Tanzu Application Platform v1.9

VMware Tanzu Application Platform 1.9

vmware by Broadcom

You can find the most up-to-date technical documentation on the VMware by Broadcom website at:

https://docs.vmware.com/

VMware by Broadcom
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

Contents

Overview of API portal for VMware Tanzu 802
Overview 802
Getting started 802

Overview of validated plug-ins for Tanzu Developer Portal 2153

Tanzu Application Platform v1.9

VMware Tanzu Application Platform (commonly known as TAP) is an application development platform with a rich set of developer tools. It offers developers a paved path to production to build and deploy software quickly and securely on any compliant public cloud or on-premises Kubernetes cluster.

Important

Tanzu Application Platform v1.9 is a short-term support release. It is no longer supported under the standard VMware Tanzu Support Life Cycle Policy.

To stay up to date with the latest software and security updates, upgrade to a supported version. For supported upgrade paths, see the topic *Upgrade Tanzu Application Platform* in the version you want to upgrade to.

Tanzu Application Platform overview

Tanzu Application Platform:

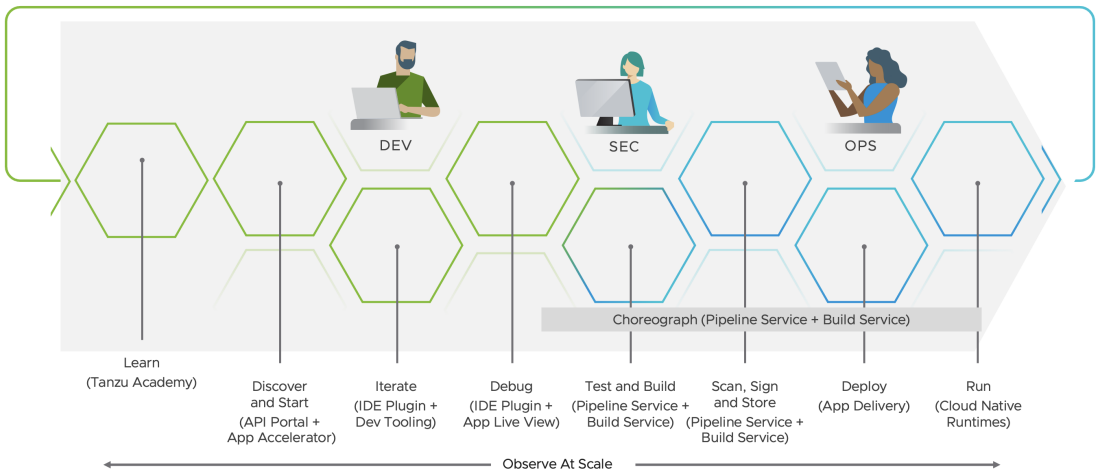
- Delivers a superior developer experience for enterprises building and deploying cloud-native applications on Kubernetes.
- Allows developers to quickly build and test applications regardless of their familiarity with Kubernetes.
- Helps application teams get to production faster by automating source-to-production pipelines.
- Clearly defines the roles of developers and operators so they can work collaboratively and integrate their efforts.

Operations teams can create application scaffolding templates with built-in security and compliance guardrails, making those considerations mostly invisible to developers. Starting with the templates, developers turn source code into a container and get a URL to test their app in minutes.

After the container is built, it updates every time there's a new code commit or dependency patch. An internal API management portal facilitates connecting to other applications and data, regardless of how they're built or the infrastructure they run on.

Simplified workflows

When creating supply chains, you can simplify workflows in both the inner and outer loop of Kubernetes-based app development with Tanzu Application Platform.



- **Inner Loop**
  - The inner loop describes a developer's development cycle of iterating on code.
  - Inner loop activities include coding, testing, and debugging before making a commit.
  - On cloud-native or Kubernetes platforms, developers in the inner loop often build container images and connect their apps to all necessary services and APIs to deploy them to a development environment.
- **Outer Loop**
  - The outer loop describes how operators deploy apps to production and maintain them over time.
  - On a cloud-native platform, outer loop activities include:
    - Building container images.
    - Adding container security.
    - Configuring continuous integration and continuous delivery (CI/CD) pipelines.
  - Outer loop activities are challenging in a Kubernetes-based development environment. App delivery platforms are constructed from various third-party and open source components with numerous configuration options.
- **Supply Chains and choreography**
  - Tanzu Application Platform uses the choreography pattern inherited from the context of microservices^1 and applies it to CI/CD to create a path to production.^2

Supply chains provide a way of codifying all of the steps of your path to production, or what is more commonly known as CI/CD. A supply chain differs from CI/CD in that with a supply chain, you can add every step necessary for an application to reach production or a lower environment.

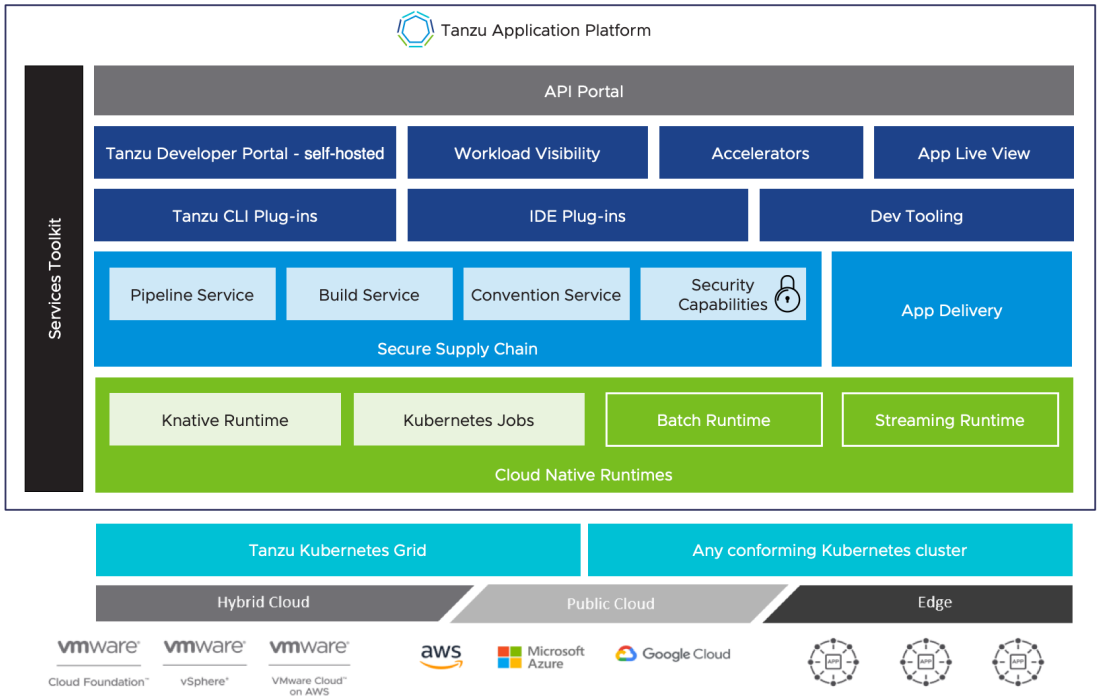
To address the developer experience gap, the path to production allows users to create a unified access point for all of the tools required for their applications to reach a customer-facing environment.

Instead of having separate tools that are loosely coupled to each other for testing and building, security, deploying, and running apps, a path to production defines all four tools in a single, unified layer of abstraction. Where tools typically can't integrate with one another and additional scripting

or webhooks are necessary, a unified automation tool codifies all interactions between each of the tools.

Tanzu Application Platform provides a default set of components that automates pushing an app to staging and production on Kubernetes. This removes the pain points for both inner and outer loops. It also allows operators to customize the platform by replacing Tanzu Application Platform components with other products.

The following diagram shows the layered API and capabilities of Tanzu Application Platform.



For more information about Tanzu Application Platform components, see Components and installation profiles.

Notice of telemetry collection for Tanzu Application Platform

Tanzu Application Platform participates in the VMware Customer Experience Improvement Program (CEIP). As part of CEIP, VMware collects technical information about your organization's use of VMware products and services in association with your organization's VMware license keys. For information about CEIP, see the Trust & Assurance Center. You may join or leave CEIP at any time. The CEIP Standard Participation Level provides VMware with information to improve its products and services, identify and fix problems, and advise you on how to best deploy and use VMware products. For example, this information can enable a proactive product deployment discussion with your VMware account team or VMware support team to help resolve your issues. This information cannot directly identify any individual.

You must acknowledge that you have read the VMware CEIP policy before you can proceed with the installation. For more information, see Install your Tanzu Application Platform profile. To opt out of telemetry participation after installation, see Opting out of telemetry collection.

Tanzu Application Platform release notes

This topic contains release notes for Tanzu Application Platform v1.9.

Important

Tanzu Application Platform v1.9 is a short-term support release. It is no longer supported under the standard VMware Tanzu Support Life Cycle Policy.

To stay up to date with the latest software and security updates, upgrade to a supported version. For supported upgrade paths, see the topic *Upgrade Tanzu Application Platform* in the version you want to upgrade to.

v1.9.1

Release Date: 09 May 2024

v1.9.1 Security fixes

This release has the following security fixes, listed by component and area.

Package Name	Vulnerabilities Resolved
base-jammy-stack-lite.buildpacks.tanzu.vmware.com	▼ Expand to see the list • CVE-2024-2961 • CVE-2024-28835 • CVE-2024-28834 • CVE-2024-28085 • CVE-2024-26633 • CVE-2024-26631 • CVE-2024-26598 • CVE-2024-26597 • CVE-2024-26591 • CVE-2024-26589 • CVE-2024-26586 • CVE-2024-26581 • CVE-2024-24860 • CVE-2024-2398 • CVE-2024-23851 • CVE-2024-23850 • CVE-2024-22705 • CVE-2023-52612 • CVE-2023-52610 • CVE-2023-52609 • CVE-2023-52603 • CVE-2023-52600 • CVE-2023-52480 • CVE-2023-52470 • CVE-2023-52469 • CVE-2023-52467 • CVE-2023-52464 • CVE-2023-52463 • CVE-2023-52462 • CVE-2023-52458 • CVE-2023-52457 • CVE-2023-52456 • CVE-2023-52454 • CVE-2023-52451 • CVE-2023-52449 • CVE-2023-52448 • CVE-2023-52445 • CVE-2023-52444 • CVE-2023-52443 • CVE-2023-52442 • CVE-2023-52441 • CVE-2023-52436

Package Name	Vulnerabilities Resolved
	• CVE-2023-52429 • CVE-2023-52340 • CVE-2023-46838 • CVE-2023-3867 • CVE-2023-38431 • CVE-2023-38430 • CVE-2023-38427 • CVE-2023-32258 • CVE-2023-32254 • CVE-2023-24023 • CVE-2023-1194
cert-manager.tanzu.vmware.com	▼ Expand to see the list • GHSA-8r3f-844c-mc37 • GHSA-45x7-px36-x8w8

Package Name	Vulnerabilities Resolved
conventions.component.apps.tanzu.vmware.com	▼ Expand to see the list • CVE-2024-2961 • CVE-2024-28835 • CVE-2024-28834 • CVE-2024-28182 • CVE-2024-28085 • CVE-2024-26633 • CVE-2024-26631 • CVE-2024-26598 • CVE-2024-26597 • CVE-2024-26591 • CVE-2024-26589 • CVE-2024-26586 • CVE-2024-26581 • CVE-2024-24860 • CVE-2024-2398 • CVE-2024-23851 • CVE-2024-23850 • CVE-2024-22705 • CVE-2023-52612 • CVE-2023-52610 • CVE-2023-52609 • CVE-2023-52603 • CVE-2023-52600 • CVE-2023-52480 • CVE-2023-52470 • CVE-2023-52469 • CVE-2023-52467 • CVE-2023-52464 • CVE-2023-52463 • CVE-2023-52462 • CVE-2023-52458 • CVE-2023-52457 • CVE-2023-52456 • CVE-2023-52454 • CVE-2023-52451 • CVE-2023-52449 • CVE-2023-52448 • CVE-2023-52445 • CVE-2023-52444 • CVE-2023-52443 • CVE-2023-52442 • CVE-2023-52441

Package Name	Vulnerabilities Resolved
	• CVE-2023-52436 • CVE-2023-52429 • CVE-2023-52340 • CVE-2023-46838 • CVE-2023-3867 • CVE-2023-38431 • CVE-2023-38430 • CVE-2023-38427 • CVE-2023-32258 • CVE-2023-32254 • CVE-2023-24023 • CVE-2023-1194
dotnet-core-lite.buildpacks.tanzu.vmware.com	▼ Expand to see the list • GHSA-xw73-rw38-6vjc • GHSA-mq39-4gv4-mvpx • GHSA-8r3f-844c-mc37 • CVE-2024-29018

Package Name	Vulnerabilities Resolved
git-writer.component.apps.tanzu.vmware.com	▼ Expand to see the list • GHSA-8r3f-844c-mc37 • CVE-2024-2961 • CVE-2024-28835 • CVE-2024-28834 • CVE-2024-28085 • CVE-2024-26633 • CVE-2024-26631 • CVE-2024-26598 • CVE-2024-26597 • CVE-2024-26591 • CVE-2024-26589 • CVE-2024-26586 • CVE-2024-24860 • CVE-2024-2398 • CVE-2024-23851 • CVE-2024-23850 • CVE-2024-22705 • CVE-2023-52612 • CVE-2023-52610 • CVE-2023-52609 • CVE-2023-52480 • CVE-2023-52470 • CVE-2023-52469 • CVE-2023-52467 • CVE-2023-52464 • CVE-2023-52463 • CVE-2023-52462 • CVE-2023-52458 • CVE-2023-52457 • CVE-2023-52456 • CVE-2023-52454 • CVE-2023-52451 • CVE-2023-52449 • CVE-2023-52448 • CVE-2023-52445 • CVE-2023-52444 • CVE-2023-52443 • CVE-2023-52442 • CVE-2023-52441 • CVE-2023-52436 • CVE-2023-52429 • CVE-2023-52340

Package Name	Vulnerabilities Resolved
	• CVE-2023-46838 • CVE-2023-3867 • CVE-2023-38431 • CVE-2023-38430 • CVE-2023-38427 • CVE-2023-32258 • CVE-2023-32254 • CVE-2023-1194
managed-resource-controller.apps.tanzu.vmware.com	▼ Expand to see the list • GHSA-w2h3-vvvq-3m53 • GHSA-qppj-fm5r-hxr3 • GHSA-m425-mq94-257g • GHSA-45x7-px36-x8w8 • CVE-2023-39326
nodejs-lite.buildpacks.tanzu.vmware.com	▼ Expand to see the list • GHSA-xw73-rw38-6vjc • GHSA-mq39-4gv4-mvpx • GHSA-8r3f-844c-mc37 • CVE-2024-29018

Package Name	Vulnerabilities Resolved
ootb-templates.tanzu.vmware.com	▼ Expand to see the list • GHSA-mw99-9chc-xw7r • GHSA-45x7-px36-x8w8 • GHSA-449p-3h89-pw88 • CVE-2024-28757 • CVE-2024-24855 • CVE-2024-22365 • CVE-2024-1086 • CVE-2024-1085 • CVE-2024-0646 • CVE-2024-0641 • CVE-2024-0607 • CVE-2024-0567 • CVE-2024-0565 • CVE-2024-0553 • CVE-2024-0340 • CVE-2024-0193 • CVE-2023-7192 • CVE-2023-6932 • CVE-2023-6931 • CVE-2023-6918 • CVE-2023-6817 • CVE-2023-6622 • CVE-2023-6606 • CVE-2023-6546 • CVE-2023-6176 • CVE-2023-6121 • CVE-2023-6040 • CVE-2023-6039 • CVE-2023-6004 • CVE-2023-5981 • CVE-2023-5717 • CVE-2023-5363 • CVE-2023-52425 • CVE-2023-5197 • CVE-2023-5178 • CVE-2023-51782 • CVE-2023-51781 • CVE-2023-51780 • CVE-2023-51779 • CVE-2023-5158 • CVE-2023-5156 • CVE-2023-4921

Package Name	Vulnerabilities Resolved
	• CVE-2023-4911 • CVE-2023-4881 • CVE-2023-48795 • CVE-2023-4813 • CVE-2023-4806 • CVE-2023-47038 • CVE-2023-46862 • CVE-2023-46813 • CVE-2023-4641 • CVE-2023-46343 • CVE-2023-4623 • CVE-2023-4622 • CVE-2023-46218 • CVE-2023-45871 • CVE-2023-45862 • CVE-2023-4569 • CVE-2023-44466 • CVE-2023-42756 • CVE-2023-42755 • CVE-2023-42754 • CVE-2023-42753 • CVE-2023-42752 • CVE-2023-4273 • CVE-2023-4244 • CVE-2023-4208 • CVE-2023-4207 • CVE-2023-4206 • CVE-2023-4194 • CVE-2023-4155 • CVE-2023-4147 • CVE-2023-4134 • CVE-2023-4132 • CVE-2023-4128 • CVE-2023-40283 • CVE-2023-4016 • CVE-2023-4015 • CVE-2023-4004 • CVE-2023-3995 • CVE-2023-39804 • CVE-2023-39198 • CVE-2023-39197 • CVE-2023-39194 • CVE-2023-39193

Package Name	Vulnerabilities Resolved
	• CVE-2023-39192 • CVE-2023-39189 • CVE-2023-3866 • CVE-2023-3865 • CVE-2023-3863 • CVE-2023-38546 • CVE-2023-38545 • CVE-2023-38432 • CVE-2023-38429 • CVE-2023-38428 • CVE-2023-38426 • CVE-2023-3817 • CVE-2023-3777 • CVE-2023-3776 • CVE-2023-3773 • CVE-2023-3772 • CVE-2023-37453 • CVE-2023-3611 • CVE-2023-3610 • CVE-2023-3609 • CVE-2023-36054 • CVE-2023-35829 • CVE-2023-35828 • CVE-2023-35827 • CVE-2023-35824 • CVE-2023-35823 • CVE-2023-35001 • CVE-2023-3446 • CVE-2023-3439 • CVE-2023-34324 • CVE-2023-34319 • CVE-2023-34256 • CVE-2023-3390 • CVE-2023-3389 • CVE-2023-3338 • CVE-2023-33288 • CVE-2023-33203 • CVE-2023-3268 • CVE-2023-32257 • CVE-2023-32252 • CVE-2023-32250 • CVE-2023-32248 • CVE-2023-32247

Package Name	Vulnerabilities Resolved
	• CVE-2023-3212 • CVE-2023-3141 • CVE-2023-31248 • CVE-2023-3117 • CVE-2023-31085 • CVE-2023-31084 • CVE-2023-31083 • CVE-2023-3090 • CVE-2023-30772 • CVE-2023-2975 • CVE-2023-2953 • CVE-2023-2898 • CVE-2023-28466 • CVE-2023-28322 • CVE-2023-28321 • CVE-2023-25775 • CVE-2023-23004 • CVE-2023-23000 • CVE-2023-22995 • CVE-2023-2269 • CVE-2023-2235 • CVE-2023-2194 • CVE-2023-2163 • CVE-2023-2156 • CVE-2023-21400 • CVE-2023-21255 • CVE-2023-2124 • CVE-2023-20593 • CVE-2023-20588 • CVE-2023-20569 • CVE-2023-2002 • CVE-2023-1990 • CVE-2023-1855 • CVE-2023-1611 • CVE-2023-1206 • CVE-2023-1192 • CVE-2023-0597 • CVE-2022-48522 • CVE-2022-48502 • CVE-2022-48425 • CVE-2022-48065 • CVE-2022-48063 • CVE-2022-47695

Package Name	Vulnerabilities Resolved
	• CVE-2022-47011 • CVE-2022-47010 • CVE-2022-47008 • CVE-2022-47007 • CVE-2022-45919 • CVE-2022-45886 • CVE-2022-45703 • CVE-2022-44840 • CVE-2022-4285 • CVE-2022-4269 • CVE-2022-41715 • CVE-2022-40982 • CVE-2022-3715 • CVE-2022-35205 • CVE-2022-32148 • CVE-2022-30629 • CVE-2022-29526 • CVE-2022-1962 • CVE-2022-1705
ruby-lite.buildpacks.tanzu.vmware.com	▼ Expand to see the list • GHSA-xw73-rw38-6vjc • GHSA-mq39-4gv4-mvpx • GHSA-8r3f-844c-mc37 • CVE-2024-29018
servicebinding.tanzu.vmware.com	▼ Expand to see the list • CVE-2024-24785 • CVE-2024-24784 • CVE-2024-24783 • CVE-2023-45290 • CVE-2023-45289
services-toolkit.tanzu.vmware.com	▼ Expand to see the list • CVE-2024-24785 • CVE-2024-24784 • CVE-2024-24783 • CVE-2023-45290 • CVE-2023-45289

Package Name	Vulnerabilities Resolved
spring-cloud-gateway.tanzu.vmware.com	▼ Expand to see the list • GHSA-hgjh-9rj2-g67j • CVE-2024-28835 • CVE-2024-28834 • CVE-2024-28757 • CVE-2024-28085 • CVE-2023-52425 • CVE-2022-3715
sso.apps.tanzu.vmware.com	▼ Expand to see the list • GHSA-hgjh-9rj2-g67j • GHSA-2wrp-6fg6-hmc5

Package Name	Vulnerabilities Resolved
supply-chain-catalog.apps.tanzu.vmware.com	▼ Expand to see the list • GHSA-xw73-rw38-6vjc • CVE-2024-2961 • CVE-2024-28835 • CVE-2024-28834 • CVE-2024-28182 • CVE-2024-28085 • CVE-2024-26633 • CVE-2024-26631 • CVE-2024-26598 • CVE-2024-26597 • CVE-2024-26591 • CVE-2024-26589 • CVE-2024-26586 • CVE-2024-26581 • CVE-2024-24860 • CVE-2024-24785 • CVE-2024-24784 • CVE-2024-24783 • CVE-2024-2398 • CVE-2024-23851 • CVE-2024-23850 • CVE-2024-22705 • CVE-2023-52612 • CVE-2023-52610 • CVE-2023-52609 • CVE-2023-52603 • CVE-2023-52600 • CVE-2023-52480 • CVE-2023-52470 • CVE-2023-52469 • CVE-2023-52467 • CVE-2023-52464 • CVE-2023-52463 • CVE-2023-52462 • CVE-2023-52458 • CVE-2023-52457 • CVE-2023-52456 • CVE-2023-52454 • CVE-2023-52451 • CVE-2023-52449 • CVE-2023-52448 • CVE-2023-52445

Package Name	Vulnerabilities Resolved
	• CVE-2023-52444 • CVE-2023-52443 • CVE-2023-52442 • CVE-2023-52441 • CVE-2023-52436 • CVE-2023-52429 • CVE-2023-52340 • CVE-2023-46838 • CVE-2023-45290 • CVE-2023-45289 • CVE-2023-45288 • CVE-2023-3867 • CVE-2023-38431 • CVE-2023-38430 • CVE-2023-38427 • CVE-2023-32258 • CVE-2023-32254 • CVE-2023-24023 • CVE-2023-1194
supply-chain.apps.tanzu.vmware.com	▼ Expand to see the list • GHSA-qppj-fm5r-hxr3 • GHSA-m425-mq94-257g • GHSA-45x7-px36-x8w8 • CVE-2023-39326

Package Name	Vulnerabilities Resolved
tap-gui.tanzu.vmware.com	▼ Expand to see the list • CVE-2024-28835 • CVE-2024-28834 • CVE-2024-26633 • CVE-2024-26631 • CVE-2024-26598 • CVE-2024-26591 • CVE-2024-26589 • CVE-2024-26586 • CVE-2024-24860 • CVE-2024-23851 • CVE-2024-23850 • CVE-2024-22705 • CVE-2023-52612 • CVE-2023-52610 • CVE-2023-52609 • CVE-2023-52480 • CVE-2023-52470 • CVE-2023-52469 • CVE-2023-52467 • CVE-2023-52464 • CVE-2023-52463 • CVE-2023-52462 • CVE-2023-52458 • CVE-2023-52457 • CVE-2023-52456 • CVE-2023-52454 • CVE-2023-52451 • CVE-2023-52449 • CVE-2023-52448 • CVE-2023-52445 • CVE-2023-52444 • CVE-2023-52443 • CVE-2023-52442 • CVE-2023-52441 • CVE-2023-52436 • CVE-2023-52429 • CVE-2023-52340 • CVE-2023-46838 • CVE-2023-3867 • CVE-2023-38431 • CVE-2023-38430 • CVE-2023-38427

Package Name	Vulnerabilities Resolved
	• CVE-2023-32258 • CVE-2023-32254 • CVE-2023-1194
web-servers-lite.buildpacks.tanzu.vmware.com	▼ Expand to see the list • GHSA-mq39-4gv4-mvpx • CVE-2024-29018

v1.9.1 Resolved issues

The following issues, listed by component and area, are resolved in this release.

v1.9.1 Resolved issues: Supply Chain

- The **Workload** page in the Supply Chain UI now loads at a normal speed when there are more than 100 workloads to display.
- The **Workload Details** page, accessed by clicking on a workload name, now loads in the Supply Chain UI when there are more than 100 workloads to display.

v1.9.1 Resolved issues: Tanzu Developer Portal

- For the Runtime Resource View plug-in, resources all across the cluster are no longer needlessly queried when a Backstage namespace annotation is specified on the entity.

v1.9.1 Known issues

This release has the following known issues, listed by component and area.

v1.9.1 Known issues: Tanzu Application Platform

- On Azure Kubernetes Service (AKS), the Datadog Cluster Agent cannot reconcile the webhook, which leads to an error. For troubleshooting information, see Datadog agent cannot reconcile webhook on AKS.
- The Tanzu Application Platform integration with Tanzu Service Mesh does not work on vSphere with TKR v1.26. For more information about this integration, see Set up Tanzu Service Mesh. As a workaround, you can apply the label to update pod security on a TKr v1.26 Kubernetes namespace as advised by the release notes for TKr 1.26.5 for vSphere 8.x. However, applying this label provides more than the minimum necessary privilege to the resources in developer namespaces.

v1.9.1 Known issues: API Auto Registration

- Registering conflicting `groupId` and `version` with API portal:

  If you create two `CuratedAPIDescriptor`s with the same `groupId` and `version` combination, both reconcile without throwing an error, and the `/openapi?groupId&version` endpoint returns both specifications. If you are adding both specifications to the API portal, only one of them might show up in the API portal UI with a warning indicating that there is a conflict. If you add the route provider annotation for both of the `CuratedAPIDescriptor`s to use Spring Cloud Gateway, the generated API specspecification includes API routes from both `CuratedAPIDescriptor`s.

You can see the `groupId` and `version` information from all `CuratedAPIDescriptor`s by running:

```
$ kubectl get curatedapidescriptors -A

NAMESPACE           NAME         GROUPID            VERSION   STATUS   CURATED
API SPEC URL
my-apps             petstore     test-api-group     1.2.3     Ready    http://A
AR-CONTROLLER-FQDN/openapi/my-apps/petstore
default             mystery      test-api-group     1.2.3     Ready    http://A
AR-CONTROLLER-FQDN/openapi/default/mystery
```

- When creating an `APIDescriptor` with different `apiSpec.url` and `server.url`, the controller incorrectly uses the API spec URL as the server URL. To avoid this issue, use `server.url` only.

v1.9.1 Known issues: App Last Mile Catalog

- The app-config-web, app-config-server, and app-config-worker components do not allow developers to override the default application ports. This means that applications that use non-standard ports do not work. To work around this, you can configure ports by providing values to the resulting Carvel package. This issue is planned to be fixed in a future release.

v1.9.1 Known issues: Application Live View

- On the Run profile, Application Live View fails to reconcile if you use a non-default cluster issuer while installing through Tanzu Mission Control.

v1.9.1 Known issues: Artifact Metadata Repository Observer and CloudEvent Handler

- Periodic reconciliation or restarting of the AMR Observer causes reattempted posting of ImageVulnerabilityScan results. There is an error on duplicate submission of identical ImageVulnerabilityScans you can ignore if the previous submission was successful.

v1.9.1 Known issues: Bitnami Services

- If you try to configure private registry integration for the Bitnami Services after having already created a claim for one or more of the services using the default configuration, the updated private registry configuration does not appear to take effect. This is due to caching behavior in the system which is not accounted for during configuration updates. For a workaround, see Troubleshoot Bitnami Services.

v1.9.1 Known issues: Cartographer Conventions

- Before Tanzu Application Platform v1.9, the `cartographer.tanzu.vmware.com` package contained two products: Cartographer and Cartographer Conventions. In Tanzu Application Platform v1.9.0 the Cartographer Conventions product is removed from the `cartographer.tanzu.vmware.com` package and is distributed in its own package named `cartographer.conventions.apps.tanzu.vmware.com`.

  When you upgrade to Tanzu Application Platform v1.9, an issue might occur when installing the new package for Cartographer Conventions. The upgrade might fail to reconcile and show error messages similar to the following:

  ```
  Resource 'clusterrole/cartographer-conventions-manager-role (rbac.authorizatio
  n.k8s.io/v1) cluster' is already associated with a different app 'cartographer.
  ```

```
app'
```

This message might appear more than once, and it can refer to several resources.

These errors appear when kapp-controller on the cluster tries to install the new Cartographer Conventions package before the Cartographer package has reconciled. The new package for Cartographer Conventions tries to install resources that the existing Cartographer package still owns.

Although it looks like the upgrade fails, if you wait a few minutes, kapp-controller finishes the installation and the packages will reconcile successfully. The system works normally after the reconciliation is complete.

This error does not occur on a new installation of Tanzu Application Platform v1.9.

- While processing workloads with large SBOMs, the Cartographer Convention controller manager pod can fail with the status `CrashLoopBackOff` or `OOMKilled`. For information about how to increase the memory limit for both the convention server and webhook servers, including app-live-view-conventions, spring-boot-webhook, and developer-conventions/webhook, see Troubleshoot Cartographer Conventions.

v1.9.1 Known issues: Crossplane

- The Crossplane `validatingwebhookconfiguration` is not removed when you uninstall the Crossplane package. To workaround, delete the `validatingwebhookconfiguration` manually by running `kubectl delete validatingwebhookconfiguration crossplane`.

v1.9.1 Known issues: Services Toolkit

- An error occurs if `additionalProperties` is `true` in a CompositeResourceDefinition. For more information and a workaround, see Troubleshoot Services Toolkit.

v1.9.1 Known issues: Supply Chain

- Components cannot have more than one resumption defined. When there are multiple resumptions, `WorkloadRuns` are not correctly created after resumptions trigger changes. The current workaround is to assess all triggers in a single resumption.
- Tanzu Supply Chain currently does not include support for Red Hat OpenShift. This means you cannot individually install components for Tanzu Supply Chain and Managed Resource Controller. You also cannot install the Authoring profile that includes those components as standard. Support for Red Hat OpenShift is planned for a later release.
- Tanzu Supply Chain currently does not include support for CA certificates in the Out of the Box components. However, you can edit the components to support CA certificates and use them to construct a new Supply Chain. Support for CA certificates as standard is planned for future versions of Tanzu Supply Chain.
- When you select the **Supply Chains** tab in Tanzu Developer Portal, you might encounter an error related to `data.packaging.carvel.dev`. The error message is related to permission issues and JSON parsing errors. The error message indicates that the user `system:serviceaccount:tap-gui:tap-gui-viewer` cannot list resource `packages` in the API group `data.packaging.carvel.dev` at the cluster scope. Additionally, an unexpected non-whitespace character is reported after JSON at position 4.

  As a temporary workaround, apply an RBAC configuration that includes the get, watch, and list permissions for the resources in the `data.packaging.carvel.dev` API group. This workaround must not be mandated for supply chains that do not generate Carvel packages.

To eliminate the error message, configure RBAC to allow access to the Carvel package resource as follows:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
- apiGroups: [data.packaging.carvel.dev]
 resources: [packages]
 verbs: ['get', 'watch', 'list']
```

- In the **Workload Details** page in the Supply Chain UI, the config writer step takes longer than 20 seconds to load when more than 149 workloads are displayed in the Supply Chain UI.

v1.9.1 Known issues: Supply Chain Choreographer

- The template for the `external-deliverable-template` does not respect the `gitops_credentials_secret` parameter. The value is not present on the deliverable if it is provided in the workload parameter `gitops_credentials_secret` or the supply chain tap-value `ootb_supply_chain*.gitops.credentials_secret`. As a workaround, operators must provide the value as a tap-value for the delivery: `ootb_delivery_basic.source.credentials_secret`. The supply chain's GitOps credentials must authenticate to the same repository as the delivery's source credentials. If a deliverable must use a secret different from that specified by the delivery tap-value, the deliverable must be manually altered when being copied to the Run cluster. Add the secret name as a `source_credentials_secret` parameter on the deliverable.
- By default, Server Workload Carvel packages generated by the Carvel package supply chains no longer contain OpenAPIv3 descriptions of their parameters. These descriptions were omitted to keep the size of the Carvel Package definition under 4 KB, which is the size limit for the string output of a Tekton Task. For information about these parameters, see Carvel Package Supply Chains.
- When using the Carvel Package Supply Chains, if the operator updates the parameter `carvel_package.name_suffix`, existing workloads incorrectly output a Carvel package to the GitOps repository that uses the old value of `carvel_package.name_suffix`. You can ignore or delete this package.
- If the size of the resulting OpenAPIv3 specification exceeds a certain size, approximately 3 KB, the Supply Chain does not function. If you use the default Carvel package parameters, this issue does not occur. If you use custom Carvel package parameters, you might encounter this size limit. If you exceed the size limit, you can either deactivate this feature, or use a workaround. The workaround requires enabling a Tekton feature flag. For more information, see the Tekton documentation.

v1.9.1 Known issues: Supply Chain Security Tools - Policy

- Supply Chain Security Tools - Policy defaults to The Update Framework (TUF) enabled due to incorrect logic. This might cause the package to not reconcile correctly if the default TUF mirrors are not reachable. To work around this, explicitly configure policy controller in the `tap-values.yaml` file to enable TUF:

```
policy:
  tuf_enabled: true
```

v1.9.1 Known issues: Supply Chain Security Tools - Scan

- When using Supply Chain Security Tools (SCST) - Scan 2.0 with a ClusterImageTemplate, the value for the scanning image is overwritten with an incorrect default value from `ootb_supply_chain_testing_scanning.image_scanner_cli` in the `tap-values.yaml` file for templates other than Trivy. You can prevent this by setting the value in your `tap-values.yaml` file to the correct image. For example, for the Grype image packaged with Tanzu Application Platform:

  ```
  ootb_supply_chain_testing_scanning:
    image_scanner_template_name: image-vulnerability-scan-grype
    image_scanning_cli:
      image: registry.tanzu.vmware.com/tanzu-application-platform/tap-packages@sh
  a256:feb1cdbd5c918aae7a89bdb2aa39d486bf6ffc81000764b522842e5934578497
  ```

- The Snyk scanner outputs an incorrectly created date, resulting in an invalid date. If the workload is in a failed state due to an invalid date, wait approximately 10 hours and the workload automatically goes into the ready state. For more about this issue information, see the Snyk GitHub repository.
- Recurring scan has a maximum of approximately 5000 container images that can be scanned at a single time due to size limits configMaps.
- If the supply chain container image scanning is configured to use a different scanner or scanner version than the recurring scanning, the vulnerabilities displayed in Tanzu Developer Portal might be inaccurate.
- SCST - Scan 1.0 fails with the error `secrets 'store-ca-cert' not found` during deployment by using Tanzu Mission Control with a non-default issuer. For how to work around this issue, see Deployment failure with non-default issuer.

v1.9.1 Known issues: Supply Chain Security Tools - Store

- SCST - Store returns an expired certificate error message when a CA certificate expires before the app certificate. For more information, see CA Cert expires.
- When outputting CycloneDX v1.5 SBOMs, the report is found to be an invalid SBOM by CycloneDX validators. This issue is planned to be fixed in a future release.
- SCST - Store automatically detects PostgreSQL database index corruptions. If SCST - Store finds a PostgresSQL database index has been corrupted, SCST - Store will automatically attempt to repair, which might cause reconciliation during package updates. When this happens, the included Postgres database might take some time to complete the repair and accept connections. For more information, see Fix Postgres Database Index Corruption.
- If CA Certificate data is included in the shared Tanzu Application Platform values section, do not configure AMR Observer with CA Certificate data.
- When `observer.deploy_through_tmc` is `true`, properties are auto-configured for Tanzu Mission Control (TMC). This causes the `MultiClusterPropertyCollector` resource to overwrite existing Tanzu Application Platform values for Observer.

  When using Let's Encrypt ACME issuers, the resultant Kubernetes secret resource does not contain a `ca.crt` property. Therefore, when the `MultiClusterPropertyCollector` resource creates the Observer package configuration values secret, the required `ca_cert_data` is empty.

  To work around this issue, add the Certificate Authority (CA) Certificate to the `shared.ca_cert_data` key in the Tanzu Application Platform installation values.

v1.9.1 Known issues: Tanzu Build Service

- During upgrades a large number of builds might be created due to buildpack and stack updates. Some of these builds might fail due to transient network issues, causing the workload to be in an unhealthy state. This resolves itself on subsequent builds after a code change and does not affect the running application.

  If you do not want to wait for subsequent builds to run, you can manually trigger a build. For instructions, see Troubleshooting.

v1.9.1 Known issues: Tanzu Developer Portal

- If you do not configure any authentication providers, and do not allow guest access, the following message appears when loading Tanzu Developer Portal in a browser:

  ```
  No configured authentication providers. Please configure at least one.
  ```

  To resolve this issue, see Troubleshooting.
- Ad-blocking browser extensions and standalone ad-blocking software can interfere with telemetry collection within the VMware Customer Experience Improvement Program and restrict access to all or parts of Tanzu Developer Portal. For more information, see Troubleshooting.
- ScmAuth is a Backstage concept that abstracts Source Code Management (SCM) authentication into a package. An oversight in a recent code-base migration led to the accidental exclusion of custom ScmAuth functions. This exclusion affected some client operations, such as using Application Accelerators to create Git repositories on behalf of users.
- The back-end Kubernetes plug-in reports failure in multicluster environments. In a multicluster environment when one request to a Kubernetes cluster fails, `backstage-kubernetes-backend` reports a failure to the front end. This is a known issue with upstream Backstage and it applies to all released versions of Tanzu Developer Portal. For more information, see this Backstage code in GitHub. This behavior arises from the API at the Backstage level. There are currently no known workarounds. There are plans for upstream commits to Backstage to resolve this issue.

v1.9.1 Known issues: Tanzu Developer Tools for IntelliJ

- The error `com.vdurmont.semver4j.SemverException: Invalid version (no major version)` is shown in the error logs when attempting to perform a workload action before installing the Tanzu CLI apps plug-in.
- If you restart your computer while running Live Update without terminating the Tilt process beforehand, there is a lock that incorrectly shows that Live Update is still running and prevents it from starting again. For the fix, see Troubleshooting.
- Workload actions and Live Update do not work when in a project with spaces in its name, such as `my app`, or in its path, such as `C:\Users\My User\my-app`. For more information, see Troubleshooting.
- An **EDT Thread Exception** error is logged or reported as a notification with a message similar to `"com.intellij.diagnostic.PluginException: 2007 ms to call on EDT TanzuApplyAction#update@ProjectViewPopup"`. For more information, see Troubleshooting.

v1.9.1 Known issues: Tanzu Developer Tools for Visual Studio

- Clicking the red square Stop button in the Visual Studio top toolbar can cause a workload to fail. For more information, see Troubleshooting.

v1.9.1 Component versions

The following table lists the Tanzu Application Platform package versions included with this release. For open source component versions in this Tanzu Application Platform release, see Open source component versions.

Component Name	Version
API Auto Registration	0.5.0
API portal	1.5.0
Application Accelerator	1.9.1
Application Configuration Service	2.3.1
Application Live View APIServer	1.9.1
Application Live View back end	1.9.1
Application Live View connector	1.9.1
Application Live View conventions	1.9.1
Application Single Sign-On	5.1.5
Artifact Metadata Repository Observer	0.5.0
AWS Services	0.3.0
Bitnami Services	0.5.0
Carbon Black Scanner for SCST - Scan (beta)	1.4.0
Cartographer Conventions	0.9.0
cert-manager	2.7.4
Cloud Native Runtimes	2.5.3
Contour	2.2.0
Crossplane	0.5.0
Default Roles	1.1.0
Developer Conventions	0.16.1
External Secrets Operator	0.9.4+tanzu.3
Flux CD Source Controller	1.1.2+tanzu.3
Grype Scanner for SCST - Scan	1.9.1
Local Source Proxy	0.2.1
Managed Resource Controller (beta)	0.2.11
Namespace Provisioner	0.6.2
Out of the Box Delivery - Basic	0.16.6
Out of the Box Supply Chain - Basic	0.16.6
Out of the Box Supply Chain - Testing	0.16.6
Out of the Box Supply Chain - Testing and Scanning	0.16.6
Out of the Box Templates	0.16.6
Service Bindings	0.12.0

Component Name	Version
Service Registry	1.3.2
Services Toolkit	0.14.0
Snyk Scanner for SCST - Scan (beta)	1.3.0
Source Controller	0.9.0
Spring Boot conventions	1.9.1
Spring Cloud Gateway	2.1.10
Supply Chain Choreographer	0.9.0
Supply Chain Security Tools - Policy Controller	1.6.4
Supply Chain Security Tools - Scan	1.9.1
Supply Chain Security Tools - Scan 2.0	0.4.0
Supply Chain Security Tools - Store	1.9.0
Tanzu Application Platform Telemetry	0.7.0
Tanzu Build Service	1.13.0
Tanzu CLI	1.3.0
Tanzu Developer Portal	1.9.2
Tanzu Developer Portal Configurator	1.9.2
Tanzu Supply Chain (beta)	0.2.31
Tekton Pipelines	0.50.3+tanzu.4

v1.9.0

Release Date: 9 April 2024

What's new in Tanzu Application Platform v1.9

This release includes the following platform-wide enhancements.

New platform-wide features

- Tanzu Application Platform v1.9 supports N-2 upgrades, which allows you to upgrade from Tanzu Application Platform v1.7.x to v1.9.x or from Tanzu Application Platform v1.8.x to v1.9.x.

v1.9.0 New features by component and area

This release includes the following changes, listed by component and area.

v1.9.0 Features: Application Accelerator

- Accelerator authors can now use IntelliJ as well as VS Code to create accelerators using the local authoring experience without connecting to a Tanzu Application Platform cluster. For more information, see Use a local Application Accelerator engine server.

- Adds the Spring AI Chat sample accelerator, which provides a sample application you can use to quickly start development of a web application for AI chat based on Spring AI. This web application offers an interactive chat experience that uses Retrieval Augmented Generation (RAG) to enable a user to ask questions about their uploaded documents. For more information, see Spring AI Chat Sample Accelerator.

v1.9.0 Features: Application Live View

- By default, Application Live View connector is deployed as a Deployment to discover applications across all namespaces running in a worker node of a Kubernetes cluster. This overrides the previous behavior where the connector was deployed as a DaemonSet, which made the Kubernetes scheduling pattern unpredictable when a node restarts. For more information, see Connector deployment modes in Application Live View.

v1.9.0 Features: Bitnami Services

- Introduces the package value `claim_namespace`, which enables you to create services in the same namespace as the originating claim. You can set this value globally or on a specific service. For more information, see Package values of Bitnami Services.

v1.9.0 Features: Buildpacks and Stacks

- Adds support for the Tanzu Standard Stack for UBI 8 to the .NET Core and Web Servers buildpacks. For more information about the Tanzu Standard Stack for UBI 8, see the Tanzu Buildpacks documentation.

v1.9.0 Features: Services Toolkit

- You can configure resource limits and requests for the Services Toolkit Controller Manager and Services Toolkit Resource Claims API Server deployments through the package values. For more information, see Scalability.

v1.9.0 Features: Tanzu Developer Portal

- Added configuration to route traffic through a specified HTTP/HTTPS proxy. This includes all outgoing requests made by Backstage and Tanzu Developer Portal. For more information, see Configure HTTP Proxy.
- The DORA plug-in now has the following changes:
    - Added the date range drop-down menu filters **Last 7 Days** and **Last 30 Days**
    - The default date range filter is now **Last 7 Days** instead of **Last 90 Days**
    - Earlier filters have more accurate names: **This Week** is now **Week to Date**, **This Month** is now **Month to Date**, and **This Quarter** is now **Quarter to Date**
    - Added percentage values to show improvements or declines in metrics when compared with the previous time period
    - Added graphs to display changes in Lead Time and Deployment Frequency metrics over time
    - Performance improvements

---

v1.9.0 Breaking changes

This release includes the following changes, listed by component and area.

v1.9.0 Breaking changes: FluxCD Source Controller

- FluxCD Source Controller no longer supports the `git_implementation` field in `GitRepository` version `v1`.

v1.9.0 Breaking changes: Services Toolkit

- The `tanzu services claims` CLI plug-in command has been removed. You must now use the `tanzu services resource-claims` command instead.
- The experimental kubectl-scp plug-in has been removed.
- The experimental multicluster APIs `*.multicluster.x-tanzu.vmware.com/v1alpha1` have been removed.
    - `apiexportrolebindings.projection.apiresources.multicluster.x-tanzu.vmware.com/v1alpha1`
    - `apiresourceimports.projection.apiresources.multicluster.x-tanzu.vmware.com/v1alpha1`
    - `clusterapigroupimports.projection.apiresources.multicluster.x-tanzu.vmware.com/v1alpha1`
    - `downstreamclusterlinks.projection.apiresources.multicluster.x-tanzu.vmware.com/v1alpha1`
    - `upstreamclusterlinks.projection.apiresources.multicluster.x-tanzu.vmware.com/v1alpha1`
    - `clusterresourceexportmonitors.replication.apiresources.multicluster.x-tanzu.vmware.com/v1alpha1`
    - `clusterresourceimportmonitors.replication.apiresources.multicluster.x-tanzu.vmware.com/v1alpha1`
    - `resourceexportmonitorbindings.replication.apiresources.multicluster.x-tanzu.vmware.com/v1alpha1`
    - `resourceimportmonitorbindings.replication.apiresources.multicluster.x-tanzu.vmware.com/v1alpha1`
    - `secretexports.replication.apiresources.multicluster.x-tanzu.vmware.com/v1alpha1`
    - `secretimports.replication.apiresources.multicluster.x-tanzu.vmware.com/v1alpha1`

v1.9.0 Breaking changes: Supply Chain Choreographer

- Supply Chain Choreographer no longer supports the `git_implementation` field in `GitRepository` version `v1`. If you use a custom supply chain, when you upgrade to Tanzu Application Platform v1.9 you must ensure that the `GitRepository` resource conforms to the `v1` API.

v1.9.0 Breaking changes: Supply Chain Security Tools - Scan

- When you configure SCST - Scan with the Metadata Store CA Certificate, you can no longer manually create the secret. Configure the secret in the `values.yaml` file. For more information, see Multicluster setup for Supply Chain Security Tools.

---

v1.9.0 Security fixes

This release has the following security fixes, listed by component and area.

Package Name	Vulnerabilities Resolved
accelerator.apps.tanzu.vmware.com	▼ Expand to see the list • GHSA-8r3f-844c-mc37 • CVE-2024-28757 • CVE-2024-26308 • CVE-2024-25710 • CVE-2024-22365 • CVE-2024-0727 • CVE-2024-0567 • CVE-2024-0553 • CVE-2023-6237 • CVE-2023-6129 • CVE-2023-5678 • CVE-2023-52425 • CVE-2023-4641 • CVE-2022-3715

Package Name	Vulnerabilities Resolved
alm-catalog.component.apps.tanzu.vmware.com	▼ Expand to see the list • GHSA-45x7-px36-x8w8 • CVE-2024-28757 • CVE-2024-28085 • CVE-2024-24855 • CVE-2024-2398 • CVE-2024-0607 • CVE-2024-0565 • CVE-2024-0340 • CVE-2023-6915 • CVE-2023-6121 • CVE-2023-52425 • CVE-2023-51782 • CVE-2023-51780 • CVE-2023-51779 • CVE-2023-50868 • CVE-2023-50387 • CVE-2023-46862 • CVE-2023-46343 • CVE-2023-4134 • CVE-2023-32247 • CVE-2023-23000 • CVE-2023-22995 • CVE-2022-48065 • CVE-2022-48063 • CVE-2022-47695 • CVE-2022-3715
apis.apps.tanzu.vmware.com	▼ Expand to see the list • GHSA-45x7-px36-x8w8 • CVE-2024-0727 • CVE-2023-6237 • CVE-2023-6129 • CVE-2023-5678 • CVE-2023-5156 • CVE-2023-4813 • CVE-2023-4806
apiserver.appliveview.tanzu.vmware.com	▼ Expand to see the list • CVE-2024-0727 • CVE-2023-6237 • CVE-2023-6129 • CVE-2023-5678 • CVE-2023-39326

Package Name	Vulnerabilities Resolved
app-scanning.apps.tanzu.vmware.com	▼ Expand to see the list • CVE-2024-28757 • CVE-2023-52425 • CVE-2022-3715
application-configuration-service.tanzu.vmware.com	▼ Expand to see the list • GHSA-6qvw-249j-h44c • CVE-2024-20926 • CVE-2024-0727 • CVE-2023-6237 • CVE-2023-6129 • CVE-2023-5678
aws.services.tanzu.vmware.com	▼ Expand to see the list • GHSA-qppj-fm5r-hxr3 • GHSA-h626-pv66-hhm7 • GHSA-45x7-px36-x8w8 • GHSA-2wrh-6pvc-2jm9 • GHSA-2q89-485c-9j2x • CVE-2024-0727 • CVE-2023-6237 • CVE-2023-6129 • CVE-2023-5678 • CVE-2023-4782 • CVE-2023-45287 • CVE-2023-39326 • CVE-2023-39319 • CVE-2023-39318 • CVE-2023-29409 • CVE-2023-29406 • CVE-2023-29403 • CVE-2023-1732

Package Name	Vulnerabilities Resolved
backend.appliveview.tanzu.vmware.com	▼ Expand to see the list • CVE-2024-24785 • CVE-2024-24784 • CVE-2024-24783 • CVE-2024-0727 • CVE-2023-6237 • CVE-2023-6129 • CVE-2023-5678 • CVE-2023-45290 • CVE-2023-45289 • CVE-2022-24823 • CVE-2021-43797 • CVE-2021-21409 • CVE-2021-21295 • CVE-2021-21290 • CVE-2014-3488
base-jammy-stack-lite.buildpacks.tanzu.vmware.com	▼ Expand to see the list • CVE-2024-28757 • CVE-2024-24855 • CVE-2024-1086 • CVE-2024-1085 • CVE-2024-0646 • CVE-2024-0607 • CVE-2024-0565 • CVE-2024-0340 • CVE-2023-6915 • CVE-2023-6121 • CVE-2023-52425 • CVE-2023-51782 • CVE-2023-51781 • CVE-2023-51780 • CVE-2023-51779 • CVE-2023-46862 • CVE-2023-46343 • CVE-2023-4134 • CVE-2023-32247 • CVE-2023-23000 • CVE-2023-22995 • CVE-2022-48065 • CVE-2022-48063 • CVE-2022-47695 • CVE-2022-3715

Package Name	Vulnerabilities Resolved
cartographer.tanzu.vmware.com	▼ Expand to see the list • GHSA-xw73-rw38-6vjc
connector.appliveview.tanzu.vmware.com	▼ Expand to see the list • GHSA-wjxj-5m7g-mg7q • GHSA-w33c-445m-f8w7 • GHSA-jjfh-589g-3hjx • GHSA-hr8g-6v94-x4m9 • GHSA-cgwf-w82q-5jrr • GHSA-6qvw-249j-h44c • GHSA-5jpm-x58v-624v • CVE-2024-29025 • CVE-2024-26308 • CVE-2024-25710 • CVE-2024-0727 • CVE-2023-6237 • CVE-2023-6129 • CVE-2023-5678 • CVE-2023-42503 • CVE-2023-35116 • CVE-2023-34462 • CVE-2022-24823 • CVE-2021-43797 • CVE-2021-21409 • CVE-2021-21295 • CVE-2021-21290 • CVE-2014-3488
contour.tanzu.vmware.com	▼ Expand to see the list • CVE-2023-6780 • CVE-2023-45284 • CVE-2023-39326
controller.source.apps.tanzu.vmware.com	▼ Expand to see the list • GHSA-xw73-rw38-6vjc • GHSA-qppj-fm5r-hxr3 • GHSA-jq35-85cj-fj4p • GHSA-8r3f-844c-mc37 • GHSA-45x7-px36-x8w8

Package Name	Vulnerabilities Resolved
conventions.appliveview.tanzu.vmware.com	▼ Expand to see the list • CVE-2024-24785 • CVE-2024-24784 • CVE-2024-24783 • CVE-2024-0727 • CVE-2023-6237 • CVE-2023-6129 • CVE-2023-5678 • CVE-2023-45290 • CVE-2023-45289 • CVE-2023-39326
conventions.component.apps.tanzu.vmware.com	▼ Expand to see the list • CVE-2024-28757 • CVE-2024-24855 • CVE-2024-1086 • CVE-2024-1085 • CVE-2024-0646 • CVE-2024-0607 • CVE-2024-0565 • CVE-2024-0340 • CVE-2023-6915 • CVE-2023-6121 • CVE-2023-52425 • CVE-2023-51782 • CVE-2023-51781 • CVE-2023-51780 • CVE-2023-51779 • CVE-2023-46862 • CVE-2023-46343 • CVE-2023-4134 • CVE-2023-32247 • CVE-2023-23000 • CVE-2023-22995 • CVE-2022-48065 • CVE-2022-48063 • CVE-2022-47695 • CVE-2022-3715

Package Name	Vulnerabilities Resolved
crossplane.tanzu.vmware.com	▼ Expand to see the list • CVE-2024-0727 • CVE-2023-6237 • CVE-2023-6129 • CVE-2023-5678 • CVE-2023-39326
fluxcd.source.controller.tanzu.vmware.com	▼ Expand to see the list • GHSA-r53h-jv2g-vpx6 • GHSA-frqx-jfcm-6jjr • GHSA-6xv5-86q9-7xr8 • GHSA-6wrf-mxfj-pf5p • GHSA-33pg-m6jh-5237 • GHSA-2wrh-6pvc-2jm9 • GHSA-2q89-485c-9j2x • CVE-2024-0727 • CVE-2023-6237 • CVE-2023-6129 • CVE-2023-5678 • CVE-2023-5363 • CVE-2023-5156 • CVE-2023-4813 • CVE-2023-4806 • CVE-2023-46737 • CVE-2023-3817 • CVE-2023-3446 • CVE-2023-2975

Package Name	Vulnerabilities Resolved
git-writer.component.apps.tanzu.vmware.com	▼ Expand to see the list • CVE-2024-28757 • CVE-2024-24855 • CVE-2024-1086 • CVE-2024-1085 • CVE-2024-0646 • CVE-2024-0607 • CVE-2024-0565 • CVE-2024-0340 • CVE-2023-6915 • CVE-2023-6121 • CVE-2023-52425 • CVE-2023-51782 • CVE-2023-51781 • CVE-2023-51780 • CVE-2023-51779 • CVE-2023-46862 • CVE-2023-4641 • CVE-2023-46343 • CVE-2023-4134 • CVE-2023-32247 • CVE-2023-23000 • CVE-2023-22995 • CVE-2022-48065 • CVE-2022-48063 • CVE-2022-47695 • CVE-2022-3715
go-lite.buildpacks.tanzu.vmware.com	▼ Expand to see the list • GHSA-qppj-fm5r-hxr3 • GHSA-mq39-4gv4-mvpx • GHSA-m5m3-46gj-wch8 • GHSA-jq35-85cj-fj4p • GHSA-hmfx-3pcx-653p • GHSA-7ww5-4wqc-m92c • GHSA-6wrf-mxfj-pf5p • GHSA-45x7-px36-x8w8 • GHSA-33pg-m6jh-5237 • GHSA-2qjp-425j-52j9 • GHSA-259w-8hf6-59c2 • CVE-2024-29018

Package Name	Vulnerabilities Resolved
java-lite.buildpacks.tanzu.vmware.com	▼ Expand to see the list • GHSA-xw73-rw38-6vjc • GHSA-8r3f-844c-mc37 • GHSA-45x7-px36-x8w8
java-native-image-lite.buildpacks.tanzu.vmware.com	▼ Expand to see the list • GHSA-45x7-px36-x8w8
metadata-store.apps.tanzu.vmware.com	▼ Expand to see the list • GHSA-qppj-fm5r-hxr3 • GHSA-m425-mq94-257g • CVE-2024-25062
ootb-supply-chain-testing-scanning.tanzu.vmware.com	▼ Expand to see the list • GHSA-xw73-rw38-6vjc • GHSA-v53g-5gjp-272r • GHSA-jw44-4f3j-q396 • GHSA-9p26-698r-w4hx • GHSA-8r3f-844c-mc37 • CVE-2024-0727 • CVE-2023-6237 • CVE-2023-6129 • CVE-2023-5678 • CVE-2023-39326
ootb-templates.tanzu.vmware.com	▼ Expand to see the list • CVE-2023-50868 • CVE-2023-50387
policy.apps.tanzu.vmware.com	▼ Expand to see the list • CVE-2024-0727 • CVE-2023-6237 • CVE-2023-6129 • CVE-2023-5678 • CVE-2023-5363 • CVE-2023-5156 • CVE-2023-4813 • CVE-2023-4806 • CVE-2023-3817 • CVE-2023-3446 • CVE-2023-2975

Package Name	Vulnerabilities Resolved
service-registry.spring.apps.tanzu.vmware.com	▼ Expand to see the list • CVE-2024-0727 • CVE-2023-6237 • CVE-2023-6129 • CVE-2023-5678
servicebinding.tanzu.vmware.com	▼ Expand to see the list • CVE-2024-0727 • CVE-2023-6237 • CVE-2023-6129 • CVE-2023-5678
services-toolkit.tanzu.vmware.com	▼ Expand to see the list • CVE-2024-0727 • CVE-2023-6237 • CVE-2023-6129 • CVE-2023-5678

Package Name	Vulnerabilities Resolved
source.component.apps.tanzu.vmware.com	▼ Expand to see the list • GHSA-8r3f-844c-mc37 • CVE-2024-28757 • CVE-2024-24855 • CVE-2024-1086 • CVE-2024-1085 • CVE-2024-0646 • CVE-2024-0607 • CVE-2024-0565 • CVE-2024-0340 • CVE-2023-6915 • CVE-2023-6121 • CVE-2023-52425 • CVE-2023-51782 • CVE-2023-51781 • CVE-2023-51780 • CVE-2023-51779 • CVE-2023-46862 • CVE-2023-4641 • CVE-2023-46343 • CVE-2023-4134 • CVE-2023-32247 • CVE-2023-23000 • CVE-2023-22995 • CVE-2022-48065 • CVE-2022-48063 • CVE-2022-47695 • CVE-2022-3715
spring-boot-conventions.tanzu.vmware.com	▼ Expand to see the list • CVE-2024-0727 • CVE-2023-6237 • CVE-2023-6129 • CVE-2023-5678 • CVE-2023-5363 • CVE-2023-5156 • CVE-2023-4813 • CVE-2023-4806 • CVE-2023-3817 • CVE-2023-3446 • CVE-2023-2975

Package Name	Vulnerabilities Resolved
spring-cloud-gateway.tanzu.vmware.com	▼ Expand to see the list • GHSA-ccgv-vj62-xf9h • GHSA-4g9r-vxhx-9pgx • GHSA-4265-ccf5-phj5 • CVE-2024-26308 • CVE-2024-25710 • CVE-2024-24785 • CVE-2024-24784 • CVE-2024-24783 • CVE-2024-24549 • CVE-2024-23672 • CVE-2023-4641 • CVE-2023-45290 • CVE-2023-45289
supply-chain-catalog.apps.tanzu.vmware.com	▼ Expand to see the list • CVE-2024-28757 • CVE-2024-24855 • CVE-2024-1086 • CVE-2024-1085 • CVE-2024-0646 • CVE-2024-0607 • CVE-2024-0565 • CVE-2024-0340 • CVE-2023-6915 • CVE-2023-6121 • CVE-2023-52425 • CVE-2023-51782 • CVE-2023-51781 • CVE-2023-51780 • CVE-2023-51779 • CVE-2023-46862 • CVE-2023-46343 • CVE-2023-4134 • CVE-2023-32247 • CVE-2023-23000 • CVE-2023-22995 • CVE-2022-48065 • CVE-2022-48063 • CVE-2022-47695 • CVE-2022-3715

Package Name	Vulnerabilities Resolved
tap-gui.tanzu.vmware.com	▼ Expand to see the list • GHSA-xvch-5gv4-984h • GHSA-ww39-953v-wcq6 • GHSA-vh95-rmgr-6w4m • GHSA-p6mc-m468-83gw • GHSA-6xwr-q98w-rvg7 • GHSA-2fc9-xpp8-2g9h • CVE-2024-28757 • CVE-2024-25062 • CVE-2024-24855 • CVE-2024-24806 • CVE-2024-22667 • CVE-2024-0985 • CVE-2024-0607 • CVE-2024-0565 • CVE-2024-0340 • CVE-2023-6915 • CVE-2023-6277 • CVE-2023-6228 • CVE-2023-6121 • CVE-2023-52425 • CVE-2023-52356 • CVE-2023-51782 • CVE-2023-51780 • CVE-2023-51779 • CVE-2023-50868 • CVE-2023-50387 • CVE-2023-49468 • CVE-2023-49467 • CVE-2023-49465 • CVE-2023-47471 • CVE-2023-46862 • CVE-2023-46343 • CVE-2023-43887 • CVE-2023-4134 • CVE-2023-32247 • CVE-2023-27103 • CVE-2023-27102 • CVE-2023-25221 • CVE-2023-24758 • CVE-2023-24757 • CVE-2023-24756 • CVE-2023-24755

Package Name	Vulnerabilities Resolved
	• CVE-2023-24754 • CVE-2023-24752 • CVE-2023-24751 • CVE-2023-23000 • CVE-2023-22995 • CVE-2022-48624 • CVE-2022-47665 • CVE-2022-43250 • CVE-2022-43249 • CVE-2022-43245 • CVE-2022-43244 • CVE-2022-3715

Package Name	Vulnerabilities Resolved
tekton.tanzu.vmware.com	▼ Expand to see the list • GHSA-9763-4f94-gfch • GHSA-7ww5-4wqc-m92c • GHSA-45x7-px36-x8w8 • GHSA-2q89-485c-9j2x • GHSA-2c7c-3mj9-8fqh • CVE-2024-28757 • CVE-2024-24855 • CVE-2024-1086 • CVE-2024-1085 • CVE-2024-0727 • CVE-2024-0646 • CVE-2024-0641 • CVE-2024-0607 • CVE-2024-0565 • CVE-2024-0340 • CVE-2024-0193 • CVE-2023-6932 • CVE-2023-6931 • CVE-2023-6915 • CVE-2023-6817 • CVE-2023-6622 • CVE-2023-6606 • CVE-2023-6237 • CVE-2023-6176 • CVE-2023-6129 • CVE-2023-6121 • CVE-2023-6040 • CVE-2023-6039 • CVE-2023-5678 • CVE-2023-52425 • CVE-2023-51782 • CVE-2023-51781 • CVE-2023-51780 • CVE-2023-51779 • CVE-2023-46862 • CVE-2023-46813 • CVE-2023-4641 • CVE-2023-46343 • CVE-2023-4134 • CVE-2023-39326 • CVE-2023-35827 • CVE-2023-34324

Package Name	Vulnerabilities Resolved
	• CVE-2023-32257 • CVE-2023-32252 • CVE-2023-32250 • CVE-2023-32247 • CVE-2023-2953 • CVE-2023-23000 • CVE-2023-22995 • CVE-2023-1732 • CVE-2022-48065 • CVE-2022-48063 • CVE-2022-47695 • CVE-2022-3715
tpb.tanzu.vmware.com	▼ Expand to see the list • GHSA-xvch-5gv4-984h • GHSA-ww39-953v-wcq6 • GHSA-vh95-rmgr-6w4m • GHSA-6xwr-q98w-rvg7
trivy.app-scanning.component.apps.tanzu.vmware.com	▼ Expand to see the list • CVE-2024-28757 • CVE-2023-52425 • CVE-2022-3715

v1.9.0 Resolved issues

The following issues, listed by component and area, are resolved in this release.

v1.9.0 Resolved issues: App Last Mile Catalog

- Resolved an issue where the Deployer component output an error message that was larger than 4 KB. This caused a Tekton error. The Deployer component now outputs a smaller error message that is human readable.

v1.9.0 Resolved issues: AWS Services

- Updated the `endpoint` key name in the binding secret for Amazon MQ (RabbitMQ) claims to `addresses` so that it matches the name that the Spring Cloud Bindings library uses. This key name change is not applied to any existing Amazon MQ (RabbitMQ) claims. If new Amazon MQ (RabbitMQ) claims still do not have the updated `addresses` key name, see Troubleshoot AWS Services.

v1.9.0 Resolved issues: Crossplane

- Fixed an issue that you might encounter if you uninstall and reinstall the Crossplane package on the same cluster. You no longer receive a TLS certificate verification error with service claims never transitioning to `READY=True`.

v1.9.0 Resolved issues: Supply Chain Choreographer

- Supply Chains that use SSH auth with the `git-writer` resource no longer fail in the gitops step.

---

v1.9.0 Known issues

This release has the following known issues, listed by component and area.

v1.9.0 Known issues: Tanzu Application Platform

- On Azure Kubernetes Service (AKS), the Datadog Cluster Agent cannot reconcile the webhook, which leads to an error. For troubleshooting information, see Datadog agent cannot reconcile webhook on AKS.
- The Tanzu Application Platform integration with Tanzu Service Mesh does not work on vSphere with TKR v1.26. For more information about this integration, see Set up Tanzu Service Mesh. As a workaround, you can apply the label to update pod security on a TKr v1.26 Kubernetes namespace as advised by the release notes for TKr 1.26.5 for vSphere 8.x. However, applying this label provides more than the minimum necessary privilege to the resources in developer namespaces.

v1.9.0 Known issues: API Auto Registration

- Registering conflicting `groupId` and `version` with API portal:

  If you create two `CuratedAPIDescriptor`s with the same `groupId` and `version` combination, both reconcile without throwing an error, and the `/openapi?groupId&version` endpoint returns both specifications. If you are adding both specifications to the API portal, only one of them might show up in the API portal UI with a warning indicating that there is a conflict. If you add the route provider annotation for both of the `CuratedAPIDescriptor`s to use Spring Cloud Gateway, the generated API specspecification includes API routes from both `CuratedAPIDescriptor`s.

  You can see the `groupId` and `version` information from all `CuratedAPIDescriptor`s by running:

  ```
  $ kubectl get curatedapidescriptors -A

  NAMESPACE           NAME         GROUPID            VERSION   STATUS   CURATED
  API SPEC URL
  my-apps             petstore     test-api-group     1.2.3     Ready    http://A
  AR-CONTROLLER-FQDN/openapi/my-apps/petstore
  default             mystery      test-api-group     1.2.3     Ready    http://A
  AR-CONTROLLER-FQDN/openapi/default/mystery
  ```

- When creating an `APIDescriptor` with different `apiSpec.url` and `server.url`, the controller incorrectly uses the API spec URL as the server URL. To avoid this issue, use `server.url` only.

v1.9.0 Known issues: App Last Mile Catalog

- The app-config-web, app-config-server, and app-config-worker components do not allow developers to override the default application ports. This means that applications that use non-standard ports do not work. To work around this, you can configure ports by providing values to the resulting Carvel package. This issue is planned to be fixed in a future release.

v1.9.0 Known issues: Application Live View

- On the Run profile, Application Live View fails to reconcile if you use a non-default cluster issuer while installing through Tanzu Mission Control.

v1.9.0 Known issues: Artifact Metadata Repository Observer and CloudEvent Handler

- Periodic reconciliation or restarting of the AMR Observer causes reattempted posting of ImageVulnerabilityScan results. There is an error on duplicate submission of identical ImageVulnerabilityScans you can ignore if the previous submission was successful.

v1.9.0 Known issues: Bitnami Services

- If you try to configure private registry integration for the Bitnami Services after having already created a claim for one or more of the services using the default configuration, the updated private registry configuration does not appear to take effect. This is due to caching behavior in the system which is not accounted for during configuration updates. For a workaround, see Troubleshoot Bitnami Services.

v1.9.0 Known issues: Cartographer Conventions

- Before Tanzu Application Platform v1.9, the `cartographer.tanzu.vmware.com` package contained two products: Cartographer and Cartographer Conventions. In Tanzu Application Platform v1.9.0 the Cartographer Conventions product is removed from the `cartographer.tanzu.vmware.com` package and is distributed in its own package named `cartographer.conventions.apps.tanzu.vmware.com`.

  When you upgrade to Tanzu Application Platform v1.9, an issue might occur when installing the new package for Cartographer Conventions. The upgrade might fail to reconcile and show error messages similar to the following:

  ```
  Resource 'clusterrole/cartographer-conventions-manager-role (rbac.authorizatio
  n.k8s.io/v1) cluster' is already associated with a different app 'cartographer.
  app'
  ```

  This message might appear more than once, and it can refer to several resources.

  These errors appear when kapp-controller on the cluster tries to install the new Cartographer Conventions package before the Cartographer package has reconciled. The new package for Cartographer Conventions tries to install resources that the existing Cartographer package still owns.

  Although it looks like the upgrade fails, if you wait a few minutes, kapp-controller finishes the installation and the packages will reconcile successfully. The system works normally after the reconciliation is complete.

  This error does not occur on a new installation of Tanzu Application Platform v1.9.

- While processing workloads with large SBOMs, the Cartographer Convention controller manager pod can fail with the status `CrashLoopBackOff` or `OOMKilled`. For information about how to increase the memory limit for both the convention server and webhook servers, including app-live-view-conventions, spring-boot-webhook, and developer-conventions/webhook, see Troubleshoot Cartographer Conventions.

v1.9.0 Known issues: Crossplane

- The Crossplane `validatingwebhookconfiguration` is not removed when you uninstall the Crossplane package. To workaround, delete the `validatingwebhookconfiguration` manually by running `kubectl delete validatingwebhookconfiguration crossplane`.

v1.9.0 Known issues: Services Toolkit

- An error occurs if `additionalProperties` is `true` in a CompositeResourceDefinition. For more information and a workaround, see Troubleshoot Services Toolkit.

v1.9.0 Known issues: Supply Chain

- Components cannot have more than one resumption defined. When there are multiple resumptions, `WorkloadRuns` are not correctly created after resumptions trigger changes. The current workaround is to assess all triggers in a single resumption.
- Tanzu Supply Chain currently does not include support for Red Hat OpenShift. This means you cannot individually install components for Tanzu Supply Chain and Managed Resource Controller. You also cannot install the Authoring profile that includes those components as standard. Support for Red Hat OpenShift is planned for a later release.
- Tanzu Supply Chain currently does not include support for CA certificates in the Out of the Box components. However, you can edit the components to support CA certificates and use them to construct a new Supply Chain. Support for CA certificates as standard is planned for future versions of Tanzu Supply Chain.
- When you select the **Supply Chains** tab in Tanzu Developer Portal, you might encounter an error related to `data.packaging.carvel.dev`. The error message is related to permission issues and JSON parsing errors. The error message indicates that the user `system:serviceaccount:tap-gui:tap-gui-viewer` cannot list resource `packages` in the API group `data.packaging.carvel.dev` at the cluster scope. Additionally, an unexpected non-whitespace character is reported after JSON at position 4.

  As a temporary workaround, apply an RBAC configuration that includes the get, watch, and list permissions for the resources in the `data.packaging.carvel.dev` API group. This workaround must not be mandated for supply chains that do not generate Carvel packages.

  To eliminate the error message, configure RBAC to allow access to the Carvel package resource as follows:

  ```
  apiVersion: rbac.authorization.k8s.io/v1
  kind: ClusterRole
  - apiGroups: [data.packaging.carvel.dev]
   resources: [packages]
   verbs: ['get', 'watch', 'list']
  ```

- The **Workload** page in the Supply Chain UI takes a long time to load when there are more than 100 workloads to display.
- The **Workload Details** page, accessed by clicking on a workload name, does not load in the Supply Chain UI when there are more than 100 workloads to display.

v1.9.0 Known issues: Supply Chain Choreographer

- The template for the `external-deliverable-template` does not respect the `gitops_credentials_secret` parameter. The value is not present on the deliverable if it is provided in the workload parameter `gitops_credentials_secret` or the supply chain tap-value `ootb_supply_chain*.gitops.credentials_secret`. As a workaround, operators must provide the value as a tap-value for the delivery: `ootb_delivery_basic.source.credentials_secret`. The supply chain's GitOps credentials must authenticate to the same repository as the delivery's source credentials. If a deliverable must use a secret different from that specified by the delivery tap-value, the

deliverable must be manually altered when being copied to the Run cluster. Add the secret name as a `source_credentials_secret` parameter on the deliverable.

- By default, Server Workload Carvel packages generated by the Carvel package supply chains no longer contain OpenAPIv3 descriptions of their parameters. These descriptions were omitted to keep the size of the Carvel Package definition under 4 KB, which is the size limit for the string output of a Tekton Task. For information about these parameters, see Carvel Package Supply Chains.
- When using the Carvel Package Supply Chains, if the operator updates the parameter `carvel_package.name_suffix`, existing workloads incorrectly output a Carvel package to the GitOps repository that uses the old value of `carvel_package.name_suffix`. You can ignore or delete this package.
- If the size of the resulting OpenAPIv3 specification exceeds a certain size, approximately 3 KB, the Supply Chain does not function. If you use the default Carvel package parameters, this issue does not occur. If you use custom Carvel package parameters, you might encounter this size limit. If you exceed the size limit, you can either deactivate this feature, or use a workaround. The workaround requires enabling a Tekton feature flag. For more information, see the Tekton documentation.

v1.9.0 Known issues: Supply Chain Security Tools - Policy

- Supply Chain Security Tools - Policy defaults to The Update Framework (TUF) enabled due to incorrect logic. This might cause the package to not reconcile correctly if the default TUF mirrors are not reachable. To work around this, explicitly configure policy controller in the `tap-values.yaml` file to enable TUF:

```
policy:
  tuf_enabled: true
```

v1.9.0 Known issues: Supply Chain Security Tools - Scan

- When using Supply Chain Security Tools (SCST) - Scan 2.0 with a ClusterImageTemplate, the value for the scanning image is overwritten with an incorrect default value from `ootb_supply_chain_testing_scanning.image_scanner_cli` in the `tap-values.yaml` file for templates other than Trivy. You can prevent this by setting the value in your `tap-values.yaml` file to the correct image. For example, for the Grype image packaged with Tanzu Application Platform:

```
ootb_supply_chain_testing_scanning:
  image_scanner_template_name: image-vulnerability-scan-grype
  image_scanning_cli:
    image: registry.tanzu.vmware.com/tanzu-application-platform/tap-packages@sh
a256:feb1cdbd5c918aae7a89bdb2aa39d486bf6ffc81000764b522842e5934578497
```

- The Snyk scanner outputs an incorrectly created date, resulting in an invalid date. If the workload is in a failed state due to an invalid date, wait approximately 10 hours and the workload automatically goes into the ready state. For more about this issue information, see the Snyk GitHub repository.
- Recurring scan has a maximum of approximately 5000 container images that can be scanned at a single time due to size limits configMaps.
- If the supply chain container image scanning is configured to use a different scanner or scanner version than the recurring scanning, the vulnerabilities displayed in Tanzu Developer Portal might be inaccurate.

- SCST - Scan 1.0 fails with the error `secrets 'store-ca-cert' not found` during deployment by using Tanzu Mission Control with a non-default issuer. For how to work around this issue, see Deployment failure with non-default issuer.

v1.9.0 Known issues: Supply Chain Security Tools - Store

- SCST - Store returns an expired certificate error message when a CA certificate expires before the app certificate. For more information, see CA Cert expires.
- When outputting CycloneDX v1.5 SBOMs, the report is found to be an invalid SBOM by CycloneDX validators. This issue is planned to be fixed in a future release.
- SCST - Store automatically detects PostgreSQL database index corruptions. If SCST - Store finds a PostgresSQL database index has been corrupted, SCST - Store will automatically attempt to repair, which might cause reconciliation during package updates. When this happens, the included Postgres database might take some time to complete the repair and accept connections. For more information, see Fix Postgres Database Index Corruption.
- When `observer.deploy_through_tmc` is `true`, properties are auto-configured for Tanzu Mission Control (TMC). This causes the `MultiClusterPropertyCollector` resource to overwrite existing Tanzu Application Platform values for Observer.

  When using Let's Encrypt ACME issuers, the resultant Kubernetes secret resource does not contain a `ca.crt` property. Therefore, when the `MultiClusterPropertyCollector` resource creates the Observer package configuration values secret, the required `ca_cert_data` is empty.

  To work around this issue, add the Certificate Authority (CA) Certificate to the `shared.ca_cert_data` key in the Tanzu Application Platform installation values.
- If CA Certificate data is included in the shared Tanzu Application Platform values section, do not configure AMR Observer with CA Certificate data.

v1.9.0 Known issues: Tanzu Build Service

- During upgrades a large number of builds might be created due to buildpack and stack updates. Some of these builds might fail due to transient network issues, causing the workload to be in an unhealthy state. This resolves itself on subsequent builds after a code change and does not affect the running application.

  If you do not want to wait for subsequent builds to run, you can manually trigger a build. For instructions, see Troubleshooting.

v1.9.0 Known issues: Tanzu Developer Portal

- Tanzu Developer Portal Configurator jumps from v1.0.x in Tanzu Application Platform v1.7 to v1.8.x in Tanzu Application Platform v1.8. This version jump enables future versions of Tanzu Developer Portal and Tanzu Developer Portal Configurator to sync going forward.
- If you do not configure any authentication providers, and do not allow guest access, the following message appears when loading Tanzu Developer Portal in a browser:

  ```
  No configured authentication providers. Please configure at least one.
  ```

  To resolve this issue, see Troubleshooting.
- Ad-blocking browser extensions and standalone ad-blocking software can interfere with telemetry collection within the VMware Customer Experience Improvement Program and

restrict access to all or parts of Tanzu Developer Portal. For more information, see Troubleshooting.

- ScmAuth is a Backstage concept that abstracts Source Code Management (SCM) authentication into a package. An oversight in a recent code-base migration led to the accidental exclusion of custom ScmAuth functions. This exclusion affected some client operations, such as using Application Accelerators to create Git repositories on behalf of users.
- The back-end Kubernetes plug-in reports failure in multicluster environments. In a multicluster environment when one request to a Kubernetes cluster fails, `backstage-kubernetes-backend` reports a failure to the front end. This is a known issue with upstream Backstage and it applies to all released versions of Tanzu Developer Portal. For more information, see this Backstage code in GitHub. This behavior arises from the API at the Backstage level. There are currently no known workarounds. There are plans for upstream commits to Backstage to resolve this issue.

v1.9.0 Known issues: Tanzu Developer Tools for IntelliJ

- The error `com.vdurmont.semver4j.SemverException: Invalid version (no major version)` is shown in the error logs when attempting to perform a workload action before installing the Tanzu CLI apps plug-in.
- If you restart your computer while running Live Update without terminating the Tilt process beforehand, there is a lock that incorrectly shows that Live Update is still running and prevents it from starting again. For the fix, see Troubleshooting.
- Workload actions and Live Update do not work when in a project with spaces in its name, such as `my app`, or in its path, such as `C:\Users\My User\my-app`. For more information, see Troubleshooting.
- An **EDT Thread Exception** error is logged or reported as a notification with a message similar to `"com.intellij.diagnostic.PluginException: 2007 ms to call on EDT TanzuApplyAction#update@ProjectViewPopup"`. For more information, see Troubleshooting.

v1.9.0 Known issues: Tanzu Developer Tools for Visual Studio

- Clicking the red square Stop button in the Visual Studio top toolbar can cause a workload to fail. For more information, see Troubleshooting.

---

v1.9.0 Component versions

The following table lists the Tanzu Application Platform package versions included with this release. For open source component versions in this Tanzu Application Platform release, see Open source component versions.

Component Name	Version
API Auto Registration	0.5.0
API portal	1.5.0
Application Accelerator	1.9.1
Application Configuration Service	2.3.1
Application Live View APIServer	1.9.1
Application Live View back end	1.9.1

Component Name	Version
Application Live View connector	1.9.1
Application Live View conventions	1.9.1
Application Single Sign-On	5.1.4
Artifact Metadata Repository Observer	0.5.0
AWS Services	0.3.0
Bitnami Services	0.5.0
Carbon Black Scanner for SCST - Scan (beta)	1.4.0
Cartographer Conventions	0.9.0
cert-manager	2.7.2
Cloud Native Runtimes	2.5.3
Contour	2.2.0
Crossplane	0.5.0
Default Roles	1.1.0
Developer Conventions	0.16.1
External Secrets Operator	0.9.4+tanzu.3
Flux CD Source Controller	1.1.2+tanzu.1
Grype Scanner for SCST - Scan	1.9.0
Local Source Proxy	0.2.1
Managed Resource Controller (beta)	0.2.1
Namespace Provisioner	0.6.2
Out of the Box Delivery - Basic	0.16.1
Out of the Box Supply Chain - Basic	0.16.1
Out of the Box Supply Chain - Testing	0.16.1
Out of the Box Supply Chain - Testing and Scanning	0.16.1
Out of the Box Templates	0.16.1
Service Bindings	0.12.0
Service Registry	1.3.2
Services Toolkit	0.14.0
Snyk Scanner for SCST - Scan (beta)	1.3.0
Source Controller	0.9.0
Spring Boot conventions	1.9.1
Spring Cloud Gateway	2.1.9
Supply Chain Choreographer	0.9.0
Supply Chain Security Tools - Policy Controller	1.6.4
Supply Chain Security Tools - Scan	1.9.1

Component Name	Version
Supply Chain Security Tools - Scan 2.0	0.4.0
Supply Chain Security Tools - Store	1.9.0
Tanzu Application Platform Telemetry	0.7.0
Tanzu Build Service	1.13.0
Tanzu CLI	1.2.0
Tanzu Developer Portal	1.9.1
Tanzu Developer Portal Configurator	1.9.1
Tanzu Supply Chain (beta)	0.2.9
Tekton Pipelines	0.50.3+tanzu.4

Deprecations

The following features, listed by component, are deprecated. Deprecated features remain on this list until they are retired from Tanzu Application Platform.

Cloud Native Runtimes deprecations

- `default_tls_secret` **config option**: After changes in this release, this config option is moved to `contour.default_tls_secret`. `default_tls_secret` is marked for removal in Cloud Native Runtimes v2.7. In the meantime, both options are supported, and `contour.default_tls_secret` takes precedence over `default_tls_secret`.
- `ingress.[internal/external].namespace` **config options**: After changes in this release, these config options are moved to `contour.[internal/external].namespace`. `ingress.[internal/external].namespace` is marked for removal in Cloud Native Runtimes v2.7. In the meantime, both options are supported, and `contour.[internal/external].namespace` takes precedence over `ingress.[internal/external].namespace`.

FluxCD Source Controller deprecations

- FluxCD Source Controller updates the `GitRepository` API from `v1beta2` to `v1`. The controller accepts resources with API versions `v1beta1` and `v1beta2`, saving them as `v1`.

Services Toolkit deprecations

- The following APIs are deprecated and are marked for removal in Tanzu Application Platform v1.11:
  - `clusterexampleusages.services.apps.tanzu.vmware.com/v1alpha1`
  - `clusterresources.services.apps.tanzu.vmware.com/v1alpha1`

Source Controller deprecations

- The Source Controller `ImageRepository` API is deprecated and is marked for removal. Use the `OCIRepository` API instead. The Flux Source Controller installation includes the `OCIRepository` API. For more information about the `OCIRepository` API, see the Flux documentation.

Tekton Pipelines deprecations

- Tekton `ClusterTask` is deprecated and marked for removal. Use the `Task` API instead. For more information, see the Tekton documentation.

Linux Kernel CVEs

Kernel level vulnerabilities are regularly identified and patched by Canonical. Tanzu Application Platform releases with available images, which might contain known vulnerabilities. When Canonical makes patched images available, Tanzu Application Platform incorporates these fixed images into future releases.

The kernel runs on your container host VM, not the Tanzu Application Platform container image. Even with a patched Tanzu Application Platform image, the vulnerability is not mitigated until you deploy your containers on a host with a patched OS. An unpatched host OS might be exploitable if the base image is deployed.

Planning and Architecture Reference

This topic tells you how to access the Tanzu Application Platform (commonly known as TAP) planning and architecture reference documentation.

The Tanzu Application Platform Planning and Architecture Reference documentation is available here.

This documentation helps you to understand the main design issues you encounter when planning your Tanzu Application Platform environment.

Important

You should review the Tanzu Application Platform Planning and Architecture Reference documentation before you install Tanzu Application Platform.

Components and installation profiles for Tanzu Application Platform

This topic lists the components you can install with Tanzu Application Platform (commonly known as TAP). You can install components as individual packages or you can install them using a profile containing a predefined group of packages.

Tanzu Application Platform components

- **API Auto Registration**

  When users deploy a workload that exposes an API, they want that API to automatically show in Tanzu Developer Portal without requiring any added manual steps.

  API Auto Registration is an automated workflow that can use a supply chain to create and manage a Kubernetes Custom Resource (CR) of type `APIDescriptor`. A Kubernetes controller reconciles the CR and updates the API entity in Tanzu Developer Portal to achieve automated API registration from workloads. You can also use API Auto Registration without supply chains by directly applying an `APIDescriptor` CR to the cluster.

- **API portal**

  API portal for VMware Tanzu enables API consumers to find APIs they can use in their own applications.

  Consumers can view detailed API documentation and try out an API to see if it meets their needs. API portal assembles its dashboard and detailed API documentation views by ingesting OpenAPI documentation from the source URLs. An API portal operator can add any number of OpenAPI source URLs to appear in a single instance.

- **Application Accelerator**

  The Application Accelerator component helps app developers and app operators create application accelerators.

  Accelerators are templates that codify best practices and ensure that important configurations and structures are in place. Developers can bootstrap their applications and get started with feature development right away.

  Application operators can create custom accelerators that reflect their desired architectures and configurations and enable fleets of developers to use them. This helps ease operator concerns about whether developers are implementing their best practices.

- **Application Configuration Service**

  Application Configuration Service provides a Kubernetes-native experience to enable the runtime configuration of existing Spring applications that were previously leveraged by using Spring Cloud Config Server.

  Application Configuration Service is compatible with the existing Git repository configuration management approach. It filters runtime configuration for any application by using slices that produce secrets.

- **Application Live View**

Application Live View is a lightweight insight and troubleshooting tool that helps application developers and application operators look inside running applications.

It is based on the concept of Spring Boot Actuators. The application provides information from inside the running processes by using endpoints (in our case, HTTP endpoints). Application Live View uses those endpoints to get the data from the application and to interact with it.

- **Application Single Sign-On**

  Application Single Sign-On enables application users to sign in to their identity provider once and be authorized and identified to access any Kubernetes-deployed workload. It is a secure and straightforward approach for developers and operators to manage access across all workloads in the enterprise.

- **Aria Operations for Applications (AOA) dashboard for Tanzu Application Platform (Beta)**

  This dashboard, powered by Aria Operations for Applications (formerly Tanzu Observability), helps platform engineers monitor the health of a given cluster by showing whether the deployed Tanzu Application Platform components are behaving as expected.

- **AWS Services**

  AWS Services provides an integration with Amazon Web Services (AWS) for Tanzu Application Platform.

  Through integration with Crossplane and Services Toolkit, you can offer services from AWS to apps teams to consume with only minimal setup and configuration required from ops teams. This makes it quick and easy to get started working with these services on Tanzu Application Platform.

- **Bitnami Services**

  Bitnami Services provides a set of services for Tanzu Application Platform backed by corresponding Bitnami Helm Charts. Through integration with Crossplane and Services Toolkit, these Bitnami Services are immediately ready for apps teams to consume, with no additional setup or configuration required from ops teams. This makes it incredibly quick and easy to get started working with services on Tanzu Application Platform.

- **Cartographer Conventions**

  Use Cartographer Conventions to ensure infrastructure uniformity across workloads deployed on the cluster. Cartographer Conventions provide a way to control how applications should be deployed on Kubernetes using a convention. Use Cartographer Conventions to apply the runtime best practices, policies, and conventions of your organization to workloads as they are created on the platform.

- **cert-manager**

  cert-manager adds certificates and certificate issuers as resource types to Kubernetes clusters. It also helps you to obtain, renew, and use those certificates. For more information about cert-manager, see the cert-manager documentation.

- **Cloud Native Runtimes**

  Cloud Native Runtimes for Tanzu is a serverless application runtime for Kubernetes that is based on Knative and runs on a single Kubernetes cluster. For information about Knative, see the Knative documentation.

- **Contour**

  Contour is an ingress controller for Kubernetes that supports dynamic configuration updates and multi-team ingress delegation. It provides the control plane for the Envoy edge and

service proxy. For more information about Contour, see the Contour documentation.

- **Default roles for Tanzu Application Platform**

  This package includes five default roles for users, including app-editor, app-viewer, app-operator, and service accounts including workload and deliverable. These roles are available to help operators limit permissions a user or service account requires on a cluster that runs Tanzu Application Platform. They are built by using aggregated cluster roles in Kubernetes role-based access control (RBAC). Default roles only apply to a user interacting with the cluster by using kubectl and Tanzu CLI.

- **Crossplane**

  Crossplane is an open source, Cloud Native Computing Foundation (CNCF) project built on the foundation of Kubernetes. Tanzu Application Platform uses Crossplane to power a number of capabilities, such as dynamic provisioning of services instances with Services Toolkit and the Bitnami Services.

- **Developer Conventions**

  Developer conventions configure workloads to prepare them for inner loop development.

  It's meant to be a "deploy and forget" component for developers. After it is installed on the cluster with the Tanzu Package CLI, developers do not need to directly interact with it. Developers instead interact with the Tanzu Developer Tools for VSCode IDE Extension or Tanzu CLI Apps plug-in, which rely on the Developer Conventions to edit the workload to enable inner loop capabilities.

- **External Secrets Operator**

  The External Secrets Operator is a Kubernetes operator that integrates with external secret management systems, for example, Google Secrets Manager and Hashicorp Vault. It reads information from external APIs and automatically injects the values into a Kubernetes secret. Tanzu Application Platform uses the External Secrets Operator to simplify Kubernetes secret life cycle management.

- **Flux CD Source Controller**

  The main role of this source management component is to provide a common interface for artifact acquisition.

- **Local Source Proxy**

  Local Source Proxy is a secure and convenient means for you to interact with external registries without providing a lot of registry details.

  With Local Source Proxy, developers can interact with external registries without needing to know registry specifics, such as endpoints, credentials, and certificates. This eliminates the burden of platform and app operators having to distribute registry credentials to developer workstations.

  Developers can also seamlessly deploy their applications without managing registry credentials on their local machines or keeping track of where their local source is uploaded.

- **Namespace Provisioner**

  Namespace Provisioner provides an easy, secure, automated way for Platform Operators to provision namespaces with the resources and proper namespace-level privileges needed for developer workloads to function as intended.

- **Service Bindings**

  Service Bindings create a Kubernetes-wide specification for communicating service secrets to workloads in a consistent way.

- **Service Registry**

  Service Registry for VMware Tanzu provides on-demand Eureka servers for your Tanzu Application Platform clusters. With Service Registry, you can create Eureka servers in your namespaces and bind Spring Boot workloads to them.

- **Services Toolkit**

  Services Toolkit is responsible for backing many of the most exciting and powerful capabilities for services in Tanzu Application Platform. From the integration of an extensive list of cloud-based and on-prem services, through to the offering and discovery of those services, and finally to the claiming and binding of service instances to application workloads, Services Toolkit has the tools you need to make working with services on Tanzu Application Platform simple, easy, and effective.

- **Source Controller**

  Tanzu Source Controller provides a standard interface for artifact acquisition and extends the function of Flux CD Source Controller. Tanzu Source Controller supports the following two resource types:

  - ImageRepository (deprecated)
  - MavenArtifact

- **Spring Boot conventions**

  The Spring Boot convention server has a bundle of smaller conventions applied to any Spring Boot application that is submitted to the supply chain in which the convention controller is configured.

- **Spring Cloud Gateway**

  Spring Cloud Gateway for Kubernetes is an API gateway solution based on the open-source Spring Cloud Gateway project. It provides a simple means to route internal or external API requests to application services that expose APIs.

- **Supply Chain Choreographer**

  Supply Chain Choreographer is based on open-source Cartographer. It enables app operators to create preapproved paths to production by integrating Kubernetes resources with the elements of their existing toolchains, such as Jenkins.

  Each pre-approved supply chain creates a paved road to production. It orchestrates supply chain resources, namely test, build, scan, and deploy. Enabling developers to focus on delivering value to their users. Pre-approved supply chains also assure application operators that all code in production has passed through the steps of an approved workflow.

  Note: Tanzu Supply Chain replaces Supply Chain Choreographer, but is still in beta testing.

- **Supply Chain Security Tools - Policy Controller**

  Supply Chain Security Tools (SCST) - Policy is an admission controller that allows a cluster operator to specify policies to verify image container signatures before admitting them to a cluster. It works with cosign signature format and allows for fine-tuned configuration of policies based on image source patterns.

- **Supply Chain Security Tools - Scan**

  With Supply Chain Security Tools (SCST) - Scan, you can build and deploy secure trusted software that complies with their corporate security requirements.

  To enable this, Supply Chain Security Tools - Scan provides scanning and gate keeping capabilities that Application and DevSecOps teams can incorporate earlier in their path to

production. This is an established industry best practice for reducing security risk and ensuring more efficient remediation.

- **Supply Chain Security Tools - Store**

  Supply Chain Security Tools (SCST) - Store saves software bills of materials (SBoMs) to a database and enables you to query for image, source, package, and vulnerability relationships. It integrates with SCST - Scan to automatically store the resulting source and image vulnerability reports.

- **Tanzu Developer Portal**

  Tanzu Developer Portal lets your developers view your organization's running applications and services. It provides a central location for viewing dependencies, relationships, technical documentation, and even service status. Tanzu Developer Portal is built from the Cloud Native Computing Foundation's project Backstage.

- **Tanzu Application Platform Telemetry**

  Tanzu Application Platform Telemetry is a set of objects that collect data about the use of Tanzu Application Platform and send it back to VMware for product improvements. A benefit of remaining enrolled in telemetry and identifying your company during Tanzu Application Platform installation is that VMware can provide your organization with use reports about Tanzu Application Platform. For information about enrolling in telemetry reports, see Tanzu Application Platform usage reports.

  > **Note**
  >
  > You can opt out of telemetry collection by following the instructions in Opting out of telemetry collection.

- **Tanzu Build Service**

  Tanzu Build Service uses the open-source Cloud Native Build packs project to turn application source code into container images.

  Tanzu Build Service executes reproducible builds that align with modern container standards and keeps images up to date. It does so by leveraging Kubernetes infrastructure with kpack, a Cloud Native Build packs Platform, to orchestrate the image life cycle.

  The kpack CLI tool, kp, can aid in managing kpack resources. Build Service helps you develop and automate containerized software workflows securely and at scale.

- **Tanzu Buildpacks**

  Tanzu Buildpacks provide framework and runtime support for applications. Buildpacks typically examine your applications to work out what dependencies to download and how to configure the apps to communicate with bound services.

  Tanzu Buildpacks use open-source Paketo Buildpacks to allow Tanzu Application Platform users to turn their application source code into container images. From Tanzu Application Platform v1.6, builders, stacks, and buildpacks are packaged separately from Tanzu Build Service, but are included in the same Tanzu Application Platform profiles as Tanzu Build Service. All buildpacks follow the package name format `*.buildpacks.tanzu.vmware.com`.

- **Tanzu Supply Chain (beta)**

  Tanzu Supply Chain is a tool that provides a golden path to production for your teams. Tanzu Supply Chain replaces the supply-chain solution Supply Chain Choreographer, which is based on Cartographer.

- **Tanzu Developer Tools for IntelliJ**

  Tanzu Developer Tools for IntelliJ is the official VMware Tanzu IDE extension for IntelliJ IDEA to help you develop code by using Tanzu Application Platform. This extension enables you to rapidly iterate on your workloads on supported Kubernetes clusters that have Tanzu Application Platform installed.

- **Tanzu Developer Tools for Visual Studio**

  Tanzu Developer Tools for Visual Studio is the official VMware Tanzu IDE extension for Visual Studio to help you develop code by using Tanzu Application Platform. The Visual Studio extension enables live updates of your application while it runs on the cluster and lets you debug your application directly on the cluster.

- **Tanzu Developer Tools for Visual Studio Code**

  Tanzu Developer Tools for VS Code is the official VMware Tanzu IDE extension for VS Code to help you develop code by using Tanzu Application Platform. The VS Code extension enables live updates of your application while it runs on the cluster and lets you debug your application directly on the cluster.

- **Tanzu Application Platform GUI Client**

  Tanzu Application Platform GUI Client is a package with RBAC resources required by Tanzu Developer Portal on clusters with Build, Run and Iterate profiles. It also contains Tanzu Mission Control specific resources. You cannot install this package on the same cluster as Tanzu Developer Portal.

  This package is only used while installing Tanzu Application Platform using Tanzu Mission Control.

- **Tekton Pipelines**

  Tekton is a powerful and flexible open-source framework for creating CI/CD systems, enabling developers to build, test, and deploy across cloud providers and on-premise systems.

Installation profiles in Tanzu Application Platform v1.9

You can deploy Tanzu Application Platform through predefined profiles, each containing various packages, or you can install the packages individually. The profiles allow Tanzu Application Platform to scale across an organization's multicluster, multi-cloud, or hybrid cloud infrastructure. These profiles are not meant to cover all use cases, but serve as a starting point to allow for further customization.

The following profiles are available in Tanzu Application Platform:

- **Full** (`full`): Contains nearly all Tanzu Application Platform packages. For the exceptions to the full profile, see the packages with a check mark in the **Not in a profile** column in the table later in this section.
- **Iterate** (`iterate`): Intended for iterative application development.
- **Build** (`build`): Intended for the transformation of source revisions to workload revisions. Specifically, hosting workloads and SupplyChains.
- **Run** (`run`): Intended for the transformation of workload revisions to running pods. Specifically, hosting deliveries and deliverables.
- **View** (`view`): Intended for instances of applications related to centralized developer experiences. Specifically, Tanzu Developer Portal and Metadata Store.

- **Authoring** (`authoring`): Includes everything in the **Iterate** profile and has extra packages for installing Tanzu Supply Chain (beta).

The following tables list the packages contained in each profile. Packages not included in any profile are available to install as individual packages only. See the component documentation for the package for installation instructions. For a diagram showing the packages contained in each profile, see Overview of multicluster Tanzu Application Platform.

Packages: A to C

Package Name	Full	Iterate	Build	Run	View	Authoring	Not in a profile
API Auto Registration	✓	✓		✓		✓	
API portal	✓				✓		
Application Accelerator	✓				✓		
Application Configuration Service							✓
Application Live View APIServer	✓	✓		✓		✓	
Application Live View back end	✓				✓		
Application Live View connector	✓	✓		✓		✓	
Application Live View conventions	✓	✓	✓			✓	
Application Single Sign-On	✓	✓		✓		✓	
AOA dashboard							✓
Artifact Metadata Repository Observer	✓		✓	✓			
AWS Services							✓
Bitnami Services	✓	✓		✓		✓	
Carbon Black Scanner for SCST - Scan (beta)							✓
cert-manager	✓	✓	✓	✓	✓	✓	
Cloud Native Runtimes	✓	✓		✓		✓	
Contour	✓	✓		✓	✓	✓	
Crossplane	✓	✓		✓		✓	

Packages: D to R

Package Name	Full	Iterate	Build	Run	View	Authoring	Not in a profile
Default Roles	✓	✓	✓	✓		✓	
Developer Conventions	✓	✓				✓	
External Secrets Operator							✓
Flux Source Controller	✓	✓	✓	✓	✓	✓	
Grype Scanner for SCST - Scan							✓
Local Source Proxy	✓	✓				✓	
Namespace Provisioner	✓	✓	✓	✓		✓	
Out of the Box Delivery - Basic	✓	✓		✓		✓	

Package Name	Full	Iterate	Build	Run	View	Authoring	Not in a profile
Out of the Box Supply Chain - Basic	✓	✓	✓			✓	
Out of the Box Supply Chain - Testing	✓	✓	✓			✓	
Out of the Box Supply Chain - Testing and Scanning	✓		✓				
Out of the Box Templates	✓	✓	✓	✓		✓	

Packages: S to Z

Package Name	Full	Iterate	Build	Run	View	Authoring	Not in a profile
Service Bindings	✓	✓		✓		✓	
Service Registry							✓
Services Toolkit	✓	✓		✓		✓	
Source Controller	✓	✓	✓	✓	✓	✓	
Snyk Scanner for SCST - Scan (beta)							✓
Spring Boot conventions	✓	✓	✓			✓	
Spring Cloud Gateway							✓
Supply Chain Choreographer	✓	✓	✓	✓		✓	
SCST - Policy Controller	✓	✓		✓		✓	
SCST - Scan	✓		✓				
SCST - Scan 2.0	✓		✓				
SCST - Store	✓				✓		
Tanzu Build Packs	✓	✓	✓			✓	
Tanzu Build Service	✓	✓	✓			✓	
Tanzu Developer Portal	✓				✓		
Tanzu Supply Chain (beta)						✓	
Tekton Pipelines	✓	✓	✓			✓	
Telemetry	✓	✓	✓	✓	✓	✓	

Note

You can only install one supply chain at any given time. For information about switching supply chains, see Add testing and scanning to your application.

Language and framework support in Tanzu Application Platform

The following table shows the languages and frameworks supported by Tanzu Application Platform components.

Language or Framework	Tanzu Build Service	Runtime Conventions	Tanzu Developer Tooling	Application Live View	Extended Scanning Coverage using Buildpack SBOMs
Java	✓	✓	✓		✓
Spring Boot	✓	✓	✓	✓	✓
.NET Core	✓		✓	✓	✓
Steeltoe	✓		✓	✓	
NodeJS	✓				✓
Python	✓				✓
golang	✓				✓
PHP	✓				✓
Ruby	✓				✓

Tanzu Developer Tooling: refers to the developer conventions that enable debugging and Live Update function in the inner loop.

Extended Scanning Coverage: When building container images with the Tanzu Build Service, the Cloud Native Build Packs used in the build process for the specified languages produce a Software Bill of Materials (SBOM). Some scan engines support the enhanced ability to use this SBOM as a source for the scan. Out of the Box Supply Chain - Testing and Scanning leverages Anchore's Grype for the image scan, which suppports this capability. In addition, users have the ability to leverage Carbon Black Container image scans, which also supports this enhanced scan coverage.

Installing Tanzu Application Platform

For more information about installing Tanzu Application Platform, see Installing Tanzu Application Platform.

Install Tanzu Application Platform

You can install Tanzu Application Platform (commonly known as TAP) by using one of the following methods:

- Install Tanzu Application Platform online. For Tanzu Application Platform on a Kubernetes cluster with internet access.
- Install Tanzu Application Platform in an air-gapped environment. For Tanzu Application Platform on a Kubernetes cluster air-gapped from external traffic.
- Install Tanzu Application Platform with GitOps (beta). For Tanzu Application Platform on a Kubernetes cluster via a GitOps approach.
- Install Tanzu Application Platform in AWS. For installing Tanzu Application platform using AWS Cloud Services.
- Install Tanzu Application Platform in Azure. For installing Tanzu Application platform using Azure Cloud Services.
- Install Tanzu Application Platform on OpenShift. For Tanzu Application Platform on an OpenShift cluster with internet access.
- Install Tanzu Application Platform with Tanzu Mission Control. For installing Tanzu Application platform on a managed cluster using Tanzu Mission Control.

Install Tanzu Application Platform

You can install Tanzu Application Platform (commonly known as TAP) by using one of the following methods:

- Install Tanzu Application Platform online. For Tanzu Application Platform on a Kubernetes cluster with internet access.
- Install Tanzu Application Platform in an air-gapped environment. For Tanzu Application Platform on a Kubernetes cluster air-gapped from external traffic.
- Install Tanzu Application Platform with GitOps (beta). For Tanzu Application Platform on a Kubernetes cluster via a GitOps approach.
- Install Tanzu Application Platform in AWS. For installing Tanzu Application platform using AWS Cloud Services.
- Install Tanzu Application Platform in Azure. For installing Tanzu Application platform using Azure Cloud Services.
- Install Tanzu Application Platform on OpenShift. For Tanzu Application Platform on an OpenShift cluster with internet access.
- Install Tanzu Application Platform with Tanzu Mission Control. For installing Tanzu Application platform on a managed cluster using Tanzu Mission Control.

Prerequisites and planning for installing Tanzu Application Platform

The following are required to install Tanzu Application Platform (commonly known as TAP):

Installation planning

Before you begin a Tanzu Application Platform installation:

1. Review the Tanzu Application Platform planning and architecture documentation. For more information, see Planning and architecture reference.
2. (Optional) To gain an understanding of Tanzu Application Platform, experiment with a Tanzu Application Platform sandbox. For more information, see Access an experimental developer sandbox environment.

Installation prerequisites

Installation requires:

VMware Tanzu Network and container image registry requirements

- Access to VMware Tanzu Network:
  - A Tanzu Network account to download Tanzu Application Platform packages.
  - Network access to https://registry.tanzu.vmware.com.
- Cluster-specific registry:
  - A container image registry, such as Harbor or Docker Hub for application images, base images, and runtime dependencies. When available, VMware recommends using a paid registry account to avoid potential rate-limiting associated with some free registry offerings.
  - Recommended storage space for container image registry:
    - 1 GB of available storage if installing Tanzu Build Service with the `lite` set of dependencies.
    - 10 GB of available storage if installing Tanzu Build Service with the `full` set of dependencies, which are suitable for offline environments.

    Note

    For production environments, `full` dependencies are recommended to optimize security and performance. For more information about Tanzu Build Service dependencies, see About lite and full dependencies.
- Registry credentials with read and write access available to Tanzu Application Platform to store images.
- Network access to your chosen container image registry.

DNS Records

There are some optional but recommended DNS records you must allocate if you decide to use these particular components:

- Cloud Native Runtimes (Knative): Allocate a wildcard subdomain for your developer's applications. This is specified in the `shared.ingress_domain` key of the `tap-values.yaml` configuration file that you input with the installation. This wildcard must be pointed at the

external IP address of the `tanzu-system-ingress`'s `envoy` service. See Access with the shared Ingress method for more information about `tanzu-system-ingress`.

- Tanzu Developer Portal: If you decide to implement the shared ingress and include Tanzu Developer Portal, allocate a fully Qualified Domain Name (FQDN) that can be pointed at the `tanzu-system-ingress` service. The default host name consists of `tap-gui` and the `shared.ingress_domain` value. For example, `tap-gui.example.com`.
- Supply Chain Security Tools - Store: Similar to Tanzu Developer Portal, allocate a fully Qualified Domain Name (FQDN) that can be pointed at the `tanzu-system-ingress` service. The default host name consists of `metadata-store` and the `shared.ingress_domain` value. For example, `metadata-store.example.com`.
- Artifact Metadata Repository: Similar to the Supply Chain Security Tools (SCST) - Store, allocate a fully Qualified Domain Name (FQDN) that can be pointed at the `tanzu-system-ingress` service. The default host name consists of `amr-graphql` and the `shared.ingress_domain` value. For example, `amr-graphql.example.com`.
- Application Live View: If you select the `ingressEnabled` option, allocate a corresponding fully Qualified Domain Name (FQDN) that can be pointed at the `tanzu-system-ingress` service. The default host name consists of `appliveview` and the `shared.ingress_domain` value. For example, `appliveview.example.com`.

Supply Chain Security Tools - Store

Although Tanzu Application Platform includes a default database for Supply Chain Security Tools (SCST) - Store, VMware discourages using it for production deployments. The included database lacks capabilities typically offered by enterprise-grade databases, such as scaling, high availability, and automated backups. VMware recommends using an external database with production level capabilities. For more information about setting up the database for production, see Database configuration.

Tanzu Developer Portal

For Tanzu Developer Portal, you must have:

- Latest version of Chrome, Firefox, or Edge. Tanzu Developer Portal currently does not support Safari browser.
- Git repository for Tanzu Developer Portal's software catalogs, with a token allowing read access. For more information about how to use your Git repository, see Create an application accelerator. Supported Git infrastructure includes:
    - GitHub
    - GitLab
    - Azure DevOps
- Tanzu Developer Portal Blank Catalog from the Tanzu Application section of VMware Tanzu Network. The Blank Catalog serves as a foundation for your customization, allowing you to populate it with your own content. For more information about formatting your own catalog, see Catalog operations.
    - To install, navigate to Tanzu Network. Under the list of available files to download, there is a folder titled `tanzu-developer-portal-catalogs-latest`. Inside that folder is a compressed archive titled `Tanzu Developer Portal Blank Catalog`. You must extract that catalog to the preceding Git repository of choice. This serves as the configuration location for your organization's catalog inside Tanzu Developer Portal.
- The Tanzu Developer Portal catalog allows for two approaches to store catalog information:

    - The default option uses an in-memory database and is suitable for test and development scenarios. This reads the catalog data from Git URLs that you specify in the `tap-values.yaml` file. This data is temporary. Any operations that cause the `server` pod in the `tap-gui` namespace to be re-created also cause this data to be rebuilt from the Git location. This can cause issues when you manually register entities by using the UI, because they only exist in the database and are lost when that in-memory database gets rebuilt.
    - For production use cases, use a PostgreSQL database that exists outside the Tanzu Application Platform packaging. The PostgreSQL database stores all the catalog data persistently both from the Git locations and the UI manual entity registrations. For more information, see Configure the Tanzu Developer Portal database

Kubernetes cluster requirements

Installation requires Kubernetes cluster v1.26, v1.27, v1.28 or v1.29 on one of the following Kubernetes providers:

- Azure Kubernetes Service.
- Amazon Elastic Kubernetes Service.
    - containerd must be used as the Container Runtime Interface (CRI). Some versions of EKS default to Docker as the container runtime and must be changed to containerd.
    - EKS clusters on Kubernetes version 1.23 and above require the Amazon EBS CSI Driver due to CSIMigrationAWS is enabled by default in Kubernetes version 1.23 and above.
        - Users currently on EKS Kubernetes version 1.22 must install the Amazon EBS CSI Driver before upgrading to Kubernetes version 1.23 and above. See AWS documentation for more information.
    - AWS Fargate is not supported.
- Google Kubernetes Engine.
    - GKE Autopilot clusters do not have the required features enabled.
    - GKE clusters that are set up in zonal mode might detect Kubernetes API errors when the GKE control plane is resized after traffic increases. Users can mitigate this by creating a regional cluster with three control-plane nodes right from the start.
- Red Hat OpenShift Container Platform v4.13, v4.14 and v4.15.
    - vSphere
    - Baremetal
- Tanzu Kubernetes Grid (commonly called TKG) with Standalone Management Cluster. For more information, see the Tanzu Kubernetes Grid documentation.
- vSphere with Tanzu v8.0 Update 1c or later, v7.0 Update 3p or later.
- Tanzu Kubernetes Grid Integrated Edition with vSphere (commonly called TKGi) v1.17 and later.
    - For TKGi with NSX, the total number of Kubernetes object labels and other tags created by both TKGi and Tanzu Application Platform can exceed the number allowed by NSX. Create or update your network profile by setting the `cni_configurations` parameter `extensions.ncp.k8s.label_filtering_regex_list`. For more information, see the VMware Tanzu Kubernetes Grid Integrated Edition documentation.

For more information about the supported Kubernetes versions, see Kubernetes version support for Tanzu Application Platform.

Resource requirements

- To deploy Tanzu Application Platform packages full profile, your cluster must have at least:
  - 8 GB of RAM available per node to Tanzu Application Platform.
  - 16 vCPUs available across all nodes to Tanzu Application Platform.
  - 100 GB of disk space available per node.

  Important

  Tanzu Application Platform requires a minimum of 100 GB per node of ephemeral storage. If you do not allocate at least this amount of ephemeral storage for kubelet on all cluster nodes, you receive the error "minDiskPerNode: some cluster nodes don't meet minimum disk space requirement of '100Gi'." For more information about configuring the storage for a TKG cluster on Supervisor, see v1alpha3 Example: TKC with Default Storage and Node Volumes and v1beta1 Example: Custom Cluster Based on the Default ClusterClass.

- To deploy Tanzu Application Platform packages build, run and iterate (shared) profile, your cluster must have at least:
  - 8 GB of RAM available per node to Tanzu Application Platform.
  - 12 vCPUs available across all nodes to Tanzu Application Platform.
  - 100 GB of disk space available per node.
- To deploy Tanzu Application Platform packages view profile, your cluster must have at least:
  - 8 GB of RAM available per node to Tanzu Application Platform.
  - 8 vCPUs available across all nodes to Tanzu Application Platform.
  - 100 GB of disk space available per node.
- For the full profile or use of Security Chain Security Tools - Store, your cluster must have a configured default StorageClass.
- Pod security policies must be configured so that Tanzu Application Platform controller pods can run as root in the following optional configurations:
  - Tanzu Build Service, in which CustomStacks require root privileges. For more information, see Tanzu Build Service documentation.
  - Supply Chain, in which Kaniko usage requires root privileges to build containers.

  For more information about pod security policies, see Kubernetes documentation.

Tools and CLI requirements

Installation requires:

- The Kubernetes CLI (kubectl) v1.26, v1.27, v1.28 or v1.29 installed and authenticated with admin rights for your target cluster. See Install Tools in the Kubernetes documentation.

Next steps

- Accept Tanzu Application Platform EULAs and installing the Tanzu CLI

Prerequisites and planning for installing Tanzu Application Platform

The following are required to install Tanzu Application Platform (commonly known as TAP):

Installation planning

Before you begin a Tanzu Application Platform installation:

1. Review the Tanzu Application Platform planning and architecture documentation. For more information, see Planning and architecture reference.
2. (Optional) To gain an understanding of Tanzu Application Platform, experiment with a Tanzu Application Platform sandbox. For more information, see Access an experimental developer sandbox environment.

Installation prerequisites

Installation requires:

VMware Tanzu Network and container image registry requirements

- Access to VMware Tanzu Network:
  - A Tanzu Network account to download Tanzu Application Platform packages.
  - Network access to https://registry.tanzu.vmware.com.
- Cluster-specific registry:
  - A container image registry, such as Harbor or Docker Hub for application images, base images, and runtime dependencies. When available, VMware recommends using a paid registry account to avoid potential rate-limiting associated with some free registry offerings.
  - Recommended storage space for container image registry:
    - 1 GB of available storage if installing Tanzu Build Service with the `lite` set of dependencies.
    - 10 GB of available storage if installing Tanzu Build Service with the `full` set of dependencies, which are suitable for offline environments.

    Note

    For production environments, `full` dependencies are recommended to optimize security and performance. For more information about Tanzu Build Service dependencies, see About lite and full dependencies.
- Registry credentials with read and write access available to Tanzu Application Platform to store images.
- Network access to your chosen container image registry.

DNS Records

There are some optional but recommended DNS records you must allocate if you decide to use these particular components:

- Cloud Native Runtimes (Knative): Allocate a wildcard subdomain for your developer's applications. This is specified in the `shared.ingress_domain` key of the `tap-values.yaml` configuration file that you input with the installation. This wildcard must be pointed at the external IP address of the `tanzu-system-ingress`'s `envoy` service. See Access with the shared Ingress method for more information about `tanzu-system-ingress`.
- Tanzu Developer Portal: If you decide to implement the shared ingress and include Tanzu Developer Portal, allocate a fully Qualified Domain Name (FQDN) that can be pointed at the `tanzu-system-ingress` service. The default host name consists of `tap-gui` and the `shared.ingress_domain` value. For example, `tap-gui.example.com`.
- Supply Chain Security Tools - Store: Similar to Tanzu Developer Portal, allocate a fully Qualified Domain Name (FQDN) that can be pointed at the `tanzu-system-ingress` service. The default host name consists of `metadata-store` and the `shared.ingress_domain` value. For example, `metadata-store.example.com`.
- Artifact Metadata Repository: Similar to the Supply Chain Security Tools (SCST) - Store, allocate a fully Qualified Domain Name (FQDN) that can be pointed at the `tanzu-system-ingress` service. The default host name consists of `amr-graphql` and the `shared.ingress_domain` value. For example, `amr-graphql.example.com`.
- Application Live View: If you select the `ingressEnabled` option, allocate a corresponding fully Qualified Domain Name (FQDN) that can be pointed at the `tanzu-system-ingress` service. The default host name consists of `appliveview` and the `shared.ingress_domain` value. For example, `appliveview.example.com`.

Supply Chain Security Tools - Store

Although Tanzu Application Platform includes a default database for Supply Chain Security Tools (SCST) - Store, VMware discourages using it for production deployments. The included database lacks capabilities typically offered by enterprise-grade databases, such as scaling, high availability, and automated backups. VMware recommends using an external database with production level capabilities. For more information about setting up the database for production, see Database configuration.

Tanzu Developer Portal

For Tanzu Developer Portal, you must have:

- Latest version of Chrome, Firefox, or Edge. Tanzu Developer Portal currently does not support Safari browser.
- Git repository for Tanzu Developer Portal's software catalogs, with a token allowing read access. For more information about how to use your Git repository, see Create an application accelerator. Supported Git infrastructure includes:
  - GitHub
  - GitLab
  - Azure DevOps
- Tanzu Developer Portal Blank Catalog from the Tanzu Application section of VMware Tanzu Network. The Blank Catalog serves as a foundation for your customization, allowing you to populate it with your own content. For more information about formatting your own catalog, see Catalog operations.

    - To install, navigate to Tanzu Network. Under the list of available files to download, there is a folder titled `tanzu-developer-portal-catalogs-latest`. Inside that folder is a compressed archive titled `Tanzu Developer Portal Blank Catalog`. You must extract that catalog to the preceding Git repository of choice. This serves as the configuration location for your organization's catalog inside Tanzu Developer Portal.
- The Tanzu Developer Portal catalog allows for two approaches to store catalog information:
    - The default option uses an in-memory database and is suitable for test and development scenarios. This reads the catalog data from Git URLs that you specify in the `tap-values.yaml` file. This data is temporary. Any operations that cause the `server` pod in the `tap-gui` namespace to be re-created also cause this data to be rebuilt from the Git location. This can cause issues when you manually register entities by using the UI, because they only exist in the database and are lost when that in-memory database gets rebuilt.
    - For production use cases, use a PostgreSQL database that exists outside the Tanzu Application Platform packaging. The PostgreSQL database stores all the catalog data persistently both from the Git locations and the UI manual entity registrations. For more information, see Configure the Tanzu Developer Portal database

Kubernetes cluster requirements

Installation requires Kubernetes cluster v1.26, v1.27, v1.28 or v1.29 on one of the following Kubernetes providers:

- Azure Kubernetes Service.
- Amazon Elastic Kubernetes Service.
    - containerd must be used as the Container Runtime Interface (CRI). Some versions of EKS default to Docker as the container runtime and must be changed to containerd.
    - EKS clusters on Kubernetes version 1.23 and above require the Amazon EBS CSI Driver due to CSIMigrationAWS is enabled by default in Kubernetes version 1.23 and above.
        - Users currently on EKS Kubernetes version 1.22 must install the Amazon EBS CSI Driver before upgrading to Kubernetes version 1.23 and above. See AWS documentation for more information.
    - AWS Fargate is not supported.
- Google Kubernetes Engine.
    - GKE Autopilot clusters do not have the required features enabled.
    - GKE clusters that are set up in zonal mode might detect Kubernetes API errors when the GKE control plane is resized after traffic increases. Users can mitigate this by creating a regional cluster with three control-plane nodes right from the start.
- Red Hat OpenShift Container Platform v4.13, v4.14 and v4.15.
    - vSphere
    - Baremetal
- Tanzu Kubernetes Grid (commonly called TKG) with Standalone Management Cluster. For more information, see the Tanzu Kubernetes Grid documentation.
- vSphere with Tanzu v8.0 Update 1c or later, v7.0 Update 3p or later.
- Tanzu Kubernetes Grid Integrated Edition with vSphere (commonly called TKGi) v1.17 and later.

    - For TKGi with NSX, the total number of Kubernetes object labels and other tags created by both TKGi and Tanzu Application Platform can exceed the number allowed by NSX. Create or update your network profile by setting the `cni_configurations` parameter `extensions.ncp.k8s.label_filtering_regex_list`. For more information, see the VMware Tanzu Kubernetes Grid Integrated Edition documentation.

For more information about the supported Kubernetes versions, see Kubernetes version support for Tanzu Application Platform.

Resource requirements

- To deploy Tanzu Application Platform packages full profile, your cluster must have at least:
    - 8 GB of RAM available per node to Tanzu Application Platform.
    - 16 vCPUs available across all nodes to Tanzu Application Platform.
    - 100 GB of disk space available per node.

    Important

    Tanzu Application Platform requires a minimum of 100 GB per node of ephemeral storage. If you do not allocate at least this amount of ephemeral storage for kubelet on all cluster nodes, you receive the error "minDiskPerNode: some cluster nodes don't meet minimum disk space requirement of '100Gi'." For more information about configuring the storage for a TKG cluster on Supervisor, see v1alpha3 Example: TKC with Default Storage and Node Volumes and v1beta1 Example: Custom Cluster Based on the Default ClusterClass.

- To deploy Tanzu Application Platform packages build, run and iterate (shared) profile, your cluster must have at least:
    - 8 GB of RAM available per node to Tanzu Application Platform.
    - 12 vCPUs available across all nodes to Tanzu Application Platform.
    - 100 GB of disk space available per node.
- To deploy Tanzu Application Platform packages view profile, your cluster must have at least:
    - 8 GB of RAM available per node to Tanzu Application Platform.
    - 8 vCPUs available across all nodes to Tanzu Application Platform.
    - 100 GB of disk space available per node.
- For the full profile or use of Security Chain Security Tools - Store, your cluster must have a configured default StorageClass.
- Pod security policies must be configured so that Tanzu Application Platform controller pods can run as root in the following optional configurations:
    - Tanzu Build Service, in which CustomStacks require root privileges. For more information, see Tanzu Build Service documentation.
    - Supply Chain, in which Kaniko usage requires root privileges to build containers.

    For more information about pod security policies, see Kubernetes documentation.

Tools and CLI requirements

Installation requires:

- The Kubernetes CLI (kubectl) v1.26, v1.27, v1.28 or v1.29 installed and authenticated with admin rights for your target cluster. See Install Tools in the Kubernetes documentation.

Next steps

- Accept Tanzu Application Platform EULAs and installing the Tanzu CLI

Kubernetes version support for Tanzu Application Platform

The following is a matrix table providing details of the compatible Kubernetes cluster versions for Tanzu Application Platform v1.9.

Kubernetes Cluster	Support Information
Kubernetes	v1.26, v1.27, v1.28, v1.29
VMware Tanzu Kubernetes Grid	v2.5.0, v2.4.0
VMware Tanzu Kubernetes Grid Integrated Edition with vSphere	v1.17 or later
vSphere with Tanzu	v8.0 Update 1c or later, v7.0 Update 3p or later
OpenShift	v4.13, v4.14, v4.15
Azure Kubernetes Service	Supported
Elastic Kubernetes Service	Supported
Google Kubernetes Engine	Supported

Open source component versions

This topic contains the open source component versions for Tanzu Application Platform v1.9 (commonly known as TAP).

OSS Release Name	Release Version
Backstage	1.22.0
Cartographer Convention	0.8.0
cert injection webhook	0.6.0
cert-manager	1.13.3
contour	1.28.0
crossplane	1.15.0-up.1
External Secrets Operator	0.9.4
Flux CD Source Controller	1.1.2
Grype Vulnerability Scanner (Scan v1)	0.54.0
IntelliJ	Build 222 or later
Kafka by Bitnami (unmanaged)	22.0.0
Knative Serving	1.13.1
kpack	0.13.2
MongoDB by Bitnami (unmanaged)	13.13.1

OSS Release Name	Release Version
MySQL by Bitnami (unmanaged)	9.5.0
Paketo Syft Buildpack	1.10.1
Postgres by Bitnami (unmanaged)	12.2.0
RabbitMQ by Bitnami (unmanaged)	11.10.0
Redis by Bitnami (unmanaged)	17.8.0
Service Bindings	0.8.0
Sigstore Policy Controller	0.8.2
Source Controller	0.9.0
Tekton Pipelines	0.50.3
Tilt	0.30.12 or later
Trivy Vulnerability Scanner (Scan v2)	0.48.3
Visual Studio Code	1.64.0 or later

Install Tanzu CLI

This topic tells you how to accept the EULAs, and install the Tanzu CLI and plug-ins on Tanzu Application Platform (commonly known as TAP).

Accept the End User License Agreements

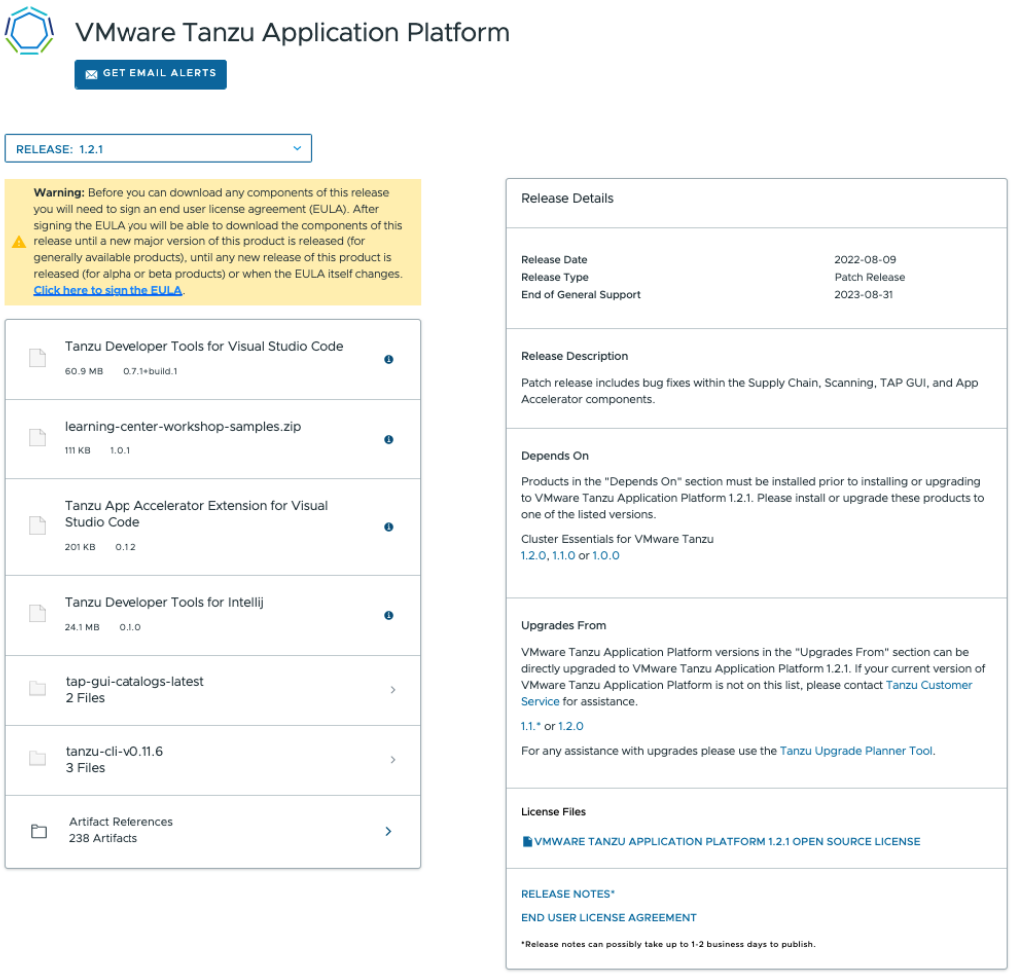
Before downloading and installing Tanzu Application Platform packages, you must accept the End User License Agreements (EULAs) as follows:

1. Sign in to VMware Tanzu Network.
2. Accept or confirm that you have accepted the EULAs for each of the following:
    - Tanzu Application Platform
    - Cluster Essentials for VMware Tanzu

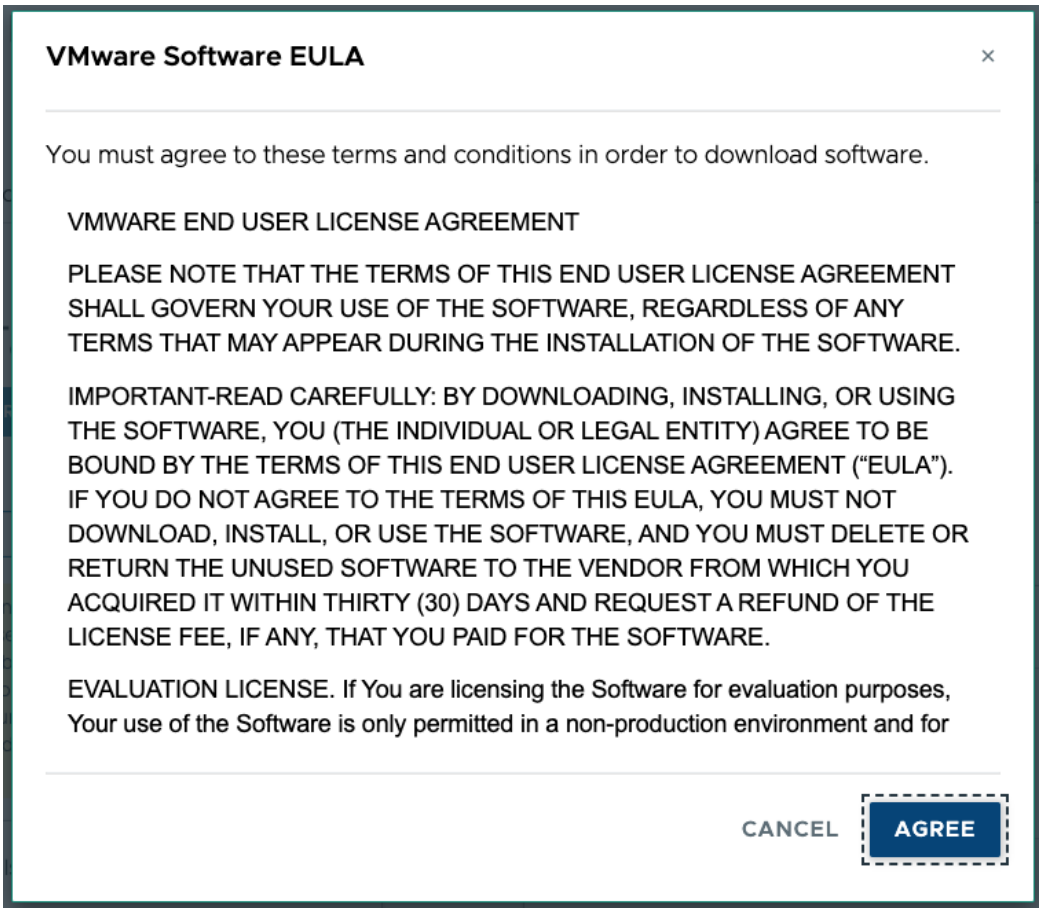
Example of accepting the Tanzu Application Platform EULA

To accept the Tanzu Application Platform EULA:

1. Go to Tanzu Application Platform.
2. Select the ***Click here to sign the EULA*** link in the yellow warning box under the release drop-down menu. If the yellow warning box is not visible, the EULA has already been accepted.



3. Select ***Agree*** in the bottom-right of the dialog box as seen in the following screenshot.



Set the Kubernetes cluster context

For information about the supported Kubernetes cluster providers and versions, see Kubernetes cluster requirements.

To set the Kubernetes cluster context:

1. List the existing contexts by running:

   ```
   kubectl config get-contexts
   ```

   For example:

   ```
   $ kubectl config get-contexts
   CURRENT   NAME                                CLUSTER           AUTHINFO
   NAMESPACE
           aks-repo-trial                      aks-repo-trial    clusterUser_aks-r
   g-01_aks-repo-trial
   *       aks-tap-cluster                     aks-tap-cluster   clusterUser_aks-r
   g-01_aks-tap-cluster
   ```

2. If you are managing multiple cluster contexts, set the context to the cluster that you want to use for the Tanzu Application Platform packages installation by running:

   ```
   kubectl config use-context CONTEXT
   ```

   Where `CONTEXT` is the cluster that you want to use. For example, `aks-tap-cluster`.

   For example:

```
$ kubectl config use-context aks-tap-cluster
Switched to context "aks-tap-cluster".
```

Install or update the Tanzu CLI and plug-ins

The Tanzu CLI and plug-ins enable you to install and use the Tanzu Application Platform functions and features.

Install the Tanzu CLI

The Tanzu CLI core 1.3 distributed with Tanzu Application Platform is forward and backward compatible with all supported Tanzu Application Platform versions. Run a single command to install the plug-in group version that matches the Tanzu Application Platform version on any target environment. For more information, see Install Plugins.

Use a package manager to install Tanzu CLI on Windows, Mac, or Linux OS. Alternatively, download and install manually from Tanzu Network, VMware Customer Connect, or GitHub.

Basic installation instructions are provided below. For more information including how to install the Tanzu CLI and CLI plug-ins in Internet-restricted environments, see the VMware Tanzu CLI documentation.

Note

To retain an existing installation of the Tanzu CLI, move the CLI binary from `/usr/local/bin/tanzu` or `C:\Program Files\tanzu` on Windows to a different location before following the steps below.

Install using a package manager

To install the Tanzu CLI using a package manager:

1. Follow the instructions for your package manager below. This installs the latest version of the CLI available in the package registry.
    - **Homebrew (MacOS):**

      ```
      brew update
      brew install vmware-tanzu/tanzu/tanzu-cli
      ```

    - **Chocolatey (Windows):**

      ```
      choco install tanzu-cli
      ```

      The `tanzu-cli` package is part of the main Chocolatey Community Repository. When a new `tanzu-cli` version is released, it might not be available immediately. If the above command fails, run:

      ```
      choco install tanzu-cli --version TANZU-CLI-VERSION
      ```

      Where `TANZU-CLI-VERSION` is the Tanzu CLI version you want to install.

      For example:

      ```
      choco install tanzu-cli --version 1.3.0
      ```

    - **APT (Debian or Ubuntu):**

```
sudo mkdir -p /etc/apt/keyrings/
sudo apt-get update
sudo apt-get install -y ca-certificates curl gpg
curl -fsSL https://packages.vmware.com/tools/keys/VMWARE-PACKAGING-GPG
-RSA-KEY.pub | sudo gpg --dearmor -o /etc/apt/keyrings/tanzu-archive-k
eyring.gpg
echo "deb [arch=amd64 signed-by=/etc/apt/keyrings/tanzu-archive-keyrin
g.gpg] https://storage.googleapis.com/tanzu-cli-os-packages/apt tanzu-
cli-jessie main" | sudo tee /etc/apt/sources.list.d/tanzu.list
sudo apt-get update
sudo apt-get install -y tanzu-cli
```

- **YUM or DNF (RHEL):**

```
cat << EOF | sudo tee /etc/yum.repos.d/tanzu-cli.repo
[tanzu-cli]
name=Tanzu CLI
baseurl=https://storage.googleapis.com/tanzu-cli-os-packages/rpm/tanzu
-cli
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.vmware.com/tools/keys/VMWARE-PACKAGING-GPG-RSA
-KEY.pub
EOF

sudo yum install -y tanzu-cli # If you are using DNF, run sudo dnf ins
tall -y tanzu-cli.
```

2. Check that the correct version of the CLI is properly installed.

```
tanzu version
version: v1.3.0
...
```

Install from a binary release

Complete the following steps:

1. Download the Tanzu CLI binary from one of the following locations:
   - **VMware Tanzu Network**
     1. Go to VMware Tanzu Network.
     2. Choose the 1.9.1 release from the Release dropdown menu.
     3. Click the tanzu-core-cli-binaries item from the result set.
     4. Download the Tanzu CLI binary for your operating system.
   - **VMware Customer Connect**
     1. Go to VMware Customer Connect.
     2. Download the Tanzu CLI binary for your operating system.
   - **GitHub**
     1. Go to Tanzu CLI release v1.3.0 on GitHub.
     2. Download the Tanzu CLI binary for your operating system, for example, `tanzu-cli-windows-amd64.tar.gz`.
2. Use an extraction tool to unpack the binary file:
   - **macOS:**

```
tar -xvf tanzu-cli-darwin-amd64.tar.gz
```

- **Linux:**

  ```
  tar -xvf tanzu-cli-linux-amd64.tar.gz
  ```

- **Windows:**

  Use the Windows extractor tool to unzip `tanzu-cli-windows-amd64.zip`.

3. Make the CLI available to the system:
   - cd to the directory containing the extracted CLI binary
   - **macOS:**

     Install the binary to `/usr/local/bin`:

     ```
     install tanzu-cli-darwin_amd64 /usr/local/bin/tanzu
     ```

   - **Linux:**

     Install the binary to `/usr/local/bin`:

     ```
     sudo install tanzu-cli-linux_amd64 /usr/local/bin/tanzu
     ```

   - **Windows:**

     1. Create a new `Program Files\tanzu` folder.
     2. Copy the `tanzu-cli-windows_amd64.exe` file into the new `Program Files\tanzu` folder.
     3. Rename `tanzu-cli-windows_amd64.exe` to `tanzu.exe`.
     4. Right-click the `tanzu` folder, select **Properties** > **Security**, and make sure that your user account has the **Full Control** permission.
     5. Use Windows Search to search for `env`.
     6. Select **Edit the system environment variables** and click the **Environment Variables** button.
     7. Select the `Path` row under **System variables**, and click **Edit**.
     8. Click **New** to add a new row and enter the path to the Tanzu CLI. The path value must not include the `.exe` extension. For example, `C:\Program Files\tanzu`.

4. Check that the correct version of the CLI is properly installed:

   ```
   tanzu version
   version: v1.3.0
   ...
   ```

Install Tanzu CLI plug-ins

There is a group of Tanzu CLI plug-ins which extend the Tanzu CLI Core with Tanzu Application Platform specific feature functionality. The plug-ins can be installed as a group with a single command. Versioned releases of the Tanzu Application Platform specific plug-in group align to each supported Tanzu Application Platform version. This makes it easy to switch between different versions of Tanzu Application Platforms environments.

Use the following commands to search for, install, and verify Tanzu CLI plug-in groups.

List the versions of each plug-in group available across Tanzu

```
tanzu plugin group search --show-details
```

List the versions of the Tanzu Application Platform specific plug-in group

```
tanzu plugin group search --name vmware-tap/default --show-details
```

Install the version of the Tanzu Application Platform plug-in group matching your target environment

```
tanzu plugin install --group vmware-tap/default:v1.9
```

Verify the plug-in group list against the plug-ins that were installed

```
tanzu plugin group get vmware-tap/default:v1.9
```

```
tanzu plugin list
```

For air-gapped installation, see the Installing the Tanzu CLI in Internet-Restricted Environments section of the Tanzu CLI documentation.

Next steps

For online installation:

- Deploy Cluster Essentials*
- Install Tanzu Application Platform package and profiles

For air-gapped installation:

- Deploy Cluster Essentials*
- Install Tanzu Application Platform in your air-gapped environment

For installation on AWS Cloud:

- Create AWS Resources for Tanzu Application Platform
- Deploy Cluster Essentials*
- Install Tanzu Application Platform package and profiles on AWS

For installation on Azure Cloud:

- Create Azure Resources for Tanzu Application Platform
- Deploy Cluster Essentials*
- Install Tanzu Application Platform package and profiles on Azure

For installation on OpenShift clusters:

- Deploy Cluster Essentials*
- Install the Tanzu Application Platform package and profiles

For GitOps (beta) installation:

- Deploy Cluster Essentials*
- Install Tanzu Application Platform through GitOps with External Secrets Operator (ESO)

- Install Tanzu Application Platform through Gitops with Secrets OPerationS (SOPS)

* *When you use a VMware Tanzu Kubernetes Grid cluster, you do not need to install Cluster Essentials because the contents of Cluster Essentials are already installed on your cluster.*

Install Tanzu Application Platform (online)

To install Tanzu Application Platform (commonly known as TAP) on your Kubernetes clusters with internet access:

Step	Task	Link
1.	Review the prerequisites to ensure you have met all requirements before installing.	Prerequisites
2.	Accept Tanzu Application Platform EULAs and install the Tanzu CLI.	Accept Tanzu Application Platform EULAs and installing the Tanzu CLI
3.	Install Cluster Essentials for Tanzu*.	Deploy Cluster Essentials
4.	Add the Tanzu Application Platform package repository, prepare your Tanzu Application Platform profile, and install the profile to the cluster.	Install the Tanzu Application Platform package and profiles
5.	(Optional) Install any additional packages that were not in the profile.	Install individual packages
6.	Set up developer namespaces to use your installed packages.	Set up developer namespaces to use your installed packages
7.	Install developer tools into your integrated development environment (IDE).	Install Tanzu Developer Tools for your VS Code

* *When you use a VMware Tanzu Kubernetes Grid cluster, there is no need to install Cluster Essentials because the contents of Cluster Essentials are already installed on your cluster.*

After installing Tanzu Application Platform on to your Kubernetes clusters, proceed with Get started with Tanzu Application Platform.

Install Tanzu Application Platform (online)

To install Tanzu Application Platform (commonly known as TAP) on your Kubernetes clusters with internet access:

Step	Task	Link
1.	Review the prerequisites to ensure you have met all requirements before installing.	Prerequisites
2.	Accept Tanzu Application Platform EULAs and install the Tanzu CLI.	Accept Tanzu Application Platform EULAs and installing the Tanzu CLI
3.	Install Cluster Essentials for Tanzu*.	Deploy Cluster Essentials
4.	Add the Tanzu Application Platform package repository, prepare your Tanzu Application Platform profile, and install the profile to the cluster.	Install the Tanzu Application Platform package and profiles
5.	(Optional) Install any additional packages that were not in the profile.	Install individual packages
6.	Set up developer namespaces to use your installed packages.	Set up developer namespaces to use your installed packages
7.	Install developer tools into your integrated development environment (IDE).	Install Tanzu Developer Tools for your VS Code

* *When you use a VMware Tanzu Kubernetes Grid cluster, there is no need to install Cluster Essentials because the contents of Cluster Essentials are already installed on your cluster.*

After installing Tanzu Application Platform on to your Kubernetes clusters, proceed with Get started with Tanzu Application Platform.

Install Tanzu Application Platform package and profiles

This topic tells you how to install Tanzu Application Platform (commonly known as TAP) packages from your Tanzu Application Platform package repository.

Before installing the packages, ensure you have:

- Completed the Prerequisites.
- Configured and verified the cluster.
- Accepted Tanzu Application Platform EULA and installed Tanzu CLI with any required plug-ins.

Relocate images to a registry

VMware recommends relocating the images from VMware Tanzu Network registry to your own container image registry before attempting installation. If you don't relocate the images, Tanzu Application Platform depends on VMware Tanzu Network for continued operation, and VMware Tanzu Network offers no uptime guarantees. The option to skip relocation is documented for evaluation and proof-of-concept only.

The supported registries are Harbor, Azure Container Registry, Google Container Registry, and Quay.io. See the following documentation for a registry to learn how to set it up:

- Harbor documentation
- Google Container Registry documentation
- Quay.io documentation

To relocate images from the VMware Tanzu Network registry to your registry:

1. Set up environment variables for installation use by running:

```
# Set tanzunet as the source registry to copy the Tanzu Application Platform pa
ckages from.
export IMGPKG_REGISTRY_HOSTNAME_0=registry.tanzu.vmware.com
export IMGPKG_REGISTRY_USERNAME_0=MY-TANZUNET-USERNAME
export IMGPKG_REGISTRY_PASSWORD_0=MY-TANZUNET-PASSWORD

# The user's registry for copying the Tanzu Application Platform package to.
export IMGPKG_REGISTRY_HOSTNAME_1=MY-REGISTRY
export IMGPKG_REGISTRY_USERNAME_1=MY-REGISTRY-USER
export IMGPKG_REGISTRY_PASSWORD_1=MY-REGISTRY-PASSWORD
# These environment variables starting with IMGPKG_* are used by the imgpkg com
mand only.

# The registry from which the Tanzu Application Platform package is retrieved.
export INSTALL_REGISTRY_USERNAME="${IMGPKG_REGISTRY_USERNAME_1}"
export INSTALL_REGISTRY_PASSWORD="${IMGPKG_REGISTRY_PASSWORD_1}"
export INSTALL_REGISTRY_HOSTNAME="${IMGPKG_REGISTRY_HOSTNAME_1}"
export TAP_VERSION=VERSION-NUMBER
export INSTALL_REPO=TARGET-REPOSITORY

# The user's registry used by Tanzu Application Platform to store built images
and the Tanzu Build Service dependencies. These credentials must have write per
mission.
export MY_REGISTRY_USERNAME="${IMGPKG_REGISTRY_USERNAME_1}"
export MY_REGISTRY_PASSWORD="${IMGPKG_REGISTRY_PASSWORD_1}"
export MY_REGISTRY_HOSTNAME="${IMGPKG_REGISTRY_HOSTNAME_1}"
```

Where:

- `MY-REGISTRY-USER` is the user with write access to `MY-REGISTRY`.
- `MY-REGISTRY-PASSWORD` is the password for `MY-REGISTRY-USER`.
- `MY-REGISTRY` is your own container registry.
- `MY-TANZUNET-USERNAME` is the user with access to the images in the VMware Tanzu Network registry `registry.tanzu.vmware.com`.
- `MY-TANZUNET-PASSWORD` is the password for `MY-TANZUNET-USERNAME`.
- `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.9.1`.
- `TARGET-REPOSITORY` is your target repository, a folder/repository on `MY-REGISTRY` that serves as the location for the installation files for Tanzu Application Platform.

VMware recommends using a JSON key file to authenticate with Google Container Registry. In this case, the value of `INSTALL_REGISTRY_USERNAME` is `_json_key` and the value of `INSTALL_REGISTRY_PASSWORD` is the content of the JSON key file. For more information about how to generate the JSON key file, see Google Container Registry documentation.

2. Install the Carvel tool imgpkg CLI.

   To query for the available versions of Tanzu Application Platform on VMWare Tanzu Network Registry, run:

   ```
   imgpkg tag list -i registry.tanzu.vmware.com/tanzu-application-platform/tap-pac
   kages | sort -V
   ```

3. Relocate the images with the `imgpkg` CLI by running:

   ```
   imgpkg copy -b registry.tanzu.vmware.com/tanzu-application-platform/tap-package
   s:${TAP_VERSION} --to-repo ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/tap-pac
   kages
   ```

Add the Tanzu Application Platform package repository

Tanzu CLI packages are accessible through repositories. By adding the Tanzu Application Platform package repository, Tanzu Application Platform and its packages become available for installation.

Relocate images to a registry is recommended but not required for installation. If you skip that step, you can run the following commands to set Tanzu Network as the source of the OCI images:

```
# The registry from which the Tanzu Application Platform package is retrieved.
export INSTALL_REGISTRY_USERNAME=TANZUNET_REGISTRY_USERNAME
export INSTALL_REGISTRY_PASSWORD=TANZUNET_REGISTRY_PASSWORD
export INSTALL_REGISTRY_HOSTNAME="registry.tanzu.vmware.com"
export TAP_VERSION=VERSION-NUMBER
export INSTALL_REPO="tanzu-application-platform"

# The user's registry used by Tanzu Application Platform to store built images and the
Tanzu Build Service dependencies. These credentials must have write permission.
export MY_REGISTRY_USERNAME=MY-REGISTRY-USER
export MY_REGISTRY_PASSWORD=MY-REGISTRY-PASSWORD
export MY_REGISTRY_HOSTNAME=MY-REGISTRY
```

Where:

- `TANZUNET_REGISTRY_USERNAME` and `TANZUNET_REGISTRY_PASSWORD` are the credentials to the VMware Tanzu Network registry `registry.tanzu.vmware.com`
- `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.9.1`

- `MY_REGISTRY_HOSTNAME` is your own container registry.
- `MY_REGISTRY_USERNAME` is the user with write access to `MY_REGISTRY_HOSTNAME`.
- `MY_REGISTRY_PASSWORD` is the password for `MY_REGISTRY_USERNAME`.

To add the Tanzu Application Platform package repository to your cluster:

1. Create a namespace called `tap-install` for deploying any component packages by running:

   ```
   kubectl create ns tap-install
   ```

   This namespace keeps the objects grouped together logically.

2. Create a registry secret by running:

   ```
   tanzu secret registry add tap-registry \
     --username ${INSTALL_REGISTRY_USERNAME} --password ${INSTALL_REGISTRY_PASSWOR
   D} \
     --server ${INSTALL_REGISTRY_HOSTNAME} \
     --export-to-all-namespaces --yes --namespace tap-install
   ```

3. Create a secret for accessing the user's registry by running:

   ```
   tanzu secret registry add registry-credentials \
       --server   ${MY_REGISTRY_HOSTNAME} \
       --username ${MY_REGISTRY_USERNAME} \
       --password ${MY_REGISTRY_PASSWORD} \
       --namespace tap-install \
       --export-to-all-namespaces \
       --yes
   ```

4. Add the Tanzu Application Platform package repository to the cluster by running:

   ```
   tanzu package repository add tanzu-tap-repository \
     --url ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/tap-packages:$TAP_VERSION
   \
     --namespace tap-install
   ```

5. Get the status of the Tanzu Application Platform package repository, and ensure the status updates to `Reconcile succeeded` by running:

   ```
   tanzu package repository get tanzu-tap-repository --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package repository get tanzu-tap-repository --namespace tap-install
   - Retrieving repository tap...
   NAME:          tanzu-tap-repository
   VERSION:       16253001
   REPOSITORY:    tapmdc.azurecr.io/mdc/1.4.0/tap-packages
   TAG:           1.9.1
   STATUS:        Reconcile succeeded
   REASON:
   ```

   Note

   The `VERSION` and `TAG` numbers differ from the earlier example if you are on Tanzu Application Platform v1.0.2 or earlier.

6. List the available packages by running:

```
tanzu package available list --namespace tap-install
```

For example:

```
$ tanzu package available list --namespace tap-install
/ Retrieving available packages...
  NAME                                                DISPLAY-NAME
SHORT-DESCRIPTION
  accelerator.apps.tanzu.vmware.com                   Application Accelerator
for VMware Tanzu                                    Used to create new projects a
nd configurations.
  api-portal.tanzu.vmware.com                         API portal
A unified user interface for API discovery and exploration at scale.
  apis.apps.tanzu.vmware.com                          API Auto Registration fo
r VMware Tanzu                                      A TAP component to automatica
lly register API exposing workloads as API entities

in TAP GUI.
  backend.appliveview.tanzu.vmware.com                Application Live View fo
r VMware Tanzu                                      App for monitoring and troubl
eshooting running apps
  buildservice.tanzu.vmware.com                       Tanzu Build Service
Tanzu Build Service enables the building and automation of containerized

software workflows securely and at scale.
  carbonblack.scanning.apps.tanzu.vmware.com          VMware Carbon Black for
Supply Chain Security Tools - Scan                  Default scan templates using
VMware Carbon Black
  cartographer.conventions.apps.tanzu.vmware.com      Convention Service for V
Mware Tanzu                                         Convention Service enables ap
p operators to consistently apply desired runtime

configurations to fleets of workloads.
  cartographer.tanzu.vmware.com                       Cartographer
Kubernetes native Supply Chain Choreographer.
  cnrs.tanzu.vmware.com                               Cloud Native Runtimes
Cloud Native Runtimes is a serverless runtime based on Knative
  connector.appliveview.tanzu.vmware.com              Application Live View Co
nnector for VMware Tanzu                            App for discovering and regis
tering running apps
  controller.source.apps.tanzu.vmware.com             Tanzu Source Controller
Tanzu Source Controller enables workload create/update from source code.
  conventions.appliveview.tanzu.vmware.com            Application Live View Co
nventions for VMware Tanzu                          Application Live View convent
ion server
  developer-conventions.tanzu.vmware.com              Tanzu App Platform Devel
oper Conventions                                    Developer Conventions
  external-secrets.apps.tanzu.vmware.com              External Secrets Operato
r                                                   External Secrets Operator is
a Kubernetes operator that integrates external

secret management systems.
  fluxcd.source.controller.tanzu.vmware.com           Flux Source Controller
The source-controller is a Kubernetes operator, specialised in artifacts

acquisition from external sources such as Git, Helm repositories and S3 bucket
s.
  grype.scanning.apps.tanzu.vmware.com                Grype for Supply Chain S
ecurity Tools - Scan                                Default scan templates using
Anchore Grype
  metadata-store.apps.tanzu.vmware.com                Supply Chain Security To
ols - Store                                         Post SBoMs and query for imag
e, package, and vulnerability metadata.
  namespace-provisioner.apps.tanzu.vmware.com         Namespace Provisioner
Automatic Provisioning of Developer Namespaces.
```

```
  ootb-delivery-basic.tanzu.vmware.com                 Tanzu App Platform Out o
f The Box Delivery Basic                           Out of The Box Delivery Basi
c.
  ootb-supply-chain-basic.tanzu.vmware.com             Tanzu App Platform Out o
f The Box Supply Chain Basic                       Out of The Box Supply Chain B
asic.
  ootb-supply-chain-testing-scanning.tanzu.vmware.com  Tanzu App Platform Out o
f The Box Supply Chain with Testing and Scanning   Out of The Box Supply Chain w
ith Testing and Scanning.
  ootb-supply-chain-testing.tanzu.vmware.com           Tanzu App Platform Out o
f The Box Supply Chain with Testing                Out of The Box Supply Chain w
ith Testing.
  ootb-templates.tanzu.vmware.com                      Tanzu App Platform Out o
f The Box Templates                                Out of The Box Templates.
  policy.apps.tanzu.vmware.com                         Supply Chain Security To
ols - Policy Controller                            Policy Controller enables def
ining of a policy to restrict unsigned container

images.
  scanning.apps.tanzu.vmware.com                       Supply Chain Security To
ols - Scan                                         Scan for vulnerabilities and
enforce policies directly within Kubernetes native

Supply Chains.
  service-bindings.labs.vmware.com                     Service Bindings for Kub
ernetes                                            Service Bindings for Kubernet
es implements the Service Binding Specification.
  services-toolkit.tanzu.vmware.com                    Services Toolkit
The Services Toolkit enables the management, lifecycle, discoverability and

connectivity of Service Resources (databases, message queues, DNS records,

etc.).
  snyk.scanning.apps.tanzu.vmware.com                  Snyk for Supply Chain Se
curity Tools - Scan                                Default scan templates using
Snyk
  spring-boot-conventions.tanzu.vmware.com             Tanzu Spring Boot Conven
tions Server                                       Default Spring Boot conventio
n server.
  sso.apps.tanzu.vmware.com                            AppSSO
Application Single Sign-On for Tanzu
  tap-auth.tanzu.vmware.com                            Default roles for Tanzu
Application Platform                               Default roles for Tanzu Appli
cation Platform
  tap-gui.tanzu.vmware.com                             Tanzu Developer Portal
web app graphical user interface for Tanzu Application Platform
  tap-telemetry.tanzu.vmware.com                       Telemetry Collector for
Tanzu Application Platform                         Tanzu Application Platform Te
lemetry
  tap.tanzu.vmware.com                                 Tanzu Application Platfo
rm                                                 Package to install a set of T
AP components to get you started based on your use

case.
  tekton.tanzu.vmware.com                              Tekton Pipelines
Tekton Pipelines is a framework for creating CI/CD systems.
```

Install your Tanzu Application Platform profile

The `tap.tanzu.vmware.com` package installs predefined sets of packages based on your profile settings. This is done by using the package manager installed by Tanzu Cluster Essentials.

For more information about profiles, see Components and installation profiles.

To prepare to install a profile:

1. List version information for the package by running:

   ```
   tanzu package available list tap.tanzu.vmware.com --namespace tap-install
   ```

2. Create a `tap-values.yaml` file by using the Full Profile sample in the following section as a guide. These samples have the minimum configuration required to deploy Tanzu Application Platform. The sample values file contains the necessary defaults for:

   - The meta-package, or parent Tanzu Application Platform package.
   - Subordinate packages, or individual child packages.

   Keep the values file for future configuration use.

   > **Note**
   >
   > `tap-values.yaml` is set as a Kubernetes secret, which provides secure means to read credentials for Tanzu Application Platform components.

3. View possible configuration settings for your package

Full profile

The following is the YAML file sample for the full-profile. The `profile:` field takes `full` as the default value, but you can also set it to `iterate`, `build`, `run` or `view`. Refer to Install multicluster Tanzu Application Platform profiles for more information.

```
shared:
  ingress_domain: "INGRESS-DOMAIN"
  ingress_issuer: # Optional, can denote a cert-manager.io/v1/ClusterIssuer of your ch
oice. Defaults to "tap-ingress-selfsigned".

  image_registry:
    project_path: "SERVER-NAME/REPO-NAME"
    secret:
      name: "KP-DEFAULT-REPO-SECRET"
      namespace: "KP-DEFAULT-REPO-SECRET-NAMESPACE"

  kubernetes_distribution: "K8S-DISTRO" # Only required if the distribution is OpenShi
ft and must be used with the following kubernetes_version key.

  kubernetes_version: "K8S-VERSION" # Required regardless of distribution when Kuberne
tes version is 1.25 or later.

  ca_cert_data: | # To be passed if using custom certificates.
      -----BEGIN CERTIFICATE-----
      MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...
      -----END CERTIFICATE-----

ceip_policy_disclosed: FALSE-OR-TRUE-VALUE # Installation fails if this is not set to
true. Not a string.

#The above keys are minimum numbers of entries needed in tap-values.yaml to get a func
tioning TAP Full profile installation.

#Below are the keys which may have default values set, but can be overridden.

profile: full # Can take iterate, build, run, view.

supply_chain: basic # Can take testing, testing_scanning.

ootb_supply_chain_basic: # Based on supply_chain set above, can be changed to ootb_sup
```

```
ply_chain_testing, ootb_supply_chain_testing_scanning.
  source:
    credentials_secret: "GIT-SOURCE-CREDENTIAL-SECRET-NAME" # (Optional) Defaults to
"".
  registry:
    server: "SERVER-NAME" # Takes the value from the shared section by default, but ca
n be overridden by setting a different value.
    repository: "REPO-NAME" # Takes the value from the shared section by default, but
can be overridden by setting a different value.
  gitops:
    credentials_secret: "GITOPS-CREDENTIAL-SECRET-NAME" # (Optional) Defaults to "".

contour:
  envoy:
    service:
      type: LoadBalancer # This is set by default, but can be overridden by setting a
different value.

buildservice:
  # Takes the value from the shared section by default, but can be overridden by setti
ng a different value.
  kp_default_repository: "KP-DEFAULT-REPO"
  kp_default_repository_secret: # Takes the value from the shared section above by def
ault, but can be overridden by setting a different value.
    name: "KP-DEFAULT-REPO-SECRET"
    namespace: "KP-DEFAULT-REPO-SECRET-NAMESPACE"

local_source_proxy:
  # Takes the value from the project_path under the image_registry section of shared b
y default, but can be overridden by setting a different value.
  repository: "EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE"
  push_secret:
    # When set to true, the secret mentioned in this section is automatically exported
to Local Source Proxy's namespace.
    name: "EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE-SECRET"
    namespace: "EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE-SECRET-NAMESPACE"
    # When set to true, the secret mentioned in this section is automatically exported
to Local Source Proxy's namespace.
    create_export: true

tap_gui:
  metadataStoreAutoconfiguration: true # Creates a service account, the Kubernetes con
trol plane token and the requisite app_config block to enable communications between T
anzu Developer Portal and SCST - Store.
  app_config:
    auth:
      allowGuestAccess: true  # This allows unauthenticated users to log in to your po
rtal. If you want to deactivate it, make sure you configure an alternative auth provid
er.
    catalog:
      locations:
        - type: url
          target: https://GIT-CATALOG-URL/catalog-info.yaml

metadata_store:
  ns_for_export_app_cert: "MY-DEV-NAMESPACE" # Verify this namespace is available with
in your cluster before initiating the Tanzu Application Platform installation.
  app_service_type: ClusterIP # Defaults to LoadBalancer. If shared.ingress_domain is
set earlier, this must be set to ClusterIP.

policy:
  tuf_enabled: false # By default, TUF initialization and keyless verification are dea
ctivated.
tap_telemetry:
  customer_entitlement_account_number: "CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER" # (Option
al) Identify data for creating the Tanzu Application Platform usage reports.
```

Important

The profile installation no longer includes Grype out of the box. Instead, you can use Namespace Provisioner to install Grype. Namespace Provisioner still uses Grype values to configure the scanner:

```
grype:
  targetImagePullSecret: "TARGET-REGISTRY-CREDENTIALS-SECRET"
```

Where:

- `INGRESS-DOMAIN` is the subdomain for the host name that you point at the `tanzu-shared-ingress` service's External IP address. It is not required to know the External IP address or set up the DNS record while installing. Installing the Tanzu Application Platform package creates the `tanzu-shared-ingress` and its External IP address. You can create the DNS record after completing the installation.
- `KP-DEFAULT-REPO` is a writable repository in your registry. Tanzu Build Service dependencies are written to this location. Examples:
    - Harbor has the form `kp_default_repository: "my-harbor.io/my-project/build-service"`.
    - Docker Hub has the form `kp_default_repository: "my-dockerhub-user/build-service"` or `kp_default_repository: "index.docker.io/my-user/build-service"`.
    - Google Cloud Registry has the form `kp_default_repository: "gcr.io/my-project/build-service"`.
- `KP-DEFAULT-REPO-SECRET` is the secret with user credentials that can write to `KP-DEFAULT-REPO`. You can `docker push` to this location with this credential.
    - You can create a secret configured with a valid registry credential with a name and namespace of your choice. For Google Cloud Registry, use `kp_default_repository_username: _json_key`.
    - You must create the secret before the installation. For example, you can use the `registry-credentials` secret created earlier.
- `KP-DEFAULT-REPO-SECRET-NAMESPACE` is the namespace where `KP-DEFAULT-REPO-SECRET` is created.
    - You must create the namespace before the installation. For example, you can use the `tap-install` namespace created earlier.
- `K8S-DISTRO` (optional) is the type of Kubernetes infrastructure in use. It is only required if the distribution is OpenShift and must be used in coordination with `kubernetes_version`. Supported value: `openshift`.
- `K8S-VERSION` (optional) is the Kubernetes version in use. You can use it independently or in coordination with `kubernetes_distribution`. For example, `1.24.x`, where `x` is the Kubernetes patch version.
- `SERVER-NAME` is the host name of the registry server. Examples:
    - Harbor has the form `server: "my-harbor.io"`.
    - Docker Hub has the form `server: "index.docker.io"`.
    - Google Cloud Registry has the form `server: "gcr.io"`.

- `REPO-NAME` is where workload images are stored in the registry. If this key is passed through the shared section earlier and AWS ECR registry is used, you must ensure that the `SERVER-NAME/REPO-NAME/buildservice` and `SERVER-NAME/REPO-NAME/workloads` exist. AWS ECR expects the paths to be pre-created. Images are written to `SERVER-NAME/REPO-NAME/workload-name`. Examples:
    - Harbor has the form `repository: "my-project/supply-chain"`.
    - Docker Hub has the form `repository: "my-dockerhub-user"`.
    - Google Cloud Registry has the form `repository: "my-project/supply-chain"`.
- `EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE` is where the developer's local source is uploaded when using Tanzu CLI to use Local Source Proxy for workload creation.

  If an AWS ECR registry is being used, ensure that the repository already exists. AWS ECR expects the repository path to already exist. This destination is represented as `REGISTRY-SERVER/REPOSITORY-PATH`. For more information, see Install Local Source Proxy.
- `EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE-SECRET` is the name of the secret with credentials that allow pushing to the `EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE` repository.
- `EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE-SECRET-NAMESPACE` is the namespace in which `EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE-SECRET` is available.
- `GIT-SOURCE-CREDENTIAL-SECRET-NAME` is the name of the Kubernetes secret in the developer namespace that supplies the Git credentials for the supply chain to fetch source code from. This field is only required if you use a private repository. See Git authentication for more information.
- `GITOPS-CREDENTIAL-SECRET-NAME` is the name of the Kubernetes secret in the developer namespace that supplies the Git credentials for the supply chain to push configuration to. See Git authentication for more information.
- `GIT-CATALOG-URL` is the path to the `catalog-info.yaml` catalog definition file. You can download either a blank or populated catalog file from the Tanzu Application Platform product page. Otherwise, you can use a Backstage-compliant catalog you've already built and posted on the Git infrastructure.
- `MY-DEV-NAMESPACE` is the name of the developer namespace. SCST - Store exports secrets to the namespace, and SCST - Scan deploys the `ScanTemplates` there. This allows the scanning feature to run in this namespace. If there are multiple developer namespaces, use `ns_for_export_app_cert: "*"` to export the SCST - Store CA certificate to all namespaces. To install Grype in multiple namespaces, use a namespace provisioner. For more information, see Namespace Provisioner.
- `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the secret that contains the credentials to pull an image from the registry for scanning.
- `CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER` (optional) refers to the Entitlement Account Number (EAN), which is a unique identifier VMware assigns to its customers. Tanzu Application Platform telemetry uses this number to identify data that belongs to a particular customers and prepare usage reports.

If you use custom CA certificates, you must provide one or more PEM-encoded CA certificates under the `ca_cert_data` key. If you configured `shared.ca_cert_data`, Tanzu Application Platform component packages inherit that value by default.

If you use AWS, the default settings creates a classic LoadBalancer. To use the Network LoadBalancer instead of the classic LoadBalancer for ingress, add the following to your `tap-values.yaml`:

```
contour:
  infrastructure_provider: aws
  envoy:
    service:
      aws:
        LBType: nlb
```

CEIP policy disclosure

Tanzu Application Platform is part of VMware's CEIP program where data is collected to help improve the customer experience. By setting `ceip_policy_disclosed` to `true` (not a string), you acknowledge the program is disclosed to you and you are aware data collection is happening. This field must be set for the installation to be completed.

See Opt out of telemetry collection for more information.

(Optional) Configure your profile with full dependencies

When you install a profile that includes Tanzu Build Service, Tanzu Application Platform is installed with the `lite` set of dependencies. These dependencies consist of buildpacks and stacks required for application builds.

The `lite` set of dependencies do not contain all buildpacks and stacks. To use all buildpacks and stacks, you must install the `full` dependencies. For more information about the differences between `lite` and `full` dependencies, see About lite and full dependencies.

To configure `full` dependencies, add the key-value pair `exclude_dependencies: true` to your `tap-values.yaml` file under the `buildservice` section. For example:

```
buildservice:
  kp_default_repository: "KP-DEFAULT-REPO"
  kp_default_repository_secret: # Takes the value from the shared section by default,
but can be overridden by setting a different value.
    name: "KP-DEFAULT-REPO-SECRET"
    namespace: "KP-DEFAULT-REPO-SECRET-NAMESPACE"
  exclude_dependencies: true
```

After configuring `full` dependencies, you must install the dependencies after you have finished installing your Tanzu Application Platform package. See Install the full dependencies package for more information.

Tanzu Application Platform v1.6.1 and later supports building applications with Ubuntu v22.04 (Jammy).

(Optional) Configure your profile with automatic dependency updates

Tanzu Build Service dependencies might be upgraded between Tanzu Application Platform releases, for example, if a CVE is discovered in the OS (stack update) or language (buildpack update).

Automatic dependency updates enable your cluster to consume the stack and buildpack updates immediately instead of waiting for the next Tanzu Application Platform patch release to pull in the updated dependencies.

- Updates are provided through a separate package repository with available version lines for all supported Tanzu Application Platform minor versions.
- Within a version line, only patch versions are incremented to avoid breaking changes.

- You can customize the packages that you want the automatic dependency updater to update through your `tap-values.yaml` file or your full dependencies values.

Prerequisites: These steps assume a registry secret already exists in the cluster for accessing `registry.tanzu.vmware.com` and your registry.

To enable automatic dependency updates:

1. Add the following to your `tap-values.yaml` file:

   ```
   buildservice:
     dependency_updates:
       allow: true
       scope: SCOPE
       include_packages: [""]
       exclude_packages: [""]
   ```

   Where:

   - `SCOPE` is the list of dependencies you want updated. The options are:
     - `stacks-only` (default): Only stacks and builders are updated. This addresses CVEs in the base image or operating system.
     - `all`: Stacks, builders, and buildpacks are updated. This addresses CVEs in the base image or operating system and CVEs in the language toolchain such as compilers, interpreters, and standard libraries.
     - `custom`: This list is empty by default. Use the `include_packages` key to add packages to be updated.

   Note

   You must update the Tanzu Application Platform package install and the Full Dependencies package install after changing the `tap-values.yaml`.

2. Add the Tanzu Build Service Dependency Updates package repository by running:

   ```
   kubectl apply -f - <<EOF
     apiVersion: packaging.carvel.dev/v1alpha1
     kind: PackageRepository
     metadata:
       name: tbs-dependencies-package-repository
       namespace: tap-install
     spec:
       fetch:
         imgpkgBundle:
           image: DEPENDENCY-UPDATER-PACKAGE-REPO
           tagSelection:
             semver:
               constraints: VERSION-CONSTRAINT
   EOF
   ```

   Where:

   - `DEPENDENCY-UPDATER-PACKAGE-REPO` is the location of the package repository. This is `registry.tanzu.vmware.com/build-service-dependency-updater/package-repo` for online installs and the internal container image registry for air-gapped installs.
   - `VERSION-CONSTRAINT` is the Tanzu Application Platform version in the form of `MAJOR.MINOR.x`. For example, `1.8.x`.

After completing this configuration, the repository you set with `DEPENDENCY-UPDATER-PACKAGE-REPO` will be polled for updates and any new releases will automatically be made available to the cluster.

(Optional) Override the default retention behavior for Crossplane CRDs

By default, the `crossplane.tanzu.vmware.com` package is configured to retain all Crossplane CRDs, providers, and managed resources when the package is uninstalled. This is in the interest of caution in relation to accidental deletion of stateful data.

You can configure Tanzu Application Platform to delete Crossplane resources to avoid orphaned resources. To do so, update the `tap-values.yaml` as follows:

```
# tap-values.yaml

crossplane:
  orphan_resources: false
```

Install your Tanzu Application Platform package

Follow these steps to install the Tanzu Application Platform package:

1. Install the package by running:

   ```
   tanzu package install tap -p tap.tanzu.vmware.com -v $TAP_VERSION --values-file
   tap-values.yaml -n tap-install
   ```

2. Verify the package install by running:

   ```
   tanzu package installed get tap -n tap-install
   ```

   This can take 5-10 minutes because it installs several packages on your cluster.

3. Verify that the necessary packages in the profile are installed by running:

   ```
   tanzu package installed list -A
   ```

4. If you configured `full` dependencies in your `tap-values.yaml` file, install the `full` dependencies by following the procedure in Install full dependencies.

Important

After installing the full profile on your cluster, you must set up developer namespaces. Otherwise, creating a workload, a Knative service or other Tanzu Application Platform packages fails. For more information, see Set up developer namespaces to use your installed packages.

You can run the following command after reconfiguring the profile to reinstall the Tanzu Application Platform:

```
tanzu package installed update tap -p tap.tanzu.vmware.com -v $TAP_VERSION  --values-f
ile tap-values.yaml -n tap-install
```

Install the full dependencies package

If you configured `full` dependencies in your `tap-values.yaml` file in Configure your profile with full dependencies earlier, you must install the `full` dependencies package.

1. (Optional) If you have an existing installation of the full dependencies package from a version earlier than Tanzu Application Platform v1.6.1, you must uninstall the full dependencies package and remove the package repository:

   Uninstall the package:

   ```
   tanzu package installed delete full-tbs-deps -n tap-install
   ```

   Remove the package repository:

   ```
   tanzu package repository delete tbs-full-deps-repository -n tap-install
   ```

2. Get the latest version of the Tanzu Application Platform package by running:

   ```
   tanzu package available list tap.tanzu.vmware.com --namespace tap-install
   ```

3. If you have not done so already, you must exclude the default dependencies by adding the key-value pair `exclude_dependencies: true` to your `tap-values.yaml` file under the `buildservice` section. For example:

   ```
   buildservice:
     exclude_dependencies: true
   ```

4. If you have not updated your Tanzu Application Platform package installation after adding the key-value pair `exclude_dependencies: true` to your values file, perform the update by running:

   ```
   tanzu package installed update tap --namespace tap-install --values-file VALUES
   -FILE
   ```

   Where `VALUES-FILE` is the path to the `tap-values.yaml` file you edited earlier.

5. Relocate the Tanzu Build Service `full` dependencies package repository by doing one of the following:

   - Relocate the images directly for online installation:

     ```
     imgpkg copy \
       -b registry.tanzu.vmware.com/tanzu-application-platform/full-deps-packa
     ge-repo:VERSION \
       --to-repo ${INSTALL_REGISTRY_HOSTNAME}/full-deps-package-repo
     ```

     Where `VERSION` is the version of the Tanzu Application Platform package you retrieved earlier.

   - Relocate the images to an external storage device and then to the registry in the air-gapped environment:

     ```
     imgpkg copy \
       -b registry.tanzu.vmware.com/tanzu-application-platform/full-deps-packa
     ge-repo:VERSION \
       --to-tar=full-deps-package-repo.tar

     # move full-deps-package-repo.tar to environment with registry access
     imgpkg copy \
       --tar full-deps-package-repo.tar \
       --to-repo=INSTALL-REGISTRY-HOSTNAME/TARGET-REPOSITORY/full-deps-package
     -repo
     ```

Where:

- `VERSION` is the version of the Tanzu Application Platform package you retrieved earlier.
- `INSTALL-REGISTRY-HOSTNAME` is your container registry.
- `TARGET-REPOSITORY` is your target repository.

6. Add the Tanzu Build Service `full` dependencies package repository by running:

```
tanzu package repository add full-deps-package-repo \
  --url INSTALL-REGISTRY-HOSTNAME/TARGET-REPOSITORY/full-deps-package-repo:VERS
ION \
  --namespace tap-install
```

Where:

- `INSTALL-REGISTRY-HOSTNAME` is your container registry.
- `TARGET-REPOSITORY` is your target repository.
- `VERSION` is the version of the Tanzu Application Platform package you retrieved earlier.

7. Create a new `tbs-full-deps-values.yaml` and copy the `kp_default_repository` key-value pair from your `tap-values.yaml` or `tbs-values.yaml`:

```
---
 kp_default_repository: "REPO-NAME"
 kp_default_repository_secret:
   name: kp-default-repository-creds
   namespace: tap-install
```

Where `REPO-NAME` is copied from the `buildservice.kp_default_repository` field in your `tap-values.yaml` or `tbs-values.yaml`.

1. (Optional) Install the UBI builder.

   The UBI builder uses Red Hat Universal Base Image (UBI) v8 for both build and run images. This builder only supports Java and Node.js. To install the UBI builder, add the key-value pair `enable_ubi_builder: true` to your `tbs-full-deps-values.yaml`.

   ```
   ---
   enable_ubi_builder: true
   ```

2. (Optional) Install the Static builder.

   The Static builder uses Ubuntu Jammy for both build images and a minimal static run image. This builder only supports Golang. To install the Static builder, add the key-value pair `enable_static_builder: true` to your `tbs-full-deps-values.yaml`.

   ```
   ---
   enable_static_builder: true
   ```

8. Install the `full` dependencies package by running:

```
tanzu package install full-deps \
  --package full-deps.buildservice.tanzu.vmware.com \
  --version "> 0.0.0" \
  --namespace tap-install \
  --values-file VALUES-FILE
```

Where `VALUES-FILE` is the path to the `tbs-full-deps-values.yaml` you created earlier.

For more information about the differences between `lite` and `full` dependencies, see About lite and full dependencies.

Access Tanzu Developer Portal

To access Tanzu Developer Portal, you can use the host name that you configured earlier. This host name is pointed at the shared ingress. To configure LoadBalancer for Tanzu Developer Portal, see Access Tanzu Developer Portal.

You're now ready to start using Tanzu Developer Portal. Proceed to the Getting Started topic or the Tanzu Developer Portal - Catalog Operations topic.

Exclude packages from a Tanzu Application Platform profile

To exclude packages from a Tanzu Application Platform profile:

1. Find the full subordinate (child) package name:

   ```
   tanzu package available list --namespace tap-install
   ```

2. Update your `tap-values` file with a section listing the exclusions:

   ```
   profile: PROFILE-VALUE
   excluded_packages:
     - tap-gui.tanzu.vmware.com
     - service-bindings.lab.vmware.com
   ```

Important

If you exclude a package after performing a profile installation including that package, you cannot see the accurate package states immediately after running `tap package installed list -n tap-install`. Also, you can break package dependencies by removing a package. Allow 20 minutes to verify that all packages have reconciled correctly while troubleshooting.

Next steps

- (Optional) Install individual packages
- Set up developer namespaces to use your installed packages
- Replace the default ingress issuer

View possible configuration settings for your package

To view possible configuration settings for a package, run:

```
tanzu package available get tap.tanzu.vmware.com/$TAP_VERSION --values-schema --namesp
ace tap-install
```

Note

The `tap.tanzu.vmware.com` package does not show all configuration settings for packages it plans to install. The package only shows top-level keys. You can view

individual package configuration settings with the same `tanzu package available get` command. For example, to find the keys for Cloud Native Runtimes, you must first identify the version of the package with `tanzu package installed list -n tap-install`, which lists all the installed packages versions. Then run the command `tanzu package available get -n tap-install cnrs.tanzu.vmware.com/CNR-VERSION --values-schema` by using the package version listed for Cloud Native Runtimes.

```
profile: full

# Shared configurations go under the shared key.
shared:
  ingress_domain: tap.example.com

# ...

# For example, Cloud Native Runtimes specific values go under its name.
cnrs:
  provider: local

# For example, App Accelerator specific values go under its name.
accelerator:
  server:
    service_type: "ClusterIP"
```

Shared Keys define values that configure multiple packages. These keys are defined under the `shared` Top-level Key, as summarized in the following table:

Shared Key	Description	Optional
`ca_cert_data`	PEM-encoded certificate data to trust TLS connections with a private CA. This shared key is used by `convention_controller`, `scanning` and the Tanzu `source_controller` (not the Flux CD Source Controller).	Yes
`ingress_domain`	Domain name to be used in service routes and host names for instances of Tanzu Application Platform components.	Yes
`ingress_issuer`	A `cert-manager.io/v1/ClusterIssuer` for issuing TLS certificates to Tanzu Application Platform components. Default value: `tap-ingress-selfsigned`	Yes
`kubernetes_distribution`	Type of Kubernetes infrastructure being used. You can use this shared key in coordination with the `kubernetes_version` key. Supported value: `openshift`.	Yes
`kubernetes_version`	Kubernetes version. You can use this shared key independently or in coordination with the `kubernetes_distribution` key. Supported value: `1.24.x`, where `x` stands for the Kubernetes patch version.	Yes
`image_registry.project_path`	Project path in the container image registry server used for builder and application images.	Yes
`image_registry.username`	User name for the container image registry. Mutually exclusive with `shared.image_registry.secret.name/namespace`	Yes
`image_registry.password`	Password for the container image registry. Mutually exclusive with `shared.image_registry.secret.name/namespace`	Yes
`secret.name`	Secret name for the container image registry credentials of type `kubernetes.io/dockerconfigjson`. Mutually exclusive with `shared.image_registry.username/password`	Yes
`secret.namespace`	Secret namespace for the container image registry credentials. Mutually exclusive with `shared.image_registry.username/password`	Yes

Shared Key	Description	Optional
`activateAppLiveViewSecureAccessControl`	Enable secure access connection between Application Live View components.	Yes

The following table summarizes the top-level keys used for package-specific configuration within your `tap-values.yaml`.

Package	Top-level Key
See table above.	`shared`
API Auto Registration	`api_auto_registration`
API portal	`api_portal`
Application Accelerator	`accelerator`
Application Live View	`appliveview`
Application Live View connector	`appliveview_connector`
Application Live View conventions	`appliveview-conventions`
Cartographer Conventions	`cartographer_conventions`
Cartographer	`cartographer`
Cloud Native Runtimes	`cnrs`
Source Controller	`source_controller`
Supply Chain	`supply_chain`
Supply Chain Basic	`ootb_supply_chain_basic`
Supply Chain Testing	`ootb_supply_chain_testing`
Supply Chain Testing Scanning	`ootb_supply_chain_testing_scanning`
Supply Chain Security Tools - Scan	`scanning`
Supply Chain Security Tools - Scan (Grype Scanner)	`grype`
Supply Chain Security Tools - Store	`metadata_store`
Build Service	`buildservice`
Tanzu Developer Portal	`tap_gui`

For information about package-specific configuration, see Install individual packages.

Install individual packages

You can install Tanzu Application Platform (commonly known as TAP) through predefined profiles or through individual packages. Use this topic to learn how to install each individual package. For more information about installing through profiles, see Components and installation profiles.

Installing individual Tanzu Application Platform packages is useful if you do not want to use a profile to install packages or if you want to install additional packages after installing a profile. Before installing the packages, be sure to complete the prerequisites, configure and verify the cluster, accept the EULA, and install the Tanzu CLI with any required plug-ins. For more information, see Prerequisites.

Install pages for individual Tanzu Application Platform packages

- Install API Auto Registration
- Install API portal
- Install Application Accelerator
- Install Application Configuration Service
- Install Application Live View
- Install Application Single Sign-On
- Install Bitnami Services
- Install cert-manager
- Install Cloud Native Runtimes
- Install Contour
- Install Crossplane
- Install default roles for Tanzu Application Platform
- Install Developer Conventions
- Install Flux CD Source Controller
- Install Out of the Box Templates
- Install Out of the Box Supply Chain with Testing
- Install Out of the Box Supply Chain with Testing and Scanning
- Install Service Bindings
- Install Services Toolkit
- Install Source Controller
- Install Spring Boot conventions
- Install Supply Chain Choreographer
- Install Supply Chain Security Tools - Store
- Install Supply Chain Security Tools - Policy Controller
- Install Supply Chain Security Tools - Scan
- Install Tanzu Developer Portal
- Install Tanzu Build Service
- Install Tekton
- Install Telemetry

Verify the installed packages

Use the following procedure to verify that the packages are installed.

1. List the installed packages by running:

   ```
   tanzu package installed list --namespace tap-install
   ```

   For example:

```
$ tanzu package installed list --namespace tap-install
\ Retrieving installed packages...
NAME                     PACKAGE-NAME                                      PAC
KAGE-VERSION  STATUS
api-portal               api-portal.tanzu.vmware.com                       1.
0.3           Reconcile succeeded
app-accelerator          accelerator.apps.tanzu.vmware.com                 1.
0.0           Reconcile succeeded
app-live-view            appliveview.tanzu.vmware.com                      1.
0.2           Reconcile succeeded
appliveview-conventions  build.appliveview.tanzu.vmware.com                1.
0.2           Reconcile succeeded
cartographer             cartographer.tanzu.vmware.com                     0.
1.0           Reconcile succeeded
cloud-native-runtimes    cnrs.tanzu.vmware.com                             1.
0.3           Reconcile succeeded
convention-controller    controller.conventions.apps.tanzu.vmware.com      0.
7.0           Reconcile succeeded
developer-conventions    developer-conventions.tanzu.vmware.com            0.
3.0-build.1   Reconcile succeeded
grype-scanner            grype.scanning.apps.tanzu.vmware.com              1.
0.0           Reconcile succeeded
image-policy-webhook     image-policy-webhook.signing.apps.tanzu.vmware.com 1.
1.2           Reconcile succeeded
metadata-store           metadata-store.apps.tanzu.vmware.com              1.
0.2           Reconcile succeeded
ootb-supply-chain-basic  ootb-supply-chain-basic.tanzu.vmware.com          0.
5.1           Reconcile succeeded
ootb-templates           ootb-templates.tanzu.vmware.com                   0.
5.1           Reconcile succeeded
scan-controller          scanning.apps.tanzu.vmware.com                    1.
0.0           Reconcile succeeded
service-bindings         service-bindings.labs.vmware.com                  0.
5.0           Reconcile succeeded
services-toolkit         services-toolkit.tanzu.vmware.com                 0.
8.0           Reconcile succeeded
source-controller        controller.source.apps.tanzu.vmware.com           0.
2.0           Reconcile succeeded
sso4k8s-install          sso.apps.tanzu.vmware.com                         1.
0.0-beta.2-31 Reconcile succeeded
tap-gui                  tap-gui.tanzu.vmware.com                          0.
3.0-rc.4      Reconcile succeeded
tekton-pipelines         tekton.tanzu.vmware.com                           0.3
0.0          Reconcile succeeded
tbs                      buildservice.tanzu.vmware.com                     1.
5.0           Reconcile succeeded
```

Next steps

- Set up developer namespaces to use your installed packages

Set up developer namespaces to use your installed packages

For details about how to automatically set up your developer namespaces, see Provision developer namespaces in Namespace Provisioner.

Additional configuration for testing and scanning

If you plan to install or have already installed Out of the Box Supply Chains with Testing and Scanning, you can use Namespace Provisioner to set up the required resources. For more

information, see Customize installation in the Namespace Provisioner documentation for configuration steps.

Legacy namespace setup

To use the legacy manual process for setting up developer namespaces, see Legacy namespace setup.

Next steps

- Install Tanzu Developer Tools for your VS Code

Provision namespaces manually

This topic tells you how to use Namespace Provisioner to provision namespaces manually in Tanzu Application Platform (commonly known as TAP).

Using Namespace Provisioner is the recommended best practice for setting up developer namespaces on Tanzu Application Platform.

To provision namespaces manually, complete the following steps:

1. Enable single user access.
2. (Optional) Enable additional users with Kubernetes RBAC.

Enable single user access

1. To add read/write registry credentials to the developer namespace, run the following command:

   ```
   tanzu secret registry add registry-credentials --server REGISTRY-SERVER --usern
   ame REGISTRY-USERNAME --password REGISTRY-PASSWORD --namespace YOUR-NAMESPACE
   ```

   Where:

   - `YOUR-NAMESPACE` is the name you give to the developer namespace. For example, use `default` for the default namespace.
   - `REGISTRY-SERVER` is the URL of the registry. You can use the same registry server as in `ootb_supply_chain_basic` - `registry` - `server`. For more information, see Install Tanzu Application Platform package and profiles.
     - For Docker Hub, the value is `https://index.docker.io/v1/`. It must have the leading `https://`, the `v1` path, and the trailing `/`.
     - For Google Container Registry (GCR), the value is `gcr.io`.
   - `REGISTRY-PASSWORD` is the password of the registry.
     - For GCR or Google Artifact Registry, this must be the concatenated version of the JSON key. For example: `"$(cat ~/gcp-key.json)"`

   If you observe the following issue:

   ```
   panic: runtime error: invalid memory address or nil pointer dereference
   [signal SIGSEGV: segmentation violation code=0x1 addr=0x128 pc=0x2bcce00]
   ```

   Use kubectl to create the secret instead:

```
kubectl create secret docker-registry registry-credentials --docker-server=REGI
STRY-SERVER --docker-username=REGISTRY-USERNAME --docker-password=REGISTRY-PASS
WORD -n YOUR-NAMESPACE
```

Note

This step is not required if you install Tanzu Application Platform on AWS with EKS and use IAM Roles for Kubernetes Service Accounts instead of secrets. You can specify the Role Amazon Resource Name (ARN) in the next step.

2. Run the following to add secrets, a service account to execute the supply chain, and RBAC rules to authorize the service account to the developer namespace:

```
cat <<EOF | kubectl -n YOUR-NAMESPACE apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: tap-registry
  annotations:
    secretgen.carvel.dev/image-pull-secret: ""
type: kubernetes.io/dockerconfigjson
data:
  .dockerconfigjson: e30K
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: default
secrets:
  - name: registry-credentials
imagePullSecrets:
  - name: registry-credentials
  - name: tap-registry
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: default-permit-deliverable
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: deliverable
subjects:
  - kind: ServiceAccount
    name: default
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: default-permit-workload
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: workload
subjects:
  - kind: ServiceAccount
    name: default
EOF
```

Note

If you install Tanzu Application Platform on AWS with EKS and use IAM Roles for Kubernetes Service Accounts, you must annotate the ARN of the IAM Role and remove the `registry-credentials` secret. Your service account entry then looks like the following:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: default
  annotations:
    eks.amazonaws.com/role-arn: <Role ARN>
imagePullSecrets:
  - name: tap-registry
```

Enable additional users with Kubernetes RBAC

Follow these steps to enable additional users in your namespace by using Kubernetes RBAC:

1. (Optional) Before you begin, ensure that you have enabled single user access. If you've set up your developer namespace using Namespace Provisioner, you can skip this step.
2. Choose either of the following options to give developers namespace-level access and view access to the appropriate cluster-level resources:
    - **Option 1:** Use the Tanzu Application Platform RBAC CLI plug-in (beta).

      To use the `tanzu rbac` plug-in to grant `app-viewer` and `app-editor` roles to an identity provider group, run:

      ```
      tanzu rbac binding add -g GROUP-FOR-APP-VIEWER -n YOUR-NAMESPACE -r app-v
      iewer
      tanzu rbac binding add -g GROUP-FOR-APP-EDITOR -n YOUR-NAMESPACE -r app-e
      ditor
      ```

      Where:

        - `YOUR-NAMESPACE` is the name you give to the developer namespace.
        - `GROUP-FOR-APP-VIEWER` is the user group from the upstream identity provider that requires access to `app-viewer` resources on the current namespace and cluster.
        - `GROUP-FOR-APP-EDITOR` is the user group from the upstream identity provider that requires access to `app-editor` resources on the current namespace and cluster.

      For more information about `tanzu rbac`, see Bind a user or group to a default role

      VMware recommends creating a user group in your identity provider's grouping system for each developer namespace and then adding the users accordingly.

      Depending on your identity provider, you might need to take further action to federate user groups appropriately with your cluster. For an example of how to set up Azure Active Directory (Azure AD) with your cluster, see Integrate Azure Active Directory.

    - **Option 2:** Use the native Kubernetes YAML.

      Run the following to apply the RBAC policy:

```
cat <<EOF | kubectl -n YOUR-NAMESPACE apply -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: dev-permit-app-viewer
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: app-viewer
subjects:
  - kind: Group
    name: GROUP-FOR-APP-VIEWER
    apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: YOUR-NAMESPACE-permit-app-viewer
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: app-viewer-cluster-access
subjects:
  - kind: Group
    name: GROUP-FOR-APP-VIEWER
    apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: dev-permit-app-editor
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: app-editor
subjects:
  - kind: Group
    name: GROUP-FOR-APP-EDITOR
    apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: YOUR-NAMESPACE-permit-app-editor
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: app-editor-cluster-access
subjects:
  - kind: Group
    name: GROUP-FOR-APP-EDITOR
    apiGroup: rbac.authorization.k8s.io
EOF
```

Where:

- `YOUR-NAMESPACE` is the name you give to the developer namespace.
- `GROUP-FOR-APP-VIEWER` is the user group from the upstream identity provider that requires access to `app-viewer` resources on the current namespace and cluster.
- `GROUP-FOR-APP-EDITOR` is the user group from the upstream identity provider that requires access to `app-editor` resources on the current namespace and cluster.

VMware recommends creating a user group in your identity provider's grouping system for each developer namespace and then adding the users accordingly.

Depending on your identity provider, you might need to take further action to federate user groups appropriately with your cluster.

Rather than granting roles directly to individuals, VMware recommends using your identity provider's user groups system to grant access to a group of developers.

For an example of how to set up Azure Active Directory (AD) with your cluster, see Integrate Azure Active Directory.

3. (Optional) Log in as a non-admin user, such as a developer, to see the effects of RBAC after the role bindings are applied.

Additional configuration for testing and scanning

If you plan to install Out of the Box Supply Chains with Testing and Scanning, see Developer Namespace.

Install Tanzu Developer Tools for your VS Code

This topic tells you how to install VMware Tanzu Developer Tools for Visual Studio Code (VS Code).

Prerequisites

Before installing the extension, you must have:

- VS Code
- kubectl
- Tilt v0.30.12 or later
- Tanzu CLI and plug-ins
- A cluster with the Tanzu Application Platform Full profile or Iterate profile

If you are an app developer, someone else in your organization might have already set up the Tanzu Application Platform environment.

Docker Desktop and local Kubernetes are not prerequisites for using Tanzu Developer Tools for VS Code.

Install

To install VMware Tanzu Developer Tools for VS Code:

1. Open Visual Studio Code.
2. Open the command palette.
3. In the search box enter `Extension`.
4. Click **Extensions: Install Extensions**.
5. The **Extensions** view opens on the left side of your screen. In the search box enter `Tanzu`.
6. Click **Tanzu Developer Tools** and then click **Install**.

Configure

To configure VMware Tanzu Developer Tools for VS Code:

1. Ensure that you are targeting the correct cluster. For more information, see the Kubernetes documentation.
2. Go to **Code** > **Preferences** > **Settings** > **Extensions** > **Tanzu Developer Tools** and set the following:
    - **Confirm Apply Config**: This controls whether the extension asks to confirm user input when applying a workload.
    - **Confirm Debug Port**: This controls whether the extension asks for the debug port when running `Tanzu: Start Java Debug`.
    - **Confirm Local Port**: This controls whether the extension asks for the local port when running `Tanzu: Start Java Debug`.
    - **Confirm Delete**: This controls whether the extension asks for confirmation when deleting a workload.
    - **Enable Live Hover**: This enables Live Hover. For more information, see Integrating Live Hover by using Spring Boot Tools. Restart VS Code for this change to take effect.
    - **Source Image**: The registry location for publishing local source code. For example, `registry.io/yourapp-source`. This must include both a registry and a project name. A source image registry location is optional when Local Source Proxy is configured.
    - **Local Path**: (Optional) The path on the local file system to a directory of source code to build. This is the current directory by default.
    - **Namespace**: (Optional) This is the namespace that workloads are deployed into. The namespace set in kubeconfig is the default.
    - **Tracked Namespaces**: (Optional) Comma-separated list of namespaces. Resources in these namespaces appear in the Tanzu Workloads panel and the Activity panel. If empty, the namespace in the current context is the default.
    - **Wait Timeout**: (Optional) This sets how long to wait for a workload to become ready.
    - **Workload Type**: (Optional) This distinguishes the workload type. Examples include web and server.

Uninstall

To uninstall VMware Tanzu Developer Tools for VS Code:

1. Go to **Code** > **Preferences** > **Settings** > **Extensions**.
2. Right-click the extension and then click **Uninstall**.

Next steps

Proceed to Getting started with Tanzu Developer Tools for Visual Studio Code.

Install Tanzu Application Platform (offline)

To install Tanzu Application Platform (commonly known as TAP) on your Kubernetes clusters in an air-gapped environment:

Step	Task	Link
1.	Review the prerequisites to ensure you have met all requirements before installing.	Prerequisites
2.	Accept Tanzu Application Platform EULAs and install the Tanzu CLI.	Accept Tanzu Application Platform EULAs and installing the Tanzu CLI
3.	Install Cluster Essentials for Tanzu*.	Deploy Cluster Essentials
4.	Add the Tanzu Application Platform package repository, prepare your Tanzu Application Platform profile, and install the profile to the cluster.	Install Tanzu Application Platform in your air-gapped environment
5.	Install Tanzu Build Service full dependencies.	Install the Tanzu Build Service dependencies
6.	Configure custom certificate authorities for Tanzu Developer Portal.	Configure custom certificate authorities for Tanzu Developer Portal
7.	Add the certificate for the private Git repository in the Accelerator system namespace.	Configure Application Accelerator
8.	(Optional) Set up offline vulnerability scanning	Use vulnerability scanning in offline and air-gapped environments
9.	Set up developer namespaces to use your installed packages.	Set up developer namespaces to use your installed packages
10.	Install IDE extensions from Marketplace	Install IDE Extensions in your air-gapped environment

* *When you use a VMware Tanzu Kubernetes Grid cluster, there is no need to install Cluster Essentials because the contents of Cluster Essentials are already installed on your cluster.*

After installing Tanzu Application Platform on to your air-gapped cluster, you can start creating workloads that run in your air-gapped containers.

For more information about the Namespace Provisioner mode, see Work with Git repositories in air-gapped environments with Namespace Provisioner.

For more information about the manual mode, see Deploy an air-gapped workload.

Install Tanzu Application Platform (offline)

To install Tanzu Application Platform (commonly known as TAP) on your Kubernetes clusters in an air-gapped environment:

Step	Task	Link
1.	Review the prerequisites to ensure you have met all requirements before installing.	Prerequisites
2.	Accept Tanzu Application Platform EULAs and install the Tanzu CLI.	Accept Tanzu Application Platform EULAs and installing the Tanzu CLI
3.	Install Cluster Essentials for Tanzu*.	Deploy Cluster Essentials
4.	Add the Tanzu Application Platform package repository, prepare your Tanzu Application Platform profile, and install the profile to the cluster.	Install Tanzu Application Platform in your air-gapped environment
5.	Install Tanzu Build Service full dependencies.	Install the Tanzu Build Service dependencies
6.	Configure custom certificate authorities for Tanzu Developer Portal.	Configure custom certificate authorities for Tanzu Developer Portal

Step	Task	Link
7.	Add the certificate for the private Git repository in the Accelerator system namespace.	Configure Application Accelerator
8.	(Optional) Set up offline vulnerability scanning	Use vulnerability scanning in offline and air-gapped environments
9.	Set up developer namespaces to use your installed packages.	Set up developer namespaces to use your installed packages
10.	Install IDE extensions from Marketplace	Install IDE Extensions in your air-gapped environment

* *When you use a VMware Tanzu Kubernetes Grid cluster, there is no need to install Cluster Essentials because the contents of Cluster Essentials are already installed on your cluster.*

After installing Tanzu Application Platform on to your air-gapped cluster, you can start creating workloads that run in your air-gapped containers.

For more information about the Namespace Provisioner mode, see Work with Git repositories in air-gapped environments with Namespace Provisioner.

For more information about the manual mode, see Deploy an air-gapped workload.

Install Tanzu Application Platform in your air-gapped environment

This topic tells you how to install Tanzu Application Platform (commonly known as TAP) on your Kubernetes cluster and registry that are air-gapped from external traffic.

Before installing the packages, ensure that you have completed the following tasks:

- Review the Prerequisites to ensure that you have set up everything required before beginning the installation.
- Accept Tanzu Application Platform EULA and install Tanzu CLI.
- Deploy Cluster Essentials. This step is optional if you are using VMware Tanzu Kubernetes Grid cluster.

Relocate images to a registry

To relocate images from the VMware Tanzu Network registry to your air-gapped registry:

1. Set up environment variables for installation use by running:

   ```
   # Set tanzunet as the source registry to copy the Tanzu Application Platform pa
   ckages from.
   export IMGPKG_REGISTRY_HOSTNAME_0=registry.tanzu.vmware.com
   export IMGPKG_REGISTRY_USERNAME_0=MY-TANZUNET-USERNAME
   export IMGPKG_REGISTRY_PASSWORD_0=MY-TANZUNET-PASSWORD

   # The user's registry for copying the Tanzu Application Platform package to.
   export IMGPKG_REGISTRY_HOSTNAME_1=MY-REGISTRY
   export IMGPKG_REGISTRY_USERNAME_1=MY-REGISTRY-USER
   export IMGPKG_REGISTRY_PASSWORD_1=MY-REGISTRY-PASSWORD
   export TAP_VERSION=VERSION-NUMBER
   export REGISTRY_CA_PATH=PATH-TO-CA
   export TO_REPO=MY-REPO
   ```

   Where:

   - `MY-REGISTRY` is your air-gapped container registry.

- `MY-REGISTRY-USER` is the user with write access to `MY-REGISTRY`.
- `MY-REGISTRY-PASSWORD` is the password for `MY-REGISTRY-USER`.
- `MY-TANZUNET-USERNAME` is the user with access to the images in the VMware Tanzu Network registry `registry.tanzu.vmware.com`
- `MY-TANZUNET-PASSWORD` is the password for `MY-TANZUNET-USERNAME`.
- `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.9.1`
- `MY-REPO` is your repository in the air-gapped container image registry. Examples:
  - Harbor has the form `MY-REGISTRY/REPO-NAME/tap-packages`.
  - Docker Hub has the form `MY-REGISTRY/tap-packages`.
  - Google Cloud Registry has the form `MY-REGISTRY/MY-PROJECT/REPO-NAME/tap-packages`.

2. Copy the images into a `.tar` file from the VMware Tanzu Network onto an external storage device with the Carvel tool imgpkg by running:

```
imgpkg copy \
  -b registry.tanzu.vmware.com/tanzu-application-platform/tap-packages:$TAP_VER
SION \
  --to-tar tap-packages-$TAP_VERSION.tar \
  --include-non-distributable-layers
```

3. Relocate the images with the Carvel tool imgpkg by running:

```
imgpkg copy \
  --tar tap-packages-$TAP_VERSION.tar \
  --to-repo $TO_REPO \
  --include-non-distributable-layers \
  --registry-ca-cert-path $REGISTRY_CA_PATH
```

4. Create a namespace called `tap-install` for deploying any component packages by running:

```
kubectl create ns tap-install
```

   This namespace keeps the objects grouped together logically.

5. Create a registry secret by running:

```
tanzu secret registry add tap-registry \
    --server   $IMGPKG_REGISTRY_HOSTNAME_1 \
    --username $IMGPKG_REGISTRY_USERNAME_1 \
    --password $IMGPKG_REGISTRY_PASSWORD_1 \
    --namespace tap-install \
    --export-to-all-namespaces \
    --yes
```

6. Create a secret for accessing the user's registry by running:

```
tanzu secret registry add registry-credentials \
    --server   $IMGPKG_REGISTRY_HOSTNAME_1 \
    --username $IMGPKG_REGISTRY_USERNAME_1 \
    --password $IMGPKG_REGISTRY_PASSWORD_1 \
    --namespace tap-install \
    --export-to-all-namespaces \
    --yes
```

7. Add the Tanzu Application Platform package repository to the cluster by running:

```
tanzu package repository add tanzu-tap-repository \
  --url $IMGPKG_REGISTRY_HOSTNAME_1/tap-packages:$TAP_VERSION \
  --namespace tap-install
```

Where `$TAP_VERSION` is the Tanzu Application Platform version environment variable you defined earlier.

8. Get the status of the Tanzu Application Platform package repository, and ensure the status updates to `Reconcile succeeded` by running:

```
tanzu package repository get tanzu-tap-repository --namespace tap-install
```

Note

The `VERSION` and `TAG` numbers differ from the earlier example if you are on Tanzu Application Platform v1.0.2 or earlier.

9. List the available packages by running:

```
tanzu package available list --namespace tap-install
```

For example:

```
$ tanzu package available list --namespace tap-install
/ Retrieving available packages...
  NAME                                                DISPLAY-NAME
SHORT-DESCRIPTION
  accelerator.apps.tanzu.vmware.com                   Application Accelerator
for VMware Tanzu                                    Used to create new projects a
nd configurations.
  api-portal.tanzu.vmware.com                         API portal
A unified user interface for API discovery and exploration at scale.
  apis.apps.tanzu.vmware.com                          API Auto Registration fo
r VMware Tanzu                                      A TAP component to automatica
lly register API exposing workloads as API entities

in TAP GUI.
  backend.appliveview.tanzu.vmware.com                Application Live View fo
r VMware Tanzu                                      App for monitoring and troubl
eshooting running apps
  buildservice.tanzu.vmware.com                       Tanzu Build Service
Tanzu Build Service enables the building and automation of containerized

software workflows securely and at scale.
  carbonblack.scanning.apps.tanzu.vmware.com          VMware Carbon Black for
Supply Chain Security Tools - Scan                  Default scan templates using
VMware Carbon Black
  cartographer.conventions.apps.tanzu.vmware.com      Convention Service for V
Mware Tanzu                                         Convention Service enables ap
p operators to consistently apply desired runtime

configurations to fleets of workloads.
  cartographer.tanzu.vmware.com                       Cartographer
Kubernetes native Supply Chain Choreographer.
  cnrs.tanzu.vmware.com                               Cloud Native Runtimes
Cloud Native Runtimes is a serverless runtime based on Knative
  connector.appliveview.tanzu.vmware.com              Application Live View Co
nnector for VMware Tanzu                            App for discovering and regis
tering running apps
  controller.source.apps.tanzu.vmware.com             Tanzu Source Controller
Tanzu Source Controller enables workload create/update from source code.
```

```
  conventions.appliveview.tanzu.vmware.com             Application Live View Co
nventions for VMware Tanzu                           Application Live View convent
ion server
  developer-conventions.tanzu.vmware.com               Tanzu App Platform Devel
oper Conventions                                     Developer Conventions
  external-secrets.apps.tanzu.vmware.com               External Secrets Operato
r                                                    External Secrets Operator is
a Kubernetes operator that integrates external

secret management systems.
  fluxcd.source.controller.tanzu.vmware.com            Flux Source Controller
The source-controller is a Kubernetes operator, specialised in artifacts

acquisition from external sources such as Git, Helm repositories and S3 bucket
s.
  grype.scanning.apps.tanzu.vmware.com                 Grype for Supply Chain S
ecurity Tools - Scan                                 Default scan templates using
Anchore Grype
  metadata-store.apps.tanzu.vmware.com                 Supply Chain Security To
ols - Store                                          Post SBoMs and query for imag
e, package, and vulnerability metadata.
  namespace-provisioner.apps.tanzu.vmware.com          Namespace Provisioner
Automatic Provisioning of Developer Namespaces.
  ootb-delivery-basic.tanzu.vmware.com                 Tanzu App Platform Out o
f The Box Delivery Basic                             Out of The Box Delivery Basi
c.
  ootb-supply-chain-basic.tanzu.vmware.com             Tanzu App Platform Out o
f The Box Supply Chain Basic                         Out of The Box Supply Chain B
asic.
  ootb-supply-chain-testing-scanning.tanzu.vmware.com  Tanzu App Platform Out o
f The Box Supply Chain with Testing and Scanning     Out of The Box Supply Chain w
ith Testing and Scanning.
  ootb-supply-chain-testing.tanzu.vmware.com           Tanzu App Platform Out o
f The Box Supply Chain with Testing                  Out of The Box Supply Chain w
ith Testing.
  ootb-templates.tanzu.vmware.com                      Tanzu App Platform Out o
f The Box Templates                                  Out of The Box Templates.
  policy.apps.tanzu.vmware.com                         Supply Chain Security To
ols - Policy Controller                              Policy Controller enables def
ining of a policy to restrict unsigned container

images.
  scanning.apps.tanzu.vmware.com                       Supply Chain Security To
ols - Scan                                           Scan for vulnerabilities and
enforce policies directly within Kubernetes native

Supply Chains.
  service-bindings.labs.vmware.com                     Service Bindings for Kub
ernetes                                              Service Bindings for Kubernet
es implements the Service Binding Specification.
  services-toolkit.tanzu.vmware.com                    Services Toolkit
The Services Toolkit enables the management, lifecycle, discoverability and

connectivity of Service Resources (databases, message queues, DNS records,

etc.).
  snyk.scanning.apps.tanzu.vmware.com                  Snyk for Supply Chain Se
curity Tools - Scan                                  Default scan templates using
Snyk
  spring-boot-conventions.tanzu.vmware.com             Tanzu Spring Boot Conven
tions Server                                         Default Spring Boot conventio
n server.
  sso.apps.tanzu.vmware.com                            AppSSO
Application Single Sign-On for Tanzu
  tap-auth.tanzu.vmware.com                            Default roles for Tanzu
Application Platform                                 Default roles for Tanzu Appli
```

```
cation Platform
  tap-gui.tanzu.vmware.com                              Tanzu Developer Portal
web app graphical user interface for Tanzu Application Platform
  tap-telemetry.tanzu.vmware.com                        Telemetry Collector for
Tanzu Application Platform                          Tanzu Application Platform Te
lemetry
  tap.tanzu.vmware.com                                  Tanzu Application Platfo
rm                                                  Package to install a set of T
AP components to get you started based on your use

case.
  tekton.tanzu.vmware.com                               Tekton Pipelines
Tekton Pipelines is a framework for creating CI/CD systems.
```

Prepare Sigstore Stack for air-gapped policy controller

Important

This section only applies if the target environment requires support for keyless authorities in `ClusterImagePolicy`. You must set the `policy.tuf_enabled` field to `true` when installing Tanzu Application Platform. By default, keyless authorities support is deactivated.

By default, the public official Sigstore “The Update Framework (TUF) server” is used. You can use an alternative Sigstore Stack by setting `policy.tuf_mirror` and `policy.tuf_root`.

The Sigstore Stack consists of:

- Trillian
- Rekor
- Fulcio
- Certificate Transparency Log (CTLog)
- The Update Framework (TUF)

For an air-gapped environment, an internally accessible Sigstore Stack is required for keyless authorities.

Install your Tanzu Application Platform profile

The `tap.tanzu.vmware.com` package installs predefined sets of packages based on your profile settings. This is done by using the package manager installed by Tanzu Cluster Essentials.

For more information about profiles, see Components and installation profiles.

To prepare to install a profile:

1. List version information for the package by running:

   ```
   tanzu package available list tap.tanzu.vmware.com --namespace tap-install
   ```

2. Create a `tap-values.yaml` file by using the Full Profile sample as a guide. These samples have the minimum configuration required to deploy Tanzu Application Platform. The sample values file contains the necessary defaults for:
   - The meta-package, or parent Tanzu Application Platform package
   - Subordinate packages, or individual child packages

Keep the values file for future configuration use.

Full Profile

To install Tanzu Application Platform with Supply Chain Basic, you must retrieve your cluster's base64 encoded ca certificate from `$HOME/.kube/config`. Retrieve the `certificate-authority-data` from the respective cluster section and input it as `B64_ENCODED_CA` in the `tap-values.yaml`.

The following is the YAML file sample for the full-profile:

```
shared:
  ingress_domain: "INGRESS-DOMAIN"
  image_registry:
    project_path: "SERVER-NAME/REPO-NAME"
    secret:
      name: "KP-DEFAULT-REPO-SECRET"
      namespace: "KP-DEFAULT-REPO-SECRET-NAMESPACE"
  ca_cert_data: |
    -----BEGIN CERTIFICATE-----
    MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...
    -----END CERTIFICATE-----
profile: full
ceip_policy_disclosed: true
buildservice:
  kp_default_repository: "KP-DEFAULT-REPO"
  kp_default_repository_secret: # Takes the value from the shared section by default,
but can be overridden by setting a different value.
    name: "KP-DEFAULT-REPO-SECRET"
    namespace: "KP-DEFAULT-REPO-SECRET-NAMESPACE"
  exclude_dependencies: true
supply_chain: basic
contour:
  infrastructure_provider: aws
  envoy:
    service:
      type: LoadBalancer
      annotations:
      # This annotation is for air-gapped AWS only.
          service.kubernetes.io/aws-load-balancer-internal: "true"

ootb_supply_chain_basic:
  source:
      credentials_secret: "GIT-SOURCE-CREDENTIAL-SECRET-NAME" # (Optional) Defaults to
"".
  registry:
      server: "SERVER-NAME" # Takes the value from the shared section by default, but
can be overridden by setting a different value.
      repository: "REPO-NAME" # Takes the value from the shared section by default, bu
t can be overridden by setting a different value.
  gitops:
      credentials_secret: "GITOPS-CREDENTIAL-SECRET-NAME" # (Optional) Defaults to "".
  maven:
      repository:
         url: https://MAVEN-URL
         secret_name: "MAVEN-CREDENTIALS"

accelerator:
  ingress:
    include: true
    enable_tls: false
  git_credentials:
    secret_name: git-credentials
    username: GITLAB-USER
    password: GITLAB-PASSWORD
```

```
appliveview:
  ingressEnabled: true

appliveview_connector:
  backend:
    ingressEnabled: true
    sslDeactivated: false
    host: appliveview.INGRESS-DOMAIN
    caCertData: |-
      -----BEGIN CERTIFICATE-----
      MIIGMzCCBBugAwIBAgIJALHHzQjxM6wMMA0GCSqGSIb3DQEBDQUAMGcxCzAJBgNV
      BAgMAklOMRQwEgYDVQQHDAtNaW5uZWFwb2xpczEPMA0GA1UECgwGVk13YXJlMRMw
      -----END CERTIFICATE-----

local_source_proxy:
  # Takes the value from the project_path under the image_registry section of shared b
y default, but can be overridden by setting a different value.
  repository: "EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE"
  push_secret:
    # When set to true, the secret mentioned in this section is automatically exported
to Local Source Proxy's namespace.
    name: "EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE-SECRET"
    namespace: "EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE-SECRET-NAMESPACE"
    # When set to true, the secret mentioned in this section is automatically exported
to Local Source Proxy's namespace.
    create_export: true

tap_gui:
  app_config:
    auth:
      allowGuestAccess: true  # This allows unauthenticated users to log in to your po
rtal. If you want to deactivate it, make sure you configure an alternative auth provid
er.
    kubernetes:
      serviceLocatorMethod:
        type: multiTenant
      clusterLocatorMethods:
        - type: config
          clusters:
            - url: https://${KUBERNETES_SERVICE_HOST}:${KUBERNETES_SERVICE_PORT}
              name: host
              authProvider: serviceAccount
              serviceAccountToken: ${KUBERNETES_SERVICE_ACCOUNT_TOKEN}
              skipTLSVerify: false
              caData: B64_ENCODED_CA
    catalog:
      locations:
        - type: url
          target: https://GIT-CATALOG-URL/catalog-info.yaml
    #Example Integration for custom GitLab:
    integrations:
      gitlab:
        - host: GITLAB-URL
          token: GITLAB-TOKEN
          apiBaseUrl: https://GITLABURL/api/v4/
    backend:
      reading:
        allow:
          - host: GITLAB-URL # Example URL: gitlab.example.com

metadata_store:
  ns_for_export_app_cert: "MY-DEV-NAMESPACE"
  app_service_type: ClusterIP # Defaults to LoadBalancer. If shared.ingress_domain is
set earlier, this must be set to ClusterIP.
```

Important

- Tanzu Build Service is installed by default with `lite` depndencies. When installing Tanzu Build Service in an air-gapped environment, the lite dependencies are not available because they require Internet access. You must install the `full` dependencies by setting `exclude_dependencies` to `true`. The existing ClusterStore instances will not be updated if you switch from `lite` dependencies to `full` dependencies after the initial installation completes.
- Installing Grype by using `tap-values.yaml` as follows is deprecated in v1.6 and will be removed in v1.8:

  ```
  grype:
   targetImagePullSecret: "TARGET-REGISTRY-CREDENTIALS-SECRET"
  ```

  You can install Grype by using Namespace Provisioner instead.

Where:

- `INGRESS-DOMAIN` is the subdomain for the host name that you point at the `tanzu-shared-ingress` service's External IP address.
- `KP-DEFAULT-REPO` is a writable repository in your registry. Tanzu Build Service dependencies are written to this location. Examples:
  - Harbor has the form `kp_default_repository: "my-harbor.io/my-project/build-service"`.
  - Docker Hub has the form `kp_default_repository: "my-dockerhub-user/build-service"` or `kp_default_repository: "index.docker.io/my-user/build-service"`.
  - Google Cloud Registry has the form `kp_default_repository: "gcr.io/my-project/build-service"`.
- `KP-DEFAULT-REPO-SECRET` is the secret with user credentials that can write to `KP-DEFAULT-REPO`. You can `docker push` to this location with this credential.
  - For Google Cloud Registry, use `kp_default_repository_username: _json_key`.
  - You must create the secret before the installation. For example, you can use the `registry-credentials` secret created earlier.
- `KP-DEFAULT-REPO-SECRET-NAMESPACE` is the namespace where `KP-DEFAULT-REPO-SECRET` is created.
- `SERVER-NAME` is the host name of the registry server. Examples:
  - Harbor has the form `server: "my-harbor.io"`.
  - Docker Hub has the form `server: "index.docker.io"`.
  - Google Cloud Registry has the form `server: "gcr.io"`.
- `REPO-NAME` is where workload images are stored in the registry. If this key is passed through the shared section earlier and AWS ECR registry is used, you must ensure that the `SERVER-NAME/REPO-NAME/buildservice` and `SERVER-NAME/REPO-NAME/workloads` exist. AWS ECR expects the paths to be pre-created.
- Images are written to `SERVER-NAME/REPO-NAME/workload-name`. Examples:
  - Harbor has the form `repository: "my-project/supply-chain"`.
  - Docker Hub has the form `repository: "my-dockerhub-user"`.

    - Google Cloud Registry has the form `repository: "my-project/supply-chain"`.

- `EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE` is where the developer's local source is uploaded when using Tanzu CLI to use Local Source Proxy for workload creation.

  If an AWS ECR registry is being used, ensure that the repository already exists. AWS ECR expects the repository path to already exist. This destination is represented as `REGISTRY-SERVER/REPOSITORY-PATH`. For more information, see Install Local Source Proxy.

- `EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE-SECRET` is the name of the secret with credentials that allow pushing to the `EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE` repository.
- `EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE-SECRET-NAMESPACE` is the namespace in which `EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE-SECRET` is available.
- `GIT-SOURCE-CREDENTIAL-SECRET-NAME` is the name of the Kubernetes secret in the developer namespace that supplies the Git credentials for the supply chain to fetch source code from. See Git authentication for more information.
- `GITOPS-CREDENTIAL-SECRET-NAME` is the name of the Kubernetes secret in the developer namespace that supplies the Git credentials for the supply chain to push configuration to. See Git authentication for more information.
- `MAVEN-CREDENTIALS` is the name of the secret with maven creds. This secret must be in the developer namespace. You can create it after the fact.
- `GIT-CATALOG-URL` is the path to the `catalog-info.yaml` catalog definition file. You can download either a blank or populated catalog file from the Tanzu Application Platform product page. Otherwise, you can use a Backstage-compliant catalog you've already built and posted on the Git infrastructure.
- `GITLABURL` is the host name of your GitLab instance.
- `GITLAB-USER` is the user name of your GitLab instance.
- `GITLAB-PASSWORD` is the password for the `GITLAB-USER` of your GitLab instance. This can also be the `GITLAB-TOKEN`.
- `GITLAB-TOKEN` is the API token for your GitLab instance.
- `MY-DEV-NAMESPACE` is the name of the developer namespace. SCST - Store exports secrets to the namespace, and SCST - Scan deploys the `ScanTemplates` there. This allows the scanning feature to run in this namespace. If there are multiple developer namespaces, use `ns_for_export_app_cert: "*"` to export the SCST - Store CA certificate to all namespaces. To install Grype in multiple namespaces, use a namespace provisioner. See Namespace Provisioner.
- `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the secret that contains the credentials to pull an image from the registry for scanning.

Note

The `appliveview_connector.backend.sslDisabled` key is deprecated and renamed to `appliveview_connector.backend.sslDeactivated`.

If you use custom CA certificates, you must provide one or more PEM-encoded CA certificates under the `ca_cert_data` key. If you configured `shared.ca_cert_data`, Tanzu Application Platform component packages inherit that value by default.

TLS is enabled by default on Application Live View back end using ClusterIssuer. Set the `ingressEnabled` key to `true` for TLS to be enabled on Application Live View back end using

ClusterIssuer. This key is set to `false` by default.

The `appliveview-cert` certificate is generated by default and its issuerRef points to the `.ingress_issuer` value. The `ingress_issuer` key consumes the value `shared.ingress_issuer` from `tap-values.yaml` by default when you don't specify the `ingress_issuer` in `tap-values.yaml`.

When `ingressEnabled` is `true`, an HTTPProxy object is created in the cluster and `appliveview-cert` certificate is generated by default in the `app_live_view` namespace. The secretName `appliveview-cert` stores this certificate.

To verify the HTTPProxy object with the secret, run:

```
kubectl get httpproxy -A
```

Expected output:

```
NAMESPACE            NAME
FQDN                                                          TLS SECRET
STATUS   STATUS DESCRIPTION
app-live-view        appliveview
appliveview.192.168.42.55.nip.io                              appliveview-cert   va
lid    Valid HTTPProxy
```

The `appliveview_connector.backend.host` key is the back end host in the view cluster. The `appliveview_connector.backend.caCertData` key is the certificate retrieved from the HTTPProxy secret exposed by Application Live View back end in the view cluster. To retrieve this certificate, run the following command in the view cluster:

```
kubectl get secret appliveview-cert -n app-live-view -o yaml |  yq '.data."ca.crt"' |
base64 -d
```

Install your Tanzu Application Platform package

Follow these steps to install the Tanzu Application Platform package:

1. Install the package by running:

   ```
   tanzu package install tap -p tap.tanzu.vmware.com -v $TAP_VERSION --values-file
   tap-values.yaml -n tap-install
   ```

   Where `$TAP_VERSION` is the Tanzu Application Platform version environment variable you defined earlier.

2. Verify the package install by running:

   ```
   tanzu package installed get tap -n tap-install
   ```

   This may take 5-10 minutes because it installs several packages on your cluster.

3. Verify that all the necessary packages in the profile are installed by running:

   ```
   tanzu package installed list -A
   ```

Next steps

- Install the Tanzu Build Service dependencies

Install the Tanzu Build Service dependencies

This topic tells you how to install the Tanzu Build Service (TBS) full dependencies on Tanzu Application Platform (commonly known as TAP).

Install full dependencies

Important

By default, Tanzu Build Service is installed with `lite` dependencies.

When installing Tanzu Build Service in an air-gapped environment, the `lite` dependencies are not available because they require Internet access. You must install the `full` dependencies.

To install `full` dependencies:

1. Get the latest version of the Tanzu Application Platform package by running:

   ```
   tanzu package available list tap.tanzu.vmware.com --namespace tap-install
   ```

2. If you have not done so already, you must exclude the default dependencies by adding the key-value pair `exclude_dependencies: true` to your `tap-values.yaml` file under the `buildservice` section. For example:

   ```
   buildservice:
     exclude_dependencies: true
   ```

3. If you have not updated your Tanzu Application Platform package installation after adding the key-value pair `exclude_dependencies: true` to your values file, perform the update by running:

   ```
   tanzu package installed update tap --namespace tap-install --values-file VALUES
   -FILE
   ```

   Where `VALUES-FILE` is the path to the `tap-values.yaml` file you edited earlier.

4. Relocate the Tanzu Build Service `full` dependencies package repository by doing one of the following:

   - Relocate the images directly for online installation:

     ```
     imgpkg copy \
       -b registry.tanzu.vmware.com/tanzu-application-platform/full-deps-packa
     ge-repo:VERSION \
       --to-repo ${INSTALL_REGISTRY_HOSTNAME}/full-deps-package-repo
     ```

     Where `VERSION` is the version of the Tanzu Application Platform package you retrieved earlier.

   - Relocate the images to an external storage device and then to the registry in the air-gapped environment:

     ```
     imgpkg copy \
       -b registry.tanzu.vmware.com/tanzu-application-platform/full-deps-packa
     ge-repo:VERSION \
       --to-tar=full-deps-package-repo.tar

     # move full-deps-package-repo.tar to environment with registry access
     imgpkg copy \
       --tar full-deps-package-repo.tar \
     ```

```
  --to-repo=INSTALL-REGISTRY-HOSTNAME/TARGET-REPOSITORY/full-deps-package
-repo
```

Where:

- `VERSION` is the version of the Tanzu Application Platform package you retrieved earlier.
- `INSTALL-REGISTRY-HOSTNAME` is your container registry.
- `TARGET-REPOSITORY` is your target repository.

5. Add the Tanzu Build Service `full` dependencies package repository by running:

```
tanzu package repository add full-deps-package-repo \
  --url INSTALL-REGISTRY-HOSTNAME/TARGET-REPOSITORY/full-deps-package-repo:VERS
ION \
  --namespace tap-install
```

Where:

- `INSTALL-REGISTRY-HOSTNAME` is your container registry.
- `TARGET-REPOSITORY` is your target repository.
- `VERSION` is the version of the Tanzu Application Platform package you retrieved earlier.

6. Create a new `tbs-full-deps-values.yaml` and copy the `kp_default_repository` key-value pair from your `tap-values.yaml` or `tbs-values.yaml`:

```
---
 kp_default_repository: "REPO-NAME"
 kp_default_repository_secret:
   name: kp-default-repository-creds
   namespace: tap-install
```

Where `REPO-NAME` is copied from the `buildservice.kp_default_repository` field in your `tap-values.yaml` or `tbs-values.yaml`.

1. (Optional) Install the UBI builder.

   The UBI builder uses Red Hat Universal Base Image (UBI) v8 for both build and run images. This builder only supports Java and Node.js. To install the UBI builder, add the key-value pair `enable_ubi_builder: true` to your `tbs-full-deps-values.yaml`.

   ```
   ---
   enable_ubi_builder: true
   ```

2. (Optional) Install the Static builder.

   The Static builder uses Ubuntu Jammy for both build images and a minimal static run image. This builder only supports Golang. To install the Static builder, add the key-value pair `enable_static_builder: true` to your `tbs-full-deps-values.yaml`.

   ```
   ---
   enable_static_builder: true
   ```

7. Install the `full` dependencies package by running:

```
tanzu package install full-deps \
  --package full-deps.buildservice.tanzu.vmware.com \
  --version "> 0.0.0" \
```

```
  --namespace tap-install \
  --values-file VALUES-FILE
```

Where `VALUES-FILE` is the path to the `tbs-full-deps-values.yaml` you created earlier.

(Optional) Update dependencies out of band of Tanzu Application Platform releases

Tanzu Build Service dependencies might be upgraded between Tanzu Application Platform releases, for example, if a CVE is discovered in the OS (stack update) or language (buildpack update).

Automatic dependency updates enable your cluster to consume the stack and buildpack updates immediately instead of waiting for the next Tanzu Application Platform patch release to pull in the updated dependencies.

- Updates are provided through a separate package repository with available version lines for all supported Tanzu Application Platform minor versions.
- Within a version line, only patch versions are incremented to avoid breaking changes.
- You can customize the packages that you want the automatic dependency updater to update through your `tap-values.yaml` file or your full dependencies values.

Prerequisites: These steps assume a registry secret already exists in the cluster for accessing `registry.tanzu.vmware.com` and your registry.

To enable automatic dependency updates:

1. Relocate the dependency updater package repository to the air-gapped container image registry:
    - If a machine with access to both the air-gapped registry and the internet is available, you can copy the images directly by running:

      ```
      imgpkg copy \
        -b registry.tanzu.vmware.com/build-service-dependency-updater/package-r
      epo:VERSION-CONSTRAINT \
        --to-repo INTERNAL-REPO
      ```

      Where:

        - `VERSION-CONSTRAINT` is the Tanzu Application Platform version in the form of `MAJOR.MINOR.x`. For example, `1.8.x`.
        - `INTERNAL-REPO` is your repository in the air-gapped container image registry. Examples:
            - Harbor has the form `MY-REGISTRY/REPO-NAME/tbs-dep-updater`.
            - Docker Hub has the form `MY-REGISTRY/tbs-dep-updater`.
            - Google Cloud Registry has the form `MY-REGISTRY/MY-PROJECT/REPO-NAME/tbs-dep-updater`.
    - If you can only transfer the data using a physical external storage device:
        1. Copy the images into a `.tar` file from VMware Tanzu Network by running:

           ```
           imgpkg copy \
             -b registry.tanzu.vmware.com/build-service-dependency-updater/pa
           ckage-repo:VERSION-CONSTRAINT \
             --to-tar dependency-updater-VERSION-CONSTRAINT.tar \
             --include-non-distributable-layers
           ```

Where `VERSION-CONSTRAINT` is the Tanzu Application Platform version in the form of `MAJOR.MINOR.x`. For example, `1.8.x`.

2. Import the `.tar` files into the air-gapped container image registry by running:

```
imgpkg copy \
  --tar dependency-updater-VERSION-CONSTRAINT.tar \
  --to-repo INTERNAL-REPO \
  --include-non-distributable-layers \
  --registry-ca-cert-path $REGISTRY_CA_PATH
```

Where:

- `VERSION-CONSTRAINT` is the Tanzu Application Platform version in the form of `MAJOR.MINOR.x`. For example, `1.8.x`.
- `INTERNAL-REPO` is your repository in the air-gapped container image registry. Examples:
  - Harbor has the form `MY-REGISTRY/REPO-NAME/tbs-dep-updater`.
  - Docker Hub has the form `MY-REGISTRY/tbs-dep-updater`.
  - Google Cloud Registry has the form `MY-REGISTRY/MY-PROJECT/REPO-NAME/tbs-dep-updater`.

2. Add the following to your `tap-values.yaml` file:

```
buildservice:
  dependency_updates:
    allow: true
    scope: SCOPE
    include_packages: [""]
    exclude_packages: [""]
```

Where:

- `SCOPE` is the list of dependencies you want updated. The options are:
  - `stacks-only` (default): Only stacks and builders are updated. This addresses CVEs in the base image or operating system.
  - `all`: Stacks, builders, and buildpacks are updated. This addresses CVEs in the base image or operating system and CVEs in the language toolchain such as compilers, interpreters, and standard libraries.
  - `custom`: This list is empty by default. Use the `include_packages` key to add packages to be updated.

Note

You must update the Tanzu Application Platform package install and the Full Dependencies package install after changing the `tap-values.yaml`.

3. Add the Tanzu Build Service Dependency Updates package repository by running:

```
kubectl apply -f - <<EOF
  apiVersion: packaging.carvel.dev/v1alpha1
  kind: PackageRepository
  metadata:
    name: tbs-dependencies-package-repository
```

```
    namespace: tap-install
  spec:
    fetch:
      imgpkgBundle:
        image: DEPENDENCY-UPDATER-PACKAGE-REPO
        tagSelection:
          semver:
            constraints: VERSION-CONSTRAINT
EOF
```

Where:

- `DEPENDENCY-UPDATER-PACKAGE-REPO` is the location of the package repository. This is `registry.tanzu.vmware.com/build-service-dependency-updater/package-repo` for online installs and the internal container image registry for air-gapped installs.
- `VERSION-CONSTRAINT` is the Tanzu Application Platform version in the form of `MAJOR.MINOR.x`. For example, `1.8.x`.

After completing this configuration, the repository you set with `DEPENDENCY-UPDATER-PACKAGE-REPO` will be polled for updates and any new releases will automatically be made available to the cluster.

Next steps

- Configure custom CAs for Tanzu Developer Portal

Configure custom certificate authorities for Tanzu Developer Portal

This topic tells you how to configure your Tanzu Developer Portal to trust unusual certificate authorities (CA) when making outbound connections.

Tanzu Developer Portal might require custom certificates when connecting to persistent databases or custom catalog locations that require SSL. You use overlays with PackageInstalls to make this possible. There are two ways to implement this workaround: you can add a custom CA or you can deactivate all SSL verification.

Add a custom CA

The overlay previously available in this section is no longer necessary. As of Tanzu Application Platform v1.3, the value `ca_cert_data` is supported at the top level of its values file. Any number of newline-delimited CA certificates in PEM format are accepted.

For example:

```
# tap-gui-values.yaml
ca_cert_data: |
  -----BEGIN CERTIFICATE-----
  cert data here
  -----END CERTIFICATE-----

  -----BEGIN CERTIFICATE-----
  other cert data here
  -----END CERTIFICATE-----
app_config:
  # ...
```

Tanzu Developer Portal also inherits `shared.ca_cert_data` from your `tap-values.yaml` file. `shared.ca_cert_data` is newline-concatenated with `ca_certs` given directly to Tanzu Developer Portal.

```
shared:
  ca_cert_data: |
    -----BEGIN CERTIFICATE-----
    cert data here
    -----END CERTIFICATE-----

tap_gui:
  ca_cert_data: |
    -----BEGIN CERTIFICATE-----
    other cert data here
    -----END CERTIFICATE-----
  app_config:
    # ...
```

To verify that Tanzu Developer Portal has processed the custom CA certificates, check that the `ca-certs-data` volume with mount path `/etc/custom-ca-certs-data` is mounted in the Tanzu Developer Portal server pod.

Deactivate all SSL verification

To deactivate SSL verification to allow for self-signed certificates, set the Tanzu Developer Portal pod's environment variable as `NODE_TLS_REJECT_UNAUTHORIZED=0`. When the value equals `0`, certificate validation is deactivated for TLS connections.

To do this, use the `package_overlays` key in the Tanzu Application Platform values file. For instructions, see Customize Package Installation.

The following YAML is an example `Secret` containing an overlay to deactivate TLS:

```
apiVersion: v1
kind: Secret
metadata:
  name: deactivate-tls-overlay
  namespace: tap-install
stringData:
  deactivate-tls-overlay.yml: |
    #@ load("@ytt:overlay", "overlay")
    #@overlay/match by=overlay.subset({"kind":"Deployment", "metadata": {"name": "se
rver", "namespace": "NAMESPACE"}}),expects="1+"
    ---
    spec:
      template:
        spec:
          containers:
            #@overlay/match by=overlay.all,expects="1+"
            #@overlay/match-child-defaults missing_ok=True
            - env:
              - name: NODE_TLS_REJECT_UNAUTHORIZED
                value: "0"
```

Where `NAMESPACE` is the namespace in which your Tanzu Developer Portal instance is deployed. For example, `tap-gui`.

Next steps

- Configure Application Accelerator

Configure Application Accelerator

This topic describes advanced configuration options available for Application Accelerator. This includes configuring Git-Ops style deployments of accelerators and configurations for use with non-

public repositories and in air-gapped environments.

Accelerators are created either using the Tanzu CLI or by applying a YAML manifest using kubectl. Another option is Using a Git-Ops style configuration for deploying a set of managed accelerators.

Application Accelerator pulls content from accelerator source repositories using either the “Flux SourceController” or the “Tanzu Application Platform Source Controller” components. If the repository used is accessible anonymously from a public server, you do not have to configure anything additional. Otherwise, provide authentication as explained in Using non-public repositories. There are also options for making these configurations easier explained in Configuring tap-values.yaml with Git credentials secret

Using a Git-Ops style configuration for deploying a set of managed accelerators

To enable a Git-Ops style of managing resources used for deploying accelerators, there is a new set of properties for the Application Accelerator configuration. The resources are managed using a Carvel kapp-controller App in the `accelerator-system` namespace that watches a Git repository containing the manifests for the accelerators. This means that you can make changes to the manifests, or to the accelerators they point to, and the changes are reconciled and reflected in the deployed resources.

You can specify the following accelerator configuration properties when installing the Application Accelerator. The same properties are provided in the `accelerator` section of the `tap-values.yaml` file:

```
accelerator:
  managed_resources:
    enable: true
    git:
      url: GIT-REPO-URL
      ref: origin/main
      sub_path: null
      secret_ref: git-credentials
```

Where:

- `GIT-REPO-URL` is the URL of a Git repository that contains manifest YAML files for the accelerators that you want to have managed. The URL must start with `https://` or `git@`. You can specify a `sub_path` if necessary and also a `secret_ref` if the repository requires authentication. If not needed, then leave these additional properties out.

  For more information, see Configure tap-values.yaml with Git credentials secret and Creating a manifest with multiple accelerators and fragments in this topic.

Functional and Organizational Considerations

Any accelerator manifest that is defined under the `GIT-REPO-URL` and optional `sub_path` is selected by the kapp-controller app. If there are multiple manifests at the defined `GIT-REPO-URL`, they are all watched for changes and displayed to the user as a merged catalog.

For example: if you have two manifests containing multiple accelerator or fragment definitions, `manifest-1.yaml`, and `manifest-2.yaml`, on the same path in the organizational considerations. The resulting catalog is (`manifest-1.yaml` + `manifest-2.yaml`).

Examples for creating accelerators

A minimal example for creating an accelerator

A minimal example might look like the following manifest:

`spring-cloud-serverless.yaml`

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: spring-cloud-serverless
spec:
  git:
    url: https://github.com/vmware-tanzu/application-accelerator-samples
    subPath: spring-cloud-serverless
    ref:
      branch: main
```

This example creates an accelerator named `spring-cloud-serverless`. The `displayName`, `description`, `iconUrl`, and `tags` text boxes are populated based on the content under the `accelerator` key in the `accelerator.yaml` file found in the `main` branch of the Git repository at Application Accelerator Samples under the sub-path `spring-cloud-serverless`. For example:

`accelerator.yaml`

```
accelerator:
  displayName: Spring Cloud Serverless
  description: A simple Spring Cloud Function serverless app
  iconUrl: https://raw.githubusercontent.com/vmware-tanzu/application-accelerator-samp
les/main/icons/icon-cloud.png
  tags:
  - java
  - spring
  - cloud
  - function
  - serverless
  - tanzu
...
```

To create this accelerator with kubectl, run:

```
kubectl apply --namespace --accelerator-system --filename spring-cloud-serverless.yaml
```

Or, you can use the Tanzu CLI and run:

```
tanzu accelerator create spring-cloud-serverless --git-repo https://github.com/vmware-
tanzu/application-accelerator-samples.git --git-branch main --git-sub-path spring-clou
d-serverless
```

An example for creating an accelerator with customized properties

You can specify the `displayName`, `description`, `iconUrl`, and `tags` text boxes and this overrides any values provided in the accelerator's Git repository. The following example explicitly sets those text boxes and the `ignore` text box:

`my-spring-cloud-serverless.yaml`

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: my-spring-cloud-serverless
spec:
  displayName: My Spring Cloud Serverless
  description: My own Spring Cloud Function serverless app
  iconUrl: https://raw.githubusercontent.com/vmware-tanzu/application-accelerator-samp
```

```
les/main/icons/icon-cloud.png
  tags:
    - spring
    - cloud
    - function
    - serverless
  git:
    ignore: ".git/, bin/"
    url: https://github.com/vmware-tanzu/application-accelerator-samples
    subPath: spring-cloud-serverless
    ref:
      branch: test
```

To create this accelerator with kubectl, run:

```
kubectl apply --namespace --accelerator-system --filename my-spring-cloud-serverless.y
aml
```

To use the Tanzu CLI, run:

```
tanzu accelerator create my-spring-cloud-serverless --git-repo https://github.com/vmwa
re-tanzu/application-accelerator-samples --git-branch main --git-sub-path spring-cloud
-serverless \
  --description "My own Spring Cloud Function serverless app" \
  --display-name "My Spring Cloud Serverless" \
  --icon-url https://raw.githubusercontent.com/vmware-tanzu/application-accelerator-sa
mples/main/icons/icon-cloud.png \
  --tags "spring,cloud,function,serverless"
```

Note

It is not possible to provide the `git.ignore` option with the Tanzu CLI.

Creating a manifest with multiple accelerators and fragments

You might have a manifest that contains multiple accelerators or fragments. For example:

`accelerator-collection.yaml`

```
---
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: spring-cloud-serverless
spec:
  git:
    url: https://github.com/vmware-tanzu/application-accelerator-samples
    subPath: spring-cloud-serverless
    ref:
      branch: main
---
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: tanzu-java-web-app
spec:
  git:
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
    subPath: tanzu-java-web-app
    ref:
      branch: main
```

For a larger example of this, see Sample Accelerators Main. Optionally, use this to create an initial catalog of accelerators and fragments during a fresh Application Accelerator install.

Configure `tap-values.yaml` with Git credentials secret

Note

For how to create a new OAuth Token for optional Git repository creation, see Create an Application Accelerator Git repository during project creation.

When deploying accelerators using Git repositories that requires authentication or are installed with custom CA certificates, you must provide some additional authentication values in a secret. The examples in the next section provide more details. This section describes how to configure a Git credentials secret that is used in later Git-based examples.

You can specify the following accelerator configuration properties when installing Application Accelerator. The same properties are provided in the `accelerator` section of the `tap-values.yaml` file:

```
accelerator:
  git_credentials:
    secret_name: git-credentials
    username: GIT-USER-NAME
    password: GIT-CREDENTIALS
    ca_file: CUSTOM-CA-CERT
```

Where:

- `GIT-USER-NAME` is the user name for authenticating with the Git repository.
- `GIT-CREDENTIALS` is the password or access token used for authenticating with the Git repository. VMware recommends using an access token for this.
- `CUSTOM-CA-CERT` is the certificate data needed when accessing the Git repository.

This is an example of this part of a `tap-values.yaml` configuration:

```
accelerator:
  git_credentials:
    secret_name: git-credentials
    username: testuser
    password: s3cret
    ca_file: |
      -----BEGIN CERTIFICATE-----
      .
      .
      .  < certificate data >
      .
      .
      -----END CERTIFICATE-----
```

You can specify the custom CA certificate data using the shared config value `shared.ca_cert_data` and it propagates to all components that can make use of it, including the App Accelerator configuration. The example earlier produces an output such as this using the shared value:

```
shared:
  ca_cert_data: |
    -----BEGIN CERTIFICATE-----
    .
    .
```

```
    .  < certificate data >
    .
    .
    -----END CERTIFICATE-----

accelerator:
  git_credentials:
    secret_name: git-credentials
    username: testuser
    password: s3cret
```

Using non-public repositories

For GitHub repositories that aren't accessible anonymously, you must provide credentials in a Secret.

- For HTTPS repositories the secret must contain user name and password fields. The password field can contain a personal access token instead of an actual password. For more information, see Fluxcd/source-controller basic access authentication.
- For HTTPS with self-signed certificates, you can add a `.data.caFile` value to the secret created for HTTPS authentication. For more information, see fluxcd/source-controller HTTPS Certificate Authority.
- For SSH repositories, the secret must contain identity, identity.pub, and known_hosts text boxes. For more information, see fluxcd/source-controller SSH authentication.
- For Image repositories that aren't publicly available, an image pull secret might be provided. For more information, see Kubernetes documentation on using imagePullSecrets.

Examples for a private Git repository

Example using http credentials

To create an accelerator using a private Git repository, first create a secret with the HTTP credentials.

Note

For better security, use an access token as the password.

```
kubectl create secret generic https-credentials \
    --namespace accelerator-system \
    --from-literal=username=<user> \
    --from-literal=password=<access-token>
```

Verify that your secret was created by running:

```
kubectl get secret --namespace accelerator-system https-credentials -o yaml
```

The output is similar to:

```
apiVersion: v1
kind: Secret
metadata:
  name: https-credentials
  namespace: accelerator-system
type: Opaque
data:
```

```
  username: <BASE64>
  password: <BASE64>
```

After you created and verified the secret, you can create the accelerator by using the `spec.git.secretRef.name` property:

`private-acc.yaml`

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: private-acc
spec:
  displayName: private
  description: Accelerator using a private repository
  git:
    url: REPOSITORY-URL
    ref:
      branch: main
    secretRef:
      name: https-credentials
```

For https credentials, the `REPOSITORY-URL` must use `https://` as the URL scheme.

If you are using the Tanzu CLI, add the `--secret-ref` flag to your `tanzu accelerator create` command and provide the name of the secret for that flag.

Example using http credentials with self-signed certificate

To create an accelerator using a private Git repository with a self-signed certificate, create a secret with the HTTP credentials and the certificate.

Note

For better security, use an access token as the password.

```
kubectl create secret generic https-ca-credentials \
    --namespace accelerator-system \
    --from-literal=username=<user> \
    --from-literal=password=<access-token> \
    --from-file=caFile=<path-to-CA-file>
```

Verify that your secret was created by running:

```
kubectl get secret --namespace accelerator-system https-ca-credentials -o yaml
```

The output is similar to:

```
apiVersion: v1
kind: Secret
metadata:
  name: https-ca-credentials
  namespace: accelerator-system
type: Opaque
data:
  username: <BASE64>
  password: <BASE64>
  caFile: <BASE64>
```

After you have the secret created, you can create the accelerator by using the `spec.git.secretRef.name` property:

`private-acc.yaml`

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: private-acc
spec:
  displayName: private
  description: Accelerator using a private repository
  git:
    url: REPOSITORY-URL
    ref:
      branch: main
    secretRef:
      name: https-ca-credentials
```

Important

For https credentials, the `REPOSITORY-URL` must use `https://` as the URL scheme.

If you are using the Tanzu CLI, add the `--secret-ref` flag to your `tanzu accelerator create` command and provide the name of the secret for that flag.

Example using SSH credentials

To create an accelerator using a private Git repository, create a secret with the SSH credentials such as this example:

```
ssh-keygen -q -N "" -f ./identity
ssh-keyscan github.com > ./known_hosts
kubectl create secret generic ssh-credentials \
    --namespace accelerator-system \
    --from-file=./identity \
    --from-file=./identity.pub \
    --from-file=./known_hosts
```

If you have a key file already created, skip the `ssh-keygen` and `ssh-keyscan` steps and replace the values for the `kubectl create secret` command. Such as:

- `--from-file=identity=<path to your identity file>`
- `--from-file=identity.pub=<path to your identity.pub file>`
- `--from-file=known_hosts=<path to your know_hosts file>`

Verify that your secret was created by running:

```
kubectl get secret --namespace accelerator-system ssh-credentials -o yaml
```

The output is similar to :

```
apiVersion: v1
kind: Secret
metadata:
  name: ssh-credentials
  namespace: accelerator-system
type: Opaque
data:
```

```
  identity: <BASE64>
  identity.pub: <BASE64>
  known_hosts: <BASE64>
```

To use this secret when creating an accelerator, provide the secret name in the `spec.git.secretRef.name` property:

`private-acc-ssh.yaml`

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: private-acc
spec:
  displayName: private
  description: Accelerator using a private repository
  git:
    url: REPOSITORY-URL
    ref:
      branch: main
    secretRef:
      name: ssh-credentials
```

When using SSH credentials, the `REPOSITORY-URL` must include the user name as part of the URL. For example: `ssh://user@example.com:22/repository.git`. For more information, see Flux documentation.

If you are using the Tanzu CLI, add the `--secret-ref` flag to your `tanzu accelerator create` command and provide the name of the secret for that flag.

Examples for a private source-image repository

If your registry uses a self-signed certificate then you must add the CA certificate data to the configuration for the “Tanzu Application Platform Source Controller” component. Add it under `source_controller.ca_cert_data` in your `tap-values.yaml` file that is used during installation.

`tap-values.yaml`

```
source_controller:
  ca_cert_data: |-
    -----BEGIN CERTIFICATE-----
    .
    .
    .  < certificate data >
    .
    .
    -----END CERTIFICATE-----
```

Example using image-pull credentials

To create an accelerator using a private source-image repository, create a secret with the image-pull credentials:

```
create secret generic registry-credentials \
    --namespace accelerator-system \
    --from-literal=username=<user> \
    --from-literal=password=<password>
```

Verify that your secret was created by running:

```
kubectl get secret --namespace accelerator-system registry-credentials -o yaml
```

The output is similar to:

```
apiVersion: v1
kind: Secret
metadata:
  name: registry-credentials
  namespace: accelerator-system
type: Opaque
data:
  username: <BASE64>
  password: <BASE64>
```

After you have the secret created, you can create the accelerator by using the `spec.git.secretRef.name` property:

`private-acc.yaml`

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: private-acc
spec:
  displayName: private
  description: Accelerator using a private repository
  source:
    image: "registry.example.com/test/private-acc-src:latest"
    imagePullSecrets:
    - name: registry-credentials
```

If you are using the Tanzu CLI, add the `--secret-ref` flag to your `tanzu accelerator create` command and provide the name of the secret for that flag.

Configure ingress timeouts when some accelerators take longer to generate

If Tanzu Application Platform is configured to use an ingress for Tanzu Developer Portal and the Accelerator Server, then it might detect a timeout during accelerator generation. This can happen if the accelerator takes a longer time to generate than the default timeout. When this happens, Tanzu Developer Portal appears to continue to run for an indefinite period. In the IDE extension, it shows a `504` error. To mitigate this, you can increase the timeout value for the HTTPProxy resources used for the ingress by applying secrets with overlays to edit the HTTPProxy resources.

Configure an ingress timeout overlay secret for each HTTPProxy

For Tanzu Developer Portal, create the following overlay secret in the `tap-install` namespace:

```
apiVersion: v1
kind: Secret
metadata:
  name: patch-tap-gui-timeout
  namespace: tap-install
stringData:
  patch.yaml: |
    #@ load("@ytt:overlay", "overlay")
    #@overlay/match by=overlay.subset({"kind": "HTTPProxy", "metadata": {"name": "tap-
gui"}})
    ---
    spec:
      routes:
        #@overlay/match by=overlay.subset({"services": [{"name": "server"}]})
        #@overlay/match-child-defaults missing_ok=True
```

```
        - timeoutPolicy:
            idle: 30s
            response: 30s
```

For Accelerator Server (used for IDE extension), create the following overlay secret in the `tap-install` namespace:

```
apiVersion: v1
kind: Secret
metadata:
  name: patch-accelerator-timeout
  namespace: tap-install
stringData:
  patch.yaml: |
    #@ load("@ytt:overlay", "overlay")
    #@overlay/match by=overlay.subset({"kind": "HTTPProxy", "metadata": {"name": "acce
lerator"}})
    ---
    spec:
      routes:
        #@overlay/match by=overlay.subset({"services": [{"name": "acc-server"}]})
        #@overlay/match-child-defaults missing_ok=True
        - timeoutPolicy:
            idle: 30s
            response: 30s
```

Apply the timeout overlay secrets in tap-values.yaml

Add the following `package_overlays` section to `tap-values.yaml` before installing or updating Tanzu Application Platform:

```
package_overlays:
- name: tap-gui
  secrets:
  - name: patch-tap-gui-timeout
- name: accelerator
  secrets:
  - name: patch-accelerator-timeout
```

Configuring skipping TLS verification for access to Source Controller

You can configure the Flux or Tanzu Application Platform Source Controller to use Transport Layer Security (TLS) and use custom certificates. In that case, configure the Accelerator System to skip the TLS verification for calls to access the sources by providing the following property in the `accelerator` section of the `tap-values.yaml` file:

```
sources:
  skip_tls_verify: true
```

Enabling TLS for Accelerator Server

To enable TLS for the Accelerator Server, the following properties must be provided in the `accelerator` section of the `tap-values.yaml` file:

```
server:
  tls:
    enabled: true
```

```
    key: SERVER-PRIVATE-KEY
    crt: SERVER-CERTIFICATE
```

Where:

- `SERVER-PRIVATE-KEY` is the pem encoded server private key.
- `SERVER-CERTIFICATE` is the pem encoded server certificate.

Here is a sample `tap-values.yaml` configuration with TLS enabled for Accelerators Server:

```
server:
  tls:
    enabled: true
    key: |
      -----BEGIN PRIVATE KEY-----
      .
      .  < private key data >
      .
      -----END PRIVATE KEY-----
    crt: |
      -----BEGIN CERTIFICATE-----
      .
      .  < certificate data >
      .
      -----END CERTIFICATE-----
```

Configuring skipping TLS verification of Engine calls for Accelerator Server

If you configure the Accelerator Engine to use TLS and use custom certificates, then you can configure the Accelerator Server to skip the TLS verification for calls to the Engine by providing the following property in the `accelerator` section of the `tap-values.yaml` file:

```
server:
  engine_skip_tls_verify: true
```

Enabling TLS for Accelerator Engine

To enable TLS for the Accelerator Engine, the following properties are provided in the `accelerator` section of the `tap-values.yaml` file:

```
engine:
  tls:
    enabled: true
    key: ENGINE-PRIVATE-KEY
    crt: ENGINE-CERTIFICATE
```

Where:

- `ENGINE-PRIVATE-KEY` is the pem encoded acc-engine private key.
- `ENGINE-CERTIFICATE` is the pem encoded acc-engine certificate.

Here is a sample `tap-values.yaml` configuration with TLS enabled for Accelerators Engine:

```
engine:
  tls:
    enabled: true
    key: |
      -----BEGIN PRIVATE KEY-----
      .
```

```
    .  < private key data >
    .
    -----END PRIVATE KEY-----
  crt: |
    -----BEGIN CERTIFICATE-----
    .
    .  < certificate data >
    .
    -----END CERTIFICATE-----
```

Next steps

- Use vulnerability scanning in offline and air-gapped environments

Use vulnerability scanning in offline and air-gapped environments

Note

For air-gapped (offline) environments, VMware recommends using Aqua Security's Trivy scanner with Scan 2.0. This greatly simplifies the installation and maintenance experiences over Anchore Grype and Scan 1.0. For more information on Scan 1.0 versus Scan 2.0, see Overview of Supply Chain Security Tools - Scan.

Using Trivy with SCST - Scan 2.0

The Aqua Trivy vulnerability scanner uses two databases to perform vulnerability scans:

1. Vulnerability Database: This database contains the vulnerability information used to scan artifacts. This database is built every six hours on GitHub and is distributed using GitHub Container registry (GHCR).
2. Java Index Database: This database enables Trivy to identify the `groupId`, `artifactId`, and `version` of JAR files. It is built once a day on GitHub and distributed using GitHub Container registry (GHCR).

In an online installation of Tanzu Application Platform, these databases are automatically downloaded from GHCR (Github Container Registry) with each scan execution. To enable scanning in an offline environment, these two databases must be relocated to your private container image registry, and Trivy must be configured to use your private location instead of downloading from GHCR.

For more information about the Trivy vulnerability databases, see the Trivy documentation.

Relocate Trivy Images

The databases are stored in an OCI-compliant registry as OCI artifacts. To copy these to your private registry, Trivy recommends using the ORAS CLI.

To make the Trivy databases available in your private registry:

1. Create a repository within your private registry. This example uses aquasecurity to remain consistent with the GHCR repository.
2. Copy the vulnerability database from GHCR to your private registry:

```
oras cp ghcr.io/aquasecurity/trivy-db:2 CONTAINER-REGISTRY/aquasecurity/trivy-d
```

```
b:2
```

> **Note**
>
> The Trivy scanner is hardcoded to use the `2` tag for the vulnerability database.

Where `CONTAINER-REGISTRY` is the URL for your registry. For example, `harbor.example.com`

3. Copy the Java index database from GHCR to your private registry:

```
oras cp ghcr.io/aquasecurity/trivy-java-db:1 harbor.CONTAINER-REGISTRY/aquasecu
rity/trivy-java-db:1
```

Where `CONTAINER-REGISTRY` is the URL for your registry. For example, `harbor.example.com`

> **Note**
>
> The Trivy scanner is hardcoded to use the `1` tag for the Java index database.

The databases are now available locally in your air-gapped environment. The next step is to configure the supply chain to use the air-gapped location for Trivy. For more information about using Trivy in an air-gapped environment, see the Trivy documentation.

Configure Trivy in the Supply Chain

Now that you have relocated the images, you must configure the scanning step of the supply chain to use Trivy with the SCST - Scan 2.0, and provide the location of the database artifacts in the private registry.

1. To enable SCST - Scan 2.0 with Trivy with private registries, update your `tap-values.yaml` file to specify the Trivy `ClusterImageTemplate`. For example:

```
ootb_supply_chain_testing_scanning:
  image_scanner_template_name: image-vulnerability-scan-trivy
  scanning:
    steps:
      env_vars:
      - name: TRIVY_DB_REPOSITORY
        value: CONTAINER-REGISTRY/aquasecurity/trivy-db
      - name: TRIVY_JAVA_DB_REPOSITORY
        value: CONTAINER-REGISTRY/aquasecurity/trivy-java-db
      - name: TRIVY_OFFLINE_SCAN
        value: "true"
```

Where `CONTAINER-REGISTRY` is the URL for your registry. For example, `harbor.example.com`

2. Update your Tanzu Application Platform installation by running:

```
tanzu package installed update tap -p tap.tanzu.vmware.com -v TAP-VERSION  --va
lues-file tap-values.yaml -n tap-install
```

Where `TAP-VERSION` is the version of Tanzu Application Platform installed.

Trivy is now configured to use resources in your air-gapped environment when executed with the supply chain. For more information about using Trivy in an air-gapped environment, see the Trivy documentation.

Using Grype with Scan 1.0

VMware recommends using Aqua Trivy with SCST - Scan 2.0 for offline or air-gapped environments due to the simplified setup and maintenance. However, if you require policy enablement based on vulnerability scanning, use the following steps to enable Grype scanning with SCST - Scan 1.0.

The Grype CLI attempts to perform two over-the-Internet calls:

- One to verify for later versions of the CLI.
- One to update the vulnerability database before scanning.

For the Grype CLI to function in an offline or air-gapped environment, the vulnerability database must be hosted in the environment. You must configure the Grype CLI with the internal URL.

The Grype CLI accepts environment variables to satisfy these needs.

Host the Grype vulnerability database

To host the Grype vulnerability database in an air-gapped environment:

1. Retrieve the Grype listing file from its public endpoint: https://toolbox-data.anchore.io/grype/databases/listing.json.
2. Create your own `listing.json` file.

   Note Different Grype versions require specific database schema versions. To avoid compatibility issues between different versions, include a database schema for each version. For example:

```
    {
      "available": {
        "1": [
          {
            "built": "2023-06-16T01:33:30Z",
            "version": 1,
            "url": "https://toolbox-data.anchore.io/grype/databases/vulnerabili
ty-db_v1_2023-06-16T01:33:30Z_1621f4169ffd15bea9e5.tar.gz",
            "checksum": "sha256:3f2c1b432945cca9a69b2e604f6fb231fec450fdd27f494
6fc5608692b63a9d1"
          }
        ],
        "2": [
          {
            "built": "2023-06-16T01:33:30Z",
            "version": 2,
            "url": "https://toolbox-data.anchore.io/grype/databases/vulnerabili
ty-db_v2_2023-06-16T01:33:30Z_d6eee5e78d9b78285e1a.tar.gz",
            "checksum": "sha256:7b7e3a2a7712c72b8c5cc777733c4d8d140d8cfee65e4f0
4540abbdfe3ef1f65"
          }
        ],
        "3": [
          {
            "built": "2023-06-16T01:33:30Z",
            "version": 3,
            "url": "https://toolbox-data.anchore.io/grype/databases/vulnerabili
ty-db_v3_2023-06-16T01:33:30Z_f96ae38a7b05987c3ece.tar.gz",
            "checksum": "sha256:8ea9fae3fda3bf3bf35bd5e5eb656fc127b59cd3c42db4c
36795556aab8a9cf0"
          }
        ],
        "4": [
          {
```

```
            "built": "2023-06-16T01:33:30Z",
            "version": 4,
            "url": "https://toolbox-data.anchore.io/grype/databases/vulnerabili
ty-db_v4_2023-06-16T01:33:30Z_13bba2fa8ff62b7f8b26.tar.gz",
            "checksum": "sha256:3b53d20241b88e5aa45feb817b325c53d6efbe9fa1fc5a6
7eeddaecafa7687e0"
          }
        ],
        "5": [
          {
            "built": "2023-06-16T01:33:30Z",
            "version": 5,
            "url": "https://toolbox-data.anchore.io/grype/databases/vulnerabili
ty-db_v5_2023-06-16T01:33:30Z_e07da3853f6db6eb1104.tar.gz",
            "checksum": "sha256:93d4d9d2f9e39f86570f832cf85b7149a949ca6f1613581
b10c12393509d884f"
          }
        ]
      }
    }
```

Where `url` points to a tarball containing Grype's `vulnerability.db` and `metadata.json` files.

3. Download and host the tarballs in your internal file server.

> **Note**
>
> Some storage solutions for internal file servers change the name of TAR files automatically because of their limits. Notice these modified names and reflect the changes in the `url`. Ensure that the timestamp in the name is correctly formatted because Grype parses the name of TAR artifact to get the timestamp.

4. Update the download `url` to point at your internal endpoint.

   For information about setting up an offline vulnerability database, see the Anchore Grype README in GitHub.

To enable Grype in offline air-gapped environments

1. Add the following to your `tap-values.yaml` file:

```
grype:
  db:
    dbUpdateUrl: INTERNAL-VULN-DB-URL
```

   Where `INTERNAL-VULN-DB-URL` is the URL that points to the internal file server.

2. Update Tanzu Application Platform:

```
tanzu package installed update tap -f tap-values.yaml -n tap-install
```

Configure Grype environmental variables

1. Create a secret that contains the ytt overlay to add the Grype environment variable to the ScanTemplates.

```
apiVersion: v1
kind: Secret
metadata:
```

```
  name: grype-airgap-environmental-variables
  namespace: tap-install
stringData:
  patch.yaml: |
    #@ load("@ytt:overlay", "overlay")

    #@overlay/match by=overlay.subset({"kind":"ScanTemplate"}),expects="1+"
    ---
    spec:
      template:
        initContainers:
          #@overlay/match by=overlay.subset({"name": "scan-plugin"}), expects
="1+"
          - name: scan-plugin
            #@overlay/match missing_ok=True
            env:
              #@overlay/append
              - name: GRYPE_CHECK_FOR_APP_UPDATE
                value: "false"
```

Where `spec.template.initContainers[]` specifies setting one or more environment variables in the `scan-plugin` `initContainer`.

Note

If you are using the Namespace Provisioner to provision a new developer namespace and want to apply a package overlay for Grype, you must import the overlay `Secret`. See Import overlay secrets.

Troubleshooting

ERROR failed to fetch latest cli version

Symptom

Error message:

```
ERROR failed to fetch latest version: Get "https://toolbox-data.anchore.io/grype/releases/latest/VERSION": dial tcp: lookup toolbox-data.anchore.io on [::1]:53: read udp [::1]:65010->[::1]:53: read: connection refused
```

Cause

The Grype CLI checks for later versions of the CLI by contacting the anchore endpoint over-the-Internet.

Solution

This message is a warning and the Grype scan still runs.

To deactivate this check, set the environment variable `GRYPE_CHECK_FOR_APP_UPDATE` to `false` by using a package overlay:

1. Create a secret that contains the ytt overlay to add the Grype environment variable to the ScanTemplates.

   ```
   apiVersion: v1
   kind: Secret
   ```

```
metadata:
  name: grype-airgap-deactivate-cli-check-overlay
  namespace: tap-install #! namespace where tap is installed
stringData:
  patch.yaml: |
    #@ load("@ytt:overlay", "overlay")

    #@overlay/match by=overlay.subset({"kind":"ScanTemplate"}),expects="1+"
    ---
    spec:
      template:
        initContainers:
          #@overlay/match by=overlay.subset({"name": "scan-plugin"}), expects
="1+"
          - name: scan-plugin
            #@overlay/match missing_ok=True
            env:
              #@overlay/append
              - name: GRYPE_CHECK_FOR_APP_UPDATE
                value: "false"
```

2. Configure `tap-values.yaml` to use `package_overlays`. Add the following to your `tap-values.yaml` file:

```
package_overlays:
  - name: "grype"
    secrets:
        - name: "grype-airgap-deactivate-cli-check-overlay"
```

3. Update Tanzu Application Platform:

```
tanzu package installed update tap -f tap-values.yaml -n tap-install
```

Database is too old

Symptom

Error message:

```
1 error occurred:
  * db could not be loaded: the vulnerability database was built N days/weeks ago (max
allowed age is 5 days)
```

Cause

Grype needs up-to-date vulnerability information to provide accurate matches. By default, it fails to run if the local database was not built in the last 5 days.

Solution

There are two options to resolve this:

1. Stale databases weaken your security posture. VMware recommends updating the database daily.
2. If you cannot update the database daily, configure the data staleness check using the environment variable `GRYPE_DB_MAX_ALLOWED_BUILT_AGE` and apply a package overlay:
    1. Create a secret that contains the ytt overlay to add the Grype environment variable to the ScanTemplates.

```
apiVersion: v1
kind: Secret
metadata:
  name: grype-airgap-override-stale-db-overlay
  namespace: tap-install #! namespace where tap is installed
stringData:
  patch.yaml: |
    #@ load("@ytt:overlay", "overlay")

    #@overlay/match by=overlay.subset({"kind":"ScanTemplate"}),expects="1
+"
    ---
    spec:
      template:
        initContainers:
          #@overlay/match by=overlay.subset({"name": "scan-plugin"}), exp
ects="1+"
          - name: scan-plugin
            #@overlay/match missing_ok=True
            env:
              #@overlay/append
              - name: GRYPE_DB_MAX_ALLOWED_BUILT_AGE #! see note on best
practices
                value: "120h"
```

Note

The default maximum allowed built age of Grype's vulnerability database is 5 days. This means that scanning with a 6 day old database causes the scan to fail. You can use the `GRYPE_DB_MAX_ALLOWED_BUILT_AGE` parameter to override the default in accordance with your security posture.

2. Configure `tap-values.yaml` to use `package_overlays`. Add the following to your `tap-values.yaml` file:

```
package_overlays:
  - name: "grype"
    secrets:
        - name: "grype-airgap-override-stale-db-overlay"
```

3. Update Tanzu Application Platform:

```
tanzu package installed update tap -f tap-values.yaml -n tap-install
```

Vulnerability database is invalid

Symptom

Error message:

```
scan-pod[scan-plugin]  1 error occurred:
scan-pod[scan-plugin]  * failed to load vulnerability db: vulnerability database is in
valid (run db update to correct): database metadata not found: /.cache/grype/db/5
```

Solution

Examine the `listing.json` file you created. This matches the format of the listing file. The listing file is located at Anchore Grype's public endpoint. See the Grype README.md in GitHub.

An example `listing.json`:

```
{
  "available": {
    "5": [
      {
        "built": "2023-03-28T01:29:38Z",
        "version": 5,
        "url": "https://toolbox-data.anchore.io/grype/databases/vulnerability-db_v5_20
23-03-28T01:29:38Z_e49d318c32a6113eed07.tar.gz",
        "checksum": "sha256:408ce2932f04dee929a5df524e92494f2d635c6b19e30ff9f0a50425b1
fc29a1"
      },
      .....
    ]
  }
}
```

Where:

- `5` refers to the Grype vulnerability database schema.
- `built` is the build timestamp in the format `yyyy-MM-ddTHH:mm:ssZ`.
- `url` is the download URL for the tarball containing the database. This points at your internal endpoint. The tarball contains the following files:
    - `vulnerability.db` is an SQLite file that is Grype's vulnerability database. Each time the data shape of the vulnerability database changes, a new schema is created. Different Grype versions require specific database schema versions. For example, Grype `v0.54.0` requires database schema version v5.
    - `metadata.json` file
- `checksum` is the SHA used to verify the database's integrity.

Verify these possible reasons why the vulnerability database is not valid:

1. The database schema is invalid. Confirm that the required database schema for the installed Grype version is used. Confirm that the top level version key matches the nested `version`. For example, the top level version `1` in the following snippet does not match the nested `version: 5`.

    ```
    {
      "available": {
        "1": [{
                "built": "2023-02-08T08_17_20Z",
                "version": 5,
                "url": "https://INTERNAL-ENDPOINT/PATH-TO-TARBALL/vulnerability-db_v
    5_2023-02-08T08_17_20Z_6ef73016d160043c630f.tar.gz",
                "checksum": "sha256:aab8d369933c845878ef1b53bb5c26ee49b91ddc5cd87c9e
    b57ffb203a88a72f"
        }]
      }
    }
    ```

    Where `PATH-TO-TARBALL` is the path to the tarball containing the vulnerability database.

    As stale databases weaken your security posture, VMware recommends using the newest entry of the relevant schema version in the `listing.json` file. See Anchore's grype-db in GitHub.

2. The `built` parameters in the `listing.json` file are incorrectly formatted. The proper format is `yyyy-MM-ddTHH:mm:ssZ`.
3. The `url` that you modified to point at an internal endpoint is not reachable from the cluster. For information about verifying connectivity, see Debug Grype database in a cluster.
4. Verify if there are syntax errors in the listing.json:

   ```
   grype db check
   ```

5. Validate the configured listing.json:

   ```
   grype db list -o raw
   ```

Debug Grype database in a cluster

1. Describe the failed source scan or image scan to verify the name of the ScanTemplate being used.
   - For `sourcescan`, run:

     ```
     kubectl describe sourcescan SCAN-NAME -n DEV-NAMESPACE
     ```

   - For `imagescan`, run:

     ```
     kubectl describe imagescan SCAN-NAME -n DEV-NAMESPACE
     ```

   Where `SCAN-NAME` is the name of the source or image scan that failed.
2. Pause reconciliation of the `grype.scanning.apps.tanzu.vmware.com` package:

   ```
   kctrl package installed pause -i <PACKAGE-INSTALL-NAME> -n tap-install
   ```

   Where `PACKAGE-INSTALL-NAME` is the name of the `grype.scanning.apps.tanzu.vmware.com` package, for example, grype
3. Edit the ScanTemplate's `scan-plugin` container to include a "sleep" entrypoint which allows you to troubleshoot inside the container:

   ```
   - name: scan-plugin
     volumeMounts:
       ...
     image: #@ data.values.scanner.image
     imagePullPolicy: IfNotPresent
     env:
       ...
     command: ["/bin/bash"]
     args:
     - "sleep 1800" # insert 30 min sleep here
   ```

4. Re-run the scan.
5. Get the name of the `scan-plugin` pod.

   ```
   kubectl get pods -n DEV-NAMESPACE
   ```

6. Get a shell to the container.

   ```
   kubectl exec --stdin --tty SCAN-PLUGIN-POD -c step-scan-plugin -- /bin/bash
   ```

Where `SCAN-PLUGIN-POD` is the name of the `scan-plugin` pod. For more information, see the Kubernetes documentation.

7. Inside the container, run Grype CLI commands to report database status and verify connectivity from the cluster to the mirror. See the Grype documentation in GitHub.
    - Report current status of Grype's database, such as location, build date, and checksum:

      ```
      grype db status
      ```

8. Ensure that the built parameters in the `listing.json` have timestamps in this proper format `yyyy-MM-ddTHH:mm:ssZ`.
9. After you complete troubleshooting, to trigger reconciliation, run:

    ```
    kctrl package installed kick -i <PACKAGE-INSTALL-NAME> -n tap-install
    ```

    Where `PACKAGE-INSTALL-NAME` is the name of the `grype.scanning.apps.tanzu.vmware.com` package, such as Grype.

Grype package overlays are not applied to scantemplates created by Namespace Provisioner

If you used the Namespace Provisioner to provision a new developer namespace and want to apply a package overlay for Grype, see Import overlay secrets.

Set up developer namespaces to use your installed packages

For details about how to automatically set up your developer namespaces, see Provision developer namespaces in Namespace Provisioner.

Additional configuration for testing and scanning

If you plan to install or have already installed Out of the Box Supply Chains with Testing and Scanning, you can use Namespace Provisioner to set up the required resources. For more information, see Customize installation in the Namespace Provisioner documentation for configuration steps.

Legacy namespace setup

To use the legacy manual process for setting up developer namespaces, see Legacy namespace setup.

Next steps

For more information about the Namespace Provisioner mode, see Work with Git repositories in air-gapped environments with Namespace Provisioner.

For more information about the manual mode, see Deploy an air-gapped workload.

Install IDE extensions in your air-gapped environment

This topic tells you how to install IDE extensions in your air-gapped environment.

To install VS Code or IntelliJ extensions in an air-gapped environment, you cannot use IDE's built-in UI, because it downloads and install extensions directly from VS Code or IntelliJ Marketplace.

The following are high-level steps to install IDE extensions in your air-gapped environment:

1. Outside the air-gapped environment:
    1. Download the extension as an archive from VS Code or IntelliJ Marketplace.
    2. Copy the extension to a location that is accessible from within the air-gapped environment.
2. In the air-gapped environment:
    1. Install the extension into the IDE by using the archive generated earlier.

Install VS Code in your air-gapped environment

Follow these steps to retrieve the `.vsix` archive and install VS Code in your air-gapped environment:

1. Find the extension you want to install on VS Code Marketplace. For example:
    - Tanzu Developer Tools for Vscode
    - Tanzu App Accelerator for Vscode
2. In a column on the right side of the screen, under **Resources**, click the **Download Extension** link.

    A file called `vmware.tanzu-dev-tools-${version}.vsix` is downloaded.
3. Save the file to a location that is accessible from your air-gapped environment. For example, a USB drive.
4. Repeat these steps for all extensions you want to install, including any dependencies.

    For example, Tanzu Developer Tools for VS Code requires all of the following extensions as dependencies:
    - Red Hat Java
    - Red Hat Yaml
    - Debugger for Java

    The Application Accelerator extension, on the other hand, does not require additional dependencies.
5. In your air-gapped environment, install VS Code extensions as follows:
    1. Open VS Code
    2. Open the command palette by pressing CTRL-SHIFT-P or CMD-SHIFT-P on Mac.
    3. In the search box, type **vsix** and select **Install from VSIX…**.

    You can script this step by using commands such as:

    ```
    code --install-extension ${path_to_vsix_file}
    ```

Install IntelliJ in your air-gapped environment

Follow these steps to retrieve the `.zip` archive and install IntelliJ in your air-gapped environment:

1. Find the extension you want to install on Jetbrains Marketplace. For example, Tanzu Developer Tools for IntelliJ.

2. Click **Get** near the top-right of the screen.
3. Find the version you want to download and click the **Download** link.

   A file called `Tanzu_Developer_Tools-${version}.zip` is downloaded.
4. Save the file to a location that is accessible from your air-gapped environment. For example, a USB drive.
5. Repeat these steps for all extensions you want to install.
6. Follow the instructions in the IntelliJ documentation to install IntelliJ.

Install Tanzu Application Platform (AWS)

You can install Tanzu Application Platform (commonly known as TAP) on Amazon Elastic Kubernetes Services (EKS) by using Amazon Elastic Container Registry (ECR).

To install, take the following steps.

Step	Task	Link
1.	Review the prerequisites to ensure that you have set up everything required before beginning the installation	Prerequisites
2.	Accept Tanzu Application Platform EULAs and install the Tanzu CLI	Accept Tanzu Application Platform EULAs and installing the Tanzu CLI
3.	Create AWS Resources (EKS Cluster, roles, etc)	Create AWS Resources
4.	Install Cluster Essentials for Tanzu	Deploy Cluster Essentials
5.	Add the Tanzu Application Platform package repository, prepare your Tanzu Application Platform profile, and install the profile to the cluster	Install the Tanzu Application Platform package and profiles
6.	(Optional) Install any additional packages that were not in the profile	Install individual packages
7.	Set up developer namespaces to use your installed packages	Set up developer namespaces to use your installed packages
8.	Install developer tools into your integrated development environment (IDE)	Install Tanzu Developer Tools for your VS Code

After installing Tanzu Application Platform on your Kubernetes clusters, get started with Tanzu Application Platform and create your ECR repositories for your workload, such as `tanzu-application-platform/tanzu-java-web-app-default`, `tanzu-application-platform/tanzu-java-web-app-default-bundle`, and `tanzu-application-platform/tanzu-java-web-app-default-source`.

Install Tanzu Application Platform (AWS)

You can install Tanzu Application Platform (commonly known as TAP) on Amazon Elastic Kubernetes Services (EKS) by using Amazon Elastic Container Registry (ECR).

To install, take the following steps.

Step	Task	Link
1.	Review the prerequisites to ensure that you have set up everything required before beginning the installation	Prerequisites
2.	Accept Tanzu Application Platform EULAs and install the Tanzu CLI	Accept Tanzu Application Platform EULAs and installing the Tanzu CLI
3.	Create AWS Resources (EKS Cluster, roles, etc)	Create AWS Resources
4.	Install Cluster Essentials for Tanzu	Deploy Cluster Essentials

Step	Task	Link
5.	Add the Tanzu Application Platform package repository, prepare your Tanzu Application Platform profile, and install the profile to the cluster	Install the Tanzu Application Platform package and profiles
6.	(Optional) Install any additional packages that were not in the profile	Install individual packages
7.	Set up developer namespaces to use your installed packages	Set up developer namespaces to use your installed packages
8.	Install developer tools into your integrated development environment (IDE)	Install Tanzu Developer Tools for your VS Code

After installing Tanzu Application Platform on your Kubernetes clusters, get started with Tanzu Application Platform and create your ECR repositories for your workload, such as `tanzu-application-platform/tanzu-java-web-app-default`, `tanzu-application-platform/tanzu-java-web-app-default-bundle`, and `tanzu-application-platform/tanzu-java-web-app-default-source`.

Create AWS Resources for Tanzu Application Platform

To install Tanzu Application Platform (commonly known as TAP) within the Amazon Web Services (AWS) Ecosystem, you must create several AWS resources. Use this topic to learn how to create:

- An Amazon Elastic Kubernetes Service (EKS) cluster to install Tanzu Application Platform.
- Identity and Access Management (IAM) roles to allow authentication and authorization to read and write from Amazon Elastic Container Registry (ECR).
- ECR Repositories for the Tanzu Application Platform container images. This is because AWS ECR does not support automatically creating container repositories on initial push. For more information, see the AWS repository in GitHub.

Creating these resources enables Tanzu Application Platform to use an IAM role bound to a Kubernetes service account for authentication, rather than the typical username and password stored in a Kubernetes secret strategy. For more information, see this AWS documentation.

This is important when using ECR because authenticating to ECR is a two-step process:

1. Retrieve a token using your AWS credentials.
2. Use the token to authenticate to the registry.

To increase security, the token has a lifetime of 12 hours. This makes storing it as a secret for a service impractical because it has to be refreshed every 12 hours.

Using an IAM role on a service account mitigates the need to retrieve the token at all because it is handled by credential helpers within the services.

Prerequisites

There are numerous methods to manage AWS cloud resources and create EKS clusters. The method presented in the following guide was chosen for simplicity.

Before installing Tanzu Application Platform on AWS, you need:

- An AWS Account. You need to create all of your resources within Amazon Web Services, so you need an Amazon account. For more information, see How do I create and activate a new AWS account?. You need your account ID for this walkthrough.
- AWS CLI. This walkthrough uses the AWS CLI to both query and configure resources in AWS, such as IAM roles. For more information, see this AWS documentation.
- `eksctl` command line. The `eksctl` command line helps you manage the life cycle of EKS clusters. This guide uses it to create clusters. To install `eksctl`, see the eksctl

documentation.

Export environment variables

Variables are used throughout this guide. To simplify the process and minimize the opportunity for errors, export these variables:

```
export AWS_ACCOUNT_ID=012345678901
export AWS_REGION=us-west-2
export EKS_CLUSTER_NAME=tap-on-aws
```

Where:

Variable	Description
AWS_ACCOUNT_ID	Your AWS account ID
AWS_REGION	The AWS region you are going to deploy to
EKS_CLUSTER_NAME	The name of your EKS Cluster

Create an EKS cluster

To create an EKS cluster in the specified region, run:

```
eksctl create cluster --name $EKS_CLUSTER_NAME --managed --region $AWS_REGION --instan
ce-types t3.xlarge --version 1.28 --with-oidc -N 5
```

Creating the control plane and node group can take anywhere from 30-60 minutes.

Note

This step is optional if you already have an existing EKS Cluster v1.23 or later with OpenID Connect (OIDC) authentication enabled. For more information about how to enable the OIDC provider, see AWS documentation.

Install the EBS CSI driver

Tanzu Application Platform requires stateful services. Starting from EKS v1.23, the EBS CSI driver is no longer installed by default. For more information about how to install the EBS CSI driver, see the AWS documentation.

Create the platform container repositories

ECR requires that the container repositories are already created for images to be pushed to them. For Tanzu Application Platform, you must create the following two repositories:

- A repository to store the Tanzu Application Platform service container images
- A repository to store Tanzu Build Service generated Base OS and Builder container images

To create these repositories, run:

```
aws ecr create-repository --repository-name tap-images --region $AWS_REGION
aws ecr create-repository --repository-name tap-build-service --region $AWS_REGION
```

Depending on your installation choices, you might also require the following additional system-related repositories:

- A repository to store Tanzu Build Service full dependencies container images
- A repository to store Tanzu Application Platform's Local Source Proxy container images
- A repository to store Tanzu Cluster Essentials container images

To create these repositories, run:

```
aws ecr create-repository --repository-name full-deps --region $AWS_REGION
aws ecr create-repository --repository-name tap-lsp --region $AWS_REGION
aws ecr create-repository --repository-name tanzu-cluster-essentials --region $AWS_REG
ION
```

Name the repositories any name you want, but remember the names for when you later build the configuration.

Create the workload container repositories

Similar to the platform container repositories, you must create repositories for each workload that Tanzu Application Platform creates before creating any workloads so that a repository is available to upload container images and workload bundles.

When installing Tanzu Application Platform, you must specify a prefix for all workload registries. This topic uses `tanzu-application-platform` as the default value, but you can customize this value in the profile configuration created in Install Tanzu Application Platform package and profiles on AWS.

To use the default value, create two workload repositories for each workload with the following format:

```
tanzu-application-platform/WORKLOADNAME-NAMESPACE
tanzu-application-platform/WORKLOADNAME-NAMESPACE-bundle
```

For example, to create these repositories for the sample workload `tanzu-java-web-app` in the `default` namespace, you can run the following ECR command:

```
aws ecr create-repository --repository-name tanzu-application-platform/tanzu-java-web-
app-default --region $AWS_REGION
aws ecr create-repository --repository-name tanzu-application-platform/tanzu-java-web-
app-default-bundle --region $AWS_REGION
```

Note

The default Supply Chain Choreographer method of storing Kubernetes configuration is RegistryOps, which requires the `bundle` repository. If you enabled the GitOps capability, this repository is not required. For more information about the differences between RegistryOps and GitOps, see Use GitOps or RegistryOps with Supply Chain Choreographer.

Create IAM roles

By default, the EKS cluster is provisioned with an EC2 instance profile that provides read-only access for the entire EKS cluster to the ECR registry within your AWS account. For more information, see this AWS documentation.

However, some of the services within Tanzu Application Platform require write access or batch read access to the container repositories. To provide that access, create IAM roles and add the ARN to the Kubernetes service accounts that those services use. This ensures that only the required services have access to write container images to ECR and the ability for batch read access, rather than a blanket policy that applies to the entire cluster.

Create the following IAM Roles:

- Tanzu Build Service: Gives write access to the repository to allow the service to automatically upload new images. Also provides elevated batch read access to the `tap-images` and `full-deps` repositories. This is limited in scope to the service account for `kpack` and the dependency updater.
- Workload: Gives write access to the entire ECR registry with a prepended path. Also provides elevated batch read access to the `full-deps` repository if you use Tanzu Build Service full dependencies. This prevents you from updating the policy for each new workload created.
- Local Source Proxy: Gives write access to the repository to allow the service to automatically upload new images. This is limited in scope to the service account for Local Source Proxy.

To create the roles, you must establish two policies:

- Trust Policy: Limits the scope to the OIDC endpoint for the Kubernetes cluster and the Kubernetes service account you attach the role to.
- Permission Policy: Limits the scope of actions the role can take on resources.

Note

These policies attempt to achieve a least privilege model. Review them to confirm they adhere to your organization's policies.

To simplify this walkthrough, use a script to create these policy documents and the roles. This script outputs the files and then creates the IAM roles by using the policy documents. If Local Source Proxy is not in your installation plan, you can omit the associated commands.

Run:

```
# Retrieve the OIDC endpoint from the Kubernetes cluster and store it for use in the p
olicy.
export OIDCPROVIDER=$(aws eks describe-cluster --name $EKS_CLUSTER_NAME --region $AWS_
REGION --output json | jq '.cluster.identity.oidc.issuer' | tr -d '"' | sed 's/http
s:\/\///')
cat << EOF > build-service-trust-policy.json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::${AWS_ACCOUNT_ID}:oidc-provider/${OIDCPROVI
DER}"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "${OIDCPROVIDER}:aud": "sts.amazonaws.com"
                },
                "StringLike": {
                    "${OIDCPROVIDER}:sub": [
```

```
                        "system:serviceaccount:kpack:controller",
                        "system:serviceaccount:build-service:dependency-updater-contro
ller-serviceaccount"
                    ]
                }
            }
        }
    ]
}
EOF

cat << EOF > build-service-policy.json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "ecr:DescribeRegistry",
                "ecr:GetAuthorizationToken",
                "ecr:GetRegistryPolicy",
                "ecr:PutRegistryPolicy",
                "ecr:PutReplicationConfiguration",
                "ecr:DeleteRegistryPolicy"
            ],
            "Resource": "*",
            "Effect": "Allow",
            "Sid": "TAPEcrBuildServiceGlobal"
        },
        {
            "Action": [
                "ecr:DescribeImages",
                "ecr:ListImages",
                "ecr:BatchCheckLayerAvailability",
                "ecr:BatchGetImage",
                "ecr:BatchGetRepositoryScanningConfiguration",
                "ecr:DescribeImageReplicationStatus",
                "ecr:DescribeImageScanFindings",
                "ecr:DescribeRepositories",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetLifecyclePolicy",
                "ecr:GetLifecyclePolicyPreview",
                "ecr:GetRegistryScanningConfiguration",
                "ecr:GetRepositoryPolicy",
                "ecr:ListTagsForResource",
                "ecr:TagResource",
                "ecr:UntagResource",
                "ecr:BatchDeleteImage",
                "ecr:BatchImportUpstreamImage",
                "ecr:CompleteLayerUpload",
                "ecr:CreatePullThroughCacheRule",
                "ecr:CreateRepository",
                "ecr:DeleteLifecyclePolicy",
                "ecr:DeletePullThroughCacheRule",
                "ecr:DeleteRepository",
                "ecr:InitiateLayerUpload",
                "ecr:PutImage",
                "ecr:PutImageScanningConfiguration",
                "ecr:PutImageTagMutability",
                "ecr:PutLifecyclePolicy",
                "ecr:PutRegistryScanningConfiguration",
                "ecr:ReplicateImage",
                "ecr:StartImageScan",
                "ecr:StartLifecyclePolicyPreview",
                "ecr:UploadLayerPart",
                "ecr:DeleteRepositoryPolicy",
                "ecr:SetRepositoryPolicy"
```

```
            ],
            "Resource": [
                "arn:aws:ecr:${AWS_REGION}:${AWS_ACCOUNT_ID}:repository/full-deps",
                "arn:aws:ecr:${AWS_REGION}:${AWS_ACCOUNT_ID}:repository/tap-build-serv
ice",
                "arn:aws:ecr:${AWS_REGION}:${AWS_ACCOUNT_ID}:repository/tap-images"
            ],
            "Effect": "Allow",
            "Sid": "TAPEcrBuildServiceScoped"
        }
    ]
}
EOF

cat << EOF > workload-policy.json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "ecr:DescribeRegistry",
                "ecr:GetAuthorizationToken",
                "ecr:GetRegistryPolicy",
                "ecr:PutRegistryPolicy",
                "ecr:PutReplicationConfiguration",
                "ecr:DeleteRegistryPolicy"
            ],
            "Resource": "*",
            "Effect": "Allow",
            "Sid": "TAPEcrWorkloadGlobal"
        },
        {
            "Action": [
                "ecr:DescribeImages",
                "ecr:ListImages",
                "ecr:BatchCheckLayerAvailability",
                "ecr:BatchGetImage",
                "ecr:BatchGetRepositoryScanningConfiguration",
                "ecr:DescribeImageReplicationStatus",
                "ecr:DescribeImageScanFindings",
                "ecr:DescribeRepositories",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetLifecyclePolicy",
                "ecr:GetLifecyclePolicyPreview",
                "ecr:GetRegistryScanningConfiguration",
                "ecr:GetRepositoryPolicy",
                "ecr:ListTagsForResource",
                "ecr:TagResource",
                "ecr:UntagResource",
                "ecr:BatchDeleteImage",
                "ecr:BatchImportUpstreamImage",
                "ecr:CompleteLayerUpload",
                "ecr:CreatePullThroughCacheRule",
                "ecr:CreateRepository",
                "ecr:DeleteLifecyclePolicy",
                "ecr:DeletePullThroughCacheRule",
                "ecr:DeleteRepository",
                "ecr:InitiateLayerUpload",
                "ecr:PutImage",
                "ecr:PutImageScanningConfiguration",
                "ecr:PutImageTagMutability",
                "ecr:PutLifecyclePolicy",
                "ecr:PutRegistryScanningConfiguration",
                "ecr:ReplicateImage",
                "ecr:StartImageScan",
                "ecr:StartLifecyclePolicyPreview",
```

```
                "ecr:UploadLayerPart",
                "ecr:DeleteRepositoryPolicy",
                "ecr:SetRepositoryPolicy"
            ],
            "Resource": [
                "arn:aws:ecr:${AWS_REGION}:${AWS_ACCOUNT_ID}:repository/full-deps",
                "arn:aws:ecr:${AWS_REGION}:${AWS_ACCOUNT_ID}:repository/tanzu-applicat
ion-platform/*"
            ],
            "Effect": "Allow",
            "Sid": "TAPEcrWorkloadScoped"
        }
    ]
}
EOF

cat << EOF > workload-trust-policy.json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::${AWS_ACCOUNT_ID}:oidc-provider/${OIDCPROVI
DER}"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringLike": {
                    "${OIDCPROVIDER}:sub": "system:serviceaccount:*:default",
                    "${OIDCPROVIDER}:aud": "sts.amazonaws.com"
                }
            }
        }
    ]
}
EOF

cat << EOF > local-source-proxy-trust-policy.json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::${AWS_ACCOUNT_ID}:oidc-provider/${OIDCPROVI
DER}"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "${OIDCPROVIDER}:aud": "sts.amazonaws.com"
                },
                "StringLike": {
                    "${OIDCPROVIDER}:sub": [
                        "system:serviceaccount:tap-local-source-system:proxy-manager"
                    ]
                }
            }
        }
    ]
}
EOF

cat << EOF > local-source-proxy-policy.json
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "ecr:GetAuthorizationToken"
            ],
            "Resource": "*",
            "Effect": "Allow",
            "Sid": "TAPLSPGlobal"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetRepositoryPolicy",
                "ecr:DescribeRepositories",
                "ecr:ListImages",
                "ecr:DescribeImages",
                "ecr:BatchGetImage",
                "ecr:GetLifecyclePolicy",
                "ecr:GetLifecyclePolicyPreview",
                "ecr:ListTagsForResource",
                "ecr:DescribeImageScanFindings",
                "ecr:InitiateLayerUpload",
                "ecr:UploadLayerPart",
                "ecr:CompleteLayerUpload",
                "ecr:PutImage"
            ],
            "Resource": [
                "arn:aws:ecr:${AWS_REGION}:${AWS_ACCOUNT_ID}:repository/tap-lsp"
            ],
            "Sid": "TAPLSPScoped"
        }
    ]
}
EOF

# Create the Tanzu Build Service Role.
aws iam create-role --role-name tap-build-service --assume-role-policy-document fil
e://build-service-trust-policy.json
# Attach the Policy to the Build Role.
aws iam put-role-policy --role-name tap-build-service --policy-name tapBuildServicePol
icy --policy-document file://build-service-policy.json

# Create the Workload Role.
aws iam create-role --role-name tap-workload --assume-role-policy-document file://work
load-trust-policy.json
# Attach the Policy to the Workload Role.
aws iam put-role-policy --role-name tap-workload --policy-name tapWorkload --policy-do
cument file://workload-policy.json

# Create the TAP Local Source Proxy Role.
aws iam create-role --role-name tap-local-source-proxy --assume-role-policy-document f
ile://local-source-proxy-trust-policy.json
# Attach the Policy to the tap-local-source-proxy Role created earlier.
aws iam put-role-policy --role-name tap-local-source-proxy --policy-name tapLocalSourc
ePolicy --policy-document file://local-source-proxy-policy.json
```

Next steps

- Deploy Cluster Essentials

Important

When you use a VMware Tanzu Kubernetes Grid cluster, you do not need to install Cluster Essentials because the contents of Cluster Essentials are already installed on your cluster.

- Install Tanzu Application Platform package and profiles on AWS

Install Tanzu Application Platform package and profiles on AWS

This topic tells you how to install Tanzu Application Platform (commonly known as TAP) packages from your Tanzu Application Platform package repository on to AWS.

Before installing the packages, ensure you have:

- Completed the Prerequisites.
- Created AWS Resources.
- Accepted Tanzu Application Platform EULA and installed Tanzu CLI with any required plug-ins.
- Installed Cluster Essentials for Tanzu.

Relocate images to a registry

VMware recommends relocating the images from VMware Tanzu Network registry to your own container image registry before attempting installation. If you don't relocate the images, Tanzu Application Platform will depend on VMware Tanzu Network for continued operation, and VMware Tanzu Network offers no uptime guarantees. The option to skip relocation is documented for evaluation and proof-of-concept only.

This section describes how to relocate images to the `tap-images` repository created in Amazon ECR. See Creating AWS Resources for more information.

To relocate images from the VMware Tanzu Network registry to the ECR registry:

1. Set up environment variables for installation use by running:

```
export AWS_ACCOUNT_ID=MY-AWS-ACCOUNT-ID
export AWS_REGION=TARGET-AWS-REGION

# Set tanzunet as the source registry to copy the Tanzu Application Platform pa
ckages from.
export IMGPKG_REGISTRY_HOSTNAME_0=registry.tanzu.vmware.com
export IMGPKG_REGISTRY_USERNAME_0=MY-TANZUNET-USERNAME
export IMGPKG_REGISTRY_PASSWORD_0=MY-TANZUNET-PASSWORD

# The user's registry for copying the Tanzu Application Platform package to.
export IMGPKG_REGISTRY_HOSTNAME_1=$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaw
s.com
export IMGPKG_REGISTRY_USERNAME_1=AWS
export IMGPKG_REGISTRY_PASSWORD_1=`aws ecr get-login-password --region $AWS_REG
ION`
# These environment variables starting with IMGPKG_* are used by the imgpkg com
mand only.

# The registry from which the Tanzu Application Platform package is retrieved.
export INSTALL_REGISTRY_HOSTNAME=$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.
com
export TAP_VERSION=VERSION-NUMBER
export INSTALL_REPO=tap-images
```

Where:

- `MY-AWS-ACCOUNT-ID` is the account ID you deploy Tanzu Application Platform in. No dashes and must be in the format `012345678901`.
- `MY-TANZUNET-USERNAME` is the user with access to the images in the VMware Tanzu Network registry `registry.tanzu.vmware.com`
- `MY-TANZUNET-PASSWORD` is the password for `MY-TANZUNET-USERNAME`.
- `TARGET-AWS-REGION` is the region you deploy the Tanzu Application Platform to.
- `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.9.1`

2. Install the Carvel tool imgpkg CLI.
3. Relocate the images with the `imgpkg` CLI by running:

```
imgpkg copy --concurrency 1 -b registry.tanzu.vmware.com/tanzu-application-plat
form/tap-packages:${TAP_VERSION} --to-repo ${INSTALL_REGISTRY_HOSTNAME}/${INSTA
LL_REPO}
```

Add the Tanzu Application Platform package repository

Tanzu CLI packages are available on repositories. Adding the Tanzu Application Platform package repository makes Tanzu Application Platform and its packages available for installation.

Relocate images to a registry is strongly recommended but not required for installation. If you skip this step, you can use the following values to replace the corresponding variables:

- `INSTALL_REGISTRY_HOSTNAME` is `registry.tanzu.vmware.com`
- `INSTALL_REPO` is `tanzu-application-platform`
- `INSTALL_REGISTRY_USERNAME` and `INSTALL_REGISTRY_PASSWORD` are the credentials to the VMware Tanzu Network registry `registry.tanzu.vmware.com`
- `TAP_VERSION` is your Tanzu Application Platform version. For example, `1.9.1`

To add the Tanzu Application Platform package repository to your cluster:

1. Create a namespace called `tap-install` for deploying any component packages by running:

```
kubectl create ns tap-install
```

This namespace keeps the objects grouped together logically.

2. (Optional) If you haven't relocated the images to ECR, create a secret to your registry by running:

```
tanzu secret registry add tap-registry \
  --username ${INSTALL_REGISTRY_USERNAME} --password ${INSTALL_REGISTRY_PASSWOR
D} \
  --server ${INSTALL_REGISTRY_HOSTNAME} \
  --export-to-all-namespaces --yes --namespace tap-install
```

3. Add the Tanzu Application Platform package repository to the cluster by running:

```
tanzu package repository add tanzu-tap-repository \
  --url ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}:${TAP_VERSION} \
  --namespace tap-install
```

4. Get the status of the Tanzu Application Platform package repository, and ensure the status updates to `Reconcile succeeded` by running:

```
tanzu package repository get tanzu-tap-repository --namespace tap-install
```

For example:

```
$ tanzu package repository get tanzu-tap-repository --namespace tap-install
- Retrieving repository tap...
NAME:          tanzu-tap-repository
VERSION:       16253001
REPOSITORY:    123456789012.dkr.ecr.us-west-2.amazonaws.com/tap-images
TAG:           1.9.1
STATUS:        Reconcile succeeded
REASON:
```

5. List the available packages by running:

```
tanzu package available list --namespace tap-install
```

For example:

```
$ tanzu package available list --namespace tap-install
  NAME                                                DISPLAY-NAME
  accelerator.apps.tanzu.vmware.com                   Application Accelerator
for VMware Tanzu
  amr-observer.apps.tanzu.vmware.com                  Supply Chain Security To
ols - AMR Observer
  api-portal.tanzu.vmware.com                         API portal
  apis.apps.tanzu.vmware.com                          API Auto Registration fo
r VMware Tanzu
  apiserver.appliveview.tanzu.vmware.com              Application Live View Ap
iServer for VMware Tanzu
  app-scanning.apps.tanzu.vmware.com                  SCST - Scan 2.0
  application-configuration-service.tanzu.vmware.com  Application Configuratio
n Service
  backend.appliveview.tanzu.vmware.com                Application Live View fo
r VMware Tanzu
  base-jammy-builder-lite.buildpacks.tanzu.vmware.com  base-jammy-builder-lite
  base-jammy-stack-lite.buildpacks.tanzu.vmware.com   base-jammy-stack
  bitnami.services.tanzu.vmware.com                   bitnami-services
  buildservice.tanzu.vmware.com                       Tanzu Build Service
  carbonblack.scanning.apps.tanzu.vmware.com          VMware Carbon Black for
Supply Chain Security Tools - Scan
  cartographer.conventions.apps.tanzu.vmware.com      Convention Service for V
Mware Tanzu
  cartographer.tanzu.vmware.com                       Cartographer
  cnrs.tanzu.vmware.com                               Cloud Native Runtimes
  connector.appliveview.tanzu.vmware.com              Application Live View Co
nnector for VMware Tanzu
  controller.source.apps.tanzu.vmware.com             Tanzu Source Controller
  conventions.appliveview.tanzu.vmware.com            Application Live View Co
nventions for VMware Tanzu
  crossplane.tanzu.vmware.com                         crossplane
  developer-conventions.tanzu.vmware.com              Tanzu App Platform Devel
oper Conventions
  dotnet-core-lite.buildpacks.tanzu.vmware.com        dotnet-core-lite
  external-secrets.apps.tanzu.vmware.com              External Secrets Operato
r
  fluxcd.source.controller.tanzu.vmware.com           Flux Source Controller
  go-lite.buildpacks.tanzu.vmware.com                 go-lite
  grype.scanning.apps.tanzu.vmware.com                Grype for Supply Chain S
ecurity Tools - Scan
  java-lite.buildpacks.tanzu.vmware.com               java-lite
  java-native-image-lite.buildpacks.tanzu.vmware.com  java-native-image-lite
  local-source-proxy.apps.tanzu.vmware.com            Local Source Proxy
  metadata-store.apps.tanzu.vmware.com                Supply Chain Security To
```

```
ols - Store
  namespace-provisioner.apps.tanzu.vmware.com          Namespace Provisioner
  nodejs-lite.buildpacks.tanzu.vmware.com              nodejs-lite
  ootb-delivery-basic.tanzu.vmware.com                 Tanzu App Platform Out o
f The Box Delivery Basic
  ootb-supply-chain-basic.tanzu.vmware.com             Tanzu App Platform Out o
f The Box Supply Chain Basic
  ootb-supply-chain-testing-scanning.tanzu.vmware.com  Tanzu App Platform Out o
f The Box Supply Chain with Testing and Scanning
  ootb-supply-chain-testing.tanzu.vmware.com           Tanzu App Platform Out o
f The Box Supply Chain with Testing
  ootb-templates.tanzu.vmware.com                      Tanzu App Platform Out o
f The Box Templates
  policy.apps.tanzu.vmware.com                         Supply Chain Security To
ols - Policy Controller
  python-lite.buildpacks.tanzu.vmware.com              python-lite
  ruby-lite.buildpacks.tanzu.vmware.com                ruby-lite
  scanning.apps.tanzu.vmware.com                       Supply Chain Security To
ols - Scan
  service-bindings.labs.vmware.com                     Service Bindings for Kub
ernetes
  services-toolkit.tanzu.vmware.com                    Services Toolkit
  snyk.scanning.apps.tanzu.vmware.com                  Snyk for Supply Chain Se
curity Tools - Scan
  spring-boot-conventions.tanzu.vmware.com             Tanzu Spring Boot Conven
tions Server
  spring-cloud-gateway.tanzu.vmware.com                Spring Cloud Gateway
  sso.apps.tanzu.vmware.com                            AppSSO
  tap-auth.tanzu.vmware.com                            Default roles for Tanzu
Application Platform
  tap-gui.tanzu.vmware.com                             Tanzu Developer Portal
  tap-telemetry.tanzu.vmware.com                       Telemetry Collector for
Tanzu Application Platform
  tap.tanzu.vmware.com                                 Tanzu Application Platfo
rm
  tekton.tanzu.vmware.com                              Tekton Pipelines
  tpb.tanzu.vmware.com                                 Tanzu Portal Builder
  web-servers-lite.buildpacks.tanzu.vmware.com         web-servers-lite
```

Install your Tanzu Application Platform profile

The `tap.tanzu.vmware.com` package installs predefined sets of packages based on your profile settings. This is done by using the package manager installed by Tanzu Cluster Essentials.

For more information about profiles, see Components and installation profiles.

To prepare to install a profile:

1. List version information for the package by running:

   ```
   tanzu package available list tap.tanzu.vmware.com --namespace tap-install
   ```

2. Create a `tap-values.yaml` file by using the Full Profile (AWS), which contains the minimum configurations required to deploy Tanzu Application Platform on AWS. The sample values file contains the necessary defaults for:

   - The meta-package, or parent Tanzu Application Platform package.
   - Subordinate packages, or individual child packages.

   Keep the values file for future configuration use.

   Note

> `tap-values.yaml` is set as a Kubernetes secret, which provides secure means to read credentials for Tanzu Application Platform components.

3. View possible configuration settings for your package

Full profile (AWS)

The following command generates the YAML file sample for the full-profile on AWS by using the ECR repositories you created earlier. The `profile:` field takes `full` as the default value, but you can also set it to `iterate`, `build`, `run`, or `view`. Refer to Install multicluster Tanzu Application Platform profiles for more information.

```
cat << EOF > tap-values.yaml
shared:
  ingress_domain: "INGRESS-DOMAIN"

ceip_policy_disclosed: true

# The above keys are minimum numbers of entries needed in tap-values.yaml to get a fun
ctioning TAP Full profile installation.

# Below are the keys which may have default values set, but can be overridden.

profile: full # Can take iterate, build, run, view.

supply_chain: basic # Can take testing, testing_scanning.

ootb_supply_chain_basic: # Based on supply_chain set above, can be changed to ootb_sup
ply_chain_testing, ootb_supply_chain_testing_scanning.
  registry:
    server: ${AWS_ACCOUNT_ID}.dkr.ecr.${AWS_REGION}.amazonaws.com
    # The prefix of the ECR repository.  Workloads will need
    # two repositories created:
    #
    # tanzu-application-platform/<workloadname>-<namespace>
    # tanzu-application-platform/<workloadname>-<namespace>-bundle
    repository: tanzu-application-platform

contour:
  envoy:
    service:
      type: LoadBalancer # This is set by default, but can be overridden by setting a
different value.

buildservice:
  kp_default_repository: ${AWS_ACCOUNT_ID}.dkr.ecr.${AWS_REGION}.amazonaws.com/tap-bui
ld-service
  # Enable the build service k8s service account to bind to the AWS IAM Role
  kp_default_repository_aws_iam_role_arn: "arn:aws:iam::${AWS_ACCOUNT_ID}:role/tap-bui
ld-service"

local_source_proxy:
  # Takes the value from the project_path under the image_registry section of shared b
y default, but can be overridden by setting a different value.
  repository: "EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE"
  push_secret:
    # When set to true, the secret mentioned in this section is automatically exported
to Local Source Proxy's namespace.
    name: "EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE-SECRET"
    namespace: "EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE-SECRET-NAMESPACE"
    # When set to true, the secret mentioned in this section is automatically exported
to Local Source Proxy's namespace.
    create_export: true
```

```
ootb_templates:
  # Enable the config writer service to use cloud based iaas authentication
  # which are retrieved from the developer namespace service account by
  # default
  iaas_auth: true

tap_gui:
  app_config:
    auth:
      allowGuestAccess: true  # This allows unauthenticated users to log in to your po
rtal. If you want to deactivate it, make sure you configure an alternative auth provid
er.
    catalog:
      locations:
        - type: url
          target: https://GIT-CATALOG-URL/catalog-info.yaml

metadata_store:
  ns_for_export_app_cert: "MY-DEV-NAMESPACE" # Verify this namespace is available with
in your cluster before initiating the Tanzu Application Platform installation.
  app_service_type: ClusterIP # Defaults to LoadBalancer. If shared.ingress_domain is
set earlier, this must be set to ClusterIP.

namespace_provisioner:
  aws_iam_role_arn: "arn:aws:iam::${AWS_ACCOUNT_ID}:role/tap-workload"

tap_telemetry:
  customer_entitlement_account_number: "CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER" # (Option
al) Identify data for creating Tanzu Application Platform usage reports.
EOF
```

Where:

- `INGRESS-DOMAIN` is the subdomain for the host name that you point at the `tanzu-shared-ingress` service's External IP address.
- `kp_default_repository_aws_iam_role_arn` is the ARN that was created to write to the ECR repository for the build service. This value is generated by the script, but you can modify it manually.
- `namspace_provisioner.aws_iam_role_arn` is the ARN that was created to write to the ECR repository for workloads. This value is generated by the script, but you can modify it manually.
- `EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE` is where the developer's local source is uploaded when using Tanzu CLI to use Local Source Proxy for workload creation.

  If an AWS ECR registry is being used, ensure that the repository already exists. AWS ECR expects the repository path to already exist. This destination is represented as `REGISTRY-SERVER/REPOSITORY-PATH`. For more information, see Install Local Source Proxy.
- `EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE-SECRET` is the name of the secret with credentials that allow pushing to the `EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE` repository.
- `EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE-SECRET-NAMESPACE` is the namespace in which `EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE-SECRET` is available.
- `GIT-CATALOG-URL` is the path to the `catalog-info.yaml` catalog definition file. You can download either a blank or populated catalog file from the Tanzu Application Platform product page. Otherwise, you can use a Backstage-compliant catalog you've already built and posted on the Git infrastructure.
- `MY-DEV-NAMESPACE` is the name of the developer namespace. SCST - Store exports secrets to the namespace, and SCST - Scan deploys the `ScanTemplates` there. This allows the

scanning feature to run in this namespace. If there are multiple developer namespaces, use `ns_for_export_app_cert: "*"` to export the SCST - Store CA certificate to all namespaces.

- `CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER` (optional) refers to the Entitlement Account Number (EAN), which is a unique identifier VMware assigns to its customers. Tanzu Application Platform telemetry uses this number to identify data that belongs to a particular customers and prepare usage reports.

For AWS, the default settings creates a classic LoadBalancer. To use the Network LoadBalancer instead of the classic LoadBalancer for ingress, add the following to your `tap-values.yaml`:

```
contour:
  infrastructure_provider: aws
  envoy:
    service:
      aws:
        LBType: nlb
```

(Optional) Additional Build Service configurations

The following tasks are optional during the Tanzu Application Platform installation process:

- (Optional) Configure your profile with full dependencies
- (Optional) Configure your profile with the Jammy stack only

(Optional) Configure your profile with full dependencies

When you install a profile that includes Tanzu Build Service, Tanzu Application Platform is installed with the `lite` set of dependencies. These dependencies consist of buildpacks and stacks required for application builds.

The `lite` set of dependencies do not contain all buildpacks and stacks. To use all buildpacks and stacks, you must install the `full` dependencies. For more information about the differences between `lite` and `full` dependencies, see About lite and full dependencies.

To configure `full` dependencies, add the key-value pair `exclude_dependencies: true` to your `tap-values.yaml` file under the `buildservice` section. For example:

```
buildservice:
  kp_default_repository: ${AWS_ACCOUNT_ID}.dkr.ecr.${AWS_REGION}.amazonaws.com/tap-bui
ld-service
  exclude_dependencies: true
```

After configuring `full` dependencies, you must install the dependencies after you have finished installing your Tanzu Application Platform package. See Install the full dependencies package for more information.

Tanzu Application Platform v1.9.1 supports building applications with Ubuntu v22.04 (Jammy).

Install your Tanzu Application Platform package

Follow these steps to install the Tanzu Application Platform package:

1. Install the package by running:

   ```
   tanzu package install tap -p tap.tanzu.vmware.com -v ${TAP_VERSION} --values-fi
   le tap-values.yaml -n tap-install
   ```

2. Verify the package install by running:

```
tanzu package installed get tap -n tap-install
```

This can take 5-10 minutes because it installs several packages on your cluster.

3. Verify that the necessary packages in the profile are installed by running:

```
tanzu package installed list -A
```

4. If you configured `full` dependencies in your `tbs-values.yaml` file, install the `full` dependencies by following the procedure in Install full dependencies.

After installing the Full profile on your cluster, you can install the Tanzu Developer Tools for VS Code Extension to help you develop against it. For instructions, see Install Tanzu Developer Tools for your VS Code.

Note

You can run the following command after reconfiguring the profile to reinstall the Tanzu Application Platform:

```
tanzu package installed update tap -p tap.tanzu.vmware.com -v ${TAP_VERSION}  --values
-file tap-values.yaml -n tap-install
```

Install the full dependencies package

If you configured `full` dependencies in your `tap-values.yaml` file in Configure your profile with full dependencies earlier, you must install the `full` dependencies package.

1. Create an ECR repository for Tanzu Build Service full dependencies by running:

```
aws ecr create-repository --repository-name full-deps --region ${AWS_REGION}
```

2. (Optional) If you have an existing installation of the full dependencies package from a version earlier than Tanzu Application Platform v1.9.1, you must uninstall the full dependencies package and remove the package repository:

    1. Uninstall the package:

```
tanzu package installed delete full-tbs-deps -n tap-install
```

    2. Remove the package repository:

```
tanzu package repository delete tbs-full-deps-repository -n tap-install
```

Important

The package and repository names might differ depending on your installation configurations.

3. Get the latest version of the Tanzu Application Platform package by running:

```
tanzu package available list tap.tanzu.vmware.com --namespace tap-install
```

4. If you have not done so already, you must exclude the default dependencies by adding the key-value pair `exclude_dependencies: true` to your `tap-values.yaml` file under the

`buildservice` section. For example:

```
buildservice:
  exclude_dependencies: true
```

5. If you have not updated your Tanzu Application Platform package installation after adding the key-value pair `exclude_dependencies: true` to your values file, perform the update by running:

   ```
   tanzu package installed update tap --namespace tap-install --values-file VALUES
   -FILE
   ```

   Where `VALUES-FILE` is the path to the `tap-values.yaml` file you edited earlier.

6. Relocate the Tanzu Build Service `full` dependencies package repository by doing one of the following:

   - Relocate the images directly for online installation:

     ```
     imgpkg copy \
       -b registry.tanzu.vmware.com/tanzu-application-platform/full-deps-packa
     ge-repo:VERSION \
       --to-repo ${INSTALL_REGISTRY_HOSTNAME}/full-deps-package-repo
     ```

     Where `VERSION` is the version of the Tanzu Application Platform package you retrieved earlier.

   - Relocate the images to an external storage device and then to the registry in the air-gapped environment:

     ```
     imgpkg copy \
       -b registry.tanzu.vmware.com/tanzu-application-platform/full-deps-packa
     ge-repo:VERSION \
       --to-tar=full-deps-package-repo.tar

     # move full-deps-package-repo.tar to environment with registry access
     imgpkg copy \
       --tar full-deps-package-repo.tar \
       --to-repo=INSTALL-REGISTRY-HOSTNAME/TARGET-REPOSITORY/full-deps-package
     -repo
     ```

     Where:

     - `VERSION` is the version of the Tanzu Application Platform package you retrieved earlier.
     - `INSTALL-REGISTRY-HOSTNAME` is your container registry.
     - `TARGET-REPOSITORY` is your target repository.

7. Add the Tanzu Build Service `full` dependencies package repository by running:

   ```
   tanzu package repository add full-deps-package-repo \
     --url INSTALL-REGISTRY-HOSTNAME/TARGET-REPOSITORY/full-deps-package-repo:VERS
   ION \
     --namespace tap-install
   ```

   Where:

   - `INSTALL-REGISTRY-HOSTNAME` is your container registry.
   - `TARGET-REPOSITORY` is your target repository.

- `VERSION` is the version of the Tanzu Application Platform package you retrieved earlier.

8. Create a new `tbs-full-deps-values.yaml` and copy the `kp_default_repository` key-value pair from your `tap-values.yaml` or `tbs-values.yaml`:

   ```
   ---
    kp_default_repository: "REPO-NAME"
    kp_default_repository_secret:
      name: kp-default-repository-creds
      namespace: tap-install
   ```

   Where `REPO-NAME` is copied from the `buildservice.kp_default_repository` field in your `tap-values.yaml` or `tbs-values.yaml`.

   1. (Optional) Install the UBI builder.

      The UBI builder uses Red Hat Universal Base Image (UBI) v8 for both build and run images. This builder only supports Java and Node.js. To install the UBI builder, add the key-value pair `enable_ubi_builder: true` to your `tbs-full-deps-values.yaml`.

      ```
      ---
      enable_ubi_builder: true
      ```

   2. (Optional) Install the Static builder.

      The Static builder uses Ubuntu Jammy for both build images and a minimal static run image. This builder only supports Golang. To install the Static builder, add the key-value pair `enable_static_builder: true` to your `tbs-full-deps-values.yaml`.

      ```
      ---
      enable_static_builder: true
      ```

9. Install the `full` dependencies package by running:

   ```
   tanzu package install full-deps \
     --package full-deps.buildservice.tanzu.vmware.com \
     --version "> 0.0.0" \
     --namespace tap-install \
     --values-file VALUES-FILE
   ```

   Where `VALUES-FILE` is the path to the `tbs-full-deps-values.yaml` you created earlier.

For more information about the differences between `lite` and `full` dependencies, see About lite and full dependencies.

Access Tanzu Developer Portal

To access Tanzu Developer Portal, you can use the host name that you configured earlier. This host name is pointed at the shared ingress. To configure LoadBalancer for Tanzu Developer Portal, see Access Tanzu Developer Portal.

You're now ready to start using Tanzu Developer Portal. Proceed to the Getting Started topic or the Tanzu Developer Portal - Catalog Operations topic.

Exclude packages from a Tanzu Application Platform profile

To exclude packages from a Tanzu Application Platform profile:

1. Find the full subordinate (child) package name:

```
tanzu package available list --namespace tap-install
```

2. Update your `tap-values` file with a section listing the exclusions:

```
profile: PROFILE-VALUE
excluded_packages:
  - tap-gui.tanzu.vmware.com
  - service-bindings.lab.vmware.com
```

Important

If you exclude a package after performing a profile installation including that package, you cannot see the accurate package states immediately after running `tap package installed list -n tap-install`. Also, you can break package dependencies by removing a package. Allow 20 minutes to verify that all packages have reconciled correctly while troubleshooting.

Next steps

- (Optional) Install Individual Packages
- Set up developer namespaces to use your installed packages

View possible configuration settings for your package

To view possible configuration settings for a package, run:

```
tanzu package available get tap.tanzu.vmware.com/$TAP_VERSION --values-schema --namesp
ace tap-install
```

Note

The `tap.tanzu.vmware.com` package does not show all configuration settings for packages it plans to install. The package only shows top-level keys. You can view individual package configuration settings with the same `tanzu package available get` command. For example, to find the keys for Cloud Native Runtimes, you must first identify the version of the package with `tanzu package installed list -n tap-install`, which lists all the installed packages versions. Then run the command `tanzu package available get -n tap-install cnrs.tanzu.vmware.com/CNR-VERSION --values-schema` by using the package version listed for Cloud Native Runtimes.

```
profile: full

# Shared configurations go under the shared key.
shared:
  ingress_domain: tap.example.com

# ...

# For example, Cloud Native Runtimes specific values go under its name.
cnrs:
  provider: local
```

```
# For example, App Accelerator specific values go under its name.
accelerator:
  server:
    service_type: "ClusterIP"
```

Shared Keys define values that configure multiple packages. These keys are defined under the `shared` Top-level Key, as summarized in the following table:

Shared Key	Description	Optional
`ca_cert_data`	PEM-encoded certificate data to trust TLS connections with a private CA. This shared key is used by `convention_controller`, `scanning` and the Tanzu `source_controller` (not the Flux CD Source Controller).	Yes
`ingress_domain`	Domain name to be used in service routes and host names for instances of Tanzu Application Platform components.	Yes
`ingress_issuer`	A `cert-manager.io/v1/ClusterIssuer` for issuing TLS certificates to Tanzu Application Platform components. Default value: `tap-ingress-selfsigned`	Yes
`kubernetes_distribution`	Type of Kubernetes infrastructure being used. You can use this shared key in coordination with the `kubernetes_version` key. Supported value: `openshift`.	Yes
`kubernetes_version`	Kubernetes version. You can use this shared key independently or in coordination with the `kubernetes_distribution` key. Supported value: `1.24.x`, where `x` stands for the Kubernetes patch version.	Yes
`image_registry.project_path`	Project path in the container image registry server used for builder and application images.	Yes
`image_registry.username`	User name for the container image registry. Mutually exclusive with `shared.image_registry.secret.name/namespace`	Yes
`image_registry.password`	Password for the container image registry. Mutually exclusive with `shared.image_registry.secret.name/namespace`	Yes
`secret.name`	Secret name for the container image registry credentials of type `kubernetes.io/dockerconfigjson`. Mutually exclusive with `shared.image_registry.username/password`	Yes
`secret.namespace`	Secret namespace for the container image registry credentials. Mutually exclusive with `shared.image_registry.username/password`	Yes
`activateAppLiveViewSecureAccessControl`	Enable secure access connection between Application Live View components.	Yes

The following table summarizes the top-level keys used for package-specific configuration within your `tap-values.yaml`.

Package	Top-level Key
See table above.	`shared`
API Auto Registration	`api_auto_registration`
API portal	`api_portal`
Application Accelerator	`accelerator`
Application Live View	`appliveview`
Application Live View connector	`appliveview_connector`
Application Live View conventions	`appliveview-conventions`
Cartographer Conventions	`cartographer_conventions`

Package	Top-level Key
Cartographer	`cartographer`
Cloud Native Runtimes	`cnrs`
Source Controller	`source_controller`
Supply Chain	`supply_chain`
Supply Chain Basic	`ootb_supply_chain_basic`
Supply Chain Testing	`ootb_supply_chain_testing`
Supply Chain Testing Scanning	`ootb_supply_chain_testing_scanning`
Supply Chain Security Tools - Scan	`scanning`
Supply Chain Security Tools - Scan (Grype Scanner)	`grype`
Supply Chain Security Tools - Store	`metadata_store`
Build Service	`buildservice`
Tanzu Developer Portal	`tap_gui`

For information about package-specific configuration, see Install individual packages.

Install individual packages

You can install Tanzu Application Platform (commonly known as TAP) through predefined profiles or through individual packages. Use this topic to learn how to install each individual package. For more information about installing through profiles, see Components and installation profiles.

Installing individual Tanzu Application Platform packages is useful if you do not want to use a profile to install packages or if you want to install additional packages after installing a profile. Before installing the packages, be sure to complete the prerequisites, configure and verify the cluster, accept the EULA, and install the Tanzu CLI with any required plug-ins. For more information, see Prerequisites.

Install pages for individual Tanzu Application Platform packages

- Install API Auto Registration
- Install API portal
- Install Application Accelerator
- Install Application Configuration Service
- Install Application Live View
- Install Application Single Sign-On
- Install Bitnami Services
- Install cert-manager
- Install Cloud Native Runtimes
- Install Contour
- Install Crossplane
- Install default roles for Tanzu Application Platform

- Install Developer Conventions
- Install Flux CD Source Controller
- Install Out of the Box Templates
- Install Out of the Box Supply Chain with Testing
- Install Out of the Box Supply Chain with Testing and Scanning
- Install Service Bindings
- Install Services Toolkit
- Install Source Controller
- Install Spring Boot conventions
- Install Supply Chain Choreographer
- Install Supply Chain Security Tools - Store
- Install Supply Chain Security Tools - Policy Controller
- Install Supply Chain Security Tools - Scan
- Install Tanzu Developer Portal
- Install Tanzu Build Service
- Install Tekton
- Install Telemetry

Verify the installed packages

Use the following procedure to verify that the packages are installed.

1. List the installed packages by running:

   ```
   tanzu package installed list --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package installed list --namespace tap-install
   \ Retrieving installed packages...
   NAME                     PACKAGE-NAME                                      PAC
   KAGE-VERSION  STATUS
   api-portal               api-portal.tanzu.vmware.com                       1.
   0.3           Reconcile succeeded
   app-accelerator          accelerator.apps.tanzu.vmware.com                 1.
   0.0           Reconcile succeeded
   app-live-view            appliveview.tanzu.vmware.com                      1.
   0.2           Reconcile succeeded
   appliveview-conventions  build.appliveview.tanzu.vmware.com                1.
   0.2           Reconcile succeeded
   cartographer             cartographer.tanzu.vmware.com                     0.
   1.0           Reconcile succeeded
   cloud-native-runtimes    cnrs.tanzu.vmware.com                             1.
   0.3           Reconcile succeeded
   convention-controller    controller.conventions.apps.tanzu.vmware.com      0.
   7.0           Reconcile succeeded
   developer-conventions    developer-conventions.tanzu.vmware.com            0.
   3.0-build.1   Reconcile succeeded
   grype-scanner            grype.scanning.apps.tanzu.vmware.com              1.
   0.0           Reconcile succeeded
   image-policy-webhook     image-policy-webhook.signing.apps.tanzu.vmware.com 1.
   1.2           Reconcile succeeded
   ```

```
metadata-store           metadata-store.apps.tanzu.vmware.com               1.
0.2            Reconcile succeeded
ootb-supply-chain-basic  ootb-supply-chain-basic.tanzu.vmware.com          0.
5.1            Reconcile succeeded
ootb-templates           ootb-templates.tanzu.vmware.com                   0.
5.1            Reconcile succeeded
scan-controller          scanning.apps.tanzu.vmware.com                    1.
0.0            Reconcile succeeded
service-bindings         service-bindings.labs.vmware.com                  0.
5.0            Reconcile succeeded
services-toolkit         services-toolkit.tanzu.vmware.com                 0.
8.0            Reconcile succeeded
source-controller        controller.source.apps.tanzu.vmware.com           0.
2.0            Reconcile succeeded
sso4k8s-install          sso.apps.tanzu.vmware.com                         1.
0.0-beta.2-31  Reconcile succeeded
tap-gui                  tap-gui.tanzu.vmware.com                          0.
3.0-rc.4       Reconcile succeeded
tekton-pipelines         tekton.tanzu.vmware.com                           0.3
0.0           Reconcile succeeded
tbs                      buildservice.tanzu.vmware.com                     1.
5.0            Reconcile succeeded
```

Next steps

- Set up developer namespaces to use your installed packages

Set up developer namespaces to use your installed packages

This topic tells you how to set up developer namespaces to use your installed packages.

You must first enable your namespace for Tanzu Application Platform Supply Chains and then optionally apply Kubernetes RBAC for additional non-admin user access. For more information, see:

1. Enable namespace for Supply Chains.
2. (Optional) Enable non-admin users access with Kubernetes RBAC.

Enable namespace for Supply Chains

The `default` Service Account in your developer namespace requires permission to your image repositories and RoleBindings for Tanzu Application Platform `ClusterRoles`. The Namespace Provisioner watches namespaces for a specific label, then takes action to enable the namespace.

To enable the Supply Chain on your namespace, run:

```
kubectl label namespace $YOUR-NAMESPACE apps.tanzu.vmware.com/tap-ns=""
```

Where `YOUR-NAMESPACE` is your developer namespace.

(Optional) Enable non-admin users access with Kubernetes RBAC

Follow these steps to enable non-admin users by using Kubernetes RBAC to submit jobs to the Supply Chain:

1. Choose either of the following options to give developers namespace-level access and view access to appropriate cluster-level resources:

- **Option 1:** Use the Tanzu Application Platform RBAC CLI plug-in (beta).

  To use the `tanzu rbac` plug-in to grant `app-viewer` and `app-editor` roles to an identity provider group, run:

  ```
  tanzu rbac binding add -g GROUP-FOR-APP-VIEWER -n YOUR-NAMESPACE -r app-v
  iewer
  tanzu rbac binding add -g GROUP-FOR-APP-EDITOR -n YOUR-NAMESPACE -r app-e
  ditor
  ```

  Where:

  - `YOUR-NAMESPACE` is the name you give to the developer namespace.
  - `GROUP-FOR-APP-VIEWER` is the user group from the upstream identity provider that requires access to `app-viewer` resources on the current namespace and cluster.
  - `GROUP-FOR-APP-EDITOR` is the user group from the upstream identity provider that requires access to `app-editor` resources on the current namespace and cluster.

  For more information about `tanzu rbac`, see Bind a user or group to a default role.

  VMware recommends creating a user group in your identity provider's grouping system for each developer namespace and then adding the users accordingly.

  Depending on your identity provider, you might need to take further action to federate user groups appropriately with your cluster. For an example of how to set up Azure Active Directory (AD) with your cluster, see Integrating Azure Active Directory.

- **Option 2:** Use the native Kubernetes YAML.

  To apply the RBAC policy, run:

  ```
  cat <<EOF | kubectl -n YOUR-NAMESPACE apply -f -
  apiVersion: rbac.authorization.k8s.io/v1
  kind: RoleBinding
  metadata:
    name: dev-permit-app-viewer
  roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: ClusterRole
    name: app-viewer
  subjects:
    - kind: Group
      name: GROUP-FOR-APP-VIEWER
      apiGroup: rbac.authorization.k8s.io
  ---
  apiVersion: rbac.authorization.k8s.io/v1
  kind: ClusterRoleBinding
  metadata:
    name: YOUR-NAMESPACE-permit-app-viewer
  roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: ClusterRole
    name: app-viewer-cluster-access
  subjects:
    - kind: Group
      name: GROUP-FOR-APP-VIEWER
      apiGroup: rbac.authorization.k8s.io
  ---
  apiVersion: rbac.authorization.k8s.io/v1
  kind: RoleBinding
  ```

```
metadata:
  name: dev-permit-app-editor
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: app-editor
subjects:
  - kind: Group
    name: GROUP-FOR-APP-EDITOR
    apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: YOUR-NAMESPACE-permit-app-editor
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: app-editor-cluster-access
subjects:
  - kind: Group
    name: GROUP-FOR-APP-EDITOR
    apiGroup: rbac.authorization.k8s.io
EOF
```

Where:

- `YOUR-NAMESPACE` is the name you give to the developer namespace.
- `GROUP-FOR-APP-VIEWER` is the user group from the upstream identity provider that requires access to `app-viewer` resources on the current namespace and cluster.
- `GROUP-FOR-APP-EDITOR` is the user group from the upstream identity provider that requires access to `app-editor` resources on the current namespace and cluster.

VMware recommends creating a user group in your identity provider's grouping system for each developer namespace and then adding the users accordingly.

Depending on your identity provider, you might need to take further action to federate user groups appropriately with your cluster. For an example of how to set up Azure Active Directory (AD) with your cluster, see Integrating Azure Active Directory.

Rather than granting roles directly to individuals, VMware recommends using your identity provider's user groups system to grant access to a group of developers. For an example of how to set up Azure AD with your cluster, see Integrating Azure Active Directory.

2. (Optional) Log in as a non-admin user, such as a developer, to see the effects of RBAC after the bindings are applied.

Next steps

- Install Tanzu Developer Tools for your VS Code

Install Tanzu Developer Tools for your VS Code

This topic tells you how to install VMware Tanzu Developer Tools for Visual Studio Code (VS Code).

Prerequisites

Before installing the extension, you must have:

- VS Code
- kubectl
- Tilt v0.30.12 or later
- Tanzu CLI and plug-ins
- A cluster with the Tanzu Application Platform Full profile or Iterate profile

If you are an app developer, someone else in your organization might have already set up the Tanzu Application Platform environment.

Docker Desktop and local Kubernetes are not prerequisites for using Tanzu Developer Tools for VS Code.

Install

To install VMware Tanzu Developer Tools for VS Code:

1. Open Visual Studio Code.
2. Open the command palette.
3. In the search box enter `Extension`.
4. Click **Extensions: Install Extensions**.
5. The **Extensions** view opens on the left side of your screen. In the search box enter `Tanzu`.
6. Click **Tanzu Developer Tools** and then click **Install**.

Configure

To configure VMware Tanzu Developer Tools for VS Code:

1. Ensure that you are targeting the correct cluster. For more information, see the Kubernetes documentation.
2. Go to **Code** > **Preferences** > **Settings** > **Extensions** > **Tanzu Developer Tools** and set the following:
    - **Confirm Apply Config**: This controls whether the extension asks to confirm user input when applying a workload.
    - **Confirm Debug Port**: This controls whether the extension asks for the debug port when running `Tanzu: Start Java Debug`.
    - **Confirm Local Port**: This controls whether the extension asks for the local port when running `Tanzu: Start Java Debug`.
    - **Confirm Delete**: This controls whether the extension asks for confirmation when deleting a workload.
    - **Enable Live Hover**: This enables Live Hover. For more information, see Integrating Live Hover by using Spring Boot Tools. Restart VS Code for this change to take effect.
    - **Source Image**: The registry location for publishing local source code. For example, `registry.io/yourapp-source`. This must include both a registry and a project name. A source image registry location is optional when Local Source Proxy is configured.
    - **Local Path**: (Optional) The path on the local file system to a directory of source code to build. This is the current directory by default.

- **Namespace**: (Optional) This is the namespace that workloads are deployed into. The namespace set in kubeconfig is the default.
- **Tracked Namespaces**: (Optional) Comma-separated list of namespaces. Resources in these namespaces appear in the Tanzu Workloads panel and the Activity panel. If empty, the namespace in the current context is the default.
- **Wait Timeout**: (Optional) This sets how long to wait for a workload to become ready.
- **Workload Type**: (Optional) This distinguishes the workload type. Examples include web and server.

Uninstall

To uninstall VMware Tanzu Developer Tools for VS Code:

1. Go to **Code** > **Preferences** > **Settings** > **Extensions**.
2. Right-click the extension and then click **Uninstall**.

Next steps

Proceed to Getting started with Tanzu Developer Tools for Visual Studio Code.

Install Tanzu Application Platform (Azure)

To install Tanzu Application Platform (commonly known as TAP) on Azure:

Step	Task	Link
1.	Review the prerequisites to ensure that you have set up everything required before beginning the installation	Prerequisites
2.	Accept Tanzu Application Platform EULAs and install the Tanzu CLI	Accept Tanzu Application Platform EULAs and installing the Tanzu CLI
3.	Create Azure Resources	Create Azure Resources
4.	Install Cluster Essentials for Tanzu	Deploy Cluster Essentials
5.	Add the Tanzu Application Platform package repository, prepare your Tanzu Application Platform profile, and install the profile to the cluster	Install the Tanzu Application Platform package and profiles
6.	(Optional) Install any additional packages that were not in the profile	Install individual packages
7.	Set up developer namespaces to use your installed packages	Set up developer namespaces to use your installed packages
8.	Install developer tools into your integrated development environment (IDE)	Install Tanzu Developer Tools for your VS Code

After installing Tanzu Application Platform on to your Kubernetes clusters, proceed with Get started with Tanzu Application Platform.

Install Tanzu Application Platform (Azure)

To install Tanzu Application Platform (commonly known as TAP) on Azure:

Step	Task	Link
1.	Review the prerequisites to ensure that you have set up everything required before beginning the installation	Prerequisites

Step	Task	Link
2.	Accept Tanzu Application Platform EULAs and install the Tanzu CLI	Accept Tanzu Application Platform EULAs and installing the Tanzu CLI
3.	Create Azure Resources	Create Azure Resources
4.	Install Cluster Essentials for Tanzu	Deploy Cluster Essentials
5.	Add the Tanzu Application Platform package repository, prepare your Tanzu Application Platform profile, and install the profile to the cluster	Install the Tanzu Application Platform package and profiles
6.	(Optional) Install any additional packages that were not in the profile	Install individual packages
7.	Set up developer namespaces to use your installed packages	Set up developer namespaces to use your installed packages
8.	Install developer tools into your integrated development environment (IDE)	Install Tanzu Developer Tools for your VS Code

After installing Tanzu Application Platform on to your Kubernetes clusters, proceed with Get started with Tanzu Application Platform.

Create Azure Resources for Tanzu Application Platform

To install Tanzu Application Platform (commonly known as TAP) within the Azure ecosystem, you must create several Azure resources. Use this topic to learn how to create:

- An Azure Kubernetes Service (AKS) cluster to install Tanzu Application Platform.
- ACR repositories for the Tanzu Application Platform container images.

Creating these resources enables Tanzu Application Platform to use an IAM role bound to a Kubernetes service account for authentication, rather than the typical username and password stored in a Kubernetes secret strategy.

This is important when using ACR because authenticating to ACR is a two-step process:

1. Retrieve a token using your Azure credentials.
2. Use the token to authenticate to the registry.

To increase security, the token has a lifetime of 12 hours. This makes storing it as a secret for a service impractical because it must be refreshed every 12 hours.

Using an IAM role on a service account mitigates the need to retrieve the token because it is handled by credential helpers within the services.

Prerequisites

Before installing Tanzu Application Platform on Azure, you need:

- **An Azure subscription:**

  You must create all of your resources within an Azure subscription and create an Azure free account.

- **Azure CLI:**

  To run CLI reference commands locally, you must install the Azure CLI. This topic uses Azure CLI to both query and configure resources in Azure, such as IAM roles. For more information, see Azure CLI documentation.

Create Azure Resource Group

1. Log in to Azure.

```
az login
az account set --subscription SUBSCRIPTION-NAME
```

2. Create a resource group with the `az group create` command.

```
az group create --name myTAPResourceGroup --location eastus
```

Create an AKS cluster

To create an AKS cluster, you can run the az aks create command with the `--enable-addons monitoring` and `--enable-msi-auth-for-monitoring` parameter to enable Azure Monitor Container insights with managed identity authentication (preview).

The following example creates a cluster named `tap-on-azure` with one node and enables a system-assigned managed identity:

```
az aks create -g myTAPResourceGroup -n tap-on-azure --enable-managed-identity --node-c
ount 6 --enable-addons monitoring --enable-msi-auth-for-monitoring --generate-ssh-keys
--node-vm-size Standard_D4ds_v4 --kubernetes-version K8S-VERSION
```

Where `K8S-VERSION` is the compatible Kubernetes version that can be retrieved by running `az aks get-versions`.

Note

You might need to increase quota for Standard DDSv4 Family vCPUs. For more information, see the Azure documentation.

After a few minutes, the command completes and returns JSON-formatted information about the cluster.

When you create an AKS cluster, a second resource group is automatically created to store the AKS resources. For more information, see Why are two resource groups created with AKS?

Connect to the AKS cluster

To manage a Kubernetes cluster, use the Kubernetes command-line client, kubectl. `kubectl` is already installed if you use Azure Cloud Shell.

1. Install `kubectl` locally by using the az aks install-cli command:

```
az aks install-cli
```

2. Configure `kubectl` to connect to your Kubernetes cluster by using the az aks get-credentials command that:
    - Downloads credentials and configures the Kubernetes CLI to use them.
    - Uses `~/.kube/config`, the default location for the Kubernetes configuration file. You can specify a different location for your Kubernetes configuration file by using the `--file` argument.

```
az aks get-credentials --resource-group myTAPResourceGroup --name tap-on-azure
```

Create the container repositories

Create the Azure Container Registry by running:

```
az acr create -n $REGISTRY_NAME -g myTAPResourceGroup --sku Standard
```

Note

Azure Container Registry (ACR) does not require that the container repositories are already created. Repositories are created automatically when images are uploaded.

Enable registry admin account

To enable push and pull to your registries, you must enable the admin user account, which is created with each registry. Run the following command to enable the admin user account:

```
az acr update -n $REGISTRY_NAME --admin-enabled true
```

There are two passwords created for each admin user account per registry. To retrieve the passwords, run the following for each registry:

```
az acr credential show --name $REGISTRY_NAME --resource-group myTAPResourceGroup
```

Expect to see the following outputs:

```
{
  "passwords": [
    {
      "name": "password",
      "value": YOUR-PASSWORD
    },
    {
      "name": "password2",
      "value": YOUR-PASSWORD-2
    }
  ],
  "username": ""
}
```

Export the username and password by running:

```
export KP_REGISTRY_USERNAME=$REGISTRY_NAME
export KP_REGISTRY_PASSWORD=YOUR-PASSWORD
```

Next steps

- Install Tanzu Application Platform package and profiles on Azure

Install Tanzu Application Platform package and profiles on Azure

This topic tells you how to install Tanzu Application Platform (commonly known as TAP) packages from your Tanzu Application Platform package repository on to Azure.

Before installing the packages, ensure you have:

- Completed the Prerequisites.
- Created Azure Resources.
- Accepted Tanzu Application Platform EULA and installed Tanzu CLI with any required plug-ins.
- Installed Cluster Essentials for Tanzu.

Relocate images to a registry

VMware recommends relocating the images from VMware Tanzu Network registry to your own container image registry before attempting installation. If you don't relocate the images, Tanzu Application Platform depends on VMware Tanzu Network for continued operation, and VMware Tanzu Network offers no uptime guarantees. The option to skip relocation is documented for evaluation and proof-of-concept only.

To relocate images from the VMware Tanzu Network registry to the ACR registry:

1. Set up environment variables for installation use by running:

   ```
   # Set tanzunet as the source registry to copy the Tanzu Application Platform pa
   ckages from.
   export IMGPKG_REGISTRY_HOSTNAME_0=registry.tanzu.vmware.com
   export IMGPKG_REGISTRY_USERNAME_0=MY-TANZUNET-USERNAME
   export IMGPKG_REGISTRY_PASSWORD_0=MY-TANZUNET-PASSWORD

   # The user's registry for copying the Tanzu Application Platform package to.
   export IMGPKG_REGISTRY_HOSTNAME_1=$INSTALL_REGISTRY_HOSTNAME
   export IMGPKG_REGISTRY_USERNAME_1=$REGISTRY_NAME
   export IMGPKG_REGISTRY_PASSWORD_1=REGISTRY-PASSWORD
   # These environment variables starting with IMGPKG_* are used by the imgpkg com
   mand only.

   # The registry from which the Tanzu Application Platform package is retrieved.
   export INSTALL_REGISTRY_HOSTNAME=$REGISTRY_NAME.azurecr.io
   export TAP_VERSION=VERSION-NUMBER
   export INSTALL_REPO=tapimages
   ```

   Where:

   - `MY-TANZUNET-USERNAME` is the user with access to the images in the VMware Tanzu Network registry `registry.tanzu.vmware.com`
   - `MY-TANZUNET-PASSWORD` is the password for `MY-TANZUNET-USERNAME`.
   - `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.9.1`

2. Install the Carvel tool imgpkg CLI.
3. Relocate the images with the `imgpkg` CLI by running:

   ```
   imgpkg copy --concurrency 1 -b ${IMGPKG_REGISTRY_HOSTNAME_0}/tanzu-application-
   platform/tap-packages:${TAP_VERSION} --to-repo ${IMGPKG_REGISTRY_HOSTNAME_1}/
   ${INSTALL_REPO}
   ```

4. Create a namespace called `tap-install` for deploying any component packages by running:

   ```
   kubectl create ns tap-install
   ```

   This namespace keeps the objects grouped together logically.

5. Create registry secret for the VMware Tanzu Network registry by running:

```
tanzu secret registry add tap-registry \
--username ${INSTALL_REGISTRY_USERNAME} --password ${INSTALL_REGISTRY_PASSWORD}
\
--server ${INSTALL_REGISTRY_HOSTNAME} \
--export-to-all-namespaces --yes --namespace tap-install
tanzu secret registry list --namespace tap-install
```

6. Add the Tanzu Application Platform package repository to the cluster by running:

```
tanzu package repository add tanzu-tap-repository \
  --url ${INSTALL_REGISTRY_HOSTNAME}/full-deps-package-repo
  --namespace tap-install
```

7. Get the status of the Tanzu Application Platform package repository, and ensure the status updates to `Reconcile succeeded` by running:

```
tanzu package repository get tanzu-tap-repository --namespace tap-install
```

For example:

```
$ tanzu package repository get tanzu-tap-repository --namespace tap-install
- Retrieving repository tap...
NAME:          tanzu-tap-repository
VERSION:       16253001
REPOSITORY:    123456789012.dkr.acr.us-east.azure.com/tap-images
TAG:           1.9.1
STATUS:        Reconcile succeeded
REASON:
```

Note

The `VERSION` and `TAG` numbers differ from the earlier example if you are on Tanzu Application Platform v1.0.2 or earlier.

8. List the available packages by running:

```
tanzu package available list --namespace tap-install
```

For example:

```
$ tanzu package available list --namespace tap-install
/ Retrieving available packages...
  NAME                                                DISPLAY-NAME
SHORT-DESCRIPTION
  accelerator.apps.tanzu.vmware.com                   Application Accelerator
for VMware Tanzu                                    Used to create new projects a
nd configurations.
  api-portal.tanzu.vmware.com                         API portal
A unified user interface for API discovery and exploration at scale.
  apis.apps.tanzu.vmware.com                          API Auto Registration fo
r VMware Tanzu                                      A TAP component to automatica
lly register API exposing workloads as API entities

in TAP GUI.
  backend.appliveview.tanzu.vmware.com                Application Live View fo
r VMware Tanzu                                      App for monitoring and troubl
eshooting running apps
  buildservice.tanzu.vmware.com                       Tanzu Build Service
Tanzu Build Service enables the building and automation of containerized
```

```
software workflows securely and at scale.
  carbonblack.scanning.apps.tanzu.vmware.com           VMware Carbon Black for
Supply Chain Security Tools - Scan                Default scan templates using
VMware Carbon Black
  cartographer.tanzu.vmware.com                        Cartographer
Kubernetes native Supply Chain Choreographer.
  cnrs.tanzu.vmware.com                                Cloud Native Runtimes
Cloud Native Runtimes is a serverless runtime based on Knative
  connector.appliveview.tanzu.vmware.com               Application Live View Co
nnector for VMware Tanzu                          App for discovering and regis
tering running apps
  controller.source.apps.tanzu.vmware.com              Tanzu Source Controller
Tanzu Source Controller enables workload create/update from source code.
  conventions.appliveview.tanzu.vmware.com             Application Live View Co
nventions for VMware Tanzu                        Application Live View convent
ion server
  developer-conventions.tanzu.vmware.com               Tanzu App Platform Devel
oper Conventions                                  Developer Conventions
  external-secrets.apps.tanzu.vmware.com               External Secrets Operato
r                                                 External Secrets Operator is
a Kubernetes operator that integrates external

secret management systems.
  fluxcd.source.controller.tanzu.vmware.com            Flux Source Controller
The source-controller is a Kubernetes operator, specialised in artifacts

acquisition from external sources such as Git, Helm repositories and S3 bucket
s.
  grype.scanning.apps.tanzu.vmware.com                 Grype for Supply Chain S
ecurity Tools - Scan                              Default scan templates using
Anchore Grype
  metadata-store.apps.tanzu.vmware.com                 Supply Chain Security To
ols - Store                                       Post SBoMs and query for imag
e, package, and vulnerability metadata.
  namespace-provisioner.apps.tanzu.vmware.com          Namespace Provisioner
Automatic Provisioning of Developer Namespaces.
  ootb-delivery-basic.tanzu.vmware.com                 Tanzu App Platform Out o
f The Box Delivery Basic                          Out of The Box Delivery Basi
c.
  ootb-supply-chain-basic.tanzu.vmware.com             Tanzu App Platform Out o
f The Box Supply Chain Basic                      Out of The Box Supply Chain B
asic.
  ootb-supply-chain-testing-scanning.tanzu.vmware.com  Tanzu App Platform Out o
f The Box Supply Chain with Testing and Scanning  Out of The Box Supply Chain w
ith Testing and Scanning.
  ootb-supply-chain-testing.tanzu.vmware.com           Tanzu App Platform Out o
f The Box Supply Chain with Testing               Out of The Box Supply Chain w
ith Testing.
  ootb-templates.tanzu.vmware.com                      Tanzu App Platform Out o
f The Box Templates                               Out of The Box Templates.
  policy.apps.tanzu.vmware.com                         Supply Chain Security To
ols - Policy Controller                           Policy Controller enables def
ining of a policy to restrict unsigned container

images.
  scanning.apps.tanzu.vmware.com                       Supply Chain Security To
ols - Scan                                        Scan for vulnerabilities and
enforce policies directly within Kubernetes native

Supply Chains.
  service-bindings.labs.vmware.com                     Service Bindings for Kub
ernetes                                           Service Bindings for Kubernet
es implements the Service Binding Specification.
  services-toolkit.tanzu.vmware.com                    Services Toolkit
The Services Toolkit enables the management, lifecycle, discoverability and
```

```
connectivity of Service Resources (databases, message queues, DNS records,

etc.).
  snyk.scanning.apps.tanzu.vmware.com                  Snyk for Supply Chain Se
curity Tools - Scan                                 Default scan templates using
Snyk
  spring-boot-conventions.tanzu.vmware.com             Tanzu Spring Boot Conven
tions Server                                        Default Spring Boot conventio
n server.
  sso.apps.tanzu.vmware.com                            AppSSO
Application Single Sign-On for Tanzu
  tap-auth.tanzu.vmware.com                            Default roles for Tanzu
Application Platform                                Default roles for Tanzu Appli
cation Platform
  tap-gui.tanzu.vmware.com                             Tanzu Developer Portal
web app graphical user interface for Tanzu Application Platform
  tap-telemetry.tanzu.vmware.com                       Telemetry Collector for
Tanzu Application Platform                          Tanzu Application Platform Te
lemetry
  tap.tanzu.vmware.com                                 Tanzu Application Platfo
rm                                                  Package to install a set of T
AP components to get you started based on your use

case.
  tekton.tanzu.vmware.com                              Tekton Pipelines
Tekton Pipelines is a framework for creating CI/CD systems.
```

Install your Tanzu Application Platform profile

The `tap.tanzu.vmware.com` package installs predefined sets of packages based on your profile settings by using the package manager installed by Tanzu Cluster Essentials. For more information about profiles, see Components and installation profiles.

Some components use `kpack` and require a repository to publish artifacts to. This registry does not have to be the same registry used for installing the Tanzu Application Platform packages. To create a registry secret and add it to a developer namespace, run:

```
export KP_REGISTRY_USERNAME=YOUR-USERNAME
export KP_REGISTRY_PASSWORD=YOUR-PASSWORD
export KP_REGISTRY_HOSTNAME=YOUR-HOSTNAME

echo $KP_REGISTRY_USERNAME
echo $KP_REGISTRY_PASSWORD
echo $KP_REGISTRY_HOSTNAME

docker login $KP_REGISTRY_HOSTNAME -u $KP_REGISTRY_USERNAME -p $KP_REGISTRY_PASSWORD

export YOUR_NAMESPACE=mydev-ns

echo $YOUR_NAMESPACE

kubectl create ns $YOUR_NAMESPACE

tanzu secret registry add registry-credentials --server $KP_REGISTRY_HOSTNAME --userna
me $KP_REGISTRY_USERNAME --password $KP_REGISTRY_PASSWORD --namespace $YOUR_NAMESPACE

kubectl get secret registry-credentials  -o jsonpath='{.data.\.dockerconfigjson}'  -n
$YOUR_NAMESPACE| base64 --decode
```

To prepare to install a profile:

1. List version information for the package by running:

```
tanzu package available list tap.tanzu.vmware.com --namespace tap-install
```

2. Create a `tap-values.yaml` file by using the Full Profile (Azure), which contains the minimum configurations required to deploy Tanzu Application Platform on Azure. The sample values file contains the necessary defaults for:
    - The meta-package, or parent Tanzu Application Platform package.
    - Subordinate packages, or individual child packages.

    Keep the values file for future configuration use.

    > **Note**
    >
    > `tap-values.yaml` is set as a Kubernetes secret, which provides secure means to read credentials for the Tanzu Application Platform components.

3. View possible configuration settings for your package

Full profile (Azure)

The following is the YAML file sample for the full-profile on Azure by using the ACR repositories you created earlier. The `profile:` field takes `full` as the default value, but you can also set it to `iterate`, `build`, `run`, or `view`. See Install multicluster Tanzu Application Platform profiles for more information.

```
cat << EOF > tap-values.yaml
shared:
  ingress_domain: YOUR_DOMAIN
  ingress_issuer: CLUSTER_ISSUER # use "" to disable SSL

  image_registry:
    project_path: ${KP_REGISTRY_HOSTNAME}/tap
    secret:
      name: registry-credentials
      namespace: ${YOUR_NAMESPACE}

ceip_policy_disclosed: true
profile: full # Can take iterate, build, run, view.

supply_chain: basic # Can take testing, testing_scanning.

ootb_templates:
  iaas_auth: true

ootb_supply_chain_basic:
  registry: # (Optional) Takes the value from the project_path under the shared.image_
registry section by default, but you can override it by setting different values.
    server: ${KP_REGISTRY_HOSTNAME}
    repository: tap-apps
  gitops:
    credentials_secret: ""

contour:
  envoy:
    service:
      type: LoadBalancer

buildservice:
  # (Optional) Takes the value from the project_path under the shared.image_registry s
ection by default, but you can override it by setting a different value.
  kp_default_repository: ${KP_REGISTRY_HOSTNAME}/buildservice
```

```
  kp_default_repository_secret:
    name: registry-credentials
    namespace: ${YOUR_NAMESPACE}

local_source_proxy:
  # (Optional) Takes the value from the project_path under the shared.image_registry s
ection by default, but you can override it by setting a different value.
  repository: "EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE"
  push_secret:
    name: "EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE-SECRET"
    namespace: "EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE-SECRET-NAMESPACE"
    create_export: true  # When set to true, the secret mentioned in this section is a
utomatically exported to Local Source Proxy's namespace.

ootb_delivery_basic:
  service_account: default

tap_gui:
  app_config:
    auth:
      allowGuestAccess: true

metadata_store:
  app_service_type: ClusterIP
  ns_for_export_app_cert: ${YOUR_NAMESPACE}

accelerator:
  server:
    service_type: "ClusterIP"

EOF
```

Where:

- `YOUR_DOMAIN` is the subdomain for the host name that you point at the `tanzu-shared-ingress` service's External IP address.

  Note

  You can update this field when knowing the external IP address of the ingress after the initial install of Tanzu Application Platform.

- `CLUSTER_ISSUER` is the name of the deployed `ClusterIssuer` to enable TLS on your ingress domain. You can disable TLS by using an empty string `""`.
- `YOUR_NAMESPACE` is the environment variable you defined earlier to describe the name of the developer namespace. Supply Chain Security Tools - Store exports secrets to the namespace, and Supply Chain Security Tools - Scan deploys the `ScanTemplates` there. This allows the scanning feature to run in this namespace. If there are multiple developer namespaces, use `ns_for_export_app_cert: "*"` to export the Supply Chain Security Tools - Store CA certificate to all namespaces.
- `EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE` is where the developer's local source is uploaded when using Tanzu CLI to use Local Source Proxy for workload creation.
- `EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE-SECRET` is the name of the secret with credentials that allow pushing to the `EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE` repository.
- `EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE-SECRET-NAMESPACE` is the namespace in which `EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE-SECRET` is available.

Important

Installing Grype by using `tap-values.yaml` as follows is deprecated in v1.6 and will be removed in v1.8:

```
grype:
  targetImagePullSecret: registry-credentials
```

You can install Grype by using Namespace Provisioner instead.

(Optional) Configure your profile with full dependencies

When you install a profile that includes Tanzu Build Service, Tanzu Application Platform is installed with the `lite` set of dependencies. These dependencies consist of buildpacks and stacks required for application builds.

The `lite` set of dependencies do not contain all buildpacks and stacks. To use all buildpacks and stacks, you must install the `full` dependencies. For more information about the differences between `lite` and `full` dependencies, see About lite and full dependencies.

To configure `full` dependencies, add the key-value pair `exclude_dependencies: true` to your `tap-values.yaml` file under the `buildservice` section. For example:

```
buildservice:
  kp_default_repository: "KP-DEFAULT-REPO"
  kp_default_repository_secret: # Takes the value from the shared section by default,
but can be overridden by setting a different value.
    name: "KP-DEFAULT-REPO-SECRET"
    namespace: "KP-DEFAULT-REPO-SECRET-NAMESPACE"
  exclude_dependencies: true
```

After configuring `full` dependencies, you must install the dependencies after you have finished installing your Tanzu Application Platform package. See Install the full dependencies package for more information.

Tanzu Application Platform v1.6.1 supports building applications with Ubuntu v22.04 (Jammy).

Install your Tanzu Application Platform package

Follow these steps to install the Tanzu Application Platform package:

1. Install the package by running:

   ```
   tanzu package install tap -p tap.tanzu.vmware.com -v $TAP_VERSION --values-file
   tap-values.yaml -n tap-install
   ```

2. Verify the package install by running:

   ```
   tanzu package installed get tap -n tap-install
   ```

   This can take 5-10 minutes because it installs several packages on your cluster.

3. Verify that the necessary packages in the profile are installed by running:

   ```
   tanzu package installed list -A
   ```

4. If you configured `full` dependencies in your `tbs-values.yaml` file, install the `full` dependencies by following the procedure in Install full dependencies.

After installing the Full profile on your cluster, you can install the Tanzu Developer Tools for VS Code Extension to help you develop against it. For more information, see Install Tanzu Developer Tools for your VS Code.

Install the full dependencies package

If you configured `full` dependencies in your `tap-values.yaml` file in Configure your profile with full dependencies earlier, you must install the `full` dependencies package.

For more information about the differences between `lite` and `full` dependencies, see About lite and full dependencies.

To install the `full` dependencies package:

1. Get the latest version of the Tanzu Application Platform package by running:

   ```
   tanzu package available list tap.tanzu.vmware.com --namespace tap-install
   ```

2. If you have not done so already, you must exclude the default dependencies by adding the key-value pair `exclude_dependencies: true` to your `tap-values.yaml` file under the `buildservice` section. For example:

   ```
   buildservice:
     exclude_dependencies: true
   ```

3. If you have not updated your Tanzu Application Platform package installation after adding the key-value pair `exclude_dependencies: true` to your values file, perform the update by running:

   ```
   tanzu package installed update tap --namespace tap-install --values-file VALUES
   -FILE
   ```

   Where `VALUES-FILE` is the path to the `tap-values.yaml` file you edited earlier.

4. Relocate the Tanzu Build Service `full` dependencies package repository by doing one of the following:

   - Relocate the images directly for online installation:

     ```
     imgpkg copy \
       -b registry.tanzu.vmware.com/tanzu-application-platform/full-deps-packa
     ge-repo:VERSION \
       --to-repo ${INSTALL_REGISTRY_HOSTNAME}/full-deps-package-repo
     ```

     Where `VERSION` is the version of the Tanzu Application Platform package you retrieved earlier.

   - Relocate the images to an external storage device and then to the registry in the air-gapped environment:

     ```
     imgpkg copy \
       -b registry.tanzu.vmware.com/tanzu-application-platform/full-deps-packa
     ge-repo:VERSION \
       --to-tar=full-deps-package-repo.tar

     # move full-deps-package-repo.tar to environment with registry access
     imgpkg copy \
       --tar full-deps-package-repo.tar \
       --to-repo=INSTALL-REGISTRY-HOSTNAME/TARGET-REPOSITORY/full-deps-package
     -repo
     ```

Where:

- `VERSION` is the version of the Tanzu Application Platform package you retrieved earlier.
- `INSTALL-REGISTRY-HOSTNAME` is your container registry.
- `TARGET-REPOSITORY` is your target repository.

5. Add the Tanzu Build Service `full` dependencies package repository by running:

```
tanzu package repository add full-deps-package-repo \
  --url INSTALL-REGISTRY-HOSTNAME/TARGET-REPOSITORY/full-deps-package-repo:VERS
ION \
  --namespace tap-install
```

Where:

- `INSTALL-REGISTRY-HOSTNAME` is your container registry.
- `TARGET-REPOSITORY` is your target repository.
- `VERSION` is the version of the Tanzu Application Platform package you retrieved earlier.

6. Create a new `tbs-full-deps-values.yaml` and copy the `kp_default_repository` key-value pair from your `tap-values.yaml` or `tbs-values.yaml`:

```
---
 kp_default_repository: "REPO-NAME"
 kp_default_repository_secret:
   name: kp-default-repository-creds
   namespace: tap-install
```

Where `REPO-NAME` is copied from the `buildservice.kp_default_repository` field in your `tap-values.yaml` or `tbs-values.yaml`.

1. (Optional) Install the UBI builder.

   The UBI builder uses Red Hat Universal Base Image (UBI) v8 for both build and run images. This builder only supports Java and Node.js. To install the UBI builder, add the key-value pair `enable_ubi_builder: true` to your `tbs-full-deps-values.yaml`.

   ```
   ---
   enable_ubi_builder: true
   ```

2. (Optional) Install the Static builder.

   The Static builder uses Ubuntu Jammy for both build images and a minimal static run image. This builder only supports Golang. To install the Static builder, add the key-value pair `enable_static_builder: true` to your `tbs-full-deps-values.yaml`.

   ```
   ---
   enable_static_builder: true
   ```

7. Install the `full` dependencies package by running:

```
tanzu package install full-deps \
  --package full-deps.buildservice.tanzu.vmware.com \
  --version "> 0.0.0" \
  --namespace tap-install \
  --values-file VALUES-FILE
```

Where `VALUES-FILE` is the path to the `tbs-full-deps-values.yaml` you created earlier.

Access Tanzu Developer Portal

You can use the host name configured in Access Tanzu Developer Portal to access Tanzu Developer Portal. This host name is pointed at the shared ingress.

You're now ready to start using Tanzu Developer Portal. Proceed to the Getting Started topic or the Tanzu Developer Portal - Catalog Operations topic.

Next steps

- (Optional) Install Individual Packages
- Set up developer namespaces to use your installed packages

View possible configuration settings for your package

To view possible configuration settings for a package, run:

```
tanzu package available get tap.tanzu.vmware.com/$TAP_VERSION --values-schema --namesp
ace tap-install
```

Note

The `tap.tanzu.vmware.com` package does not show all configuration settings for packages it plans to install. The package only shows top-level keys. You can view individual package configuration settings with the same `tanzu package available get` command. For example, to find the keys for Cloud Native Runtimes, you must first identify the version of the package with `tanzu package installed list -n tap-install`, which lists all the installed packages versions. Then run the command `tanzu package available get -n tap-install cnrs.tanzu.vmware.com/CNR-VERSION --values-schema` by using the package version listed for Cloud Native Runtimes.

```
profile: full

# Shared configurations go under the shared key.
shared:
  ingress_domain: tap.example.com

# ...

# For example, Cloud Native Runtimes specific values go under its name.
cnrs:
  provider: local

# For example, App Accelerator specific values go under its name.
accelerator:
  server:
    service_type: "ClusterIP"
```

Shared Keys define values that configure multiple packages. These keys are defined under the `shared` Top-level Key, as summarized in the following table:

Shared Key	Description	Optional
`ca_cert_data`	PEM-encoded certificate data to trust TLS connections with a private CA. This shared key is used by `convention_controller`, `scanning` and the Tanzu `source_controller` (not the Flux CD Source Controller).	Yes

Shared Key	Description	Optional
`ingress_domain`	Domain name to be used in service routes and host names for instances of Tanzu Application Platform components.	Yes
`ingress_issuer`	A `cert-manager.io/v1/ClusterIssuer` for issuing TLS certificates to Tanzu Application Platform components. Default value: `tap-ingress-selfsigned`	Yes
`kubernetes_distribution`	Type of Kubernetes infrastructure being used. You can use this shared key in coordination with the `kubernetes_version` key. Supported value: `openshift`.	Yes
`kubernetes_version`	Kubernetes version. You can use this shared key independently or in coordination with the `kubernetes_distribution` key. Supported value: `1.24.x`, where `x` stands for the Kubernetes patch version.	Yes
`image_registry.project_path`	Project path in the container image registry server used for builder and application images.	Yes
`image_registry.username`	User name for the container image registry. Mutually exclusive with `shared.image_registry.secret.name/namespace`	Yes
`image_registry.password`	Password for the container image registry. Mutually exclusive with `shared.image_registry.secret.name/namespace`	Yes
`secret.name`	Secret name for the container image registry credentials of type `kubernetes.io/dockerconfigjson`. Mutually exclusive with `shared.image_registry.username/password`	Yes
`secret.namespace`	Secret namespace for the container image registry credentials. Mutually exclusive with `shared.image_registry.username/password`	Yes
`activateAppLiveViewSecureAccessControl`	Enable secure access connection between Application Live View components.	Yes

The following table summarizes the top-level keys used for package-specific configuration within your `tap-values.yaml`.

Package	Top-level Key
See table above.	`shared`
API Auto Registration	`api_auto_registration`
API portal	`api_portal`
Application Accelerator	`accelerator`
Application Live View	`appliveview`
Application Live View connector	`appliveview_connector`
Application Live View conventions	`appliveview-conventions`
Cartographer Conventions	`cartographer_conventions`
Cartographer	`cartographer`
Cloud Native Runtimes	`cnrs`
Source Controller	`source_controller`
Supply Chain	`supply_chain`
Supply Chain Basic	`ootb_supply_chain_basic`
Supply Chain Testing	`ootb_supply_chain_testing`
Supply Chain Testing Scanning	`ootb_supply_chain_testing_scanning`

Package	Top-level Key
Supply Chain Security Tools - Scan	`scanning`
Supply Chain Security Tools - Scan (Grype Scanner)	`grype`
Supply Chain Security Tools - Store	`metadata_store`
Build Service	`buildservice`
Tanzu Developer Portal	`tap_gui`

For information about package-specific configuration, see Install individual packages.

Install individual packages

You can install Tanzu Application Platform (commonly known as TAP) through predefined profiles or through individual packages. Use this topic to learn how to install each individual package. For more information about installing through profiles, see Components and installation profiles.

Installing individual Tanzu Application Platform packages is useful if you do not want to use a profile to install packages or if you want to install additional packages after installing a profile. Before installing the packages, be sure to complete the prerequisites, configure and verify the cluster, accept the EULA, and install the Tanzu CLI with any required plug-ins. For more information, see Prerequisites.

Install pages for individual Tanzu Application Platform packages

- Install API Auto Registration
- Install API portal
- Install Application Accelerator
- Install Application Configuration Service
- Install Application Live View
- Install Application Single Sign-On
- Install Bitnami Services
- Install cert-manager
- Install Cloud Native Runtimes
- Install Contour
- Install Crossplane
- Install default roles for Tanzu Application Platform
- Install Developer Conventions
- Install Flux CD Source Controller
- Install Out of the Box Templates
- Install Out of the Box Supply Chain with Testing
- Install Out of the Box Supply Chain with Testing and Scanning
- Install Service Bindings
- Install Services Toolkit

- Install Source Controller
- Install Spring Boot conventions
- Install Supply Chain Choreographer
- Install Supply Chain Security Tools - Store
- Install Supply Chain Security Tools - Policy Controller
- Install Supply Chain Security Tools - Scan
- Install Tanzu Developer Portal
- Install Tanzu Build Service
- Install Tekton
- Install Telemetry

Verify the installed packages

Use the following procedure to verify that the packages are installed.

1. List the installed packages by running:

   ```
   tanzu package installed list --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package installed list --namespace tap-install
   \ Retrieving installed packages...
   NAME                     PACKAGE-NAME                                        PAC
   KAGE-VERSION  STATUS
   api-portal               api-portal.tanzu.vmware.com                         1.
   0.3           Reconcile succeeded
   app-accelerator          accelerator.apps.tanzu.vmware.com                   1.
   0.0           Reconcile succeeded
   app-live-view            appliveview.tanzu.vmware.com                        1.
   0.2           Reconcile succeeded
   appliveview-conventions  build.appliveview.tanzu.vmware.com                  1.
   0.2           Reconcile succeeded
   cartographer             cartographer.tanzu.vmware.com                       0.
   1.0           Reconcile succeeded
   cloud-native-runtimes    cnrs.tanzu.vmware.com                               1.
   0.3           Reconcile succeeded
   convention-controller    controller.conventions.apps.tanzu.vmware.com        0.
   7.0           Reconcile succeeded
   developer-conventions    developer-conventions.tanzu.vmware.com              0.
   3.0-build.1   Reconcile succeeded
   grype-scanner            grype.scanning.apps.tanzu.vmware.com                1.
   0.0           Reconcile succeeded
   image-policy-webhook     image-policy-webhook.signing.apps.tanzu.vmware.com  1.
   1.2           Reconcile succeeded
   metadata-store           metadata-store.apps.tanzu.vmware.com                1.
   0.2           Reconcile succeeded
   ootb-supply-chain-basic  ootb-supply-chain-basic.tanzu.vmware.com            0.
   5.1           Reconcile succeeded
   ootb-templates           ootb-templates.tanzu.vmware.com                     0.
   5.1           Reconcile succeeded
   scan-controller          scanning.apps.tanzu.vmware.com                      1.
   0.0           Reconcile succeeded
   service-bindings         service-bindings.labs.vmware.com                    0.
   5.0           Reconcile succeeded
   services-toolkit         services-toolkit.tanzu.vmware.com                   0.
   8.0           Reconcile succeeded
   source-controller        controller.source.apps.tanzu.vmware.com             0.
   ```

```
2.0            Reconcile succeeded
sso4k8s-install          sso.apps.tanzu.vmware.com                          1.
0.0-beta.2-31  Reconcile succeeded
tap-gui                  tap-gui.tanzu.vmware.com                           0.
3.0-rc.4       Reconcile succeeded
tekton-pipelines         tekton.tanzu.vmware.com                            0.3
0.0           Reconcile succeeded
tbs                      buildservice.tanzu.vmware.com                      1.
5.0            Reconcile succeeded
```

Next steps

- Set up developer namespaces to use your installed packages

Set up developer namespaces to use your installed packages

This topic tells you how to set up developer namespaces by using the legacy manual process. For more information about how to automatically set up your developer namespaces, see Namespace Provisioner.

Additional configuration for testing and scanning

If you plan to install or have already installed Out of the Box Supply Chains with Testing and Scanning, you can use Namespace Provisioner to set up the required resources. For more information, see Customize installation in the Namespace Provisioner documentation for configuration steps.

Legacy namespace setup

You can choose either one of the following two approaches to create a `Workload` for your application by using the registry credentials specified, add credentials and Role-Based Access Control (RBAC) rules to the namespace that you plan to create the `Workload` in:

- Enable single user access.
- Enable additional users access with Kubernetes RBAC.

Enable single user access

Run the following command to add secrets, a service account to execute the supply chain, and RBAC rules to authorize the service account to the developer namespace:

```
cat <<EOF | kubectl -n YOUR-NAMESPACE apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: tap-registry
  annotations:
    secretgen.carvel.dev/image-pull-secret: ""
type: kubernetes.io/dockerconfigjson
data:
  .dockerconfigjson: e30K

---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: default
```

```
secrets:
  - name: registry-credentials
imagePullSecrets:
  - name: registry-credentials
  - name: tap-registry

---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: default
rules:
- apiGroups: [source.toolkit.fluxcd.io]
  resources: [gitrepositories]
  verbs: ['*']
- apiGroups: [source.apps.tanzu.vmware.com]
  resources: [imagerepositories]
  verbs: ['*']
- apiGroups: [carto.run]
  resources: [deliverables, runnables]
  verbs: ['*']
- apiGroups: [kpack.io]
  resources: [images]
  verbs: ['*']
- apiGroups: [conventions.apps.tanzu.vmware.com]
  resources: [podintents]
  verbs: ['*']
- apiGroups: [""]
  resources: ['configmaps']
  verbs: ['*']
- apiGroups: [""]
  resources: ['pods']
  verbs: ['list']
- apiGroups: [tekton.dev]
  resources: [taskruns, pipelineruns]
  verbs: ['*']
- apiGroups: [tekton.dev]
  resources: [pipelines]
  verbs: ['list']
- apiGroups: [kappctrl.k14s.io]
  resources: [apps]
  verbs: ['*']
- apiGroups: [serving.knative.dev]
  resources: ['services']
  verbs: ['*']
- apiGroups: [servicebinding.io]
  resources: ['servicebindings']
  verbs: ['*']
- apiGroups: [services.apps.tanzu.vmware.com]
  resources: ['resourceclaims']
  verbs: ['*']
- apiGroups: [scanning.apps.tanzu.vmware.com]
  resources: ['imagescans', 'sourcescans']
  verbs: ['*']

---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: default
subjects:
```

```
  - kind: ServiceAccount
    name: default
```

Where `YOUR-NAMESPACE` is your developer namespace.

Enable additional users access with Kubernetes RBAC

Follow these steps to enable additional users by using Kubernetes RBAC to submit jobs to the Supply Chain:

1. Enable single user access.
2. Choose either of the following options to give developers namespace-level access and view access to appropriate cluster-level resources:
    - **Option 1:** Use the Tanzu Application Platform RBAC CLI plug-in (beta).

      To use the `tanzu rbac` plug-in to grant `app-viewer` and `app-editor` roles to an identity provider group, run:

      ```
      tanzu rbac binding add -g GROUP-FOR-APP-VIEWER -n YOUR-NAMESPACE -r app-v
      iewer
      tanzu rbac binding add -g GROUP-FOR-APP-EDITOR -n YOUR-NAMESPACE -r app-e
      ditor
      ```

      Where:

        - `YOUR-NAMESPACE` is the name you give to the developer namespace.
        - `GROUP-FOR-APP-VIEWER` is the user group from the upstream identity provider that requires access to `app-viewer` resources on the current namespace and cluster.
        - `GROUP-FOR-APP-EDITOR` is the user group from the upstream identity provider that requires access to `app-editor` resources on the current namespace and cluster.

      For more information about `tanzu rbac`, see Bind a user or group to a default role.

      VMware recommends creating a user group in your identity provider's grouping system for each developer namespace and then adding the users accordingly.

      Depending on your identity provider, you might need to take further action to federate user groups appropriately with your cluster. For an example of how to set up Azure Active Directory (AD) with your cluster, see Integrating Azure Active Directory.

    - **Option 2:** Use the native Kubernetes YAML.

      To apply the RBAC policy, run:

      ```
      cat <<EOF | kubectl -n YOUR-NAMESPACE apply -f -
      apiVersion: rbac.authorization.k8s.io/v1
      kind: RoleBinding
      metadata:
        name: dev-permit-app-viewer
      roleRef:
        apiGroup: rbac.authorization.k8s.io
        kind: ClusterRole
        name: app-viewer
      subjects:
        - kind: Group
          name: GROUP-FOR-APP-VIEWER
          apiGroup: rbac.authorization.k8s.io
      ---
      ```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: YOUR-NAMESPACE-permit-app-viewer
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: app-viewer-cluster-access
subjects:
  - kind: Group
    name: GROUP-FOR-APP-VIEWER
    apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: dev-permit-app-editor
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: app-editor
subjects:
  - kind: Group
    name: GROUP-FOR-APP-EDITOR
    apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: YOUR-NAMESPACE-permit-app-editor
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: app-editor-cluster-access
subjects:
  - kind: Group
    name: GROUP-FOR-APP-EDITOR
    apiGroup: rbac.authorization.k8s.io
EOF
```

Where:

- `YOUR-NAMESPACE` is the name you give to the developer namespace.
- `GROUP-FOR-APP-VIEWER` is the user group from the upstream identity provider that requires access to `app-viewer` resources on the current namespace and cluster.
- `GROUP-FOR-APP-EDITOR` is the user group from the upstream identity provider that requires access to `app-editor` resources on the current namespace and cluster.

VMware recommends creating a user group in your identity provider's grouping system for each developer namespace and then adding the users accordingly.

Depending on your identity provider, you might need to take further action to federate user groups appropriately with your cluster. For an example of how to set up Azure Active Directory (AD) with your cluster, see Integrating Azure Active Directory.

Rather than granting roles directly to individuals, VMware recommends using your identity provider's user groups system to grant access to a group of developers. For an example of how to set up Azure AD with your cluster, see Integrating Azure Active Directory.

3. (Optional) Log in as a non-admin user, such as a developer, to see the effects of RBAC after the bindings are applied.

Next steps

- Install Tanzu Developer Tools for your VS Code

Install Tanzu Developer Tools for your VS Code

This topic tells you how to install VMware Tanzu Developer Tools for Visual Studio Code (VS Code).

Prerequisites

Before installing the extension, you must have:

- VS Code
- kubectl
- Tilt v0.30.12 or later
- Tanzu CLI and plug-ins
- A cluster with the Tanzu Application Platform Full profile or Iterate profile

If you are an app developer, someone else in your organization might have already set up the Tanzu Application Platform environment.

Docker Desktop and local Kubernetes are not prerequisites for using Tanzu Developer Tools for VS Code.

Install

To install VMware Tanzu Developer Tools for VS Code:

1. Open Visual Studio Code.
2. Open the command palette.
3. In the search box enter `Extension`.
4. Click **Extensions: Install Extensions**.
5. The **Extensions** view opens on the left side of your screen. In the search box enter `Tanzu`.
6. Click **Tanzu Developer Tools** and then click **Install**.

Configure

To configure VMware Tanzu Developer Tools for VS Code:

1. Ensure that you are targeting the correct cluster. For more information, see the Kubernetes documentation.
2. Go to **Code** > **Preferences** > **Settings** > **Extensions** > **Tanzu Developer Tools** and set the following:
    - **Confirm Apply Config**: This controls whether the extension asks to confirm user input when applying a workload.
    - **Confirm Debug Port**: This controls whether the extension asks for the debug port when running `Tanzu: Start Java Debug`.

- **Confirm Local Port**: This controls whether the extension asks for the local port when running `Tanzu: Start Java Debug`.
- **Confirm Delete**: This controls whether the extension asks for confirmation when deleting a workload.
- **Enable Live Hover**: This enables Live Hover. For more information, see Integrating Live Hover by using Spring Boot Tools. Restart VS Code for this change to take effect.
- **Source Image**: The registry location for publishing local source code. For example, `registry.io/yourapp-source`. This must include both a registry and a project name. A source image registry location is optional when Local Source Proxy is configured.
- **Local Path**: (Optional) The path on the local file system to a directory of source code to build. This is the current directory by default.
- **Namespace**: (Optional) This is the namespace that workloads are deployed into. The namespace set in kubeconfig is the default.
- **Tracked Namespaces**: (Optional) Comma-separated list of namespaces. Resources in these namespaces appear in the Tanzu Workloads panel and the Activity panel. If empty, the namespace in the current context is the default.
- **Wait Timeout**: (Optional) This sets how long to wait for a workload to become ready.
- **Workload Type**: (Optional) This distinguishes the workload type. Examples include web and server.

Uninstall

To uninstall VMware Tanzu Developer Tools for VS Code:

1. Go to **Code** > **Preferences** > **Settings** > **Extensions**.
2. Right-click the extension and then click **Uninstall**.

Next steps

Proceed to Getting started with Tanzu Developer Tools for Visual Studio Code.

Install Tanzu Application Platform (OpenShift)

To install Tanzu Application Platform (commonly known as TAP) on your OpenShift clusters with internet access:

Step	Task	Link
1.	Review the prerequisites to ensure you have met all requirements before installing.	Prerequisites
2.	Accept Tanzu Application Platform EULAs and install the Tanzu CLI.	Accept Tanzu Application Platform EULAs and installing the Tanzu CLI
3.	Install Cluster Essentials for Tanzu.	Deploy Cluster Essentials
4.	Add the Tanzu Application Platform package repository, prepare your Tanzu Application Platform profile, and install the profile to the cluster.	Install the Tanzu Application Platform package and profiles
5.	(Optional) Install any additional packages that were not in the profile.	Install individual packages

Step	Task	Link
6.	Set up developer namespaces to use your installed packages.	Set up developer namespaces to use your installed packages
7.	Install developer tools into your integrated development environment (IDE).	Install Tanzu Developer Tools for your VS Code

After installing Tanzu Application Platform on to your OpenShift clusters, proceed with Get started with Tanzu Application Platform.

Install Tanzu Application Platform (OpenShift)

To install Tanzu Application Platform (commonly known as TAP) on your OpenShift clusters with internet access:

Step	Task	Link
1.	Review the prerequisites to ensure you have met all requirements before installing.	Prerequisites
2.	Accept Tanzu Application Platform EULAs and install the Tanzu CLI.	Accept Tanzu Application Platform EULAs and installing the Tanzu CLI
3.	Install Cluster Essentials for Tanzu.	Deploy Cluster Essentials
4.	Add the Tanzu Application Platform package repository, prepare your Tanzu Application Platform profile, and install the profile to the cluster.	Install the Tanzu Application Platform package and profiles
5.	(Optional) Install any additional packages that were not in the profile.	Install individual packages
6.	Set up developer namespaces to use your installed packages.	Set up developer namespaces to use your installed packages
7.	Install developer tools into your integrated development environment (IDE).	Install Tanzu Developer Tools for your VS Code

After installing Tanzu Application Platform on to your OpenShift clusters, proceed with Get started with Tanzu Application Platform.

Install Tanzu Application Platform on your OpenShift clusters

This topic tells you how to install Tanzu Application Platform (commonly known as TAP) packages on your OpenShift clusters.

Before installing the packages, ensure you have:

- Completed the Prerequisites.
- Configured and verified the cluster.
- Accepted Tanzu Application Platform EULA and installed Tanzu CLI with any required plug-ins.

Relocate images to a registry

VMware recommends relocating the images from VMware Tanzu Network registry to your own container image registry before attempting installation. If you don't relocate the images, Tanzu Application Platform will depend on VMware Tanzu Network for continued operation, and VMware Tanzu Network offers no uptime guarantees. The option to skip relocation is documented for evaluation and proof-of-concept only.

The supported registries are Harbor, Azure Container Registry, Google Container Registry, and Quay.io. See the following documentation for a registry to learn how to set it up:

- Harbor documentation
- Google Container Registry documentation
- Quay.io documentation

To relocate images from the VMware Tanzu Network registry to your registry:

1. Set up environment variables for installation use by running:

   ```
   # Set tanzunet as the source registry to copy Tanzu Application Platform packag
   es from.
   export IMGPKG_REGISTRY_HOSTNAME_0=registry.tanzu.vmware.com
   export IMGPKG_REGISTRY_USERNAME_0=MY-TANZUNET-USERNAME
   export IMGPKG_REGISTRY_PASSWORD_0=MY-TANZUNET-PASSWORD

   # The user's registry for copying the Tanzu Application Platform package to.
   export IMGPKG_REGISTRY_HOSTNAME_1=MY-REGISTRY
   export IMGPKG_REGISTRY_USERNAME_1=MY-REGISTRY-USER
   export IMGPKG_REGISTRY_PASSWORD_1=MY-REGISTRY-PASSWORD
   # These environment variables starting with IMGPKG_* are used by the imgpkg com
   mand only.

   # The registry from which the Tanzu Application Platform package is retrieved.
   export INSTALL_REGISTRY_USERNAME="${IMGPKG_REGISTRY_USERNAME_1}"
   export INSTALL_REGISTRY_PASSWORD="${IMGPKG_REGISTRY_PASSWORD_1}"
   export INSTALL_REGISTRY_HOSTNAME=MY-REGISTRY
   export TAP_VERSION=VERSION-NUMBER
   export INSTALL_REPO=TARGET-REPOSITORY
   ```

   Where:

   - `MY-REGISTRY-USER` is the user with write access to `MY-REGISTRY`.
   - `MY-REGISTRY-PASSWORD` is the password for `MY-REGISTRY-USER`.
   - `MY-REGISTRY` is your own container registry.
   - `MY-TANZUNET-USERNAME` is the user with access to the images in the VMware Tanzu Network registry `registry.tanzu.vmware.com`
   - `MY-TANZUNET-PASSWORD` is the password for `MY-TANZUNET-USERNAME`.
   - `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.9.1`.
   - `TARGET-REPOSITORY` is your target repository, a folder/repository on `MY-REGISTRY` that serves as the location for the installation files for Tanzu Application Platform.

2. Install the Carvel tool imgpkg CLI.
3. Relocate the images with the `imgpkg` CLI by running:

   ```
   imgpkg copy -b registry.tanzu.vmware.com/tanzu-application-platform/tap-package
   s:${TAP_VERSION} --to-repo ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/tap-pac
   kages
   ```

4. Create a namespace called `tap-install` for deploying any component packages by running:

   ```
   kubectl create ns tap-install
   ```

   This namespace keeps the objects grouped together logically.

5. Create a registry secret by running:

```
tanzu secret registry add tap-registry \
  --username ${INSTALL_REGISTRY_USERNAME} --password ${INSTALL_REGISTRY_PASSWOR
D} \
  --server ${INSTALL_REGISTRY_HOSTNAME} \
  --export-to-all-namespaces --yes --namespace tap-install
```

6. (Optional) Create a registry secret for your writable image repository used for:
    - Tanzu Build Service Dependencies
    - Workloads when using the `shared.image_registry` key

```
tanzu secret registry add image-registry-creds \
  --server "${REGISTRY_HOSTNAME}" \
  --username "${REGISTRY_USERNAME}" \
  --password "${REGISTRY_PASSWORD}" \
  --namespace tap-install
```

Where:

- `REGISTRY_HOSTNAME` is the host name for the registry that contains your writable repository. Examples:
    - Harbor has the form `--server "my-harbor.io"`.
    - Docker Hub has the form `--server "index.docker.io"`.
    - Google Cloud Registry has the form `--server "gcr.io"`.
- `REGISTRY_USERNAME` and `REGISTRY_PASSWORD` are the user name and password for the user that can write to the repository used in the following step. For Google Cloud Registry, use `_json_key` as the user name and the contents of the service account JSON file for the password.

Note

If using the same repository as `tap-registry`, you can skip this step and use the `tap-registry` secret in your `tap-values.yaml` instead of `image-registry-creds`.

7. Add the Tanzu Application Platform package repository to the cluster by running:

```
tanzu package repository add tanzu-tap-repository \
  --url ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/tap-packages:$TAP_VERSION
\
  --namespace tap-install
```

8. Get the status of the Tanzu Application Platform package repository, and ensure the status updates to `Reconcile succeeded` by running:

```
tanzu package repository get tanzu-tap-repository --namespace tap-install
```

For example:

```
$ tanzu package repository get tanzu-tap-repository --namespace tap-install
- Retrieving repository tap...
NAME:          tanzu-tap-repository
VERSION:       16253001
REPOSITORY:    tapmdc.azurecr.io/mdc/1.4.0/tap-packages
TAG:           1.9.1
```

```
STATUS:        Reconcile succeeded
REASON:
```

Note

The `VERSION` and `TAG` numbers differ from the earlier example if you are on Tanzu Application Platform v1.0.2 or earlier.

9. List the available packages by running:

```
tanzu package available list --namespace tap-install
```

For example:

```
$ tanzu package available list --namespace tap-install
/ Retrieving available packages...
  NAME                                                DISPLAY-NAME
SHORT-DESCRIPTION
  accelerator.apps.tanzu.vmware.com                   Application Accelerator
for VMware Tanzu                                    Used to create new projects a
nd configurations.
  api-portal.tanzu.vmware.com                         API portal
A unified user interface for API discovery and exploration at scale.
  apis.apps.tanzu.vmware.com                          API Auto Registration fo
r VMware Tanzu                                      A TAP component to automatica
lly register API exposing workloads as API entities

in TAP GUI.
  backend.appliveview.tanzu.vmware.com                Application Live View fo
r VMware Tanzu                                      App for monitoring and troubl
eshooting running apps
  buildservice.tanzu.vmware.com                       Tanzu Build Service
Tanzu Build Service enables the building and automation of containerized

software workflows securely and at scale.
  carbonblack.scanning.apps.tanzu.vmware.com          VMware Carbon Black for
Supply Chain Security Tools - Scan                  Default scan templates using
VMware Carbon Black
  cartographer.conventions.apps.tanzu.vmware.com      Convention Service for V
Mware Tanzu                                         Convention Service enables ap
p operators to consistently apply desired runtime

configurations to fleets of workloads.
  cartographer.tanzu.vmware.com                       Cartographer
Kubernetes native Supply Chain Choreographer.
  cnrs.tanzu.vmware.com                               Cloud Native Runtimes
Cloud Native Runtimes is a serverless runtime based on Knative
  connector.appliveview.tanzu.vmware.com              Application Live View Co
nnector for VMware Tanzu                            App for discovering and regis
tering running apps
  controller.source.apps.tanzu.vmware.com             Tanzu Source Controller
Tanzu Source Controller enables workload create/update from source code.
  conventions.appliveview.tanzu.vmware.com            Application Live View Co
nventions for VMware Tanzu                          Application Live View convent
ion server
  developer-conventions.tanzu.vmware.com              Tanzu App Platform Devel
oper Conventions                                    Developer Conventions
  external-secrets.apps.tanzu.vmware.com              External Secrets Operato
r                                                   External Secrets Operator is
a Kubernetes operator that integrates external

secret management systems.
  fluxcd.source.controller.tanzu.vmware.com           Flux Source Controller
```

```
The source-controller is a Kubernetes operator, specialised in artifacts

acquisition from external sources such as Git, Helm repositories and S3 bucket
s.
  grype.scanning.apps.tanzu.vmware.com                Grype for Supply Chain S
ecurity Tools - Scan                                Default scan templates using
Anchore Grype
  metadata-store.apps.tanzu.vmware.com                Supply Chain Security To
ols - Store                                         Post SBoMs and query for imag
e, package, and vulnerability metadata.
  namespace-provisioner.apps.tanzu.vmware.com         Namespace Provisioner
Automatic Provisioning of Developer Namespaces.
  ootb-delivery-basic.tanzu.vmware.com                Tanzu App Platform Out o
f The Box Delivery Basic                            Out of The Box Delivery Basi
c.
  ootb-supply-chain-basic.tanzu.vmware.com            Tanzu App Platform Out o
f The Box Supply Chain Basic                        Out of The Box Supply Chain B
asic.
  ootb-supply-chain-testing-scanning.tanzu.vmware.com  Tanzu App Platform Out o
f The Box Supply Chain with Testing and Scanning  Out of The Box Supply Chain w
ith Testing and Scanning.
  ootb-supply-chain-testing.tanzu.vmware.com          Tanzu App Platform Out o
f The Box Supply Chain with Testing                 Out of The Box Supply Chain w
ith Testing.
  ootb-templates.tanzu.vmware.com                     Tanzu App Platform Out o
f The Box Templates                                 Out of The Box Templates.
  policy.apps.tanzu.vmware.com                        Supply Chain Security To
ols - Policy Controller                             Policy Controller enables def
ining of a policy to restrict unsigned container

images.
  scanning.apps.tanzu.vmware.com                      Supply Chain Security To
ols - Scan                                          Scan for vulnerabilities and
enforce policies directly within Kubernetes native

Supply Chains.
  service-bindings.labs.vmware.com                    Service Bindings for Kub
ernetes                                             Service Bindings for Kubernet
es implements the Service Binding Specification.
  services-toolkit.tanzu.vmware.com                   Services Toolkit
The Services Toolkit enables the management, lifecycle, discoverability and

connectivity of Service Resources (databases, message queues, DNS records,

etc.).
  snyk.scanning.apps.tanzu.vmware.com                 Snyk for Supply Chain Se
curity Tools - Scan                                 Default scan templates using
Snyk
  spring-boot-conventions.tanzu.vmware.com            Tanzu Spring Boot Conven
tions Server                                        Default Spring Boot conventio
n server.
  sso.apps.tanzu.vmware.com                           AppSSO
Application Single Sign-On for Tanzu
  tap-auth.tanzu.vmware.com                           Default roles for Tanzu
Application Platform                                Default roles for Tanzu Appli
cation Platform
  tap-gui.tanzu.vmware.com                            Tanzu Developer Portal
web app graphical user interface for Tanzu Application Platform
  tap-telemetry.tanzu.vmware.com                      Telemetry Collector for
Tanzu Application Platform                          Tanzu Application Platform Te
lemetry
  tap.tanzu.vmware.com                                Tanzu Application Platfo
rm                                                  Package to install a set of T
AP components to get you started based on your use

case.
```

```
  tekton.tanzu.vmware.com                                Tekton Pipelines
Tekton Pipelines is a framework for creating CI/CD systems.
```

Install your Tanzu Application Platform profile

The `tap.tanzu.vmware.com` package installs predefined sets of packages based on your profile settings. This is done by using the package manager installed by Tanzu Cluster Essentials.

For more information about profiles, see Components and installation profiles.

To prepare to install a profile:

1. List version information for the package by running:

   ```
   tanzu package available list tap.tanzu.vmware.com --namespace tap-install
   ```

2. Create a `tap-values.yaml` file by using the Full Profile sample in the following section as a guide. These samples have the minimum configuration required to deploy Tanzu Application Platform. The sample values file contains the necessary defaults for:

   - The meta-package, or parent Tanzu Application Platform package.
   - Subordinate packages, or individual child packages.

   Keep the values file for future configuration use.

   Note

   `tap-values.yaml` is set as a Kubernetes secret, which provides secure means to read credentials for Tanzu Application Platform components.

3. View possible configuration settings for your package

Full profile

The following is the YAML file sample for the full-profile. The `profile:` field takes `full` as the default value, but you can also set it to `iterate`, `build`, `run` or `view`. Refer to Install multicluster Tanzu Application Platform profiles for more information.

```
shared:
  ingress_domain: "INGRESS-DOMAIN"
  image_registry:
    project_path: "SERVER-NAME/REPO-NAME"
    secret:
      name: image-registry-creds
      namespace: tap-install
  kubernetes_distribution: "openshift" # To be passed only for OpenShift. Defaults to
"".
  kubernetes_version: "K8S-VERSION"
  ca_cert_data: | # To be passed if using custom certificates.
      -----BEGIN CERTIFICATE-----
      MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...
      -----END CERTIFICATE-----

ceip_policy_disclosed: FALSE-OR-TRUE-VALUE # Installation fails if this is not set to
true. Not a string.

#The above keys are minimum numbers of entries needed in tap-values.yaml to get a func
tioning TAP Full profile installation.

#Below are the keys which may have default values set, but can be overridden.
```

```
profile: full # Can take iterate, build, run, view.

supply_chain: basic # Can take testing, testing_scanning.

ootb_supply_chain_basic: # Based on supply_chain set above, can be changed to ootb_sup
ply_chain_testing, ootb_supply_chain_testing_scanning.
  source:
    credentials_secret: "GIT-SOURCE-CREDENTIAL-SECRET-NAME" # (Optional) Defaults to
"".
  registry:
    server: "SERVER-NAME" # Takes the value from shared section above by default, but
can be overridden by setting a different value.
    repository: "REPO-NAME" # Takes the value from shared section above by default, bu
t can be overridden by setting a different value.
  gitops:
    credentials_secret: "GITOPS-CREDENTIAL-SECRET-NAME" # (Optional) Defaults to "".

contour:
  envoy:
    service:
      type: LoadBalancer # This is set by default, but can be overridden by setting a
different value.

buildservice:
  kp_default_repository: "KP-DEFAULT-REPO"
  kp_default_repository_secret: # Takes the value from the shared section by default,
but can be overridden by setting a different value.
    name: image-registry-creds
    namespace: tap-install

local_source_proxy:
  # Takes the value from the project_path under the image_registry section of shared b
y default, but can be overridden by setting a different value.
  repository: "EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE"
  push_secret:
    # When set to true, the secret mentioned in this section is automatically exported
to Local Source Proxy's namespace.
    name: "EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE-SECRET"
    namespace: "EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE-SECRET-NAMESPACE"
    # When set to true, the secret mentioned in this section is automatically exported
to Local Source Proxy's namespace.
    create_export: true

tap_gui:
  app_config:
    auth:
      allowGuestAccess: true  # This allows unauthenticated users to log in to your po
rtal. If you want to deactivate it, make sure you configure an alternative auth provid
er.
    catalog:
      locations:
        - type: url
          target: https://GIT-CATALOG-URL/catalog-info.yaml

metadata_store:
  ns_for_export_app_cert: "MY-DEV-NAMESPACE"
  app_service_type: ClusterIP # Defaults to LoadBalancer. If shared.ingress_domain is
set earlier, this must be set to ClusterIP.
```

Important

Installing Grype by using `tap-values.yaml` as follows is deprecated in v1.6 and will be removed in v1.8:

```
grype:
  targetImagePullSecret: "TARGET-REGISTRY-CREDENTIALS-SECRET"
```

You can install Grype by using Namespace Provisioner instead.

Where:

- `INGRESS-DOMAIN` is the subdomain for the host name that you point at the `tanzu-shared-ingress` service's External IP address.
- `KP-DEFAULT-REPO` is a writable repository in your registry. Tanzu Build Service dependencies are written to this location. Examples:
  - Harbor has the form `kp_default_repository: "my-harbor.io/my-project/build-service"`.
  - Docker Hub has the form `kp_default_repository: "my-dockerhub-user/build-service"` or `kp_default_repository: "index.docker.io/my-user/build-service"`.
  - Google Cloud Registry has the form `kp_default_repository: "gcr.io/my-project/build-service"`.
- `K8S-VERSION` is the Kubernetes version used by your OpenShift cluster. It must be in the form of `1.26.3`, `1.27.x` or `1.28.x`, where `x` stands for the patch version. Examples:
  - Red Hat OpenShift Container Platform v4.13 uses the Kubernetes version `1.26.3`.
  - Red Hat OpenShift Container Platform v4.14 uses the Kubernetes version `1.27.6`.
  - Red Hat OpenShift Container Platform v4.15 uses the Kubernetes version `1.28.5`.
- `SERVER-NAME` is the host name of the registry server. Examples:
  - Harbor has the form `server: "my-harbor.io"`.
  - Docker Hub has the form `server: "index.docker.io"`.
  - Google Cloud Registry has the form `server: "gcr.io"`.
- `REPO-NAME` is where workload images are stored in the registry. If this key is passed through the shared section earlier and AWS ECR registry is used, you must ensure that the `SERVER-NAME/REPO-NAME/buildservice` and `SERVER-NAME/REPO-NAME/workloads` exist. AWS ECR expects the paths to be pre-created. Images are written to `SERVER-NAME/REPO-NAME/workload-name`. Examples:
  - Harbor has the form `repository: "my-project/supply-chain"`.
  - Docker Hub has the form `repository: "my-dockerhub-user"`.
  - Google Cloud Registry has the form `repository: "my-project/supply-chain"`.
- `EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE` is where the developer's local source is uploaded when using Tanzu CLI to use Local Source Proxy for workload creation.

  If an AWS ECR registry is being used, ensure that the repository already exists. AWS ECR expects the repository path to already exist. This destination is represented as `REGISTRY-SERVER/REPOSITORY-PATH`. For more information, see Install Local Source Proxy.
- `EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE-SECRET` is the name of the secret with credentials that allow pushing to the `EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE` repository.
- `EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE-SECRET-NAMESPACE` is the namespace in which `EXTERNAL-REGISTRY-FOR-LOCAL-SOURCE-SECRET` is available.
- `GIT-SOURCE-CREDENTIAL-SECRET-NAME` is the name of the Kubernetes secret in the developer namespace that supplies the Git credentials for the supply chain to fetch source code from.

This field is only required if you use a private repository. See Git authentication for more information.

- `GITOPS-CREDENTIAL-SECRET-NAME` is the name of the Kubernetes secret in the developer namespace that supplies the Git credentials for the supply chain to push configuration to. See Git authentication for more information.
- `GIT-CATALOG-URL` is the path to the `catalog-info.yaml` catalog definition file. You can download either a blank or populated catalog file from the Tanzu Application Platform product page. Otherwise, you can use a Backstage-compliant catalog you've already built and posted on the Git infrastructure.
- `MY-DEV-NAMESPACE` is the name of the developer namespace. SCST - Store exports secrets to the namespace, and SCST - Scan deploys the `ScanTemplates` there. This allows the scanning feature to run in this namespace. If there are multiple developer namespaces, use `ns_for_export_app_cert: "*"` to export the SCST - Store CA certificate to all namespaces.
- `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the secret that contains the credentials to pull an image from the registry for scanning.

Tanzu Application Platform is part of VMware's CEIP program where data is collected to help improve the customer experience. By setting `ceip_policy_disclosed` to `true` (not a string), you acknowledge the program is disclosed to you and you are aware data collection is happening. This field must be set for the installation to be completed. See Opt out of telemetry collection for more information.

If you use custom CA certificates, you must provide one or more PEM-encoded CA certificates under the `ca_cert_data` key. If you configured `shared.ca_cert_data`, Tanzu Application Platform component packages inherit that value by default.

If you use AWS, the default settings creates a classic LoadBalancer. To use the Network LoadBalancer instead of the classic LoadBalancer for ingress, add the following to your `tap-values.yaml`:

```
contour:
  infrastructure_provider: aws
  envoy:
    service:
      aws:
        LBType: nlb
```

(Optional) Additional Build Service configurations

The following tasks are optional during the Tanzu Application Platform installation process:

- (Optional) Configure your profile with full dependencies
- (Optional) Configure your profile with the Jammy stack only

(Optional) Configure your profile with full dependencies

When you install a profile that includes Tanzu Build Service, Tanzu Application Platform is installed with the `lite` set of dependencies. These dependencies consist of buildpacks and stacks required for application builds.

The `lite` set of dependencies do not contain all buildpacks and stacks. To use all buildpacks and stacks, you must install the `full` dependencies. For more information about the differences between `lite` and `full` dependencies, see About lite and full dependencies.

To configure `full` dependencies, add the key-value pair `exclude_dependencies: true` to your `tap-values.yaml` file under the `buildservice` section. For example:

```
buildservice:
  ...
  exclude_dependencies: true
  ...
```

After configuring `full` dependencies, you must install the dependencies after you have finished installing your Tanzu Application Platform package. See Install the full dependencies package for more information.

(Optional) Configure your profile with the Jammy stack only

Tanzu Application Platform v1.5.0 supports building applications with both the Ubuntu v22.04 (Jammy) and v18.04 (Bionic) stack. For more information, see Bionic and Jammy stacks.

To install Tanzu Application Platform with Jammy as the only available stack, include the `stack_configuration: jammy-only` field under the `buildservice:` section in `tap-values.yaml`.

Security Context Constraints

Security Context Constraints (SCC) define a set of rules that a pod must satisfy to be created. Tanzu Application Platform components use the built-in nonroot-v2 or restricted-v2 SCC.

In Red Hat OpenShift, SCC are used to restrict privileges for pods. In Tanzu Application Platform v1.4 there is no custom SCC.

Tanzu Application Platform packages reconcile without any issues when using OpenShift v4.11 with restricted-v2 or nonroot-v2.

Install your Tanzu Application Platform package

Follow these steps to install the Tanzu Application Platform package:

1. Install the package by running:

   ```
   tanzu package install tap -p tap.tanzu.vmware.com -v $TAP_VERSION --values-file
   tap-values.yaml -n tap-install
   ```

2. Verify the package install by running:

   ```
   tanzu package installed get tap -n tap-install
   ```

   This can take 5-10 minutes because it installs several packages on your cluster.

3. Verify that the necessary packages in the profile are installed by running:

   ```
   tanzu package installed list -A
   ```

4. If you configured `full` dependencies in your `tbs-values.yaml` file, install the `full` dependencies by following the procedure in Install full dependencies.

After installing the Full profile on your cluster, you can install the Tanzu Developer Tools for VS Code Extension to help you develop against it. For instructions, see Install Tanzu Developer Tools for your VS Code.

Note

You can run the following command after reconfiguring the profile to reinstall the Tanzu Application Platform:

```
tanzu package installed update tap -p tap.tanzu.vmware.com -v $TAP_VERSION  --values-f
ile tap-values.yaml -n tap-install
```

Install the full dependencies package

If you configured `full` dependencies in your `tap-values.yaml` file in Configure your profile with full dependencies earlier, you must install the `full` dependencies package.

For more information about the differences between `lite` and `full` dependencies, see About lite and full dependencies.

To install the `full` dependencies package:

1. Get the latest version of the Tanzu Application Platform package by running:

   ```
   tanzu package available list tap.tanzu.vmware.com --namespace tap-install
   ```

2. If you have not done so already, you must exclude the default dependencies by adding the key-value pair `exclude_dependencies: true` to your `tap-values.yaml` file under the `buildservice` section. For example:

   ```
   buildservice:
     exclude_dependencies: true
   ```

3. If you have not updated your Tanzu Application Platform package installation after adding the key-value pair `exclude_dependencies: true` to your values file, perform the update by running:

   ```
   tanzu package installed update tap --namespace tap-install --values-file VALUES
   -FILE
   ```

   Where `VALUES-FILE` is the path to the `tap-values.yaml` file you edited earlier.

4. Relocate the Tanzu Build Service `full` dependencies package repository by doing one of the following:

   - Relocate the images directly for online installation:

     ```
     imgpkg copy \
       -b registry.tanzu.vmware.com/tanzu-application-platform/full-deps-packa
     ge-repo:VERSION \
       --to-repo ${INSTALL_REGISTRY_HOSTNAME}/full-deps-package-repo
     ```

     Where `VERSION` is the version of the Tanzu Application Platform package you retrieved earlier.

   - Relocate the images to an external storage device and then to the registry in the air-gapped environment:

     ```
     imgpkg copy \
       -b registry.tanzu.vmware.com/tanzu-application-platform/full-deps-packa
     ge-repo:VERSION \
       --to-tar=full-deps-package-repo.tar

     # move full-deps-package-repo.tar to environment with registry access
     imgpkg copy \
       --tar full-deps-package-repo.tar \
       --to-repo=INSTALL-REGISTRY-HOSTNAME/TARGET-REPOSITORY/full-deps-package
     -repo
     ```

     Where:

- `VERSION` is the version of the Tanzu Application Platform package you retrieved earlier.
- `INSTALL-REGISTRY-HOSTNAME` is your container registry.
- `TARGET-REPOSITORY` is your target repository.

5. Add the Tanzu Build Service `full` dependencies package repository by running:

```
tanzu package repository add full-deps-package-repo \
  --url INSTALL-REGISTRY-HOSTNAME/TARGET-REPOSITORY/full-deps-package-repo:VERS
ION \
  --namespace tap-install
```

Where:

- `INSTALL-REGISTRY-HOSTNAME` is your container registry.
- `TARGET-REPOSITORY` is your target repository.
- `VERSION` is the version of the Tanzu Application Platform package you retrieved earlier.

6. Create a new `tbs-full-deps-values.yaml` and copy the `kp_default_repository` key-value pair from your `tap-values.yaml` or `tbs-values.yaml`:

```
---
 kp_default_repository: "REPO-NAME"
 kp_default_repository_secret:
   name: kp-default-repository-creds
   namespace: tap-install
```

Where `REPO-NAME` is copied from the `buildservice.kp_default_repository` field in your `tap-values.yaml` or `tbs-values.yaml`.

1. (Optional) Install the UBI builder.

   The UBI builder uses Red Hat Universal Base Image (UBI) v8 for both build and run images. This builder only supports Java and Node.js. To install the UBI builder, add the key-value pair `enable_ubi_builder: true` to your `tbs-full-deps-values.yaml`.

   ```
   ---
   enable_ubi_builder: true
   ```

2. (Optional) Install the Static builder.

   The Static builder uses Ubuntu Jammy for both build images and a minimal static run image. This builder only supports Golang. To install the Static builder, add the key-value pair `enable_static_builder: true` to your `tbs-full-deps-values.yaml`.

   ```
   ---
   enable_static_builder: true
   ```

7. Install the `full` dependencies package by running:

```
tanzu package install full-deps \
  --package full-deps.buildservice.tanzu.vmware.com \
  --version "> 0.0.0" \
  --namespace tap-install \
  --values-file VALUES-FILE
```

Where `VALUES-FILE` is the path to the `tbs-full-deps-values.yaml` you created earlier.

Access Tanzu Developer Portal

To access Tanzu Developer Portal, you can use the host name that you configured earlier. This host name is pointed at the shared ingress. To configure LoadBalancer for Tanzu Developer Portal, see Access Tanzu Developer Portal.

You're now ready to start using Tanzu Developer Portal. Proceed to the Getting Started topic or the Tanzu Developer Portal - Catalog Operations topic.

Exclude packages from a Tanzu Application Platform profile

To exclude packages from a Tanzu Application Platform profile:

1. Find the full subordinate (child) package name:

   ```
   tanzu package available list --namespace tap-install
   ```

2. Update your `tap-values` file with a section listing the exclusions:

   ```
   profile: PROFILE-VALUE
   excluded_packages:
     - tap-gui.tanzu.vmware.com
     - service-bindings.lab.vmware.com
   ```

Important

If you exclude a package after performing a profile installation including that package, you cannot see the accurate package states immediately after running `tap package installed list -n tap-install`. Also, you can break package dependencies by removing a package. Allow 20 minutes to verify that all packages have reconciled correctly while troubleshooting.

View possible configuration settings for your package

To view possible configuration settings for a package, run:

```
tanzu package available get tap.tanzu.vmware.com/$TAP_VERSION --values-schema --namesp
ace tap-install
```

Note

The `tap.tanzu.vmware.com` package does not show all configuration settings for packages it plans to install. The package only shows top-level keys. You can view individual package configuration settings with the same `tanzu package available get` command. For example, to find the keys for Cloud Native Runtimes, you must first identify the version of the package with `tanzu package installed list -n tap-install`, which lists all the installed packages versions. Then run the command `tanzu package available get -n tap-install cnrs.tanzu.vmware.com/CNR-VERSION --values-schema` by using the package version listed for Cloud Native Runtimes.

```
profile: full

# Shared configurations go under the shared key.
shared:
  ingress_domain: tap.example.com

# ...

# For example, Cloud Native Runtimes specific values go under its name.
cnrs:
  provider: local

# For example, App Accelerator specific values go under its name.
accelerator:
  server:
    service_type: "ClusterIP"
```

Shared Keys define values that configure multiple packages. These keys are defined under the `shared` Top-level Key, as summarized in the following table:

Shared Key	Description	Optional
`ca_cert_data`	PEM-encoded certificate data to trust TLS connections with a private CA. This shared key is used by `convention_controller`, `scanning` and the Tanzu `source_controller` (not the Flux CD Source Controller).	Yes
`ingress_domain`	Domain name to be used in service routes and host names for instances of Tanzu Application Platform components.	Yes
`ingress_issuer`	A `cert-manager.io/v1/ClusterIssuer` for issuing TLS certificates to Tanzu Application Platform components. Default value: `tap-ingress-selfsigned`	Yes
`kubernetes_distribution`	Type of Kubernetes infrastructure being used. You can use this shared key in coordination with the `kubernetes_version` key. Supported value: `openshift`.	Yes
`kubernetes_version`	Kubernetes version. You can use this shared key independently or in coordination with the `kubernetes_distribution` key. Supported value: `1.24.x`, where `x` stands for the Kubernetes patch version.	Yes
`image_registry.project_path`	Project path in the container image registry server used for builder and application images.	Yes
`image_registry.username`	User name for the container image registry. Mutually exclusive with `shared.image_registry.secret.name/namespace`	Yes
`image_registry.password`	Password for the container image registry. Mutually exclusive with `shared.image_registry.secret.name/namespace`	Yes
`secret.name`	Secret name for the container image registry credentials of type `kubernetes.io/dockerconfigjson`. Mutually exclusive with `shared.image_registry.username/password`	Yes
`secret.namespace`	Secret namespace for the container image registry credentials. Mutually exclusive with `shared.image_registry.username/password`	Yes
`activateAppLiveViewSecureAccessControl`	Enable secure access connection between Application Live View components.	Yes

The following table summarizes the top-level keys used for package-specific configuration within your `tap-values.yaml`.

Package	Top-level Key
See table above.	`shared`
API Auto Registration	`api_auto_registration`

Package	Top-level Key
API portal	`api_portal`
Application Accelerator	`accelerator`
Application Live View	`appliveview`
Application Live View connector	`appliveview_connector`
Application Live View conventions	`appliveview-conventions`
Cartographer Conventions	`cartographer_conventions`
Cartographer	`cartographer`
Cloud Native Runtimes	`cnrs`
Source Controller	`source_controller`
Supply Chain	`supply_chain`
Supply Chain Basic	`ootb_supply_chain_basic`
Supply Chain Testing	`ootb_supply_chain_testing`
Supply Chain Testing Scanning	`ootb_supply_chain_testing_scanning`
Supply Chain Security Tools - Scan	`scanning`
Supply Chain Security Tools - Scan (Grype Scanner)	`grype`
Supply Chain Security Tools - Store	`metadata_store`
Build Service	`buildservice`
Tanzu Developer Portal	`tap_gui`

For information about package-specific configuration, see Install individual packages.

Install individual packages

You can install Tanzu Application Platform (commonly known as TAP) through predefined profiles or through individual packages. Use this topic to learn how to install each individual package. For more information about installing through profiles, see Components and installation profiles.

Installing individual Tanzu Application Platform packages is useful if you do not want to use a profile to install packages or if you want to install additional packages after installing a profile. Before installing the packages, be sure to complete the prerequisites, configure and verify the cluster, accept the EULA, and install the Tanzu CLI with any required plug-ins. For more information, see Prerequisites.

Install pages for individual Tanzu Application Platform packages

- Install API Auto Registration
- Install API portal
- Install Application Accelerator
- Install Application Configuration Service
- Install Application Live View
- Install Application Single Sign-On

- Install Bitnami Services
- Install cert-manager
- Install Cloud Native Runtimes
- Install Contour
- Install Crossplane
- Install default roles for Tanzu Application Platform
- Install Developer Conventions
- Install Flux CD Source Controller
- Install Out of the Box Templates
- Install Out of the Box Supply Chain with Testing
- Install Out of the Box Supply Chain with Testing and Scanning
- Install Service Bindings
- Install Services Toolkit
- Install Source Controller
- Install Spring Boot conventions
- Install Supply Chain Choreographer
- Install Supply Chain Security Tools - Store
- Install Supply Chain Security Tools - Policy Controller
- Install Supply Chain Security Tools - Scan
- Install Tanzu Developer Portal
- Install Tanzu Build Service
- Install Tekton
- Install Telemetry

Verify the installed packages

Use the following procedure to verify that the packages are installed.

1. List the installed packages by running:

   ```
   tanzu package installed list --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package installed list --namespace tap-install
   \ Retrieving installed packages...
   NAME                     PACKAGE-NAME                                    PAC
   KAGE-VERSION  STATUS
   api-portal               api-portal.tanzu.vmware.com                     1.
   0.3           Reconcile succeeded
   app-accelerator          accelerator.apps.tanzu.vmware.com               1.
   0.0           Reconcile succeeded
   app-live-view            appliveview.tanzu.vmware.com                    1.
   0.2           Reconcile succeeded
   appliveview-conventions  build.appliveview.tanzu.vmware.com              1.
   0.2           Reconcile succeeded
   cartographer             cartographer.tanzu.vmware.com                   0.
   1.0           Reconcile succeeded
   ```

```
cloud-native-runtimes    cnrs.tanzu.vmware.com                                1.
0.3            Reconcile succeeded
convention-controller    controller.conventions.apps.tanzu.vmware.com         0.
7.0            Reconcile succeeded
developer-conventions    developer-conventions.tanzu.vmware.com               0.
3.0-build.1    Reconcile succeeded
grype-scanner            grype.scanning.apps.tanzu.vmware.com                 1.
0.0            Reconcile succeeded
image-policy-webhook     image-policy-webhook.signing.apps.tanzu.vmware.com 1.
1.2            Reconcile succeeded
metadata-store           metadata-store.apps.tanzu.vmware.com                 1.
0.2            Reconcile succeeded
ootb-supply-chain-basic  ootb-supply-chain-basic.tanzu.vmware.com             0.
5.1            Reconcile succeeded
ootb-templates           ootb-templates.tanzu.vmware.com                      0.
5.1            Reconcile succeeded
scan-controller          scanning.apps.tanzu.vmware.com                       1.
0.0            Reconcile succeeded
service-bindings         service-bindings.labs.vmware.com                     0.
5.0            Reconcile succeeded
services-toolkit         services-toolkit.tanzu.vmware.com                    0.
8.0            Reconcile succeeded
source-controller        controller.source.apps.tanzu.vmware.com              0.
2.0            Reconcile succeeded
sso4k8s-install          sso.apps.tanzu.vmware.com                            1.
0.0-beta.2-31  Reconcile succeeded
tap-gui                  tap-gui.tanzu.vmware.com                             0.
3.0-rc.4       Reconcile succeeded
tekton-pipelines         tekton.tanzu.vmware.com                              0.3
0.0           Reconcile succeeded
tbs                      buildservice.tanzu.vmware.com                        1.
5.0            Reconcile succeeded
```

Next steps

- Set up developer namespaces to use your installed packages

Set up developer namespaces to use your installed packages

For details about how to automatically set up your developer namespaces, see Provision developer namespaces in Namespace Provisioner.

Additional configuration for testing and scanning

If you plan to install or have already installed Out of the Box Supply Chains with Testing and Scanning, you can use Namespace Provisioner to set up the required resources. For more information, see Customize installation in the Namespace Provisioner documentation for configuration steps.

Legacy namespace setup

To use the legacy manual process for setting up developer namespaces, see Legacy namespace setup.

Next steps

- Install Tanzu Developer Tools for your VS Code

Install Tanzu Developer Tools for your VS Code

This topic tells you how to install VMware Tanzu Developer Tools for Visual Studio Code (VS Code).

Prerequisites

Before installing the extension, you must have:

- VS Code
- kubectl
- Tilt v0.30.12 or later
- Tanzu CLI and plug-ins
- A cluster with the Tanzu Application Platform Full profile or Iterate profile

If you are an app developer, someone else in your organization might have already set up the Tanzu Application Platform environment.

Docker Desktop and local Kubernetes are not prerequisites for using Tanzu Developer Tools for VS Code.

Install

To install VMware Tanzu Developer Tools for VS Code:

1. Open Visual Studio Code.
2. Open the command palette.
3. In the search box enter `Extension`.
4. Click **Extensions: Install Extensions**.
5. The **Extensions** view opens on the left side of your screen. In the search box enter `Tanzu`.
6. Click **Tanzu Developer Tools** and then click **Install**.

Configure

To configure VMware Tanzu Developer Tools for VS Code:

1. Ensure that you are targeting the correct cluster. For more information, see the Kubernetes documentation.
2. Go to **Code** > **Preferences** > **Settings** > **Extensions** > **Tanzu Developer Tools** and set the following:
    - **Confirm Apply Config**: This controls whether the extension asks to confirm user input when applying a workload.
    - **Confirm Debug Port**: This controls whether the extension asks for the debug port when running `Tanzu: Start Java Debug`.
    - **Confirm Local Port**: This controls whether the extension asks for the local port when running `Tanzu: Start Java Debug`.
    - **Confirm Delete**: This controls whether the extension asks for confirmation when deleting a workload.
    - **Enable Live Hover**: This enables Live Hover. For more information, see Integrating Live Hover by using Spring Boot Tools. Restart VS Code for this change to take effect.

- **Source Image**: The registry location for publishing local source code. For example, `registry.io/yourapp-source`. This must include both a registry and a project name. A source image registry location is optional when Local Source Proxy is configured.
- **Local Path**: (Optional) The path on the local file system to a directory of source code to build. This is the current directory by default.
- **Namespace**: (Optional) This is the namespace that workloads are deployed into. The namespace set in kubeconfig is the default.
- **Tracked Namespaces**: (Optional) Comma-separated list of namespaces. Resources in these namespaces appear in the Tanzu Workloads panel and the Activity panel. If empty, the namespace in the current context is the default.
- **Wait Timeout**: (Optional) This sets how long to wait for a workload to become ready.
- **Workload Type**: (Optional) This distinguishes the workload type. Examples include web and server.

Uninstall

To uninstall VMware Tanzu Developer Tools for VS Code:

1. Go to **Code** > **Preferences** > **Settings** > **Extensions**.
2. Right-click the extension and then click **Uninstall**.

Next steps

Proceed to Getting started with Tanzu Developer Tools for Visual Studio Code.

Custom Security Context Constraint details for Tanzu Application Platform

Custom Security Context Constraint (commonly known as SCC) details for Tanzu Application Platform (commonly known as TAP) components are as follows:

- Application Accelerator on OpenShift cluster
- Application Live View on OpenShift
- Application Single Sign-On for OpenShift cluster
- Contour for OpenShift cluster
- Developer Conventions for OpenShift cluster
- Tanzu Build Service for OpenShift cluster

Application Accelerator on OpenShift

On OpenShift clusters, Application Accelerator must run with a custom SecurityContextConstraint (SCC) to enable compliance with restricted Kubernetes pod security standards. Tanzu Application Platform configures the following SCC for Application Accelerator when you configure the `kubernetes_distribution: openshift` key in the `tap-values.yaml` file.

Specification follows:

```
#@ load("@ytt:data", "data")
#@ load("@ytt:assert", "assert")
```

```
#@ kubernetes_distribution = data.values.kubernetes_distribution
#@ validDistributions = [None, "", "openshift"]
#@ if kubernetes_distribution not in validDistributions:
#@   assert.fail("{} not in {}".format(kubernetes_distribution, validDistributions))
#@ end

#@ if kubernetes_distribution == "openshift":
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: accelerator-system-nonroot-scc
  namespace: accelerator-system
rules:
- apiGroups:
  - security.openshift.io
  resourceNames:
  - nonroot
  resources:
  - securitycontextconstraints
  verbs:
  - use
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: accelerator-system-nonroot-scc
  namespace: accelerator-system
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: accelerator-system-nonroot-scc
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: system:serviceaccounts:accelerator-system
#@ end
```

Application Live View on OpenShift

Application Live View must run with a custom SecurityContextConstraint (SCC) to enable compliance with restricted Kubernetes Pod Security Standards on OpenShift. Tanzu Application Platform configures the following SCC for Application Live View back end, Application Live View connector, and Application Live View convention service when you configure the `kubernetes_distribution: openshift` key in the `tap-values.yaml` file.

The following is a `SecurityContextConstraints` specification for Application Live View connector:

```
---
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  name: appliveview-connector-restricted-with-seccomp
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: false
allowPrivilegedContainer: false
allowedCapabilities: null
defaultAddCapabilities: null
fsGroup:
  type: MustRunAs
```

```
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities:
  - ALL
runAsUser:
  type: MustRunAsNonRoot
seLinuxContext:
  type: MustRunAs
supplementalGroups:
  type: RunAsAny
volumes:
  - configMap
  - downwardAPI
  - emptyDir
  - persistentVolumeClaim
  - projected
  - secret
seccompProfiles:
  - runtime/default
```

The preceding `SecurityContextConstraints` specification is applicable to Application Live View back end and Application Live View convention service as well.

Application Single Sign-On for OpenShift cluster

On OpenShift clusters, AppSSO must run with a custom SecurityContextConstraint (SCC) to enable compliance with restricted Kubernetes Pod Security Standards. Tanzu Application Platform configures the following SCC for AppSSO controller and its `AuthServer` managed resources when you configure the `kubernetes_distribution: openshift` key in the `tap-values.yaml` file.

Specification follows:

```
---
kind: SecurityContextConstraints
apiVersion: security.openshift.io/v1
metadata:
  name: appsso-scc
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: false
allowPrivilegedContainer: false
allowedCapabilities: null
defaultAddCapabilities: null
fsGroup:
  type: MustRunAs
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities:
  - KILL
  - MKNOD
  - SETUID
  - SETGID
runAsUser:
  type: MustRunAsNonRoot
seLinuxContext:
  type: MustRunAs
volumes:
  - configMap
  - downwardAPI
  - emptyDir
  - persistentVolumeClaim
```

```
  - projected
  - secret
seccompProfiles:
  - 'runtime/default'
```

AppSSO controller's `ServiceAccount` is given the following additional permissions, including a `use` permission for AppSSO SCC, so `AuthServer` can use the custom SCC:

```
- apiGroups:
    - security.openshift.io
  resources:
    - securitycontextconstraints
  verbs:
    - "get"
    - "list"
    - "watch"
```

```
- apiGroups:
    - security.openshift.io
  resourceNames:
    - appsso-scc
  resources:
    - securitycontextconstraints
  verbs:
    - "use"
```

Contour for OpenShift cluster

On OpenShift clusters, Contour must run with a custom SecurityContextConstraint (SCC) to enable compliance with restricted Kubernetes Pod Security Standards. Tanzu Application Platform configures the following SCC for the service accounts in the `tanzu-system-ingress` namespace, which applies to Contour's controller and Envoy pods, when you configure the `kubernetes_distribution: openshift` key in the `tap-values.yaml` file.

Specification follows:

```
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  annotations:
    include.release.openshift.io/ibm-cloud-managed: "true"
    include.release.openshift.io/self-managed-high-availability: "true"
    include.release.openshift.io/single-node-developer: "true"
    kubernetes.io/description: nonroot provides all features of the restricted SCC
      but allows users to run with any non-root UID.  The user must specify the UID
      or it must be specified on the by the manifest of the container runtime. On
      top of the legacy 'nonroot' SCC, it also requires to drop ALL capabilities and
      does not allow privilege escalation binaries. It will also default the seccomp
      profile to runtime/default if unset, otherwise this seccomp profile is required.
  name: contour-seccomp-nonroot-v2
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: false
allowPrivilegedContainer: false
allowedCapabilities:
- NET_BIND_SERVICE
defaultAddCapabilities: null
fsGroup:
```

```
  type: RunAsAny
groups: []
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities:
- ALL
runAsUser:
  type: MustRunAsNonRoot
seLinuxContext:
  type: MustRunAs
seccompProfiles:
- runtime/default
supplementalGroups:
  type: RunAsAny
users: []
volumes:
- configMap
- downwardAPI
- emptyDir
- persistentVolumeClaim
- projected
- secret
```

The SCC is bound to the service accounts by using the following Role and RoleBinding:

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: contour-seccomp-nonroot-v2
  namespace: tanzu-system-ingress
rules:
- apiGroups:
  - security.openshift.io
  resourceNames:
  - contour-seccomp-nonroot-v2
  resources:
  - securitycontextconstraints
  verbs:
  - use
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: contour-seccomp-nonroot-v2
  namespace: tanzu-system-ingress
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: contour-seccomp-nonroot-v2
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: system:serviceaccounts:tanzu-system-ingress
```

Developer Conventions for OpenShift cluster

On OpenShift clusters, Developer Conventions must run with a custom SecurityContextConstraint (SCC) to enable compliance with restricted Kubernetes pod security standards. Tanzu Application Platform configures the following SCC for the Developer Convention's webhook when you configure the `kubernetes_distribution: openshift` key in the `tap-values.yaml` file.

Specification follows:

```
---
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  name: developer-conventions-scc
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: false
allowPrivilegedContainer: false
defaultAddCapabilities: null
fsGroup:
  type: RunAsAny
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities: null
runAsUser:
  type: MustRunAsNonRoot
seLinuxContext:
  type: MustRunAs
supplementalGroups:
  type: RunAsAny
volumes:
  - secret
seccompProfiles: []
groups:
  - system:serviceaccounts:developer-conventions
```

Tanzu Build Service for OpenShift cluster

On OpenShift clusters Tanzu Build Service must run with a custom Security Context Constraint (SCC) to enable compliance. Tanzu Application Platform configures the following SCC for Tanzu Build Service when you configure the `kubernetes_distribution: openshift` key in the `tap-values.yaml` file.

```
---
kind: SecurityContextConstraints
apiVersion: security.openshift.io/v1
metadata:
  name: tbs-restricted-scc-with-seccomp
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: false
allowPrivilegedContainer: false
allowedCapabilities:
  - NET_BIND_SERVICE
defaultAddCapabilities: null
fsGroup:
  type: RunAsAny
groups: []
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities:
  - ALL
runAsUser:
  type: MustRunAsNonRoot
seLinuxContext:
  type: MustRunAs
```

```
seccompProfiles:
  - runtime/default
supplementalGroups:
  type: RunAsAny
users: []
volumes:
  - configMap
  - downwardAPI
  - emptyDir
  - persistentVolumeClaim
  - projected
  - secret
```

It also applies the following RBAC to allow Tanzu Build Service services to use the SCC:

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    apps.tanzu.vmware.com/aggregate-to-workload: "true"
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  name: system:tbs:scc:restricted-with-seccomp
rules:
  - apiGroups:
      - security.openshift.io
    resourceNames:
      - tbs-restricted-scc-with-seccomp
    resources:
      - securitycontextconstraints
    verbs:
      - use
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: system:tbs:scc:restricted-with-seccomp
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:tbs:scc:restricted-with-seccomp
subjects:
  - kind: ServiceAccount
    namespace: build-service
    name: dependency-updater-serviceaccount
  - kind: ServiceAccount
    namespace: build-service
    name: dependency-updater-controller-serviceaccount
  - kind: ServiceAccount
    namespace: build-service
    name: secret-syncer-service-account
  - kind: ServiceAccount
    namespace: build-service
    name: warmer-service-account
  - kind: ServiceAccount
    namespace: build-service
    name: build-service-daemonset-serviceaccount
  - kind: ServiceAccount
    namespace: cert-injection-webhook
    name: cert-injection-webhook-sa
  - kind: ServiceAccount
    namespace: kpack
    name: kp-default-repository-serviceaccount
  - kind: ServiceAccount
    namespace: kpack
```

```
    name: kpack-pull-lifecycle-serviceaccount
  - kind: ServiceAccount
    namespace: kpack
    name: controller
  - kind: ServiceAccount
    namespace: kpack
    name: webhook
  - kind: ServiceAccount
    namespace: stacks-operator-system
    name: controller-manager
```

Install Tanzu Application Platform (GitOps)

GitOps is a set of practices and principles to manage Kubernetes infrastructure and application deployments using Git as the single source of truth. It promotes declarative configurations and automated workflows to ensure consistency, reliability, and traceability for your application deployments.

The key components involved in implementing GitOps with Kubernetes include:

- **Git as the single source of truth**: The desired state is stored in a Git repository. To change the cluster state, you must change it in the Git repository instead of modifying it directly on the cluster.
- **Declarative configuration**: GitOps follows a declarative approach, where the desired state is defined in the declarative configuration files.
- **Pull-based synchronization**: GitOps follows a pull-based model. Kubernetes cluster periodically pulls the desired state from the Git repository. This approach ensures that the cluster is always in sync with the desired configuration.

Caution

Tanzu Application Platform (GitOps) is currently in beta and is intended for evaluation and test purposes only. Do not use in a production environment.

How Tanzu RI supports GitOps

The Tanzu GitOps Reference Implementation (RI) is built upon Carvel, which shares the same packaging APIs as the Tanzu Application Platform. Carvel packaging APIs support all the GitOps features and enables a native GitOps flow.

- All the packaging APIs are declarative in nature.
- Among many options to fetch the manifest to be deployed, it can also pull the content from the Git repository, making Git the source of truth.
- Packages installed are reconciled every time after the SyncPeriod expires (10 minutes by default). As part of the reconciliation, it fetches the manifest from the Git repository and when the desired state is different from the actual state on Kubernetes, it converges the resources to their desired state declared in Git.

GitOps benefits

GitOps offers the following benefits:

- **Compliance and auditing capabilities**: In GitOps, Git is the single source of truth, enabling auditors to access a complete audit trail of all configuration changes.

- **Disaster recovery**: Disaster recovery involves an organization's efforts to restore access and function to its IT infrastructure. With all configurations securely stored in Git, disaster recovery becomes as straightforward as reapplying the desired configuration version.
- **Repeatable**: Running Tanzu CLI commands with environment variables or configuration files on a local machine is no longer required. Instead, all the necessary configurations and service accounts for access are configured in a shared Git repository. This approach allows any operator to make edits to a file, and the system's behavior remains independent of their local environment.

GitOps install paths

Choose one of the following install paths to install Tanzu Application Platform on your Kubernetes clusters through GitOps:

Note

To decide which approach to use, see Choosing SOPS or ESO.

GitOps with Secrets OPerationS (SOPS)

Applies to the scenario when you want a simple instance and store sensitive data encrypted in your Git repo:

Step	Task	Link
1.	Review the prerequisites to ensure you have met all requirements before installing.	Prerequisites
2.	Accept Tanzu Application Platform EULAs and install the Tanzu CLI.	Accept Tanzu Application Platform EULAs and installing the Tanzu CLI
3.	Install Cluster Essentials for Tanzu*.	Deploy Cluster Essentials
4.	Install Tanzu Application Platform.	Install Tanzu Application Platform through Gitops with Secrets OPerationS (SOPS)
5.	(Optional) Install any additional packages that were not in the profile.	Install individual packages
6.	Set up developer namespaces to use your installed packages.	Set up developer namespaces to use your installed packages
7.	Install developer tools into your integrated development environment (IDE).	Install Tanzu Developer Tools for your VS Code

GitOps with External Secrets Operator (ESO)

AWS Secrets Manager

Applies to the scenario when you want to store sensitive data in AWS Secrets Manager:

Step	Task	Link
1.	Review the prerequisites to ensure you have met all requirements before installing.	Prerequisites
2.	Accept Tanzu Application Platform EULAs and install the Tanzu CLI.	Accept Tanzu Application Platform EULAs and installing the Tanzu CLI
3.	Create AWS Resources such as EKS cluster and roles)	Create AWS Resources

Step	Task	Link
4.	Install Cluster Essentials for Tanzu*.	Deploy Cluster Essentials
5.	Install Tanzu Application Platform.	Install Tanzu Application Platform through GitOps using AWS Secrets Manager
6.	(Optional) Install any additional packages that were not in the profile.	Install individual packages
7.	Set up developer namespaces to use your installed packages.	Set up developer namespaces to use your installed packages
8.	Install developer tools into your integrated development environment (IDE).	Install Tanzu Developer Tools for your VS Code

HashiCorp Vault

Applies to the scenario when you want to store sensitive data in HashiCorp Vault:

Step	Task	Link
1.	Review the prerequisites to ensure you have met all requirements before installing.	Prerequisites
2.	Accept Tanzu Application Platform EULAs and install the Tanzu CLI.	Accept Tanzu Application Platform EULAs and installing the Tanzu CLI
3.	Install Cluster Essentials for Tanzu*.	Deploy Cluster Essentials
4.	Install Tanzu Application Platform.	Install Tanzu Application Platform through GitOps using HashiCorp Vault
5.	(Optional) Install any additional packages that were not in the profile.	Install individual packages
6.	Set up developer namespaces to use your installed packages.	Set up developer namespaces to use your installed packages
7.	Install developer tools into your integrated development environment (IDE).	Install Tanzu Developer Tools for your VS Code

** When you use a VMware Tanzu Kubernetes Grid cluster, there is no need to install Cluster Essentials because the contents of Cluster Essentials are already installed on your cluster.*

After installing Tanzu Application Platform on to your Kubernetes clusters, proceed with Get started with Tanzu Application Platform.

Install Tanzu Application Platform (GitOps)

GitOps is a set of practices and principles to manage Kubernetes infrastructure and application deployments using Git as the single source of truth. It promotes declarative configurations and automated workflows to ensure consistency, reliability, and traceability for your application deployments.

The key components involved in implementing GitOps with Kubernetes include:

- **Git as the single source of truth**: The desired state is stored in a Git repository. To change the cluster state, you must change it in the Git repository instead of modifying it directly on the cluster.
- **Declarative configuration**: GitOps follows a declarative approach, where the desired state is defined in the declarative configuration files.
- **Pull-based synchronization**: GitOps follows a pull-based model. Kubernetes cluster periodically pulls the desired state from the Git repository. This approach ensures that the

cluster is always in sync with the desired configuration.

Caution

Tanzu Application Platform (GitOps) is currently in beta and is intended for evaluation and test purposes only. Do not use in a production environment.

How Tanzu RI supports GitOps

The Tanzu GitOps Reference Implementation (RI) is built upon Carvel, which shares the same packaging APIs as the Tanzu Application Platform. Carvel packaging APIs support all the GitOps features and enables a native GitOps flow.

- All the packaging APIs are declarative in nature.
- Among many options to fetch the manifest to be deployed, it can also pull the content from the Git repository, making Git the source of truth.
- Packages installed are reconciled every time after the SyncPeriod expires (10 minutes by default). As part of the reconciliation, it fetches the manifest from the Git repository and when the desired state is different from the actual state on Kubernetes, it converges the resources to their desired state declared in Git.

GitOps benefits

GitOps offers the following benefits:

- **Compliance and auditing capabilities**: In GitOps, Git is the single source of truth, enabling auditors to access a complete audit trail of all configuration changes.
- **Disaster recovery**: Disaster recovery involves an organization's efforts to restore access and function to its IT infrastructure. With all configurations securely stored in Git, disaster recovery becomes as straightforward as reapplying the desired configuration version.
- **Repeatable**: Running Tanzu CLI commands with environment variables or configuration files on a local machine is no longer required. Instead, all the necessary configurations and service accounts for access are configured in a shared Git repository. This approach allows any operator to make edits to a file, and the system's behavior remains independent of their local environment.

GitOps install paths

Choose one of the following install paths to install Tanzu Application Platform on your Kubernetes clusters through GitOps:

Note

To decide which approach to use, see Choosing SOPS or ESO.

GitOps with Secrets OPerationS (SOPS)

Applies to the scenario when you want a simple instance and store sensitive data encrypted in your Git repo:

Step	Task	Link
1.	Review the prerequisites to ensure you have met all requirements before installing.	Prerequisites
2.	Accept Tanzu Application Platform EULAs and install the Tanzu CLI.	Accept Tanzu Application Platform EULAs and installing the Tanzu CLI
3.	Install Cluster Essentials for Tanzu*.	Deploy Cluster Essentials
4.	Install Tanzu Application Platform.	Install Tanzu Application Platform through Gitops with Secrets OPerationS (SOPS)
5.	(Optional) Install any additional packages that were not in the profile.	Install individual packages
6.	Set up developer namespaces to use your installed packages.	Set up developer namespaces to use your installed packages
7.	Install developer tools into your integrated development environment (IDE).	Install Tanzu Developer Tools for your VS Code

GitOps with External Secrets Operator (ESO)

AWS Secrets Manager

Applies to the scenario when you want to store sensitive data in AWS Secrets Manager:

Step	Task	Link
1.	Review the prerequisites to ensure you have met all requirements before installing.	Prerequisites
2.	Accept Tanzu Application Platform EULAs and install the Tanzu CLI.	Accept Tanzu Application Platform EULAs and installing the Tanzu CLI
3.	Create AWS Resources such as EKS cluster and roles)	Create AWS Resources
4.	Install Cluster Essentials for Tanzu*.	Deploy Cluster Essentials
5.	Install Tanzu Application Platform.	Install Tanzu Application Platform through GitOps using AWS Secrets Manager
6.	(Optional) Install any additional packages that were not in the profile.	Install individual packages
7.	Set up developer namespaces to use your installed packages.	Set up developer namespaces to use your installed packages
8.	Install developer tools into your integrated development environment (IDE).	Install Tanzu Developer Tools for your VS Code

HashiCorp Vault

Applies to the scenario when you want to store sensitive data in HashiCorp Vault:

Step	Task	Link
1.	Review the prerequisites to ensure you have met all requirements before installing.	Prerequisites
2.	Accept Tanzu Application Platform EULAs and install the Tanzu CLI.	Accept Tanzu Application Platform EULAs and installing the Tanzu CLI
3.	Install Cluster Essentials for Tanzu*.	Deploy Cluster Essentials
4.	Install Tanzu Application Platform.	Install Tanzu Application Platform through GitOps using HashiCorp Vault

Step	Task	Link
5.	(Optional) Install any additional packages that were not in the profile.	Install individual packages
6.	Set up developer namespaces to use your installed packages.	Set up developer namespaces to use your installed packages
7.	Install developer tools into your integrated development environment (IDE).	Install Tanzu Developer Tools for your VS Code

** When you use a VMware Tanzu Kubernetes Grid cluster, there is no need to install Cluster Essentials because the contents of Cluster Essentials are already installed on your cluster.*

After installing Tanzu Application Platform on to your Kubernetes clusters, proceed with Get started with Tanzu Application Platform.

Install Tanzu Application Platform through GitOps with External Secrets Operator (ESO)

Caution

Tanzu Application Platform (GitOps) is currently in beta and is intended for evaluation and test purposes only. Do not use in a production environment.

This topic tells you how to install Tanzu Application Platform (commonly known as TAP) through GitOps with secrets managed externally in AWS Secrets Manager.

Select a secrets manager for external secret storage:

- AWS Secrets Manager
- HashiCorp Vault

Important

Tanzu GitOps Reference Implementation (RI) does not support changing the secrets management strategy for a cluster.

Install Tanzu Application Platform through GitOps with AWS Secrets Manager

Caution

Tanzu Application Platform (GitOps) is currently in beta and is intended for evaluation and test purposes only. Do not use in a production environment.

This topic tells you how to install Tanzu Application Platform (commonly known as TAP) through GitOps with secrets managed externally in AWS Secrets Manager. To decide which approach to use, see Choosing Secrets OPerationS (SOPS) or External Secrets Operator (ESO).

Tanzu GitOps Reference Implementation (RI) does not support changing the secrets management strategy for a cluster, for example, SOPS to ESO. However, changing between AWS Secrets Manager and HashiCorp Vault is supported. The External Secrets Operator integration in this release of Tanzu GitOps RI is verified to support AWS Elastic Kubernetes Service (EKS) cluster with

AWS Secrets Manager. Other combinations of Kubernetes distribution and ESO providers are not verified.

Prerequisites

Before installing Tanzu Application Platform, ensure you have:

- Completed the Prerequisites.
- Created AWS Resources.
- Accepted Tanzu Application Platform EULA and installed Tanzu CLI with any required plug-ins.
- Installed Cluster Essentials for Tanzu.
- Installed eksctl CLI.

Relocate images to a registry

VMware recommends relocating the images from VMware Tanzu Network registry to your own container image registry before attempting installation. If you don't relocate the images, Tanzu Application Platform depends on VMware Tanzu Network for continued operation, and VMware Tanzu Network offers no uptime guarantees. The option to skip relocation is documented for evaluation and proof-of-concept only.

The supported registries are Harbor, Azure Container Registry, Google Container Registry, and Quay.io. See the following the documentation for instructions on setting up a registry:

- Harbor documentation
- Google Container Registry documentation
- Quay.io documentation

To relocate images from the VMware Tanzu Network registry to your registry:

1. Set up environment variables for installation use by running:

   ```
   export IMGPKG_REGISTRY_HOSTNAME_0=registry.tanzu.vmware.com
   export IMGPKG_REGISTRY_USERNAME_0=MY-TANZUNET-USERNAME
   export IMGPKG_REGISTRY_PASSWORD_0=MY-TANZUNET-PASSWORD
   export IMGPKG_REGISTRY_HOSTNAME_1=MY-REGISTRY
   export IMGPKG_REGISTRY_USERNAME_1=MY-REGISTRY-USER
   export IMGPKG_REGISTRY_PASSWORD_1=MY-REGISTRY-PASSWORD
   export INSTALL_REGISTRY_USERNAME=MY-REGISTRY-USER
   export INSTALL_REGISTRY_PASSWORD=MY-REGISTRY-PASSWORD
   export INSTALL_REGISTRY_HOSTNAME=MY-REGISTRY
   export TAP_VERSION=VERSION-NUMBER
   export INSTALL_REPO=TARGET-REPOSITORY
   ```

   Where:

   - `MY-REGISTRY-USER` is the user with write access to `MY-REGISTRY`.
   - `MY-REGISTRY-PASSWORD` is the password for `MY-REGISTRY-USER`.
   - `MY-REGISTRY` is your own container registry.
   - `MY-TANZUNET-USERNAME` is the user with access to the images in the VMware Tanzu Network registry `registry.tanzu.vmware.com`.
   - `MY-TANZUNET-PASSWORD` is the password for `MY-TANZUNET-USERNAME`.
   - `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.9.1`.

- `TARGET-REPOSITORY` is your target repository, a folder or repository on `MY-REGISTRY` that serves as the location for the installation files for Tanzu Application Platform.

VMware recommends using a JSON key file to authenticate with Google Container Registry. In this case, the value of `INSTALL_REGISTRY_USERNAME` is `_json_key` and the value of `INSTALL_REGISTRY_PASSWORD` is the content of the JSON key file. For more information about how to generate the JSON key file, see Google Container Registry documentation.

2. Install the Carvel tool `imgpkg` CLI.

   To query for the available versions of Tanzu Application Platform on VMWare Tanzu Network Registry, run:

   ```
   imgpkg tag list -i registry.tanzu.vmware.com/tanzu-application-platform/tap-packages | sort -V
   ```

3. Relocate the images with the `imgpkg` CLI by running:

   ```
   imgpkg copy -b registry.tanzu.vmware.com/tanzu-application-platform/tap-packages:${TAP_VERSION} --to-repo ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/tap-packages
   ```

(Optional) Install Tanzu Application Platform in an air-gapped environment

Complete the following steps if you install Tanzu Application Platform in an air-gapped environment:

1. Relocate the Tanzu Build Service images to your registry:

   ```
   imgpkg copy -b registry.tanzu.vmware.com/tanzu-application-platform/full-tbs-deps-package-repo:VERSION --to-repo ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/full-tbs-deps-package-repo
   ```

   Where `VERSION` is the version of Tanzu Build Service. You can retrieve this value by running `kubectl get package -n tap-install | grep buildservice`

2. Configure custom certificate authorities for Tanzu Developer Portal.
3. Host a `grype` database in the air-gapped environment. For more information, see Use vulnerability scanning in offline and air-gapped environments.

Create a new Git repository

Follow these steps to create a new Git repository:

1. In a hosted Git service, for example, GitHub or GitLab, create a new repository.

   This version of Tanzu GitOps RI supports authenticating to a hosted Git repository by using SSH and Basic Authentication.

2. Initialize a new Git repository:

   ```
   mkdir -p $HOME/tap-gitops
   cd $HOME/tap-gitops

   git init
   git remote add origin git@github.com:my-organization/tap-gitops.git
   ```

3. Set up the authentication method:

SSH
Create a read-only deploy key for this new repository (recommended) or SSH key for an account with read access to this repository. The private portion of this key is referred to as `GIT_SSH_PRIVATE_KEY`.

Basic Authentication
Have a user name with read access to the Git repository and password or personal access token for the same user.

Important

Only use one of `ssh` or `Basic Authentication`, not both.

Download and unpack Tanzu GitOps Reference Implementation (RI)

Follow these steps to download and unpack Tanzu GitOps Reference Implementation (RI):

1. Sign in to VMware Tanzu Network.
2. Go to the Tanzu Application Platform product page.
3. Select **Release 1.9.1** from the release drop-down menu.
4. Click **Tanzu GitOps Reference Implementation**.
5. Unpack the downloaded TGZ file into the `$HOME/tap-gitops` directory by running:

   ```
   tar -xvf tanzu-gitops-ri-*.tgz -C $HOME/tap-gitops
   ```

6. Commit the initial state:

   ```
   cd $HOME/tap-gitops

   git add . && git commit -m "Initialize Tanzu GitOps RI"
   git push -u origin
   ```

Create your cluster configuration

Follow these steps to create your cluster configuration:

1. Seed configuration for a cluster by using the provided convenience script:

   ```
   cd $HOME/tap-gitops

   ./setup-repo.sh CLUSTER-NAME aws-secrets-manager
   ```

   Where:

   - `CLUSTER-NAME` is the name for your cluster. Typically, this is the same as your EKS cluster's name, the name of the cluster as it appears in `eksctl get clusters`
   - `aws-secrets-manager` selects the AWS Secrets Manager external Secret Store.

   For example, if the name of your cluster is `iterate-green`:

   ```
   cd $HOME/tap-gitops
   ```

```
./setup-repo.sh iterate-green aws-secrets-manager
```

This script creates the directory `clusters/iterate-green/` and copies in the configuration required to sync this Git repository with the cluster and installing Tanzu Application Platform.

2. Commit and push:

```
git add . && git commit -m 'Add "iterate-green" cluster'
git push
```

Saving the base configuration in an initial commit makes it easier to review customizations in the future.

Customize your cluster configuration

Configuring the Tanzu Application Platform installation involves setting up two components:

- An installation of Tanzu Application Platform.
- An instance of Tanzu Sync, the component that implements the GitOps workflow, fetching configuration from Git and applying it to the cluster.

Follow these steps to customize your Tanzu Application Platform cluster configuration:

1. Navigate to the created directory:

```
cd clusters/CLUSTER-NAME
```

For example, if the name of your cluster is `iterate-green`:

```
cd clusters/iterate-green
```

2. Define the following environment variables:

```
export AWS_ACCOUNT_ID=MY-AWS-ACCOUNT-ID
export AWS_REGION=AWS-REGION
export CLUSTER_NAME=CLUSTER-NAME
export TAP_PKGR_REPO=TAP-PACKAGE-OCI-REPOSITORY
```

Where:

- `MY-AWS-ACCOUNT-ID` is your AWS account ID as it appears in the output of `aws sts get-caller-identity`.
- `AWS-REGION` is the region where the Secrets Manager is and the EKS cluster was created.
- `CLUSTER-NAME` is the name of the target cluster as it appears in the output of `eksctl get clusters`.
- `TAP-PACKAGE-OCI-REPOSITORY` is the fully-qualified path to the OCI repository hosting the Tanzu Application Platform images. If they are relocated to a different registry as described in Relocate images to a registry, the value is `${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/tap-packages`.

Grant read access to secret data

AWS Secrets Manager secrets store all sensitive configurations, which are accessed by both Tanzu Sync and the Tanzu Application Platform installation.

Follow these steps to configure the IAM Role for a Service Account:

1. In AWS Identity and Access Manager, create two IAM Policies, one to read the Tanzu Sync secrets and another to read the Tanzu Application Platform installation secrets, by using the supplied script:

   ```
   tanzu-sync/scripts/setup/create-policies.sh
   ```

2. Create two IAM Role-to-Service Account pairs for your cluster, one for Tanzu Sync and another for the Tanzu Application Platform installation, by using the supplied script:

   ```
   tanzu-sync/scripts/setup/create-irsa.sh
   ```

   For example, if the name of the EKS cluster is `iterate-green` using the defaults, there are two IAM roles in the AWS account:

   ```
   $ aws iam list-roles --query 'Roles[?starts_with(RoleName,`iterate-green`)]'
   [
       {
           "RoleName": "iterate-green--tanzu-sync-secrets",
           ...
       },
       {
           "RoleName": "iterate-green--tap-install-secrets",
           ...
       }
   ]
   ```

Generate the default configuration

You can use the following script to generate the default configuration for both Tanzu Sync and Tanzu Application Platform installation:

```
tanzu-sync/scripts/configure.sh
```

The following sections tell you how to edit the configuration values to suit your specific needs.

Review and store Tanzu Sync config

Configuration for Tanzu is stored in two locations:

- Sensitive configuration is stored in AWS Secrets Manager.
- Non-sensitive configuration are stored in YAML files in the Git repository.

Follow these steps to create the sensitive configuration and review the non-sensitive configuration:

1. Save the credentials that Tanzu Sync uses to authenticate with the Git repository.

   SSH
   Create a secret named `dev/CLUSTER-NAME/tanzu-sync/sync-git/ssh` containing the following information as plaintext:

   ```
   {
     "privatekey": "... (private key portion here) ...",
     "knownhosts": "... (known_hosts for git host here) ..."
   }
   ```

   Where `CLUSTER-NAME` is the name as it appears in `eksctl get clusters`.

   For example, if the Git repository is hosted on GitHub, and the private key created in Create a new Git repository is stored in the file `~/.ssh/id_ed25519`:

```
aws secretsmanager create-secret \
  --name dev/${CLUSTER_NAME}/tanzu-sync/sync-git/ssh \
  --secret-string "$(cat <<EOF
{
  "privatekey": "$(cat ~/.ssh/id_ed25519 | awk '{printf "%s\\n", $0}')",
  "knownhosts": "$(ssh-keyscan github.com | awk '{printf "%s\\n", $0}')"
}
EOF
)"
```

Where:

- The content of `~/.ssh/id_ed25519` is the private portion of the SSH key.
- `ssh-keyscan` obtains the public keys for the SSH host.
- `awk '{printf "%s\n", $0}'` converts a multiline string into a single-line string with embedded newline chars (`\n`). JSON does not support multiline strings.

Basic Authentication

Create a secret named `dev/CLUSTER-NAME/tanzu-sync/sync-git/basic_auth` containing the following information as plaintext:

```
{
  "username": "... (username) ...",
  "password": "... (password) ..."
}
```

Where:

- `CLUSTER-NAME` is the name as it appears in `eksctl get clusters`.
- `username` is the username of a user account with read access to the Git repository.
- `password` is the password or personal access token for the user.

2. To securely store the authentication credentials required for accessing the OCI registry that hosts the Tanzu Application Platform images, create a secret called `dev/CLUSTER-NAME/tanzu-sync/install-registry-dockerconfig`. This secret contains the following information in plaintext:

```
{
  "auths": {
    "MY-REGISTRY": {
      "username": "MY-REGISTRY-USER",
      "password": "MY-REGISTRY-PASSWORD"
    }
  }
}
```

Where:

- `CLUSTER-NAME` is the name as it appears in `eksctl get clusters`
- `MY-REGISTRY-USER` is the user with write access to `MY-REGISTRY`.
- `MY-REGISTRY-PASSWORD` is the password for `MY-REGISTRY-USER`.
- `MY-REGISTRY` is the container registry where the Tanzu Application Platform images are located.

For example:

```
aws secretsmanager create-secret \
   --name dev/${CLUSTER_NAME}/tanzu-sync/install-registry-dockerconfig \
   --secret-string "$(cat <<EOF
{
 "auths": {
   "${INSTALL_REGISTRY_HOSTNAME}": {
     "username": "${INSTALL_REGISTRY_USERNAME}",
     "password": "${INSTALL_REGISTRY_PASSWORD}"
   }
 }
}
EOF
)"
```

3. Review the hosted Git URL and branch used by Tanzu Sync.

   This configuration was generated by the `configure.sh` script. It reported:

   ```
   ...
   wrote non-sensitive Tanzu Sync configuration to: tanzu-sync/app/values/tanzu-sy
   nc.yaml
   ...
   ```

   For example, for the `iterate-green` cluster, if the Git repository is hosted on GitHub under `my-organization/tap-gitops` and is on the `main` branch, `tanzu-sync.yaml` contains the following information:

   ```
   ---
   git:
    url: git@github.com:my-organization/tap-gitops.git
    ref: origin/main
    sub_path: clusters/iterate-green/cluster-config
   ```

   You can review and edit these values as needed.

4. Review the integration with External Secrets Operator.

   This configuration was generated by the `configure.sh` script. It reported:

   ```
   ...
   wrote ESO configuration for Tanzu Sync to: tanzu-sync/app/values/tanzu-sync-aws
   -secrets-manager-values.yaml
   ...
   ```

   For example, for the `iterate-green` cluster, if the AWS account is `665100000000`, `tanzu-sync-aws-secrets-manager-values.yaml` contains the following information:

   ```
   ---
   secrets:
    eso:
      aws:
        region: us-west-2
        tanzu_sync_secrets:
          role_arn: arn:aws:iam::665100000000:role/iterate-green--tanzu-sync-secre
   ts
      remote_refs:
        sync_git:
        # TO DO: Fill in your configuration for ssh or basic authentication here.
   See tanzu-sync/app/config/.tanzu-managed/schema--eso.yaml for details.
        install_registry_dockerconfig:
          dockerconfigjson:
            key: dev/iterate-green/tanzu-sync/install-registry-dockerconfig
   ```

Where:

- `role_arn` is the IAM role that grants permission to Tanzu Sync to read secrets specific to Tanzu Sync. This role was created in the Grant read access to secret data section.
- `install_registry_dockerconfig` contains the AWS Secrets Manager secret name that contains the Docker config authentication to the OCI registry hosting the Tanzu Application Platform images created earlier.

5. Replace any `TO DO` sections from line 10 in the earlier example with the relevant values.

   Configuration example for SSH authentication:

```
---
secrets:
 eso:
  aws:
    region: us-west-2
    tanzu_sync_secrets:
      role_arn: arn:aws:iam::665100000000:role/iterate-green--tanzu-sync-secret
s
  remote_refs:
    sync_git:
      ssh:
        private_key:
          key: dev/iterate-green/tanzu-sync/sync-git/ssh
          property: privatekey
        known_hosts:
          key: dev/iterate-green/tanzu-sync/sync-git/ssh
          property: knownhosts
    install_registry_dockerconfig:
      dockerconfigjson:
        key: dev/iterate-green/tanzu-sync/install-registry-dockerconfig
```

   Configuration example for basic authentication:

```
---
secrets:
 eso:
  aws:
    region: us-west-2
    tanzu_sync_secrets:
      role_arn: arn:aws:iam::665100000000:role/iterate-green--tanzu-sync-secret
s
  remote_refs:
    sync_git:
      basic_auth:
        username:
          key: dev/iterate-green/tanzu-sync/sync-git/basic_auth
          property: username
        password:
          key: dev/iterate-green/tanzu-sync/sync-git/basic_auth
          property: password
    install_registry_dockerconfig:
      dockerconfigjson:
        key: dev/iterate-green/tanzu-sync/install-registry-dockerconfig
```

6. Commit the Tanzu Sync configuration.

   For example, for the “iterate-green” cluster, run:

```
git add tanzu-sync/
git commit -m 'Configure Tanzu Sync on "iterate-green"'
```

Review and store the Tanzu Application Platform installation config

Configuration for the Tanzu Application Platform installation is stored in two places:

- Sensitive configuration is stored in AWS Secrets Manager.
- Non-sensitive configuration is stored in YAML files in the Git repository.

Follow these steps to create the sensitive configuration and review the non-sensitive configuration:

1. Create a secret named `dev/${CLUSTER_NAME}/tap/sensitive-values.yaml` that stores the sensitive data such as username, password, private key from the `tap-values.yaml` file:

   ```
   aws secretsmanager create-secret \
      --name dev/${CLUSTER_NAME}/tap/sensitive-values.yaml \
      --secret-string "$(cat <<EOF
   ---
   # this document is intentionally initially blank.
   EOF
   )"
   ```

   You can start with an empty document and edit it later on as described in the Configure and push the Tanzu Application Platform values section.

2. Review the integration with External Secrets Operator.

   This configuration was generated by the `configure.sh` script. It reported:

   ```
   ...
   wrote AWS Secrets Manager configuration for TAP Install to: cluster-config/valu
   es/tap-install-aws-secrets-manager-values.yaml
   ...
   ```

   For example, for the `iterate-green` cluster, if the AWS account is `665100000000`, `tap-install-aws-secrets-manager-values.yaml` contains the following information:

   ```
   ---
   tap_install:
    secrets:
      eso:
        aws:
          region: us-west-2
          tap_install_secrets:
            role_arn: arn:aws:iam:665100000000:iterate-green--tap-install-secrets
        remote_refs:
          tap_sensitive_values:
            sensitive_tap_values_yaml:
              key: dev/iterate-green/tap/sensitive-values.yaml
   ```

   Where:

   - `role_arn` is the IAM role that grants permission to the Tanzu Application Platform installation to read its associated secrets. This role was created in the Grant read access to secret data section.
   - `sensitive_tap_values_yaml.key` is the AWS Secrets Manager secret name that contains the sensitive data from the `tap-values.yaml` file for this cluster in a YAML format.

3. (Optional) Update Tanzu Application Platform to use the latest patch:

   Update the Tanzu Application Platform version in `GIT-REPO-ROOT/clusters/CLUSTER-NAME/cluster-config/values/tap-install-values.yaml`:

```
tap_install:
    ...
    version:
        package_repo_bundle_tag: "1.9.1" # Populate these values with the lates
t patch version.
        package_version: "1.9.1"
```

Where:

- `package_repo_bundle_tag` is the version of Tanzu Application Platform you want to upgrade to.
- `package_version` is the version of Tanzu Application Platform you want to upgrade to. This version must match `package_repo_bundle_tag`.

Note

Tanzu GitOps RI does not provide a separate artifact for each patch version within a minor line. For example, Tanzu Application Platform v1.6.x only contains the v1.6.1 GitOps artifact.

4. Commit the Tanzu Application Platform installation configuration.

   For example, for the `iterate-green` cluster, run:

```
git add cluster-config/
git commit -m 'Configure installer for TAP 1.6.1 on "iterate-green"'
```

Configure and push the Tanzu Application Platform values

The configuration for the Tanzu Application Platform is divided into two separate locations:

- Sensitive configuration is stored in a AWS Secrets Manager secret created as described in the Review and store Tanzu Application Platform installation config section.
- Non-sensitive configuration is stored in a plain YAML file `cluster-config/values/tap-values.yaml`

Follow these steps to split the Tanzu Application Platform values:

1. Create the `cluster-config/values/tap-values.yaml` file by using the Full Profile (AWS), which contains the minimum configurations required to deploy Tanzu Application Platform on AWS.

   The Tanzu Application Platform values are input configurations to the Tanzu Application Platform installation and are placed under the `tap_install.values` path.

```
tap_install:
  values:
    # Tanzu Application Platform values go here.
    shared:
      ingress_domain: "INGRESS-DOMAIN"
    ceip_policy_disclosed: true
...
```

   To install Tanzu Application Platform in an offline environment, you must configure `Tanzu Build Service` and `Grype` to work in an air-gapped environment:

```
---
tap_install:
```

```
  values:
    ...
    buildservice:
      exclude_dependencies: true
    grype:
      db:
        dbUpdateUrl: INTERNAL-VULN-DB-URL
```

Where `INTERNAL-VULN-DB-URL` is the URL that points to the internal file server.

For more information, see Components and installation profiles.

2. Review the contents of `tap-values.yaml` and move all the sensitive values into the AWS Secrets Store secret created in the Review and store Tanzu Application Platform installation config section.

   For example, if the `iterate-green` cluster is configured with the basic Out of the Box Supply Chain, this might include a passphrase for that supply chain's GitOps flow:

   ```
   ---
   tap_install:
    values:
      ...
      ootb_supply_chain_basic:
        registry:
          server: "SERVER-NAME"
          repository: "REPO-NAME"
        gitops:
          ssh_secret: "SSH-SECRET-KEY"
      ...
   ```

   To maintain the secrecy of `ootb_supply_chain_basic.gitops.ssh_secret`, move this value from the `tap-values.yaml` file:

   ```
   ---
   tap_install:
    values:
      ...
      ootb_supply_chain_basic:
        registry:
          server: "SERVER-NAME"
          repository: "REPO-NAME"
      ...
   ```

   Add it to the AWS Secrets Store secret named `dev/iterate-green/tap/sensitive-values.yaml`, by default, without the `tap_install.values` root:

   ```
   ---
   ...
   ootb_supply_chain_basic:
    gitops:
      ssh_secret: "SSH-SECRET-KEY"
   ...
   ```

   To update the secret value, follow the instructions in Modify an AWS Secrets Manager secret.

   When moving values, you must omit the `tap_install.values` root, but keep the remaining structure. All of the parent keys, such as `ootb_supply_chain_basic.gitops` and `ssh_secret`, must be copied to the sensitive value YAML.

3. Commit and push the Tanzu Application Platform values:

```
git add cluster-config/
git commit -m "Configure initial values for TAP 1.6.1"
git push
```

Tanzu Sync fetches configuration from the hosted clone of the Git repository. For changes to take effect on the cluster, they must be pushed to that clone of the Git repository.

Deploy Tanzu Sync

Deploying Tanzu Sync starts the GitOps workflow that initiates the Tanzu Application Platform installation.

After deployed, Tanzu Sync periodically polls the Git repository for changes. The following deployment process is only required once per cluster:

1. Install the Carvel tools `kapp` and `ytt` onto your `$PATH`:

   ```
   sudo cp $HOME/tanzu-cluster-essentials/kapp /usr/local/bin/kapp
   sudo cp $HOME/tanzu-cluster-essentials/ytt /usr/local/bin/ytt
   ```

   This step is required to ensure the successful deployment of the `tanzu-sync` app.

2. Ensure the Kubernetes cluster context is set to the EKS cluster.

   1. List the existing contexts:

      ```
      kubectl config get-contexts
      ```

   2. Set the context to the cluster that you want to deploy:

      ```
      kubectl config use-context CONTEXT-NAME
      ```

      Where `CONTEXT-NAME` can be retrieved from the outputs of the previous step.

3. Bootstrap the deployment.

   External Secrets Operator is installed from the package included in the Tanzu Application Platform package repository. That repository must be fetched from the OCI registry initially.

   1. Set the following environment variables:

      ```
      export INSTALL_REGISTRY_HOSTNAME=MY-REGISTRY
      export INSTALL_REGISTRY_USERNAME=MY-REGISTRY-USER
      export INSTALL_REGISTRY_PASSWORD=MY-REGISTRY-PASSWORD
      ```

      Where:

      - `MY-REGISTRY` is your container registry.
      - `MY-REGISTRY-USER` is the user with read access to `MY-REGISTRY`.
      - `MY-REGISTRY-PASSWORD` is the password for `MY-REGISTRY-USER`.

   2. Create a secret containing credentials to fetch from that OCI registry by using the provided script:

      ```
      tanzu-sync/scripts/bootstrap.sh
      ```

      These credentials are used exactly once to install the External Secrets Operator (ESO) package.

4. Install Tanzu Sync and start the GitOps workflow by deploying it to the cluster using `kapp` and `ytt`.

```
tanzu-sync/scripts/deploy.sh
```

Depending on the profile and components included, it may take 5-10 minutes for the Tanzu Application Platform to install. During this time, `kapp` waits for the deployment of Tanzu Sync to reconcile successfully. This is normal.

You can track the progress of the installation by watching the installation of those packages in a separate terminal window:

```
watch kubectl get pkgi -n tap-install
```

Install Tanzu Application Platform through GitOps with HashiCorp Vault

Caution

Tanzu Application Platform (GitOps) is currently in beta and is intended for evaluation and test purposes only. Do not use in a production environment.

This topic tells you how to install Tanzu Application Platform (commonly known as TAP) through GitOps with secrets managed in an external secrets store. To decide which approach to use, see Choosing Secrets OPerationS (SOPS) or External Secrets Operator (ESO).

Tanzu GitOps Reference Implementation (RI) does not support changing the secrets management strategy for a cluster, for example, SOPS to ESO. However, changing between AWS Secrets Manager and HashiCorp Vault is supported. The External Secrets Operator integration in this release of Tanzu GitOps RI is verified to support Kubernetes integration with HashiCorp Vault.

Prerequisites

Before installing Tanzu Application Platform, ensure you have:

- Completed the Prerequisites.
- Accepted Tanzu Application Platform EULA and installed Tanzu CLI with any required plug-ins.
- Installed Cluster Essentials for Tanzu.
- Installed eksctl CLI.

Relocate images to a registry

VMware recommends relocating the images from VMware Tanzu Network registry to your own container image registry before attempting installation. If you don't relocate the images, Tanzu Application Platform depends on VMware Tanzu Network for continued operation, and VMware Tanzu Network offers no uptime guarantees. The option to skip relocation is documented for evaluation and proof-of-concept only.

The supported registries are Harbor, Azure Container Registry, Google Container Registry, and Quay.io. See the following the documentation for instructions on setting up a registry:

- Harbor documentation
- Google Container Registry documentation
- Quay.io documentation

To relocate images from the VMware Tanzu Network registry to your registry:

1. Set up environment variables for installation use by running:

   ```
   export IMGPKG_REGISTRY_HOSTNAME_0=registry.tanzu.vmware.com
   export IMGPKG_REGISTRY_USERNAME_0=MY-TANZUNET-USERNAME
   export IMGPKG_REGISTRY_PASSWORD_0=MY-TANZUNET-PASSWORD
   export IMGPKG_REGISTRY_HOSTNAME_1=MY-REGISTRY
   export IMGPKG_REGISTRY_USERNAME_1=MY-REGISTRY-USER
   export IMGPKG_REGISTRY_PASSWORD_1=MY-REGISTRY-PASSWORD
   export INSTALL_REGISTRY_USERNAME=MY-REGISTRY-USER
   export INSTALL_REGISTRY_PASSWORD=MY-REGISTRY-PASSWORD
   export INSTALL_REGISTRY_HOSTNAME=MY-REGISTRY
   export TAP_VERSION=VERSION-NUMBER
   export INSTALL_REPO=TARGET-REPOSITORY
   ```

   Where:

   - `MY-REGISTRY-USER` is the user with write access to `MY-REGISTRY`.
   - `MY-REGISTRY-PASSWORD` is the password for `MY-REGISTRY-USER`.
   - `MY-REGISTRY` is your own container registry.
   - `MY-TANZUNET-USERNAME` is the user with access to the images in the VMware Tanzu Network registry `registry.tanzu.vmware.com`.
   - `MY-TANZUNET-PASSWORD` is the password for `MY-TANZUNET-USERNAME`.
   - `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.9.1`.
   - `TARGET-REPOSITORY` is your target repository, a folder or repository on `MY-REGISTRY` that serves as the location for the installation files for Tanzu Application Platform.

   VMware recommends using a JSON key file to authenticate with Google Container Registry. In this case, the value of `INSTALL_REGISTRY_USERNAME` is `_json_key` and the value of `INSTALL_REGISTRY_PASSWORD` is the content of the JSON key file. For more information about how to generate the JSON key file, see Google Container Registry documentation.

2. Install the Carvel tool `imgpkg` CLI.

   To query for the available versions of Tanzu Application Platform on VMWare Tanzu Network Registry, run:

   ```
   imgpkg tag list -i registry.tanzu.vmware.com/tanzu-application-platform/tap-pac
   kages | sort -V
   ```

3. Relocate the images with the `imgpkg` CLI by running:

   ```
   imgpkg copy -b registry.tanzu.vmware.com/tanzu-application-platform/tap-package
   s:${TAP_VERSION} --to-repo ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/tap-pac
   kages
   ```

(Optional) Install Tanzu Application Platform in an air-gapped environment

Complete the following steps if you install Tanzu Application Platform in an air-gapped environment:

1. Relocate the Tanzu Build Service images to your registry:

   ```
   imgpkg copy -b registry.tanzu.vmware.com/tanzu-application-platform/full-tbs-de
   ps-package-repo:VERSION --to-repo ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/
   ```

```
full-tbs-deps-package-repo
```

Where:

- `VERSION` is the version of Tanzu Build Service. You can retrieve this value by running `kubectl get package -n tap-install | grep buildservice`

2. Configure custom certificate authorities for Tanzu Developer Portal.
3. Host a `grype` database in the air-gapped environment. For more information, see Use vulnerability scanning in offline and air-gapped environments.

Create a new Git repository

1. In a hosted Git service, for example, GitHub or GitLab, create a new repository.

   This version of Tanzu GitOps RI supports authenticating to a hosted Git repository by using SSH as well as Basic Authentication.

2. Initialize a new Git repository:

```
mkdir -p $HOME/tap-gitops
cd $HOME/tap-gitops

git init
git remote add origin git@github.com:my-organization/tap-gitops.git
```

3. Set up the authentication method.

SSH
Create a read-only deploy key for this new repository (recommended) or SSH key for an account with read access to this repository. The private portion of this key is referred to as `GIT_SSH_PRIVATE_KEY`.

Basic Authentication
Have a username with read access to the Git repository and password or personal access token for the same user.

Important

Only use one of `ssh` or `Basic Authentication`, not both.

Download and unpack Tanzu GitOps Reference Implementation (RI)

1. Sign in to VMware Tanzu Network.
2. Go to the Tanzu Application Platform product page.
3. Select **Release 1.9.1** from the release drop-down menu.
4. Click **Tanzu GitOps Reference Implementation**.
5. Unpack the downloaded TGZ file into the `$HOME/tap-gitops` directory by running:

```
tar -xvf tanzu-gitops-ri-*.tgz -C $HOME/tap-gitops
```

6. Commit the initial state:

```
cd $HOME/tap-gitops

git add . && git commit -m "Initialize Tanzu GitOps RI"
git push -u origin
```

Create cluster configuration

1. Seed configuration for a cluster using Vault through the provided convenience script:

   ```
   cd $HOME/tap-gitops

   ./setup-repo.sh CLUSTER-NAME vault
   ```

   Where:

   - `CLUSTER-NAME` is the name for your cluster. Typically, this is the same as your EKS cluster's name, the name of the cluster as it appears in `eksctl get clusters`.
   - `vault` selects the Vault external Secret Store.

   For example, if the name of your cluster is `iterate-green`:

   ```
   cd $HOME/tap-gitops

   ./setup-repo.sh iterate-green vault
   ```

   This script creates the directory `clusters/iterate-green/` and copies in the configuration required to sync this Git repository with the cluster and installing Tanzu Application Platform.

2. Commit and push:

   ```
   git add . && git commit -m 'Add "iterate-green" cluster'
   git push
   ```

   Saving the base configuration in an initial commit makes it easier to review customizations in the future.

Customize cluster configuration

Configuring the Tanzu Application Platform installation involves setting up two components:

- An installation of Tanzu Application Platform.
- An instance of Tanzu Sync, the component that implements the GitOps workflow, fetching configuration from Git and applying it to the cluster.

Follow these steps to customize your Tanzu Application Platform cluster configuration:

1. Navigate to the created directory:

   ```
   cd clusters/CLUSTER-NAME
   ```

   For example, if the name of your cluster is `iterate-green`:

   ```
   cd clusters/iterate-green
   ```

2. Define the following environment variables:

   ```
   export VAULT_ADDR=MY-VAULT-ADDR
   export CLUSTER_NAME=CLUSTER-NAME
   ```

```
export TAP_PKGR_REPO=TAP-PACKAGE-OCI-REPOSITORY
```

Where:

- `MY-VAULT-ADDR` is the accessible URL of your vault instance (Vault must be reachable by the Kubernetes cluster).
- `CLUSTER-NAME` is the name of the target cluster.
- `TAP-PACKAGE-OCI-REPOSITORY` is the fully-qualified path to the OCI repository hosting the Tanzu Application Platform images. If they are relocated to a different registry as described in Relocate images to a registry, the value is `${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/tap-packages`.

Connect Vault to a Kubernetes cluster

Tanzu GitOps RI uses the Vault Kubernetes authentication method for establishing trust between the Kubernetes cluster and Vault, see Vault Kubernetes auth for more. This authentication method uses the Kubernetes Control Plane TokenReview API to authenticate the Kubernetes service accounts with Vault. For this reason, the clusters control plane must be able to communicate over the network to the Vault instance.

To configure Kubernetes authentication for Vault, you can create a new Kubernetes authentication engine instance on Vault and two IAM Roles by using the supplied script:

```
tanzu-sync/scripts/setup/create-kubernetes-auth.sh
```

Important

- If you use an Enterprise Vault Server with namespaces, run `export VAULT_NAMESPACE=MY-VAULT-NAMESPACE` before using the script.
- If you use token to access server, run `export VAULT_TOKEN=MY-VAULT-TOKEN` before using the script.

This creates a new vault Kubernetes authentication instance using the information for the current context in your `KUBECONFIG`.

Example:

```
vault write auth/iterate-green/config \
  kubernetes_host="MY-KUBERNETES-API-URL" \
  kubernetes_ca_cert="MY-KUBERNETES-CA-CERT" \
  ttl=1h
```

Grant read access to secret data

Vault secrets store all sensitive configurations, which are accessed by both Tanzu Sync and the Tanzu Application Platform installation.

Follow these steps to configure Roles in Vault:

1. Create two Policies, one to read the Tanzu Sync secrets and another to read the Tanzu Application Platform installation secrets, by using the supplied script:

   ```
   tanzu-sync/scripts/setup/create-policies.sh
   ```

2. Create two Roles, one to read the Tanzu Sync secrets and another to read the Tanzu Application Platform installation secrets, by using the supplied script:

```
tanzu-sync/scripts/setup/create-roles.sh
```

Generate default configuration

You can use the following script to generate default configuration for both Tanzu Sync and Tanzu Application Platform installation:

```
tanzu-sync/scripts/configure.sh
```

The following sections tell you how to edit the configuration values to suit your specific needs.

Review and store Tanzu Sync config

Configuration for Tanzu is stored in two locations:

- Sensitive configuration is stored in Vault.
- Non-sensitive configuration are stored in YAML files in the Git repository.

Follow these steps to create the sensitive configuration and review the non-sensitive configuration:

Important

You must enable a key-value secret engine named `secret`.

1. Save the credentials that Tanzu Sync uses to authenticate with the Git repository. There are two supported authentication methods:

   SSH
   Create a secret named `secret/dev/CLUSTER-NAME/tanzu-sync/sync-git/ssh` containing the following information as plaintext:

   ```
   {
     "privatekey": "... (private key portion here) ...",
     "knownhosts": "... (known_hosts for git host here) ..."
   }
   ```

   Where `CLUSTER-NAME` is the name of your cluster.

   For example, if the Git repository is hosted on GitHub, and the private key created in Create a new Git repository is stored in the file `~/.ssh/id_ed25519`:

   ```
   export GIT_SSH_PRIVATE_KEY_FILE=~/.ssh/id_ed25519
   export GIT_KNOWN_HOSTS=$(ssh-keyscan github.com)
   printf '%s\n'  "$(cat <<EOF
   {
       "privatekey": "$(cat $GIT_SSH_PRIVATE_KEY_FILE | awk '{printf "%s\\n",
   $0}')",
       "knownhosts": "$(echo $GIT_KNOWN_HOSTS | awk '{printf "%s\\n", $0}')"
   }
   EOF
   )" | vault kv put secret/dev/${CLUSTER_NAME}/tanzu-sync/sync-git/ssh -
   ```

   Where:

   - The content of `~/.ssh/id_ed25519` is the private portion of the SSH key.
   - `ssh-keyscan` obtains the public keys for the SSH host.

- `awk '{printf "%s\\n", $0}'` converts a multiline string into a single-line string with embedded newline chars (`\n`). JSON does not support multiline strings.

Basic Authentication

Create a secret named `secret/dev/CLUSTER-NAME/tanzu-sync/sync-git/basic_auth` containing the following information as plaintext:

```
{
  "username": "... (username) ...",
  "password": "... (password) ..."
}
```

Where:

- `CLUSTER-NAME` is the name of your cluster.
- `username` is the username of a user account with read access to the Git repository.
- `password` is the password or personal access token for the user.

2. To securely store the authentication credentials required for accessing the OCI registry hosting the Tanzu Application Platform images, create a secret called `dev/CLUSTER-NAME/tanzu-sync/install-registry-dockerconfig`. This secret contains the following information in plaintext:

```
{
  "auths": {
    "MY-REGISTRY": {
      "username": "MY-REGISTRY-USER",
      "password": "MY-REGISTRY-PASSWORD"
    }
  }
}
```

Where:

- `CLUSTER-NAME` is the name of your cluster.
- `MY-REGISTRY-USER` is the user with write access to `MY-REGISTRY`.
- `MY-REGISTRY-PASSWORD` is the password for `MY-REGISTRY-USER`.
- `MY-REGISTRY` is the container registry where the Tanzu Application Platform images are located.

For example:

```
printf '%s\n'  "$(cat <<EOF
{
  "auths": {
      "${INSTALL_REGISTRY_HOSTNAME}": {
          "username": "${INSTALL_REGISTRY_USERNAME}",
          "password": "${INSTALL_REGISTRY_PASSWORD}"
      }
  }
}
EOF
)" | vault kv put secret/dev/${CLUSTER_NAME}/tanzu-sync/install-registry-docker
config -
```

3. Review the hosted Git URL and branch used by Tanzu Sync.

   This configuration was generated by the `configure.sh` script. It reported:

```
...
wrote non-sensitive Tanzu Sync configuration to: tanzu-sync/app/values/tanzu-sy
nc.yaml
...
```

For example, for the `iterate-green` cluster, if the Git repository is hosted on GitHub at `my-organization/tap-gitops` on the `main` branch, `tanzu-sync.yaml` contains the following information:

```
---
git:
 url: git@github.com:my-organization/tap-gitops.git
 ref: origin/main
 sub_path: clusters/iterate-green/cluster-config
```

You can review and edit these values as needed.

4. Review the integration with External Secrets Operator.

   This configuration was generated by the `configure.sh` script. It reported:

   ```
   ...
   wrote ESO configuration for Tanzu Sync to: tanzu-sync/app/values/tanzu-sync-vau
   lt-values.yaml
   ...
   ```

   For example, for the `iterate-green` cluster, if the Vault URL is `https://vault.example.com`, `tanzu-sync-vault-values.yaml` contains the following information:

   ```
   ---
   secrets:
    eso:
     vault:
       server: https://vault.example.com
       namespace: ""
       auth:
         kubernetes:
           mountPath: iterate-green
           role: iterate-green--tanzu-sync-secrets
     remote_refs:
       sync_git:
       # TO DO: Fill in your configuration for ssh or basic authentication here. S
   ee tanzu-sync/app/config/.tanzu-managed/schema--eso.yaml for details.
       install_registry_dockerconfig:
         dockerconfigjson:
           key: secret/dev/iterate-green/tanzu-sync/install-registry-dockerconfig
   ```

   Where:

   - `kubernetes.role` is the IAM role that grants permission to Tanzu Application Platform installation to read its associated secrets. This role was created in the Grant read access to secret data section.
   - `install_registry_dockerconfig` contains the Vault secret name that contains the Docker config authentication to the OCI registry hosting the Tanzu Application Platform images created earlier.

5. Replace any `TO DO` sections from line 12 in the earlier example with the relevant values.

   Configuration example for SSH authentication:

   ```
   ---
   secrets:
   ```

```
 eso:
  vault:
    server: https://vault.example.com
    namespace: ""
    auth:
      kubernetes:
        mountPath: iterate-green
        role: iterate-green--tanzu-sync-secrets
  remote_refs:
    sync_git:
      ssh:
        private_key:
          key: secret/dev/iterate-green/tanzu-sync/sync-git/ssh
          property: privatekey
        known_hosts:
          key: secret/dev/iterate-green/tanzu-sync/sync-git/ssh
          property: knownhosts
    install_registry_dockerconfig:
      dockerconfigjson:
        key: secret/dev/iterate-green/tanzu-sync/install-registry-dockerconfig
```

Configuration example for basic authentication:

```
---
secrets:
 eso:
  vault:
    server: https://vault.example.com
    namespace: ""
    auth:
      kubernetes:
        mountPath: iterate-green
        role: iterate-green--tanzu-sync-secrets
  remote_refs:
    sync_git:
      basic_auth:
        username:
          key: secret/dev/iterate-green/tanzu-sync/sync-git/basic_auth
          property: username
        password:
          key: secret/dev/iterate-green/tanzu-sync/sync-git/basic_auth
          property: password
    install_registry_dockerconfig:
      dockerconfigjson:
        key: secret/dev/iterate-green/tanzu-sync/install-registry-dockerconfig
```

6. (Optional) Update Tanzu Application Platform to use the latest patch:

   Update the Tanzu Application Platform version in `GIT-REPO-ROOT/clusters/CLUSTER-NAME/cluster-config/values/tap-install-values.yaml`:

```
tap_install:
    ...
    version:
        package_repo_bundle_tag: "1.9.1" # Populate these values with the lates
t patch version.
        package_version: "1.9.1"
```

   Where:

   - `package_repo_bundle_tag` is the version of Tanzu Application Platform you want to upgrade to.
   - `package_version` is the version of Tanzu Application Platform you want to upgrade to. This version must match `package_repo_bundle_tag`.

> **Note**
>
> Tanzu Application Platform (GitOps) does not provide a separate artifact for each patch version within a minor line. For example, Tanzu Application Platform v1.6.x only contains the v1.6.1 GitOps artifact.

7. Commit the Tanzu Sync configuration.

   For example, for the “iterate-green” cluster, run:

   ```
   git add tanzu-sync/
   git commit -m 'Configure Tanzu Sync on "iterate-green"'
   ```

Review and store the Tanzu Application Platform installation config

Configuration for the Tanzu Application Platform installation is stored in two places:

- Sensitive configuration is stored in Vault.
- Non-sensitive configuration is stored in YAML files in the Git repository.

Follow these steps to create the sensitive configuration and review the non-sensitive configuration:

1. Create a secret named `secret/dev/${CLUSTER_NAME}/tap/sensitive-values.yaml` that stores the sensitive data such as username, password, private key from the `tap-values.yaml` file:

   ```
   printf '%s\n'  "$(cat <<EOF
   ---
   # this document is intentionally initially blank.
   EOF
   )" | vault kv put secret/dev/${CLUSTER_NAME}/tap/sensitive-values.yaml -
   ```

   You can start with an empty document and edit it later on as described in the Configure and push the Tanzu Application Platform values section.

   Vault does not support storing YAML files, all secrets must be in `key-value` format. You must convert your sensitive-values YAML file to `json` before storage.

2. Review the integration with External Secrets Operator.

   This configuration was generated by the `configure.sh` script. It reported:

   ```
   ...
   wrote Vault configuration for TAP Install to: cluster-config/values/tap-install
   -vault-values.yaml
   ...
   ```

   For example, for the `iterate-green` cluster `tap-install-vault-values.yaml` contains the following information:

   ```
   ---
   tap_install:
    secrets:
      eso:
       vault:
         server: https://vault.example.com
         namespace: ""
         auth:
           kubernetes:
             mountPath: iterate-green
             role: iterate-green--tap-install-secrets
   ```

```
    remote_refs:
      tap_sensitive_values:
        sensitive_tap_values_yaml:
          key: secret/dev/iterate-green/tap/sensitive-values.yaml
```

Where:

- `kubernetes.role` is the IAM role that grants permission to Tanzu Application Platform installation to read its associated secrets. This role was created in the Grant read access to secret data section.
- `sensitive_tap_values_yaml.key` is the Vault secret name that contains the sensitive data from the `tap-values.yaml` file for this cluster in a YAML format.

3. Commit the Tanzu Application Platform installation configuration.

   For example, for the `iterate-green` cluster, run:

   ```
   git add cluster-config/
   git commit -m 'Configure installer for TAP 1.6.1 on "iterate-green"'
   ```

Configure and push the Tanzu Application Platform values

The configuration for the Tanzu Application Platform is divided into two separate locations:

- Sensitive configuration is stored in a Vault secret created in the Review and store Tanzu Application Platform installation config section.
- Non-sensitive configuration is stored in a plain YAML file `cluster-config/values/tap-values.yaml`

Follow these steps to split the Tanzu Application Platform values:

1. Create the file `cluster-config/values/tap-values.yaml` by using the Full Profile sample as a guide:

   The Tanzu Application Platform values are input configurations to the Tanzu Application Platform installation and are placed under the `tap_install.values` path.

   ```
   tap_install:
     values:
       # Tanzu Application Platform values go here.
       shared:
         ingress_domain: "INGRESS-DOMAIN"
       ceip_policy_disclosed: true
   ...
   ```

   To install Tanzu Application Platform in an offline environment, you must configure `Tanzu Build Service` and `Grype` to work in an air-gapped environment:

   ```
   ---
   tap_install:
     values:
       ...
       buildservice:
         exclude_dependencies: true
       grype:
         db:
           dbUpdateUrl: INTERNAL-VULN-DB-URL
   ```

   Where `INTERNAL-VULN-DB-URL` is the URL that points to the internal file server.

   For more information, see Components and installation profiles.

2. Review the contents of `tap-values.yaml` and move all the sensitive values into the Vault secret created in the Review and store Tanzu Application Platform installation config section.

   For example, if the `iterate-green` cluster is configured with the basic Out of the Box Supply Chain, this might include a passphrase for that supply chain's GitOps flow:

   ```
   ---
   tap_install:
    values:
      ...
      ootb_supply_chain_basic:
        registry:
          server: "SERVER-NAME"
          repository: "REPO-NAME"
        gitops:
          ssh_secret: "SSH-SECRET-KEY"
      ...
   ```

   To maintain the secrecy of `ootb_supply_chain_basic.gitops.ssh_secret`, move this value from the `tap-values.yaml` file:

   ```
   ---
   tap_install:
    values:
      ...
      ootb_supply_chain_basic:
        registry:
          server: "SERVER-NAME"
          repository: "REPO-NAME"
      ...
   ```

   Add it to the Vault secret named `secret/dev/iterate-green/tap/sensitive-values.yaml`, by default, without the `tap_install.values` root:

   ```
   ---
   ...
   ootb_supply_chain_basic:
    gitops:
      ssh_secret: "SSH-SECRET-KEY"
   ...
   ```

   When moving values, you must omit the `tap_install.values` root, but keep the remaining structure. All of the parent keys, such as `ootb_supply_chain_basic.gitops` and `ssh_secret`, must be copied to the sensitive value YAML.

3. Commit and push the Tanzu Application Platform values:

   ```
   git add cluster-config/
   git commit -m "Configure initial values for TAP 1.6.1"
   git push
   ```

   Tanzu Sync fetches configuration from the hosted clone of the Git repository. For changes to take effect on the cluster, they must be pushed to that clone of the Git repository.

Deploy Tanzu Sync

Deploying Tanzu Sync starts the GitOps workflow that initiates the Tanzu Application Platform installation.

After deployed, Tanzu Sync periodically polls the Git repository for changes. The following deployment process is only required once per cluster:

1. Install the Carvel tools `kapp` and `ytt` onto your `$PATH`:

   ```
   sudo cp $HOME/tanzu-cluster-essentials/kapp /usr/local/bin/kapp
   sudo cp $HOME/tanzu-cluster-essentials/ytt /usr/local/bin/ytt
   ```

   This step is required to ensure the successful deployment of the `tanzu-sync` app.

2. Ensure the Kubernetes cluster context is set to the correct cluster.

   1. List the existing contexts:

      ```
      kubectl config get-contexts
      ```

   2. Set the context to the cluster that you want to deploy:

      ```
      kubectl config use-context CONTEXT-NAME
      ```

      Where `CONTEXT-NAME` can be retrieved from the outputs of the previous step.

3. Bootstrap the deployment.

   External Secrets Operator is installed from the package included in the Tanzu Application Platform package repository. That repository must be fetched from the OCI registry initially.

   1. Set the following environment variables:

      ```
      export INSTALL_REGISTRY_HOSTNAME=MY-REGISTRY
      export INSTALL_REGISTRY_USERNAME=MY-REGISTRY-USER
      export INSTALL_REGISTRY_PASSWORD=MY-REGISTRY-PASSWORD
      ```

      Where:

      - `MY-REGISTRY` is your container registry.
      - `MY-REGISTRY-USER` is the user with read access to `MY-REGISTRY`.
      - `MY-REGISTRY-PASSWORD` is the password for `MY-REGISTRY-USER`.

   2. Create a secret containing credentials to fetch from that OCI registry by using the provided script:

      ```
      tanzu-sync/scripts/bootstrap.sh
      ```

      These credentials are used exactly once to install the External Secrets Operator (ESO) package.

4. Install Tanzu Sync and start the GitOps workflow by deploying it to the cluster using `kapp` and `ytt`.

   ```
   tanzu-sync/scripts/deploy.sh
   ```

   Depending on the profile and components included, it may take 5-10 minutes for the Tanzu Application Platform to install. During this time, `kapp` waits for the deployment of Tanzu Sync to reconcile successfully. This is normal.

   You can track the progress of the installation by watching the installation of those packages in a separate terminal window:

   ```
   watch kubectl get pkgi -n tap-install
   ```

Install Tanzu Application Platform through Gitops with Secrets OPerationS (SOPS)

This topic tells you how to install Tanzu Application Platform (commonly known as TAP) through GitOps with secrets managed in a Git repository.

Caution

- Tanzu Application Platform (GitOps) is currently in beta and is intended for evaluation and test purposes only. Do not use in a production environment.
- Tanzu GitOps Reference Implementation (RI) does not support changing the secrets management strategy for a cluster.

Prerequisites

Before installing Tanzu Application Platform, you need:

- **SOPS CLI** to view and edit SOPS encrypted files. To install the SOPS CLI, see SOPS documentation in GitHub.
- **Age CLI** to create an encryption key used to encrypt and decrypt sensitive data. To install the Age CLI, see age documentation in GitHub.
- Completed the Prerequisites.
- Accepted Tanzu Application Platform EULA and installed Tanzu CLI with any required plug-ins.
- Installed Cluster Essentials for Tanzu.

Relocate images to a registry

VMware recommends relocating the images from VMware Tanzu Network registry to your own container image registry before attempting installation. If you don't relocate the images, Tanzu Application Platform depends on VMware Tanzu Network for continued operation, and VMware Tanzu Network offers no uptime guarantees. The option to skip relocation is documented for evaluation and proof-of-concept only.

The supported registries are Harbor, Azure Container Registry, Google Container Registry, and Quay.io. See the following documentation for a registry to learn how to set it up:

- Harbor documentation
- Google Container Registry documentation
- Quay.io documentation

To relocate images from the VMware Tanzu Network registry to your registry:

1. Set up environment variables for installation use by running:

```
# Set tanzunet as the source registry to copy the Tanzu Application Platform pa
ckages from.
export IMGPKG_REGISTRY_HOSTNAME_0=registry.tanzu.vmware.com
export IMGPKG_REGISTRY_USERNAME_0=MY-TANZUNET-USERNAME
export IMGPKG_REGISTRY_PASSWORD_0=MY-TANZUNET-PASSWORD

# The user's registry for copying the Tanzu Application Platform package to.
export IMGPKG_REGISTRY_HOSTNAME_1=MY-REGISTRY
export IMGPKG_REGISTRY_USERNAME_1=MY-REGISTRY-USER
```

```
export IMGPKG_REGISTRY_PASSWORD_1=MY-REGISTRY-PASSWORD
# These environment variables starting with IMGPKG_* are used by the imgpkg com
mand only.

# The registry from which the Tanzu Application Platform package is retrieved.
export INSTALL_REGISTRY_USERNAME=MY-REGISTRY-USER
export INSTALL_REGISTRY_PASSWORD=MY-REGISTRY-PASSWORD
export INSTALL_REGISTRY_HOSTNAME=MY-REGISTRY
export TAP_VERSION=VERSION-NUMBER
export INSTALL_REPO=TARGET-REPOSITORY
```

Where:

- `MY-REGISTRY-USER` is the user with write access to `MY-REGISTRY`.
- `MY-REGISTRY-PASSWORD` is the password for `MY-REGISTRY-USER`.
- `MY-REGISTRY` is your own container image registry.
- `MY-TANZUNET-USERNAME` is the user with access to the images in the VMware Tanzu Network registry `registry.tanzu.vmware.com`.
- `MY-TANZUNET-PASSWORD` is the password for `MY-TANZUNET-USERNAME`.
- `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.9.1`.
- `TARGET-REPOSITORY` is your target repository, a folder or repository on `MY-REGISTRY` that serves as the location for the installation files for Tanzu Application Platform.

VMware recommends using a JSON key file to authenticate with Google Container Registry. In this case, the value of `INSTALL_REGISTRY_USERNAME` is `_json_key` and the value of `INSTALL_REGISTRY_PASSWORD` is the content of the JSON key file. For more information about how to generate the JSON key file, see Google Container Registry documentation.

2. Install the Carvel tool `imgpkg` CLI.

   To query for the available versions of Tanzu Application Platform on VMWare Tanzu Network Registry, run:

   ```
   imgpkg tag list -i registry.tanzu.vmware.com/tanzu-application-platform/tap-pac
   kages | sort -V
   ```

3. Relocate the images with the `imgpkg` CLI by running:

   ```
   imgpkg copy -b registry.tanzu.vmware.com/tanzu-application-platform/tap-package
   s:${TAP_VERSION} --to-repo ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/tap-pac
   kages
   ```

(Optional) Install Tanzu Application Platform in an air-gapped environment

Complete the following steps if you install Tanzu Application Platform in an air-gapped environment:

1. Relocate the Tanzu Build Service images to your registry:

   ```
   imgpkg copy -b registry.tanzu.vmware.com/tanzu-application-platform/full-tbs-de
   ps-package-repo:VERSION --to-repo ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/
   full-tbs-deps-package-repo
   ```

   Where `VERSION` is the version of Tanzu Build Service. You can retrieve this value by running `kubectl get package -n tap-install | grep buildservice`

2. Host a `grype` database in the air-gapped environment. For more information, see Use vulnerability scanning in offline and air-gapped environments.

Create a new Git repository

Follow these steps to create a new Git repository and set up the authentication:

1. In a hosted Git service, for example, GitHub or GitLab, create a new repository.

   This version of Tanzu GitOps RI supports authenticating to a hosted Git repository by using SSH and Basic Authentication.

2. Initialize a new Git repository:

```
mkdir -p $HOME/tap-gitops
cd $HOME/tap-gitops

git init
git remote add origin git@github.com:my-organization/tap-gitops.git
```

3. Set up the authentication method.

   SSH
   Create a read-only deploy key for this new repository (recommended) or SSH key for an account with read access to this repository. The private portion of this key is called `GIT_SSH_PRIVATE_KEY`.

   Basic Authentication
   Have a user name with read access to the Git repository and password or personal access token for the same user.

Important

Only use one of `ssh` or `Basic Authentication`, not both.

Download and unpack Tanzu GitOps Reference Implementation (RI)

1. Sign in to VMware Tanzu Network.
2. Go to the Tanzu Application Platform product page.
3. Select **Release 1.9.1** from the release drop-down menu.
4. Click **Tanzu GitOps Reference Implementation**.
5. Unpack the downloaded TGZ file into the `$HOME/tap-gitops` directory by running:

```
tar xvf tanzu-gitops-ri-*.tgz -C $HOME/tap-gitops
```

6. Commit the initial state:

```
cd $HOME/tap-gitops

git add . && git commit -m "Initialize Tanzu GitOps RI"
git push -u origin
```

Create cluster configuration

1. Seed configuration for a cluster using SOPS:

   ```
   cd $HOME/tap-gitops

   ./setup-repo.sh CLUSTER-NAME sops
   ```

   Where:

   - `CLUSTER-NAME` the name of your cluster.
   - `sops` selects the Secrets OPerationS-based secrets management variant.

   Example:

   ```
   cd $HOME/tap-gitops

   ./setup-repo.sh full-tap-cluster sops
   Created cluster configuration in ./clusters/full-tap-cluster.
   ...
   ```

   This script creates the directory `clusters/full-tap-cluster/` and copies in the configuration required to sync this Git repository with the cluster and installing Tanzu Application Platform.

2. Commit and push:

   ```
   git add . && git commit -m "Add full-tap-cluster"
   git push
   ```

Configure Tanzu Application Platform

Tanzu Sync Reference Implementation (RI) splits the values configuration of Tanzu Application Platform into two categories:

- Sensitive TAP values, for example, credentials, encryptions keys and so on.
- Non-sensitive TAP values, for example, packages to exclude, namespace configuration and so on.

The following sections describe how to create these values files.

Prepare the sensitive Tanzu Application Platform values

Follow these steps to prepare the sensitive Tanzu Application Platform values:

1. Generate Age public or secrets keys:

   Note

   Skip this step if you already have an Age key to encrypt or decrypt secrets.

   ```
   mkdir -p $HOME/tmp-enc
   chmod 700 $HOME/tmp-enc
   cd $HOME/tmp-enc

   age-keygen -o key.txt

   cat key.txt
   ```

```
# created: 2023-02-08T10:55:35-07:00
# public key: age1ql3z7hjy54pw3hyww5ayyfg7zqgvc7w3j2elw8zmrj2kg5sfn9aqmcac8p
AGE-SECRET-KEY-my-secret-key
```

2. Create a plain YAML file `tap-sensitive-values.yaml` outside of your Git repository that contains a placeholder for the sensitive portion of the Tanzu Application Platform values:

   ```
   ---
   tap_install:
     sensitive_values:
   ```

3. Encrypt `tap-sensitive-values.yaml` with Age using SOPS:

   ```
   export SOPS_AGE_RECIPIENTS=$(cat key.txt | grep "# public key: " | sed 's/# pub
   lic key: //')
   sops --encrypt tap-sensitive-values.yaml > tap-sensitive-values.sops.yaml
   ```

   Where:

   - `grep` is used to find the line containing the public key portion of the generated secret.
   - `sed` is used to extract the public key from the line found by `grep`.

4. (Optional) Verify the encrypted file can be decrypted:

   ```
   export SOPS_AGE_KEY_FILE=key.txt
   sops --decrypt tap-sensitive-values.sops.yaml
   ```

   (Optional) Verify the encrypted file can be edited directly by using SOPS:

   ```
   sops tap-sensitive-values.sops.yaml
   ```

5. Move the sensitive Tanzu Application Platform values into the cluster config:

   ```
   mv tap-sensitive-values.sops.yaml GIT-REPO-ROOT/clusters/CLUSTER-NAME/cluster-c
   onfig/values/
   ```

   Example:

   ```
   mv tap-sensitive-values.sops.yaml $HOME/tap-gitops/clusters/full-tap-cluster/cl
   uster-config/values/
   ```

6. (Optional) Retain the Age identity key file in a safe and secure place such as a password manager, and purge the scratch space:

   ```
   mv key.txt SAFE-LOCATION/
   export SOPS_AGE_KEY_FILE=SAFE-LOCATION/key.txt
   rm -rf $HOME/tmp-enc
   ```

Prepare the non-sensitive Tanzu Application Platform values

Create a plain YAML file `GIT-REPO-ROOT/clusters/CLUSTER-NAME/cluster-config/values/tap-non-sensitive-values.yaml` by using the Full Profile sample as a guide:

Example:

```
---
tap_install:
```

```
  values:
    ceip_policy_disclosed: true
    excluded_packages:
    - policy.apps.tanzu.vmware.com
...
```

To install Tanzu Application Platform in an offline environment, you must configure `Tanzu Build Service` and `Grype` to work in an air-gapped environment:

```
---
tap_install:
  values:
    ...
    buildservice:
      exclude_dependencies: true
    grype:
      db:
        dbUpdateUrl: INTERNAL-VULN-DB-URL
```

Where `INTERNAL-VULN-DB-URL` is the URL that points to the internal file server.

For more information, see Components and installation profiles.

Update the sensitive Tanzu Application Platform values

After filling in the non-sensitive values, follow these steps to extract the sensitive values into `tap-sensitive-values.sops.yaml` that you prepared earlier:

1. Open an editor through SOPS to edit the encrypted sensitive values file:

   ```
   sops GIT-REPO-ROOT/clusters/CLUSTER-NAME/cluster-config/values/tap-sensitive-values.sops.yaml
   ```

   Example:

   ```
   sops $HOME/tap-gitops/clusters/full-tap-cluster/cluster-config/values/tap-sensitive-values.sops.yaml
   ```

2. Add the sensitive values:

   Example of the container registry credentials using basic authentication:

   ```
   ---
   tap_install:
    sensitive_values:
      shared:
        image_registry:
          project_path: "example.com/my-project/tap"
          username: "my_username"
          password: "my_password"
   ```

   Example of the container registry credentials using Google Container Registry:

   ```
   ---
   tap_install:
    sensitive_values:
     shared:
       image_registry:
          project_path: "gcr.io/my-project/tap"
          username: "_json_key"
          password: |
            {
              "type": "service_account",
   ```

```
            "project_id": "my-project",
            "private_key_id": "my-private-key-id",
            "private_key": "-----BEGIN PRIVATE KEY-----\n.........\n-----END PR
IVATE KEY-----\n",
            ...
          }
```

Prepare the sensitive Tanzu Sync values

Follow these steps to prepare the sensitive Tanzu Sync values:

1. Create a plain YAML file `tanzu-sync-values.yaml` outside of your Git repository that contains a placeholder for the sensitive portion of Tanzu Sync values:

   ```
   ---
   secrets:
       sops:
   ```

2. Encrypt `tanzu-sync-values.yaml` with Age using SOPS:

   ```
   export SOPS_AGE_RECIPIENTS=$(cat key.txt | grep "# public key: " | sed 's/# pub
   lic key: //')
   sops --encrypt tanzu-sync-values.yaml > tanzu-sync-values.sops.yaml
   ```

   Where:

   - `grep` is used to find the line containing the public key portion of the generated secret.
   - `sed` is used to extract the public key from the line found by `grep`.

3. Move the encrypted sensitive Tanzu Application Platform values into the Tanzu Sync config:

   ```
   mv tanzu-sync-values.sops.yaml GIT-REPO-ROOT/clusters/CLUSTER-NAME/tanzu-sync/a
   pp/sensitive-values/tanzu-sync-values.sops.yaml
   ```

   Example:

   ```
   mv tanzu-sync-values.sops.yaml $HOME/tap-gitops/clusters/full-tap-cluster/tanzu
   -sync/app/sensitive-values/tanzu-sync-values.sops.yaml
   ```

Update the sensitive Tanzu Sync values

Follow these steps to populate `tap-sensitive-values.sops.yaml` with credentials:

1. Open an editor and use SOPS to edit the encrypted sensitive values file:

   ```
   sops GIT-REPO-ROOT/clusters/CLUSTER-NAME/tanzu-sync/app/sensitive-values/tanzu-
   sync-values.sops.yaml
   ```

   Example:

   ```
   sops $HOME/tap-gitops/clusters/full-tap-cluster/tanzu-sync/app/sensitive-value
   s/tanzu-sync-values.sops.yaml
   ```

2. Add the sensitive values:

   Configuration example for the Git SSH authentication:

   ```
   ---
   secrets:
   ```

```
    sops:
        age_key: |
            AGE-KEY
        registry:
            hostname: MY-REGISTRY
            username: MY-REGISTRY-USER
            password: MY-REGISTRY-PASSWORD
        git:
            ssh:
                private_key: |
                    PRIVATE-KEY
                known_hosts: |
                    HOST-LIST
```

Configuration example for the Git basic authentication:

```
---
secrets:
    sops:
        age_key: |
            AGE-KEY
        registry:
            hostname: MY-REGISTRY
            username: MY-REGISTRY-USER
            password: MY-REGISTRY-PASSWORD
        git:
            basic_auth:
                username: MY-GIT-USERNAME
                password: MY-GIT-PASSWORD
```

Where:

- `AGE-KEY` is the contents of the Age key generated earlier.
- `MY-REGISTRY` is your container image registry.
- `MY-REGISTRY-USER` is the user with read access to `MY-REGISTRY`.
- `MY-REGISTRY-PASSWORD` is the password for `MY-REGISTRY-USER`.
- `PRIVATE-KEY` is the contents of an SSH private key file with read access to your Git repository. Only applies when you use `ssh`.
- `HOST-LIST` is the list of known hosts for Git host service. Only applies when you use `ssh`.
- `MY-GIT-USERNAME` is the user with read access to your Git repository. Only applies when you use `basic authentication`.
- `MY-GIT-PASSWORD` is the password for `MY-GIT-USERNAME`. Only applies when you use `basic authentication`.

You can find the schema for Tanzu Sync credentials in `GIT-REPO-ROOT/clusters/CLUSTER-NAME/tanzu-sync/app/config/.tanzu-managed/schema--sops.yaml`

Generate Tanzu Application Platform installation and Tanzu Sync configuration

Follow these steps to generate the Tanzu Application Platform installation and Tanzu Sync configuration:

1. Set up environment variables by running:

```
export TAP_PKGR_REPO=TAP-PACKAGE-OCI-REPOSITORY
export SOPS_AGE_KEY=AGE-KEY
```

Where:

- `TAP-PACKAGE-OCI-REPOSITORY` is the fully-qualified path to the OCI repository hosting the Tanzu Application Platform images. If the images are relocated as described in Relocate images to a registry, this value is `${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/tap-packages`.
- `AGE-KEY` is the contents of the Age key generated earlier.

Example of the Git repo hosted on GitHub:

```
export TAP_PKGR_REPO=registry.tanzu.vmware.com/tanzu-application-platform/tap-p
ackages
export SOPS_AGE_KEY=$(cat $HOME/tmp-enc/key.txt)
```

2. Generate the Tanzu Application Platform install and the Tanzu Sync configuration files by using the provided script:

```
cd GIT-REPO-ROOT/clusters/CLUSTER-NAME

./tanzu-sync/scripts/configure.sh
```

Example:

```
cd $HOME/tap-gitops/clusters/full-tap-cluster

./tanzu-sync/scripts/configure.sh
```

3. (Optional) Update Tanzu Application Platform to use the latest patch:

Update the Tanzu Application Platform version in `GIT-REPO-ROOT/clusters/CLUSTER-NAME/cluster-config/values/tap-install-values.yaml`:

```
tap_install:
    ...
    version:
        package_repo_bundle_tag: "1.9.1" # Populate these values with the lates
t patch version.
        package_version: "1.9.1"
```

Where:

- `package_repo_bundle_tag` is the version of Tanzu Application Platform you want to upgrade to.
- `package_version` is the version of Tanzu Application Platform you want to upgrade to. This version must match `package_repo_bundle_tag`.

Note

Tanzu GitOps RI does not provide a separate artifact for each patch version within a minor line. For example, Tanzu Application Platform v1.6.x only contains the v1.6.1 GitOps artifact.

4. Commit the generated configured to Git repository:

```
git add cluster-config/ tanzu-sync/
git commit -m "Configure install of TAP 1.6.1"
git push
```

Deploy Tanzu Sync

Follow these steps to deploy Tanzu Sync:

1. Install the Carvel tools `kapp` and `ytt` onto your `$PATH`:

   ```
   sudo cp $HOME/tanzu-cluster-essentials/kapp /usr/local/bin/kapp
   sudo cp $HOME/tanzu-cluster-essentials/ytt /usr/local/bin/ytt
   ```

2. Set the Kubernetes cluster context.
   1. List the existing contexts:

      ```
      kubectl config get-contexts
      ```

   2. Set the context to the cluster that you want to deploy:

      ```
      kubectl config use-context CONTEXT-NAME
      ```

      Where `CONTEXT-NAME` can be retrieved from the outputs of the previous step.

3. Deploy the Tanzu Sync component:

   ```
   cd GIT-REPO-ROOT/clusters/CLUSTER-NAME

   ./tanzu-sync/scripts/deploy.sh
   ```

   Example:

   ```
   cd $HOME/tap-gitops/clusters/full-tap-cluster

   ./tanzu-sync/scripts/deploy.sh
   ```

   Note

   Depending on the profile and components included, it may take 5-10 minutes for Tanzu Application Platform to install. During this time, `kapp` waits for the deployment of Tanzu Sync to reconcile successfully. This is normal.

Install individual packages

You can install Tanzu Application Platform (commonly known as TAP) through predefined profiles or through individual packages. Use this topic to learn how to install each individual package. For more information about installing through profiles, see Components and installation profiles.

Installing individual Tanzu Application Platform packages is useful if you do not want to use a profile to install packages or if you want to install additional packages after installing a profile. Before installing the packages, be sure to complete the prerequisites, configure and verify the cluster, accept the EULA, and install the Tanzu CLI with any required plug-ins. For more information, see Prerequisites.

Install pages for individual Tanzu Application Platform packages

- Install API Auto Registration
- Install API portal
- Install Application Accelerator
- Install Application Configuration Service
- Install Application Live View
- Install Application Single Sign-On
- Install Bitnami Services
- Install cert-manager
- Install Cloud Native Runtimes
- Install Contour
- Install Crossplane
- Install default roles for Tanzu Application Platform
- Install Developer Conventions
- Install Flux CD Source Controller
- Install Out of the Box Templates
- Install Out of the Box Supply Chain with Testing
- Install Out of the Box Supply Chain with Testing and Scanning
- Install Service Bindings
- Install Services Toolkit
- Install Source Controller
- Install Spring Boot conventions
- Install Supply Chain Choreographer
- Install Supply Chain Security Tools - Store
- Install Supply Chain Security Tools - Policy Controller
- Install Supply Chain Security Tools - Scan
- Install Tanzu Developer Portal
- Install Tanzu Build Service
- Install Tekton
- Install Telemetry

Verify the installed packages

Use the following procedure to verify that the packages are installed.

1. List the installed packages by running:

   ```
   tanzu package installed list --namespace tap-install
   ```

   For example:

```
$ tanzu package installed list --namespace tap-install
\ Retrieving installed packages...
NAME                     PACKAGE-NAME                                      PAC
KAGE-VERSION  STATUS
api-portal               api-portal.tanzu.vmware.com                       1.
0.3           Reconcile succeeded
app-accelerator          accelerator.apps.tanzu.vmware.com                 1.
0.0           Reconcile succeeded
app-live-view            appliveview.tanzu.vmware.com                      1.
0.2           Reconcile succeeded
appliveview-conventions  build.appliveview.tanzu.vmware.com                1.
0.2           Reconcile succeeded
cartographer             cartographer.tanzu.vmware.com                     0.
1.0           Reconcile succeeded
cloud-native-runtimes    cnrs.tanzu.vmware.com                             1.
0.3           Reconcile succeeded
convention-controller    controller.conventions.apps.tanzu.vmware.com      0.
7.0           Reconcile succeeded
developer-conventions    developer-conventions.tanzu.vmware.com            0.
3.0-build.1   Reconcile succeeded
grype-scanner            grype.scanning.apps.tanzu.vmware.com              1.
0.0           Reconcile succeeded
image-policy-webhook     image-policy-webhook.signing.apps.tanzu.vmware.com 1.
1.2           Reconcile succeeded
metadata-store           metadata-store.apps.tanzu.vmware.com              1.
0.2           Reconcile succeeded
ootb-supply-chain-basic  ootb-supply-chain-basic.tanzu.vmware.com          0.
5.1           Reconcile succeeded
ootb-templates           ootb-templates.tanzu.vmware.com                   0.
5.1           Reconcile succeeded
scan-controller          scanning.apps.tanzu.vmware.com                    1.
0.0           Reconcile succeeded
service-bindings         service-bindings.labs.vmware.com                  0.
5.0           Reconcile succeeded
services-toolkit         services-toolkit.tanzu.vmware.com                 0.
8.0           Reconcile succeeded
source-controller        controller.source.apps.tanzu.vmware.com           0.
2.0           Reconcile succeeded
sso4k8s-install          sso.apps.tanzu.vmware.com                         1.
0.0-beta.2-31 Reconcile succeeded
tap-gui                  tap-gui.tanzu.vmware.com                          0.
3.0-rc.4      Reconcile succeeded
tekton-pipelines         tekton.tanzu.vmware.com                           0.3
0.0          Reconcile succeeded
tbs                      buildservice.tanzu.vmware.com                     1.
5.0           Reconcile succeeded
```

Next steps

- Set up developer namespaces to use your installed packages

Set up developer namespaces to use your installed packages

For details about how to automatically set up your developer namespaces, see Provision developer namespaces in Namespace Provisioner.

Additional configuration for testing and scanning

If you plan to install or have already installed Out of the Box Supply Chains with Testing and Scanning, you can use Namespace Provisioner to set up the required resources. For more

information, see Customize installation in the Namespace Provisioner documentation for configuration steps.

Legacy namespace setup

To use the legacy manual process for setting up developer namespaces, see Legacy namespace setup.

Next steps

- Install Tanzu Developer Tools for your VS Code

Install Tanzu Developer Tools for your VS Code

This topic tells you how to install VMware Tanzu Developer Tools for Visual Studio Code (VS Code).

Prerequisites

Before installing the extension, you must have:

- VS Code
- kubectl
- Tilt v0.30.12 or later
- Tanzu CLI and plug-ins
- A cluster with the Tanzu Application Platform Full profile or Iterate profile

If you are an app developer, someone else in your organization might have already set up the Tanzu Application Platform environment.

Docker Desktop and local Kubernetes are not prerequisites for using Tanzu Developer Tools for VS Code.

Install

To install VMware Tanzu Developer Tools for VS Code:

1. Open Visual Studio Code.
2. Open the command palette.
3. In the search box enter `Extension`.
4. Click **Extensions: Install Extensions**.
5. The **Extensions** view opens on the left side of your screen. In the search box enter `Tanzu`.
6. Click **Tanzu Developer Tools** and then click **Install**.

Configure

To configure VMware Tanzu Developer Tools for VS Code:

1. Ensure that you are targeting the correct cluster. For more information, see the Kubernetes documentation.
2. Go to **Code** > **Preferences** > **Settings** > **Extensions** > **Tanzu Developer Tools** and set the following:

- **Confirm Apply Config**: This controls whether the extension asks to confirm user input when applying a workload.
- **Confirm Debug Port**: This controls whether the extension asks for the debug port when running `Tanzu: Start Java Debug`.
- **Confirm Local Port**: This controls whether the extension asks for the local port when running `Tanzu: Start Java Debug`.
- **Confirm Delete**: This controls whether the extension asks for confirmation when deleting a workload.
- **Enable Live Hover**: This enables Live Hover. For more information, see Integrating Live Hover by using Spring Boot Tools. Restart VS Code for this change to take effect.
- **Source Image**: The registry location for publishing local source code. For example, `registry.io/yourapp-source`. This must include both a registry and a project name. A source image registry location is optional when Local Source Proxy is configured.
- **Local Path**: (Optional) The path on the local file system to a directory of source code to build. This is the current directory by default.
- **Namespace**: (Optional) This is the namespace that workloads are deployed into. The namespace set in kubeconfig is the default.
- **Tracked Namespaces**: (Optional) Comma-separated list of namespaces. Resources in these namespaces appear in the Tanzu Workloads panel and the Activity panel. If empty, the namespace in the current context is the default.
- **Wait Timeout**: (Optional) This sets how long to wait for a workload to become ready.
- **Workload Type**: (Optional) This distinguishes the workload type. Examples include web and server.

Uninstall

To uninstall VMware Tanzu Developer Tools for VS Code:

1. Go to **Code** > **Preferences** > **Settings** > **Extensions**.
2. Right-click the extension and then click **Uninstall**.

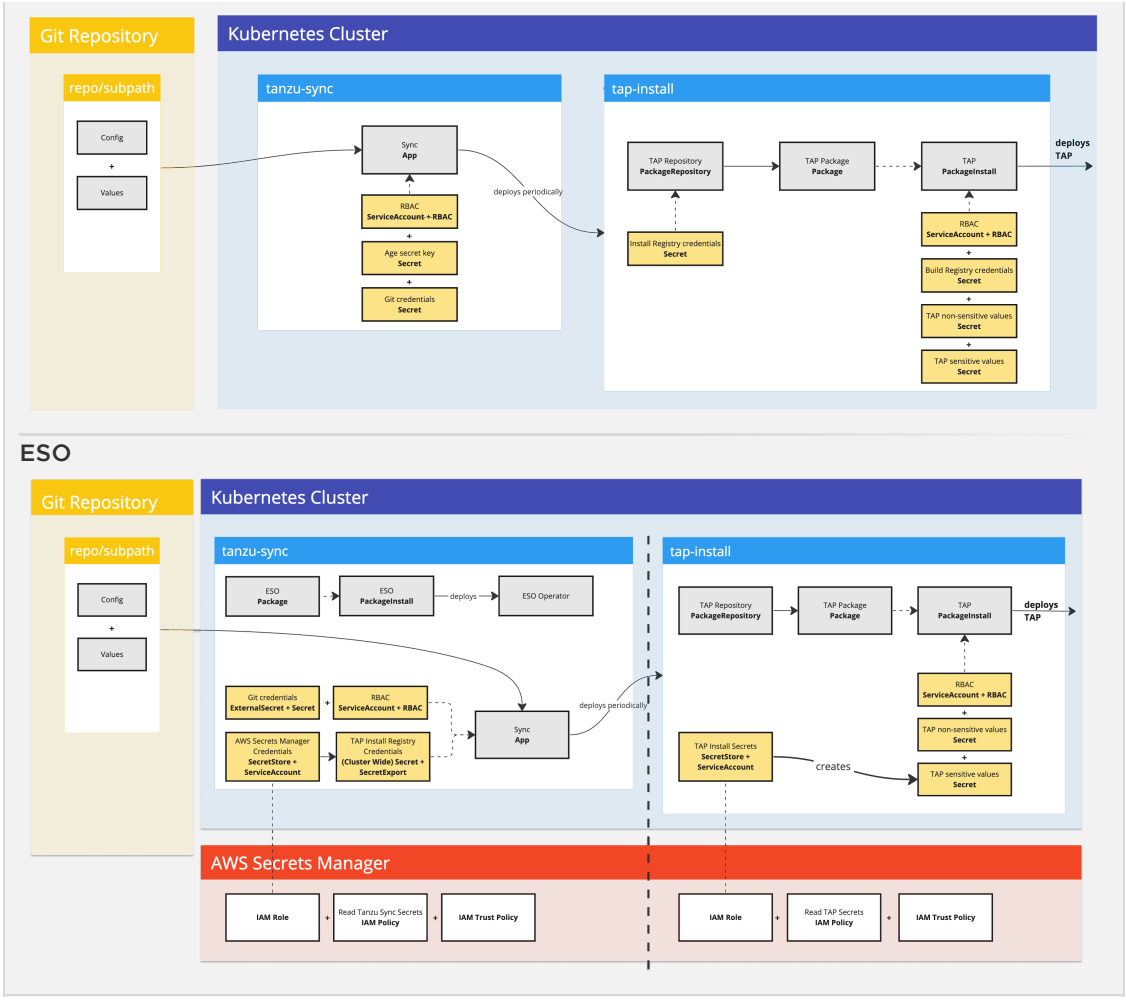
Next steps

Proceed to Getting started with Tanzu Developer Tools for Visual Studio Code.

Tanzu GitOps RI Reference Documentation

The following diagrams shows you the components that are installed as part of Tanzu GitOps Reference Implementation (RI) and how they work together to automate the installation of Tanzu Application Platform (commonly known as TAP):

SOPS



Caution

Tanzu Application Platform (GitOps) is currently in beta and is intended for evaluation and test purposes only. Do not use in a production environment.

Tanzu Sync Carvel Application

Tanzu Sync consists of a Carvel application named `sync` that is installed in the `tanzu-sync` namespace. The sync application:

1. Fetches a Git repository that contains configuration for Tanzu Application Platform.
2. Templates with `ytt` a set of resources and data values.
3. Deploys with `kapp` a set of resources to install Tanzu Application Platform, with any other user specified configuration in the Git Repository.

Choosing Secrets OPerationS (SOPS) or External Secrets Operator (ESO)

The following table outlines the Kubernetes distributions and secret management solutions that SOPS and ESO support:

Choose	IaaS	Secrets Manager
SOPS	Any IaaS supported by Tanzu Application Platform	N/A
ESO	AWS EKS	AWS Secrets Manager
ESO	Azure, GKE and AWS EKS are supported by Tanzu Application Platform	Vault

Note

Future release will include additional Secrets Managers for ESO.

The following table describes a few common use cases and scenarios for SOPS and ESO:

I want ...	SOPS	ESO
Sensitive data encrypted inside the Git repository.		
Sensitive data to be stored outside the Git repository.		
Minimal setup. No external secret storage system		
To manage sensitive data myself. For example, storing keys, rotation and usage auditing.)		
To utilize sensitive data management. For example, storage, rotation and usage auditing by a third-party solution.		

Git Repository structure

Tanzu Sync Application fetches our deployable content from a Git repository that must match the following structure:

Git repository for a cluster named `full-tap-cluster`:

```
├── .catalog
│   ├── tanzu-sync
│   │   └── 0.0.3
│   └── tap-install
│       └── 1.5.0
├── README.md
├── clusters
│   └── full-tap-cluster
│       ├── README.md
│       ├── cluster-config
│       │   ├── config
│       │   │   └── tap-install
│       │   │       └── .tanzu-managed
│       │   └── values
│       └── tanzu-sync
│           ├── app
│           │   ├── config
│           │   │   └── .tanzu-managed
│           │   └── values
│           ├── bootstrap
│           └── scripts
└── setup-repo.sh
```

Where:

- `.catalog`: VMware supplied directory of resources and configuration to install Tanzu Sync and Tanzu Application Platform.

    - `tanzu-sync`: Contains the Carvel Packaging App which supports a GitOps workflow for fetching, templating and deploying the `clusters/full-tap-cluster/cluster-config` directory of this repository.
    - `tap-install`: Contains the configuration to install Tanzu Application Platform.
- `clusters/full-tap-cluster`
    - `cluster-config`
        - `config`: Contains the Tanzu Application Platform installation configuration. This directory can be extended to include any desired resources managed through GitOps to your cluster.
        - `.tanzu-managed`: Contains VMware managed Kubernetes resource files to install Tanzu Application Platform. Do not alter this value.
        - `values`: Contains the plain YAML data files which configure the application.
    - `tanzu-sync`
        - `app`: Contains the main Carvel Packaging App that runs on the cluster. It fetches, templates and deploys your Tanzu Application Platform installation from `clusters/full-tap-cluster/cluster-config`.
        - `bootstrap`: Contains secret provider specific bootstrapping if required.
        - `scripts`: Contains helper scripts to assist with the configuration and deployment of Tanzu GitOps RI.

Configuration of Tanzu Sync without helper scripts

1. The following plain YAML values files are required to run Tanzu Sync:
    - Tanzu Sync App:

      `clusters/full-tap-cluster/tanzu-sync/app/values/values.yaml` adhering to the following schema:

      ```
      #@data/values-schema
      #@overlay/match-child-defaults missing_ok=True
      ---
      git:
        url: ""
        ref: ""
        sub_path: ""

      tap_package_repository:
        oci_repository: ""
      ```

      Example:

      ```
      ---
      git:
        url: git@github.com:my-org/gitops-tap.git
        ref: origin/main
        sub_path: clusters/full-tap-cluster/cluster-config

      tap_package_repository:
        oci_repository: registry.example.com/tanzu-application-platform/tap-pac
      kages
      ```
    - Tanzu Application Platform Install:

      `clusters/full-tap-cluster/cluster-config/config/values/install-values.yaml` adhering to the following schema:

```
#@data/values-schema
#@overlay/match-child-defaults missing_ok=True
---
tap_install:
  package_repository:
    oci_repository: ""
  #@schema/type any=True
  values: {}
```

Example:

```
tap_install:
  package_repository:
    oci_repository: registry.example.com/tanzu-application-platform/tap-p
ackages
  values:
    shared:
      ingress_domain: example.vmware.com
    ceip_policy_disclosed: true
```

`clusters/full-tap-cluster/cluster-config/config/values/sensitive-values.sops.yaml` adhering to the following schema:

```
#@data/values-schema
#@overlay/match-child-defaults missing_ok=True
---
tap_install:
  #@schema/nullable
  #@schema/validation not_null=True
  #@schema/type any=True
  sensitive_values: {}
```

Example:

```
tap_install:
  sensitive_values:
    shared:
      image_registry:
        project_path: example.registry.com/my-project/my-user/tap
        username: my-username
        password: my-password
```

2. The following is used to deploy the application by using `kapp`:

```
kapp deploy --app tanzu-sync --file <(ytt \
    --file tanzu-sync/app/config \
    --file cluster-config/config/tap-install/.tanzu-managed/version.yaml \
    --data-values-file tanzu-sync/app/values/ \
    --data-value secrets.sops.age_key=$(cat $HOME/key.txt) \
    --data-value secrets.sops.registry.hostname="hostname" \
    --data-value secrets.sops.registry.username="foo@example.com" \
    --data-value secrets.sops.registry.password="password" \
    --data-value secrets.sops.git.ssh.private_key=$(cat $HOME/.ssh/my_private_k
ey) \
    --data-value secrets.sops.git.ssh.known_hosts=$(ssh-keyscan github.com) \
)
```

Tanzu Sync Scripts

Caution

The provided scripts are intended to help set up your Git repository to work with a GitOps approach, they are subject to change or removal between releases.

VMware provides a set of convenience bash scripts in `clusters/MY-CLUSTER/tanzu-sync/scripts` to help you set up your Git repository and configure the values as described in the previous section:

- `setup-repo.sh`: Populates a Git repository with the structure described in the Git Repository structure section.
- `configure.sh`: Generates the values files described in the Configuration of values without helper scripts section.
- `deploy.sh`: A light wrapper around a simple `kapp deploy` given the data values from the previous section and sensitive values which must not be stored on disk.

Customize your package installation

You can customize your package configuration that is not exposed through data values by using annotations and ytt overlays.

You can customize a package that was installed manually or that was installed by using a Tanzu Application Platform profile.

Customize a package that was manually installed

To customize a package that was installed manually:

1. Create a `secret.yml` file with a `Secret` that contains your ytt overlay. For example:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
    name: tap-overlay
    namespace: tap-install
   stringData:
    custom-package-overlay.yml: |
      CUSTOM-OVERLAY
   ```

   For more information about ytt overlays, see the Carvel documentation.

2. Apply the `Secret` to your cluster by running:

   ```
   kubectl apply -f secret.yml
   ```

3. Update your `PackageInstall` to include the `ext.packaging.carvel.dev/ytt-paths-from-secret-name.x` annotation to reference your new overlay `Secret`. For example:

   ```
   apiVersion: packaging.carvel.dev/v1alpha1
   kind: PackageInstall
   metadata:
    name: PACKAGE-NAME
    namespace: tap-install
    annotations:
      ext.packaging.carvel.dev/ytt-paths-from-secret-name: tap-overlay
   ...
   ```

   Note

> You can suffix the extension annotation with `.x`, where `x` is a number, to apply multiple overlays. For more information, see the Carvel documentation.

Customize a package that was installed by using a profile

To add an overlay to a package that was installed by using a Tanzu Application Platform profile:

1. Create a `Secret` with your ytt overlay. For more information about ytt overlays, see the Carvel documentation.
2. Update your values file to include a `package_overlays` field:

   ```
   package_overlays:
   - name: PACKAGE-NAME
     secrets:
     - name: SECRET-NAME
   ```

   Where `PACKAGE-NAME` is the target package for the overlay. For example, `tap-gui`.
3. Update Tanzu Application Platform by running:

   ```
   tanzu package installed update tap -p tap.tanzu.vmware.com -v 1.9.1  --values-f
   ile tap-values.yaml -n tap-install
   ```

For information about Tanzu Application Platform profiles, see Installing Tanzu Application Platform package and profiles.

Upgrade Tanzu Application Platform

This document tells you how to upgrade your Tanzu Application Platform (commonly known as TAP).

You can perform a fresh install of Tanzu Application Platform by following the instructions in Installing Tanzu Application Platform.

Prerequisites

Before you upgrade Tanzu Application Platform:

- Verify that you meet all the prerequisites and resource requirements of the target Tanzu Application Platform version. If the target Tanzu Application Platform version does not support your existing Kubernetes version, VMware recommends upgrading to a supported version before proceeding with the upgrade.
- For information about installing your Tanzu Application Platform, see Install your Tanzu Application Platform profile.
- Ensure that Tanzu CLI is updated to the version recommended by the target Tanzu Application Platform version. For information about installing or updating the Tanzu CLI and plug-ins, see Install or update the Tanzu CLI and plug-ins.
- For information about Tanzu Developer Portal considerations, see Tanzu Developer Portal Considerations.
- Verify all packages are reconciled by running `tanzu package installed list -A`.
- To avoid the temporary warning state that is described in Update the new package repository, upgrade to Cluster Essentials v1.9. For more information about the upgrade

procedures, see the Cluster Essentials documentation.

- The previously deprecated field `scanning.metadataStore.url` is removed from the values for installing or upgrading Tanzu Application Platform v1.7 and later. This field must not present in the `tap-non-sensitive-values.yaml` file when performing the upgrade.
- Note that this upgrade will update all workloads and pods that are using service bindings. This is done automatically after upgrading to 1.7 or later and requires no user action.
- All pods with service bindings are recreated concurrently at the time of the upgrade. You must have sufficient Kubernetes resources in your clusters to support the pod rollout.
- If you manually created a secret to configure the Metadata Store CA Certificate for Supply Chain Security Tools (SCST) - Scan, you must configure this certificate in Tanzu Application Platform values file before upgrading. For more information, see v1.9.0 Breaking changes: Supply Chain Security Tools - Scan.

Supported upgrade paths

Tanzu Application Platform v1.9 supports upgrading from the following versions:

- v1.8.x long-term support release
- v1.7.x long-term support release

Update the new package repository

Follow these steps to update the new package repository:

1. Relocate the latest version of Tanzu Application Platform images by following step 1 through step 6 in Relocate images to a registry.

   Important

   Make sure to update the `TAP_VERSION` to the target version of Tanzu Application Platform you are migrating to. For example, `1.9.1`.

2. Add the target version of the Tanzu Application Platform package repository by running:

   Cluster Essentials 1.2 or above

   ```
   tanzu package repository add tanzu-tap-repository \
   --url ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/tap-packages:$TAP_VERSION
   \
   --namespace tap-install
   ```

   Cluster Essentials 1.1 or 1.0

   ```
   tanzu package repository update tanzu-tap-repository \
   --url ${INSTALL_REGISTRY_HOSTNAME}/TARGET-REPOSITORY/tap-packages:${TAP_VERSI
   ON} \
   --namespace tap-install
   ```

   Expect to see the installed Tanzu Application Platform packages in a temporary “Reconcile Failed” state, following a “Package not found” warning. These warnings will disappear after you upgrade the installed Tanzu Application Platform packages to version 1.2.0.

3. Verify you have added the new package repository by running:

   ```
   tanzu package repository get TAP-REPO-NAME --namespace tap-install
   ```

   Where `TAP-REPO-NAME` is the package repository name. It must match with either `NEW-TANZU-TAP-REPOSITORY` or `tanzu-tap-repository` in the previous step.

Perform the upgrade of Tanzu Application Platform

The following sections describe how to upgrade in different scenarios.

Upgrade instructions for Profile-based installation

The following changes affect the upgrade procedures:

- **Introduced Artifact Metadata Repository**

  In Tanzu Application Platform v1.7.0 and later, the Artifact Metadata Repository component is introduced into the Supply Chain Security Tools (SCST) - Store package. In a multicluster deployment, this component requires additional configuration during upgrade. For more information, see Upgrading from AMR Beta to AMR GA release.

- **Keyless support deactivated by default**

  In Tanzu Application Platform v1.5.0, keyless support is deactivated by default. For more information, see Install Supply Chain Security Tools - Policy Controller.

  To support the keyless authorities in `ClusterImagePolicy`, Policy Controller no longer initializes TUF by default. To continue using keyless authorities, you must set the `policy.tuf_enabled` field to `true` in the `tap-values.yaml` file during the upgrade process.

  By default, the public official Sigstore “The Update Framework (TUF) server” is used. You can use an alternative Sigstore Stack by setting `policy.tuf_mirror` and `policy.tuf_root`.

- **Image Policy Webhook no longer in use**

  Tanzu Application Platform v1.5.0 removes Image Policy Webhook. If you use Image Policy Webhook in the previous version of Tanzu Application Platform, you must migrate the `ClusterImagePolicy` resource from Image Policy Webhook to Policy Controller.

- **CVE results require a read-write service account**

  Tanzu Application Platform v1.3.0 uses a read-only service account. In Tanzu Application Platform v1.4.0 and later, enabling CVE results for the Supply Chain Choreographer and Security Analysis GUI plug-ins requires a read-write service account. For more information, see Enable CVE scan results.

If you installed Tanzu Application Platform by using a profile, you can perform the upgrade by running the following command in the directory where the `tap-values.yaml` file resides:

```
tanzu package installed update tap -p tap.tanzu.vmware.com -v ${TAP_VERSION}  --values
-file tap-values.yaml -n tap-install
```

When upgrading to Tanzu Application Platform v1.5, you might encounter a temporary resource reconciliation failure. This error does not persist and the packages will reconcile subsequently. To facilitate the reconciliation of packages, you can execute the `tanzu package installed kick -n tap-install tap -y` command repeatedly.

Upgrade the full dependencies package

If you installed the full dependencies package, you can upgrade the package by following these steps:

1. (Optional) If you have an existing installation of the full dependencies from a version earlier than TAP v1.6.1, you must uninstall the full dependencies package and remove the package repository. Subsequent upgrades will not require a removal:

   Uninstall the package

   ```
   tanzu package installed delete full-tbs-deps -n tap-install
   ```

   Remove the package repository

   ```
   tanzu package repository delete tbs-full-deps-repository -n tap-install
   ```

2. After upgrading Tanzu Application Platform, retrieve the latest version of the Tanzu Application Platform package by running:

   ```
   tanzu package available list tap.tanzu.vmware.com --namespace tap-install
   ```

3. Relocate the Tanzu Build Service `full` dependencies package repository by running:

   ```
   imgpkg copy -b registry.tanzu.vmware.com/tanzu-application-platform/full-deps-p
   ackage-repo:VERSION \
   --to-repo ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/full-deps-package-repo
   ```

   Where `VERSION` is the version of the Tanzu Application Platform package you retrieved in the previous step.

4. Update the Tanzu Build Service `full` dependencies package repository by running:

   ```
   tanzu package repository add full-deps-package-repo \
     --url ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/full-deps-package-repo:VER
   SION \
     --namespace tap-install
   ```

5. Update the `full` dependencies package by running:

   ```
   tanzu package install full-deps -p full-deps.buildservice.tanzu.vmware.com -v
   "> 0.0.0" -n tap-install --values-file PATH-TO-TAP-VALUES-FILE
   ```

   Important

   The values file is only required for this command if you install this package for the first time or if the values have changed.

Multicluster upgrade order

Upgrading a multicluster deployment requires updating multiple clusters with different profiles. If upgrades are not performed at the exact same time, different clusters have different versions of profiles installed temporarily. This might cause a temporary API mismatch that leads to errors. Those errors eventually disappear when the versions are consistent across all clusters.

To reduce the likelihood of temporary failures, follow these steps to upgrade your multicluster deployment:

1. Upgrade the view-profile cluster.
2. Upgrade the remaining clusters in any order.

Upgrade instructions for component-specific installation

You can upgrade the following components outside of a Tanzu Application Platform profile upgrade:

- Tanzu Developer Portal: Upgrade Tanzu Developer Portal.
- Supply Chain Security Tools - Scan: Upgrade Supply Chain Security Tools - Scan.
- Supply Chain Security Tools - Store: Upgrading Supply Chain Security Tools - Store.

Verify the upgrade

Verify the versions of packages after the upgrade by running:

```
tanzu package installed list --namespace tap-install
```

Your output is similar, but probably not identical, to the following example output:

```
- Retrieving installed packages...
  NAME                                PACKAGE-NAME
PACKAGE-VERSION  STATUS
  accelerator                         accelerator.apps.tanzu.vmware.com
1.3.0            Reconcile succeeded
  api-auto-registration               apis.apps.tanzu.vmware.com
0.1.1            Reconcile succeeded
  api-portal                          api-portal.tanzu.vmware.com
1.2.2            Reconcile succeeded
  appliveview                         backend.appliveview.tanzu.vmware.com
1.3.0            Reconcile succeeded
  appliveview-connector               connector.appliveview.tanzu.vmware.com
1.3.0            Reconcile succeeded
  appliveview-conventions             conventions.appliveview.tanzu.vmware.com
1.3.0            Reconcile succeeded
  appsso                              sso.apps.tanzu.vmware.com
2.0.0            Reconcile succeeded
  buildservice                        buildservice.tanzu.vmware.com
1.7.1            Reconcile succeeded
  cartographer                        cartographer.tanzu.vmware.com
0.5.3            Reconcile succeeded
  cert-manager                        cert-manager.tanzu.vmware.com
1.7.2+tap.1      Reconcile succeeded
  cnrs                                cnrs.tanzu.vmware.com
2.0.1            Reconcile succeeded
  contour                             contour.tanzu.vmware.com
1.22.0+tap.3     Reconcile succeeded
  conventions-controller              controller.conventions.apps.tanzu.vmware.com
0.7.1            Reconcile succeeded
  developer-conventions               developer-conventions.tanzu.vmware.com
0.8.0            Reconcile succeeded
  fluxcd-source-controller            fluxcd.source.controller.tanzu.vmware.com
0.27.0+tap.1     Reconcile succeeded
  grype                               grype.scanning.apps.tanzu.vmware.com
1.3.0            Reconcile succeeded
  image-policy-webhook                image-policy-webhook.signing.apps.tanzu.vmware.c
om   1.1.7            Reconcile succeeded
  metadata-store                      metadata-store.apps.tanzu.vmware.com
1.3.3            Reconcile succeeded
  ootb-delivery-basic                 ootb-delivery-basic.tanzu.vmware.com
0.10.2           Reconcile succeeded
  ootb-supply-chain-testing-scanning  ootb-supply-chain-testing-scanning.tanzu.vmware.
com  0.10.2           Reconcile succeeded
  ootb-templates                      ootb-templates.tanzu.vmware.com
0.10.2           Reconcile succeeded
  policy-controller                   policy.apps.tanzu.vmware.com
```

```
1.1.1            Reconcile succeeded
  scanning                           scanning.apps.tanzu.vmware.com
1.3.0            Reconcile succeeded
  service-bindings                   service-bindings.labs.vmware.com
0.8.0            Reconcile succeeded
  services-toolkit                   services-toolkit.tanzu.vmware.com
0.8.0            Reconcile succeeded
  source-controller                  controller.source.apps.tanzu.vmware.com
0.5.0            Reconcile succeeded
  spring-boot-conventions            spring-boot-conventions.tanzu.vmware.com
0.5.0            Reconcile succeeded
  tap                                tap.tanzu.vmware.com
1.3.0            Reconcile succeeded
  tap-auth                           tap-auth.tanzu.vmware.com
1.1.0            Reconcile succeeded
  tap-gui                            tap-gui.tanzu.vmware.com
1.3.0            Reconcile succeeded
  tap-telemetry                      tap-telemetry.tanzu.vmware.com
0.3.1            Reconcile succeeded
  tekton-pipelines                   tekton.tanzu.vmware.com
0.39.0+tap.2     Reconcile succeeded
```

Upgrade your Tanzu Application Platform by using GitOps

This document tells you how to upgrade your Tanzu Application Platform (commonly known as TAP) by using GitOps.

Caution

Tanzu Application Platform (GitOps) is currently in beta and is intended for evaluation and test purposes only. Do not use in a production environment.

You can perform a fresh install of Tanzu Application Platform by following the instructions in Installing Tanzu Application Platform.

Prerequisites

Before you upgrade your Tanzu Application Platform:

- Verify that you meet all the prerequisites of the target Tanzu Application Platform version. If the target Tanzu Application Platform version does not support your existing Kubernetes version, VMware recommends upgrading to a supported version before proceeding with the upgrade.
- For information about installing your Tanzu Application Platform, see Install Tanzu Application Platform through Gitops with Secrets OPerationS (SOPS) or Install Tanzu Application Platform through GitOps with External Secrets Operator (ESO).
- For information about Tanzu Developer Portal considerations, see Tanzu Developer Portal Considerations.
- Verify all packages are reconciled by running `kubectl get packageinstall --namespace tap-install`.
- The previously deprecated field `scanning.metadataStore.url` is removed from the values for installing or upgrading Tanzu Application Platform v1.7 and later. This field must not present in the `tap-non-sensitive-values.yaml` file when performing the upgrade.

Relocate Tanzu Application Platform images to a registry

VMware recommends relocating the images from VMware Tanzu Network registry to your own container image registry before attempting installation. If you don't relocate the images, Tanzu Application Platform depends on VMware Tanzu Network for continued operation, and VMware Tanzu Network offers no uptime guarantees. The option to skip relocation is documented for evaluation and proof-of-concept only.

The supported registries are Harbor, Azure Container Registry, Google Container Registry, and Quay.io. See the following the documentation for instructions on setting up a registry:

- Harbor documentation
- Google Container Registry documentation
- Quay.io documentation

To relocate images from the VMware Tanzu Network registry to your registry:

1. Set up environment variables for installation use by running:

   ```
   export IMGPKG_REGISTRY_HOSTNAME_0=registry.tanzu.vmware.com
   export IMGPKG_REGISTRY_USERNAME_0=MY-TANZUNET-USERNAME
   export IMGPKG_REGISTRY_PASSWORD_0=MY-TANZUNET-PASSWORD
   export IMGPKG_REGISTRY_HOSTNAME_1=MY-REGISTRY
   export IMGPKG_REGISTRY_USERNAME_1=MY-REGISTRY-USER
   export IMGPKG_REGISTRY_PASSWORD_1=MY-REGISTRY-PASSWORD
   export INSTALL_REGISTRY_USERNAME=MY-REGISTRY-USER
   export INSTALL_REGISTRY_PASSWORD=MY-REGISTRY-PASSWORD
   export INSTALL_REGISTRY_HOSTNAME=MY-REGISTRY
   export TAP_VERSION=VERSION-NUMBER
   export INSTALL_REPO=TARGET-REPOSITORY
   ```

   Where:

   - `MY-REGISTRY-USER` is the user with write access to `MY-REGISTRY`.
   - `MY-REGISTRY-PASSWORD` is the password for `MY-REGISTRY-USER`.
   - `MY-REGISTRY` is your own container registry.
   - `MY-TANZUNET-USERNAME` is the user with access to the images in the VMware Tanzu Network registry `registry.tanzu.vmware.com`.
   - `MY-TANZUNET-PASSWORD` is the password for `MY-TANZUNET-USERNAME`.
   - `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.9.1`.
   - `TARGET-REPOSITORY` is your target repository, a folder or repository on `MY-REGISTRY` that serves as the location for the installation files of Tanzu Application Platform.

   VMware recommends using a JSON key file to authenticate with Google Container Registry. In this case, the value of `INSTALL_REGISTRY_USERNAME` is `_json_key` and the value of `INSTALL_REGISTRY_PASSWORD` is the content of the JSON key file. For more information about how to generate the JSON key file, see Google Container Registry documentation.

2. Install the Carvel tool `imgpkg` CLI.

   To query for the available versions of Tanzu Application Platform on VMWare Tanzu Network Registry, run:

   ```
   imgpkg tag list -i registry.tanzu.vmware.com/tanzu-application-platform/tap-pac
   kages | sort -V
   ```

3. Relocate the images with the `imgpkg` CLI by running:

   ```
   imgpkg copy -b registry.tanzu.vmware.com/tanzu-application-platform/tap-package
   s:${TAP_VERSION} --to-repo ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/tap-pac
   ```

```
kages
```

Upgrade to a patch version

Caution

Tanzu Application Platform (GitOps) does not provide a separate artifact for each patch version within a minor line. For example, Tanzu Application Platform v1.6.x only contains the v1.6.1 GitOps artifact.

Follow these steps to upgrade to the latest patch:

1. Download and unpack the latest version of Tanzu GitOps RI.
2. Update the Tanzu Application Platform version in `GIT-REPO-ROOT/clusters/CLUSTER-NAME/cluster-config/values/tap-install-values.yaml`:

   ```
   tap_install:
       ...
       version:
           package_repo_bundle_tag: "1.9.1"
           package_version: "1.9.1"
   ```

   Where:

   - `package_repo_bundle_tag` is the version of Tanzu Application Platform you want to upgrade to.
   - `package_version` is the version of Tanzu Application Platform you want to upgrade to. This version must match `package_repo_bundle_tag`.
3. Commit the upgrade configurations:

   ```
   git add . && git commit -m "Upgrade TAP to 1.9.1"
   git push
   ```

Upgrade the existing SOPs based installation

In previous versions of Tanzu GitOps RI, sensitive values were provided to Tanzu Sync by using the command line and environment variables. This is replaced by a SOPs encrypted file that is committed to the repository.

Follow these steps to upgrade the existing SOPs based installation:

1. Download and unpack the new version of Tanzu GitOps RI.
2. Overwrite the configuration for TAP installation and Tanzu Sync from the catalog:

   ```
   cd $HOME/tap-gitops

   ./setup-repo.sh CLUSTER-NAME sops
   ```

   Where:

   - `CLUSTER-NAME` the name of your cluster you want to upgrade.
   - `sops` selects the Secrets OPerationS-based secrets management variant. Changing between SOPs and any other secret management variant is not supported!

   Example:

```
cd $HOME/tap-gitops

./setup-repo.sh full-tap-cluster sops
Created cluster configuration in ./clusters/full-tap-cluster.
...
```

3. Prepare sensitive Tanzu Sync values.

Upgrade the existing ESO based installation

Follow these steps to upgrade the existing ESO based installation with secrets managed externally in AWS Secrets Manager:

1. Download and unpack the new version of Tanzu GitOps RI.
2. Overwrite the configuration for Tanzu Application Platform installation and Tanzu Sync from the catalog:

   ```
   cd $HOME/tap-gitops

   ./setup-repo.sh CLUSTER-NAME aws-secrets-manager
   ```

   Where:

   - `CLUSTER-NAME` the name of your cluster you want to upgrade.
   - `aws-secrets-manager` selects the AWS Secrets Manager external Secret Store.

   Example:

   ```
   cd $HOME/tap-gitops

   ./setup-repo.sh full-tap-cluster aws-secrets-manager
   Created cluster configuration in ./clusters/full-tap-cluster.
   ...
   ```

3. Generate the default configuration.

Verify the upgrade

Verify the versions of packages after the upgrade by running:

```
kubectl get packageinstall --namespace tap-install
```

Your output is similar, but probably not identical, to the following example output:

```
- Retrieving installed packages...
  NAME                                PACKAGE-NAME
PACKAGE-VERSION  DESCRIPTION
  accelerator                         accelerator.apps.tanzu.vmware.com
1.3.0            Reconcile succeeded
  api-auto-registration               apis.apps.tanzu.vmware.com
0.1.1            Reconcile succeeded
  api-portal                          api-portal.tanzu.vmware.com
1.2.2            Reconcile succeeded
  appliveview                         backend.appliveview.tanzu.vmware.com
1.3.0            Reconcile succeeded
  appliveview-connector               connector.appliveview.tanzu.vmware.com
1.3.0            Reconcile succeeded
  appliveview-conventions             conventions.appliveview.tanzu.vmware.com
1.3.0            Reconcile succeeded
  appsso                              sso.apps.tanzu.vmware.com
```

```
2.0.0            Reconcile succeeded
  buildservice                        buildservice.tanzu.vmware.com
1.7.1            Reconcile succeeded
  cartographer                        cartographer.tanzu.vmware.com
0.5.3            Reconcile succeeded
  cert-manager                        cert-manager.tanzu.vmware.com
1.7.2+tap.1      Reconcile succeeded
  cnrs                                cnrs.tanzu.vmware.com
2.0.1            Reconcile succeeded
  contour                             contour.tanzu.vmware.com
1.22.0+tap.3     Reconcile succeeded
  conventions-controller              controller.conventions.apps.tanzu.vmware.com
0.7.1            Reconcile succeeded
  developer-conventions               developer-conventions.tanzu.vmware.com
0.8.0            Reconcile succeeded
  fluxcd-source-controller            fluxcd.source.controller.tanzu.vmware.com
0.27.0+tap.1     Reconcile succeeded
  grype                               grype.scanning.apps.tanzu.vmware.com
1.3.0            Reconcile succeeded
  image-policy-webhook                image-policy-webhook.signing.apps.tanzu.vmware.c
om   1.1.7            Reconcile succeeded
  metadata-store                      metadata-store.apps.tanzu.vmware.com
1.3.3            Reconcile succeeded
  ootb-delivery-basic                 ootb-delivery-basic.tanzu.vmware.com
0.10.2           Reconcile succeeded
  ootb-supply-chain-testing-scanning  ootb-supply-chain-testing-scanning.tanzu.vmware.
com  0.10.2           Reconcile succeeded
  ootb-templates                      ootb-templates.tanzu.vmware.com
0.10.2           Reconcile succeeded
  policy-controller                   policy.apps.tanzu.vmware.com
1.1.1            Reconcile succeeded
  scanning                            scanning.apps.tanzu.vmware.com
1.3.0            Reconcile succeeded
  service-bindings                    service-bindings.labs.vmware.com
0.8.0            Reconcile succeeded
  services-toolkit                    services-toolkit.tanzu.vmware.com
0.8.0            Reconcile succeeded
  source-controller                   controller.source.apps.tanzu.vmware.com
0.5.0            Reconcile succeeded
  spring-boot-conventions             spring-boot-conventions.tanzu.vmware.com
0.5.0            Reconcile succeeded
  tap                                 tap.tanzu.vmware.com
1.6.1            Reconcile succeeded
  tap-auth                            tap-auth.tanzu.vmware.com
1.1.0            Reconcile succeeded
  tap-gui                             tap-gui.tanzu.vmware.com
1.3.0            Reconcile succeeded
  tap-telemetry                       tap-telemetry.tanzu.vmware.com
0.3.1            Reconcile succeeded
  tekton-pipelines                    tekton.tanzu.vmware.com
0.39.0+tap.2     Reconcile succeeded
```

Opt out of telemetry collection

This topic tells you how to opt out of the VMware Customer Experience Improvement Program (CEIP) and out of Pendo telemetry on an organizational level.

There are two components for telemetry collection in Tanzu Application Platform (commonly known as TAP) under the VMware Customer Experience Improvement Program (CEIP):

1. The standard CEIP telemetry collection
2. Pendo telemetry from Tanzu Developer Portal

Each telemetry component has its own opt-in and opt-out process. The CEIP telemetry opt-out decision can be made at an organizational level, whereas the decision regarding the Pendo telemetry is available both on an organizational level and at an individual user level.

When you install Tanzu Application Platform, both standard CEIP and Pendo telemetry are turned on by default. If you opt out of standard CEIP telemetry collection, VMware cannot offer you proactive support and the other benefits that accompany participation in the CEIP.

Turn off standard CEIP telemetry collection

To deactivate Pendo telemetry collection, see Enable or deactivate the Pendo telemetry for the organization later in the topic.

Note

If you decide to opt in to Pendo telemetry collection, each user is given the option to opt in or opt out. For more information, see Opt in or opt out of Pendo telemetry for Tanzu Developer Portal.

To turn off CEIP telemetry collection, follow these instructions:

kubectl

To turn off telemetry collection on Tanzu Application Platform by using kubectl:

1. Ensure that your Kubernetes context is pointing to the cluster where Tanzu Application Platform is installed.
2. Run:

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: Namespace
metadata:
  name: vmware-system-telemetry
---
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: vmware-system-telemetry
  name: vmware-telemetry-cluster-ceip
data:
  level: disabled
EOF
```

3. If you already have Tanzu Application Platform installed, restart the telemetry collector to apply the change:

```
kubectl delete pods --namespace tap-telemetry --all
```

Your Tanzu Application Platform deployment now no longer emits telemetry, and you have opted out of the CEIP.

Tanzu CLI

The Tanzu CLI provides a telemetry plug-in enabled by the Tanzu Framework v0.25.0, which is included in Tanzu Application Platform v1.3 and later.

To turn off telemetry collection on your Tanzu Application Platform by using the Tanzu CLI, run:

```
tanzu telemetry update --CEIP-opt-out
```

To learn more about how to update the telemetry settings, run:

```
tanzu telemetry update --help
```

Your Tanzu Application Platform deployment now no longer emits telemetry, and you have opted out of the CEIP.

Turn off Pendo telemetry collection

To deactivate the program for the entire organization, add the following parameters to your `tap-values.yaml` file:

```
tap_gui:
  app_config:
    pendoAnalytics:
      enabled: false
```

To enable Pendo telemetry for the organization, add the following parameters to your `tap-values.yaml` file:

```
tap_gui:
  app_config:
    pendoAnalytics:
      enabled: true
```

Opt in or opt out of Pendo telemetry for Tanzu Developer Portal

Tanzu Developer Portal uses Pendo.io to better understand the way users interact with it to provide a better user experience for VMware customers and to improve VMware products and services.

Pendo.io collects data based on your interaction with the software, such as clickstream data and page loads, hashed user ID, and limited browser and device information.

By default, each instance of Tanzu Developer Portal is assigned to a random organization ID to ensure that your sensitive information is not revealed.

However, you can choose to customize your organization ID and self-identify. Doing so allows VMware to observe account-level telemetry, such as frequency of portal use, most popular function, and so on. All personally identifiable information remains anonymized in any case. The organization name is hashed to prevent VMware from identifying you by the value.

To customize your organization ID, see Customize the Tanzu Developer Portal telemetry collection.

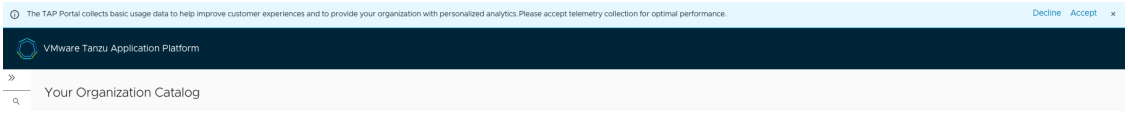
To enable or deactivate Pendo telemetry for the organization, see Enable or deactivate the Pendo telemetry for the organization.

Note

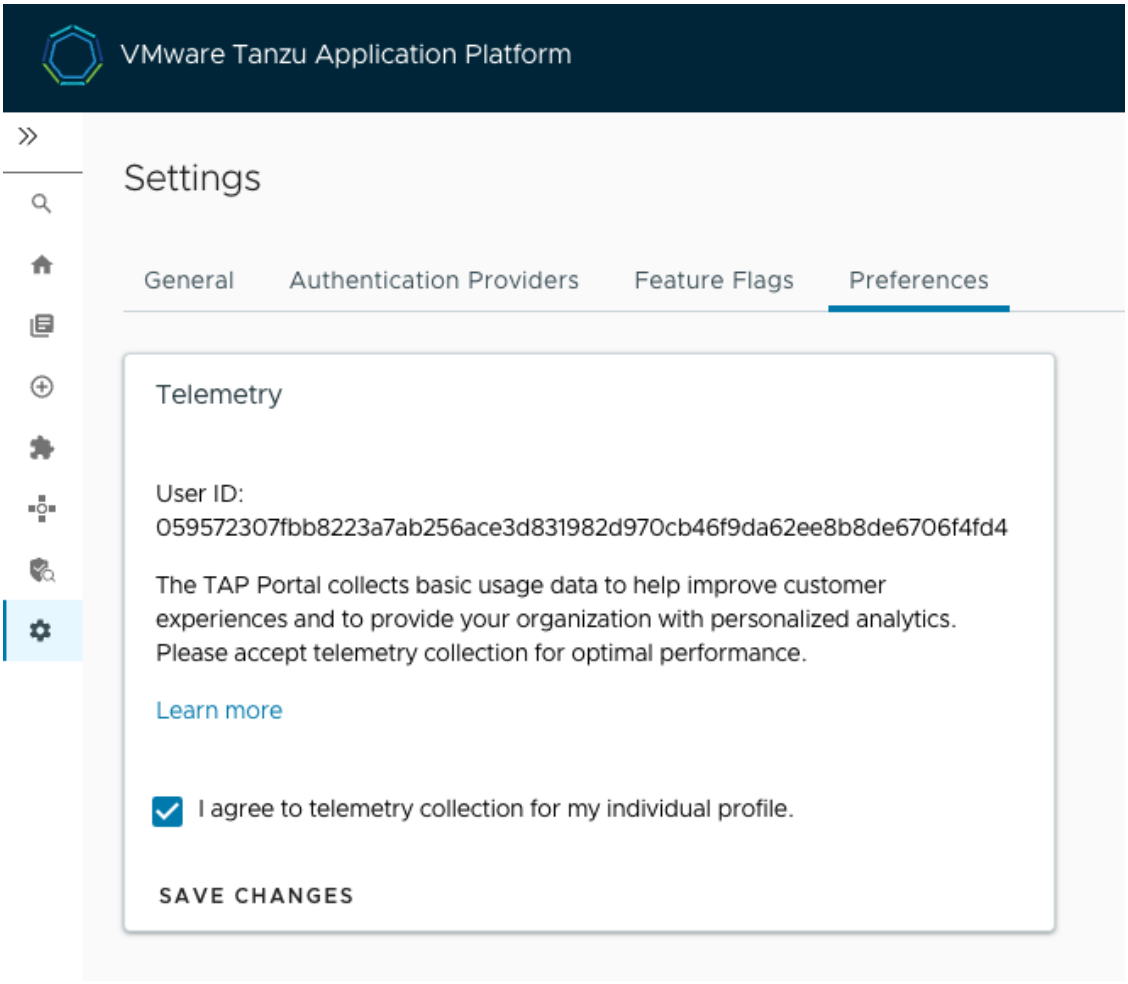
Pendo telemetry is separate from the VMware CEIP telemetry. There is a separate process for opting in or out of the VMware CEIP. For more information, see Opt out of telemetry collection.

Opt in or opt out of Pendo telemetry from Tanzu Developer Portal

After the Pendo telemetry is enabled for the organization, in accordance with VMware policy each user is prompted to agree to participate in the program or decline.



Each individual's preference is stored in Tanzu Developer Portal and can be modified at any time. To change your preferences, go to **Settings** > **Preferences**.



Request to delete your anonymized data

If you no longer want to participate in the program and you want VMware to delete all your anonymized data, please send an email requesting deletion, with your hashed User ID, to tap-pendo@groups.vmware.com.

This enables VMware to identify your anonymized data and delete it in accordance with the applicable regulations.

To find your hashed User ID, go to **Settings** > **Preferences** in Tanzu Developer Portal.

Access an experimental sandbox environment

This topic describes how you can access a hosted experimental Tanzu Application Platform (formerly known as TAP) environment.

VMware provides an experimental developer sandbox that you can use for 8 hours. Your sandbox session takes 2 minutes to start, though it can take up to 25 minutes. The sandbox environment includes the following services:

- Spring Cloud Services: Application Config Service, Spring Cloud Gateway and AppSSO
- Data Services: PostgreSQL, MySQL, RabbitMQ, Kafka and MongoDB

Procedure

1. Create a VMware Customer Connect account and log in to Tanzu Academy.
2. Get started with your developer sandbox in Tanzu Academy.

Note

This is an evaluation environment only, do not use it for production applications or to store sensitive data.

Scale workloads

This topic describes the best practices required to build and deploy workloads at scale.

Sample application reference

The following sections describe the configuration of the different-size applications used to derive scalability best practices.

Small

This is the simplest configuration and consists of the following services and workloads:

- API Gateway workload
- Search workload with in-memory database
- Search processor workload
- Availability workload with in-memory database
- UI workload
- 3 Node RabbitMQ cluster

Medium

This includes all of the services of the small-size application and the following services and workloads:

- Notify workload
- Persistent Database, MySQL or Postgres

Large

This includes all of the services of the medium size application and the following services and workloads:

- Crawler Service
- Redis

Application Configuration

The following section describes the application configuration used to derive the scalability best practices.

Supply chains used:

- Out of the Box Supply Chain with Testing and Scanning (Build+Run)
- Out of the Box Supply Chain Basic + Out of the Box Supply Chain with Testing (Iterate)

Workload type: `web`, `server` + `worker`

Kubernetes Distribution: Azure Kubernetes Service

Number of applications deployed concurrently: 50–55

	CPU	Memory Range	Workload CRs in Iterate	Workload CRs in Build+Run	Workload Transactions per second
Small	500m - 700m	3-5 GB	4	5	4
Medium	700m - 1000m	4-6 GB	NA	6	4
Large	1000m - 1500m	6-8 GB	NA	7	4

Scale configuration for workload deployments

This section describes cluster sizes for deploying a 1K workload.

Node configuration: 4 vCPUs, 16 GB RAM, 120 GB Disk size

Cluster Type / Workload Details	Shared Iterate Cluster	Build Cluster	Run Cluster 1	Run Cluster 2	Run Cluster 3
No. of Namespaces	300	333	333	333	333
Small	300	233	233	233	233
Medium		83	83	83	83
Large		17	17	17	17
No. of Nodes	90	60	135	135	135

Best Practices

The following table describes the resource limit changes that are required for components to support the scale configuration described in the previous table.

Controller/Pod	CPU Requests/Limits	Memory Requests/Limits	Other changes	Build	Run	Iterate	Changes made in
AMR Observer	200 m/1000 m	2 Gi/**3 Gi**	n/a	Yes	Yes	No	`tap-values.yaml`
Build Service/kpack controller	20 m/100 m	**1 Gi/2 Gi**	n/a	Yes	No	Yes	`tap-values.yaml`
Scanning/scan-link	200 m/500 m	**1 Gi/3 Gi**	“SCAN_JOB_TTL_SECONDS_AFTER_FINISHED” - 10800*	Yes	No	No	`tap-values.yaml`
Cartographer	**3000 m/4000 m**	**10 Gi/10 Gi**	In `tap-values.yaml`, change `concurrency` to 25.	Yes	Partial (only CPU)	Yes	`tap-values.yaml`
Cartographer conventions	100 m/100 m	20 Mi/**1.8 Gi**	n/a	Yes	Yes	Yes	`tap-values.yaml`

Controller/Pod	CPU Requests/Limits	Memory Requests/Limits	Other changes	Build	Run	Iterate	Changes made in
Namespace Provisioner	100 m/500 m	**500 Mi/2 Gi**	n/a	Yes	Yes	Yes	`tap-values.yaml`
Cnrs/knative-controller	100 m/1000 m	**1 Gi/3 Gi**	n/a	No	Yes	Yes	`tap-values.yaml`
Cnrs/net-contour	40 m/400 m	**512 Mi/2 Gi**	In `tap-values.yaml`, change Contour envoy workload type from `Daemonset` to `Deployment`.	No	Yes	Yes	`tap-values.yaml`
Cnrs/activator	300 m/1000 m	**5 Gi/5 Gi**	n/a	No	Yes	No	`tap-values.yaml`
Cnrs/autoscaler	100 m/1000 m	**2 Gi/2 Gi**	n/a	No	Yes	No	`tap-values.yaml`
Services Toolkit Controller	100 m/200 m	**750 Mi/1.5 Gi**	n/a	No	Yes	Yes	overlay
tap-telemetry/tap-telemetry-informer	100 m/1000 m	100 Mi/**2 Gi**	n/a	Yes	No	Yes	`tap-values.yaml`
App SSO/App SSO Controller	20 m/500 m	**512 Mi/2 Gi**	n/a	No	Yes	Yes	`tap-values.yaml`
Tekton	n/a	n/a	Update default timeout for Tekton pipeline	Yes	No	Yes	`tap-values.yaml`
App Scanning	n/a	n/a	Update `maxConcurrentScan` to 100	Yes	No	No	`tap-values.yaml`
SpringBoot Conventions	n/a	n/a	Update `livenessProbe`	No	Yes	Yes	`tap-values.yaml`

- CPU is measured in millicores. m = millicore. 1000 millicores = 1 vCPU.
- Memory is measured in Mebibyte and Gibibyte. Mi = Mebibyte. Gi = Gibibyte
- In the CPU Requests/Limits column and the Memory Requests/Limits, the changed values are bolded. Non-bolded values are the default ones set during a Tanzu Application Platform installation.
- In the CPU Requests/Limits column, some of the request and limits values are set equally so that the pod is allocated in a node where the requested limit is available.

* Only when there is an issue with scan pods getting deleted before Cartographer can process it

Example resource limit changes

The following section provides examples of the changes required to the default limits to achieve scalability:

Cartographer

The default Cartographer concurrency limits are:

```
cartographer:
  cartographer:
    concurrency:
      max_workloads: 2
      max_deliveries: 2
      max_runnables: 2
```

Edit `values.yaml` to scale Cartographer concurrency limits. Configure the node with 4 vCPUs, 16 GB RAM, and 120 GB disk size:

```
cartographer:
  cartographer:
    concurrency:
      max_workloads: 25
      max_deliveries: 25
      max_runnables: 25
```

The default resource limits are:

```
resources:
  limits:
    cpu: 1
    memory: 1Gi
  requests:
    cpu: 500m
    memory: 512Mi
```

Edit `values.yaml` to scale resource limit:

```
# build-cluster
cartographer:
  cartographer:
    resources:
      limits:
        cpu: 4
        memory: 10Gi
      requests:
        cpu: 3
        memory: 10Gi

# run-cluster
cartographer:
  cartographer:
    resources:
      limits:
        cpu: 4
        memory: 2Gi
      requests:
        cpu: 3
        memory: 1G
```

Cartographer Conventions

The default resource limits are:

```
resources:
  limits:
    cpu: 100m
    memory: 256Mi
  requests:
    cpu: 100m
    memory: 20Mi
```

Edit `values.yaml` to scale resource limit:

```
cartographer_conventions:
  resources:
    limits:
      memory: 1.8Gi
```

Scan-link-controller

The default resource limits are:

```
resources:
  limits:
    cpu: 250m
    memory: 256Mi
  requests:
    cpu: 100m
    memory: 128Mi
```

Edit `values.yaml` to scale resource limit:

```
scanning:
  resources:
    limits:
      cpu: 500m
      memory: 3Gi
    requests:
      cpu: 200m
      memory: 1Gi
```

AMR Observer

The default resource limits are:

```
resources:
  limits:
    cpu: 500m
    memory: 512Mi
  requests:
    cpu: 100m
    memory: 256Mi
```

Edit `values.yaml` to scale resource limits:

```
amr:
  observer:
    app_limit_cpu: 1000m
    app_limit_memory: 3Gi
    app_req_cpu: 200m
    app_req_memory: 2Gi
```

kpack-controller in Tanzu Build Service

The default resource limits are:

```
resources:
  limits:
    memory: 1Gi
  requests:
    cpu: 20m
    memory: 1Gi
```

Edit `values.yaml` to scale resource limits:

```
buildservice:
  controller:
    resources:
      limits:
        memory: 2Gi
        cpu: 100m
      requests:
        memory: 1Gi
        cpu: 20m
```

Namespace Provisioner

The default resource limits are:

```
resources:
  limits:
    cpu: 500m
    memory: 100Mi
  requests:
    cpu: 100m
    memory: 20Mi
```

Edit `values.yaml` to scale resource limits:

```
namespace_provisioner:
  controller_resources:
    resources:
      limits:
        cpu: 500m
        memory: 2Gi
      requests:
        cpu: 100m
        memory: 500Mi
```

Cloud Native Runtimes Knative Serving

The default resource limits are:

```
resources:
  limits:
    cpu: 1
    memory: 1000Mi
  requests:
    cpu: 100m
    memory: 100Mi
```

Edit `values.yaml` to scale resource limits:

```
cnrs:
  resource_management:
  - name: "controller"
```

```
    limits:
      cpu: 1000m
      memory: 3Gi
    requests:
      cpu: 100m
      memory: 1Gi
```

net-contour controller

Change deployment type from Daemonset to Deployment.

```
contour:
  envoy:
    workload:
      type: Deployment
      replicas: 3
```

The default resource limits are:

```
resources:
  limits:
    cpu: 400m
    memory: 400Mi
  requests:
    cpu: 40m
    memory: 40Mi
```

Edit `values.yaml` to scale resource limits:

```
cnrs:
  resource_management:
  - name: "net-contour-controller"
    limits:
      cpu: 400m
      memory: 2Gi
    requests:
      cpu: 40m
      memory: 512Mi
```

Autoscaler

The default resource limits are:

```
resources:
  limits:
    cpu: 1
    memory: 1000Mi
  requests:
    cpu: 100m
    memory: 100Mi
```

Edit `values.yaml` to scale resource limits:

```
cnrs:
  resource_management:
  - name: "autoscaler"
    limits:
      cpu: 1000m
      memory: 2Gi
    requests:
      cpu: 100m
      memory: 2Gi
```

Activator

The default resource limits are:

```
resources:
  limits:
    cpu: 1
    memory: 600Mi
  requests:
    cpu: 300m
    memory: 60Mi
```

Edit `values.yaml` to scale resource limits:

```
cnrs:
  resource_management:
  - name: "activator"
    limits:
      cpu: 1000m
      memory: 5Gi
    requests:
      cpu: 300m
      memory: 5Gi
```

Tanzu Application Platform Telemetry

The default resource limits are:

```
resources:
  limits:
    cpu: 1
    memory: 1000Mi
  requests:
    cpu: 100m
    memory: 100Mi
```

Edit `values.yaml` to scale resource limits:

```
tap_telemetry:
  limit_memory: 2Gi
```

Application Single Sign-On

The default resource limits are:

```
resources:
  limits:
    cpu: 500m
    memory: 5000Mi
  requests:
    cpu: 20m
    memory: 100Mi
```

Edit `values.yaml` to scale resource limits:

```
appsso:
  resources:
    limits:
      memory: 1Gi
    requests:
      memory: 512Mi
```

Services Toolkit Controller

The default resource limits are:

```
resources:
  limits:
    cpu: 200m
    memory: 500Mi
  requests:
    cpu: 100m
    memory: 100Mi
```

Edit `values.yaml` to scale resource limits:

```
services_toolkit:
  controller:
    resources:
      requests:
        cpu: "200m"
        memory: "750Mi"
      limits:
        cpu: "200m"
        memory: "1.5Gi"
```

Services Toolkit Resource Claims API Server

The default resource limits are:

```
resources:
  limits:
    cpu: 120m
    memory: 500Mi
  requests:
    cpu: 100m
    memory: 100Mi
```

Edit `values.yaml` to scale resource limits:

```
services_toolkit:
  resource_claims_apiserver:
    resources:
      requests:
        cpu: "200m"
        memory: "750Mi"
      limits:
        cpu: "200m"
        memory: "1.5Gi"
```

FluxCD Source Controller

The default resource limits are:

```
resources:
  limits:
    cpu: 1000m
    memory: 1Gi
  requests:
    cpu: 50m
    memory: 64Mi
```

To change the resource limits for FluxCD Source controller, follow the procedure in Configure resource Limits.

Tekton Pipeline default timeout

The default resource limits are:

```
tekton_pipelines:
  defaults:
    timeout_minutes: "60"
```

Edit `values.yaml` to scale resource limits:

```
tekton_pipelines:
  defaults:
    timeout_minutes: "120"
```

App Scanning maximum concurrent scan (Scan 2.0)

The default resource limits are:

```
app_scanning:
  scans:
    maxConcurrentScans: 10
```

Edit `values.yaml` to scale resource limits:

```
app_scanning:
  scans:
    maxConcurrentScans: 100
```

For more information, see Supply Chain Security Tools - Scan 2.0.

Spring Boot Conventions

The default resource limits are:

```
springboot_conventions:
  livenessProbe:
    initialDelaySeconds: 0
```

Edit `values.yaml` to scale resource limits:

```
springboot_conventions:
  livenessProbe:
    initialDelaySeconds: 45
```

kube-dns resource limit changes for GKE clusters

The default memory limit for kube-dns pods is set to 210 Mi. VMware recommends that you set it to 1 Gi. For instructions, see the GKE documentation.

Overview of security and compliance in Tanzu Application Platform

Security is a primary focus for Tanzu Application Platform (commonly known as TAP).

This section describes how to:

Secure exposed ingress endpoints in Tanzu Application Platform

Plan ingress certificates inventory in Tanzu Application Platform

Use custom CA certificates in Tanzu Application Platform

Assess Tanzu Application Platform against the NIST 800-53 Moderate Assessment

Harden Tanzu Application Platform

See also:

External Secrets Operator

Secure exposed ingress endpoints in Tanzu Application Platform

This topic tells you how to secure exposed ingress endpoints with TLS in Tanzu Application Platform (commonly known as TAP).

Tanzu Application Platform exposes ingress endpoints so that:

- Platform operators and application developers can interact with the platform.
- End users can interact with applications running on the platform.

There are two ways of configuring ingress certificates to secure endpoints with TLS, such as `https://`:

- Shared ingress issuer
- Component-level configuration

Shared ingress issuer

Ingress communication uses TLS by default. VMware recommends a shared ingress issuer for issuing ingress certificates on Tanzu Application Platform.

The shared ingress issuer is an on-platform representation of a certificate authority. It provides a method to set up TLS for the entire platform. The shared ingress issuer issues ingress certificates for all participating components.

Default self-signed ingress issuer

By default, Tanzu Application Platform installs and uses a self-signed CA ingress issuer for all components. The default ingress issuer is a self-signed `cert-manager.io/v1/ClusterIssuer` provided by the cert-manager package.

The ingress issuer is designated by the Tanzu Application Platform configuration value `shared.ingress_issuer`. It defaults to `tap-ingress-selfsigned`.

Caution

The default ingress issuer is appropriate for testing and evaluation. VMware recommends that you replace the default self-signed issuer with your own issuer.

Default self-signed issuer limitations

The default ingress issuer represents a self-signed certificate authority. This is not problematic as far as security is concerned, however, it is not included in any trust chain configured. As a result, nothing trusts the default ingress issuer implicitly, not even Tanzu Application Platform components. While the issued certificates are valid in principle, they are rejected, for example, by your browser. Furthermore, some interactions between components are not functional by default.

Default self-signed issuer prerequisites

To use the Tanzu Application Platform ingress issuer, your certificate authority must be representable by a cert-manager `ClusterIssuer`. You need one of the following:

- Your own CA certificate
- Your CA is an ACME, Venafi, or Vault-based issuer, for example, LetsEncrypt
- Your CA can be represented by an external cert-manager `ClusterIssuer`.

Without one of the above, you cannot use the ingress issuer, but you can still configure TLS for components. For more information, see Ingress certificates inventory.

Important

If `cert-manager.tanzu.vmware.com` is excluded from the installation, then `tap-ingress-selfsigned` is not installed either. In this case, bring your own ingress issuer.

Trust the default, self-signed issuer

You can trust the default ingress issuer by including `tap-ingress-selfsigned`'s certificate in the Tanzu Application Platform trusted CA certificates and your device's certificate chain.

Caution

This approach is discouraged. Instead, replace the default ingress issuer.

1. Obtain `tap-ingress-selfsigned`'s PEM-encoded certificate

   ```
   kubectl get secret \
     tap-ingress-selfsigned-root-ca \
     --namespace cert-manager \
     --output go-template='{{ index .data "tls.crt" | base64decode }}'
   ```

2. Add the certificate to custom CA certificates by appending it to `shared.ca_cert_data` and applying Tanzu Application Platform's installation values.

3. Add the certificate to your device's trust chain. The trust chain will vary depending on your operating system and privileges.

Replace the default ingress issuer

Tanzu Application Platform's default ingress issuer can be replaced by any other cert-manager-compliant ClusterIssuer.

To replace the default ingress issuer:

Custom CA
Complete the following steps:

Prerequisites

You need your own CA certificates and private key for this.

1. Create your `ClusterIssuer`

   Create a `Secret` and `ClusterIssuer` which represent your CA on the platform:

   ```
   ---
   apiVersion: v1
   kind: Secret
   type: kubernetes.io/tls
   metadata:
     name: my-company-ca
     namespace: cert-manager
   stringData:
     tls.crt: #! your CA's PEM-encoded certificate
     tls.key: #! your CA's PEM-encoded private key

   ---
   apiVersion: cert-manager.io/v1
   kind: ClusterIssuer
   metadata:
     name: my-company
   spec:
     ca:
       secretName: my-company-ca
   ```

2. Set `shared.ingress_issuer` to the name of your issuer:

   ```
   #! my-tap-values.yaml
   #! ...
   shared:
     ingress_issuer: my-company-ca
   #! ...
   ```

3. Apply the Tanzu Application Platform installation values file.

   When the configuration is applied, components obtain certificates from the new issuer and serve them.

LetsEncrypt production
Complete the following steps

Prerequisites

- Public CAs, like LetsEncrypt, record signed certificates in publicly-available certificate logs for the purpose of certificate transparency. Ensure that you are OK with this before using LetsEncrypt.
- LetsEncrypt's production API has rate limits.

- LetsEncrypt requires your `shared.ingress_domain` to be accessible from the Internet.
- Depending on your setup, you might need to adjust .spec.acme.solvers
- Replace `.spec.acme.email` with the email that should receive notices for certificates from LetsEncrypt.

Caution

ACME HTTP01 challenges can fail under certain conditions. For more information, see ACME challenges.

1. Create a `ClusterIssuer` for Let's Encrypts production API:

```
---
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-production
spec:
  acme:
    email: certificate-notices@my-company.com
    privateKeySecretRef:
      name: letsencrypt-production
    server: https://acme-v02.api.letsencrypt.org/directory
    solvers:
      - http01:
          ingress:
            class: contour
```

2. Set `shared.ingress_issuer` to the name of your issuer

```
#! my-tap-values.yaml
#! ...
shared:
  ingress_issuer: letsencrypt-production
#! ...
```

3. Apply Tanzu Application Platform installation values

   When the configuration is applied, components obtain certificates from the new issuer and serve them.

LetsEncrypt staging

Complete the following steps

Prerequisites

- Public CAs - like LetsEncrypt - record signed certificates in publicly-available certificate logs for the purpose of certificate transparency. Ensure that you are OK with this before using LetsEncrypt.
- LetsEncrypt's staging API is not a publicly-trusted CA. You have to add its certificate to your devices trust chain and Tanzu Application Platform's custom CA certificates.
- LetsEncrypt requires your `shared.ingress_domain` to be accessible from the Internet.
- Depending on your setup you might need to adjust .spec.acme.solvers.
- Replace `.spec.acme.email` with the email that should receive notices for certificates from LetsEncrypt.

Caution

ACME HTTP01 challenges can fail under certain conditions. For more information, see ACME challenges.

1. Create a `ClusterIssuer` for Let's Encrypts staging API:

```
---
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-staging
spec:
  acme:
    email: certificate-notices@my-company.com
    privateKeySecretRef:
      name: letsencrypt-production
    server: https://acme-staging-v02.api.letsencrypt.org/directory
    solvers:
      - http01:
          ingress:
            class: contour
```

2. Set `shared.ingress_issuer` to the name of your issuer

```
#! my-tap-values.yaml
#! ...
shared:
  ingress_issuer: letsencrypt-staging
#! ...
```

3. Apply Tanzu Application Platform installation values

   After the configuration is applied, components obtain certificates from the new issuer and serve them.

Other

Complete the following steps

You can use any other cert-manager-supported `ClusterIssuer` as an ingress issuer for Tanzu Application Platform.

cert-manager supports a host of in-tree and out-of-tree issuers. See cert-manager's documentation of issuers.

1. Set `shared.ingress_issuer` to the name of your issuer

```
#! my-tap-values.yaml
#! ...
shared:
  ingress_issuer: my-company-ca
#! ...
```

2. Apply Tanzu Application Platform's installation values

   After the configuration is applied, components obtain certificates from the new issuer and serve them.

There are many ways and tools to assert that new certificates are issued and served. It is best to connect to one of the ingress endpoints and inspect the certificate it serves.

The `openssl` command-line utility is available on most operating systems. The following command retrieves the certificate from an ingress endpoint and shows its text representation:

```
# replace tap.example.com with your Tanzu Application Platform installation's ingress
domain
openssl s_client -showcerts -servername tap-gui.tap.example.com -connect tap-gui.tap.e
xample.com:443 <<< Q | openssl x509 -text -noout
```

Alternatively, use a browser to navigate to the ingress endpoint and click the lock icon in the navigation bar to inspect the certificate.

Component-level configuration

In some situations, depending on prerequisites, the shared ingress issuer is not the right choice. You can override configuration of TLS and certificates per component. A component's ingress and TLS configuration take precedence over the shared ingress issuer.

For a list of components with ingress and how to customize them, see Plan Ingress certificates inventory in Tanzu Application Platform.

Tanzu Application Platform also has limited support for wildcard certificates. For more information, see Wildcards.

Deactivate TLS for ingress

While VMware does not recommend it, you can deactivate the ingress issuer by setting `shared.ingress_issuer: ""`.

Plan ingress certificates inventory in Tanzu Application Platform

This topic tells you how to plan for TLS certificates in Tanzu Application Platform (commonly known as TAP).

The effective number of ingress endpoints can vary widely, depending on the installation profile, excluded packages, and end-user-facing resources such as `Workload`, and `AuthServer`. As a result, the number of TLS certificates is not fixed but is a function of the platform's configuration and tenancy.

Ingress refers to any resource that facilitates ingress, for example, core `Ingress` and Contour's `HTTPProxy`.

Wildcards

You can use wildcard certificates but Tanzu Application Platform does not offer support. Wildcard certificates require component-level configuration.

You can use a mixed approach for configuring TLS for components. For example, use a shared ingress issuer, but override TLS configuration for select components while using wildcard certificates for some.

When using wildcard certificates the approach differs between components that have a fixed set of ingress endpoints and those that have a variable set of ingress endpoints:

Components with ingress endpoints can have a fixed or variable number of ingress endpoints:

- Components with a fixed set of ingress endpoints can receive a reference to the wildcard certificate's `Secret` and an ingress domain, for example, Tanzu Developer Portal.
- Components with a variable set of ingress endpoints usually offer Kubernetes APIs that create ingress resources. These components allow configuration of domain templating so that wildcard certificates can be used, for example, Cloud Native Runtimes and Application Single Sign-On.

Ingress support for components

Use the following table to help with the planning and accounting of TLS certificates. For a full list of components and the profiles supported for each component, see About Tanzu Application Platform components and profiles.

Package name	Ingress purpose	Supports ingress issuer	Supports wildcards	Number of ingress	SANs*
api-portal.tanzu.vmware.com	Serves the API portal	No	Yes	`1`	`api-portal.INGRESS-DOMAIN`
cnrs.tanzu.vmware.com	Instances of Knative's `Service` have ingress	Yes	Yes	`Number of Services`	SANs depend on the component's `domain_template`
metadata-store.apps.tanzu.vmware.com	Serves the Supply Chain Security Tools store	Yes	Yes	`1`	`metadata-store.INGRESS-DOMAIN`
spring-cloud-gateway.tanzu.vmware.com	Instances of SpringCloudGateway have ingress	No	Yes	`Number of SpringCloudGateways`	See Using an Ingress Resource in the Spring Cloud Gateway documentation
sso.apps.tanzu.vmware.com	Instances of AuthServer have ingress	Yes	Yes	`Number of AuthServers`	Depends on the component's `domain_template`
tap-gui.tanzu.vmware.com	Serves the platform-internal developer and service portal	Yes	Yes	`1`	`tap-gui.INGRESS-DOMAIN`

*The SANs is configurable for components in the following two ways:

- Components that install a single ingress resource in the form of `COMPONENT.DOMAIN-NAME`, such as Tanzu Developer Portal
- Components that install an ingress resource per API instance that gets templated from a `domain_template` feeding `DOMAIN-NAME`, such as `cnrs.tanzu.vmware.com` and `sso.apps.tanzu.vmware.com`

Use custom CA certificates in Tanzu Application Platform

This topic tells you about configuring custom CA certificates in Tanzu Application Platform (commonly known as TAP).

For egress communication, you can establish trust for custom CA certificates. This is helpful if any Tanzu Application Platforms components are connecting to services that serve certificates issued by private certificate authorities.

The `shared.ca_cert_data` installation value can contain a PEM-encoded CA bundle. Each component then trusts the CAs contained in the bundle.

You can also configure trust per component by providing a CA bundle in the component's installation values. The component then trusts those CAs and the CAs configured in `shared.ca_cert_data`. For more information, see components.

For example:

```
#! my-tap-values.yaml

shared:
  ca_cert_data: |
    Corporate CA 1
    -----BEGIN CERTIFICATE-----
    MIIFmDCCA4....
    -----END CERTIFICATE-----
    Corporate CA 2
    -----BEGIN CERTIFICATE-----
    MIIFkzCCA3....
    -----END CERTIFICATE-----
```

Some Tanzu Application Platform components do not support the `shared.ca_cert_data` feature, such as Flux CD, Tekton, and the External Secrets operator. Any custom CA certificates must be configured directly by those components.

For information about using Git with a custom CA in supply chains and configuring the `caFile` parameter, see Git Authentication.

Assess Tanzu Application Platform against the NIST 800-53 Moderate Assessment

This topic provides you with an assessment of Tanzu Application Platform (commonly known as TAP) against the NIST SP 800-53 Revision 4 Moderate baseline controls. This translates to FISMA Moderate and CNSSI 1253 Mod/Mod/Mod for use in US Federal systems accreditation.

The Moderate baseline applies to only technical controls. Organizational policy controls, physical security, media policies, and similar are excluded as they are not applicable to Tanzu Application Platform. These excluded controls are still relevant to the system at large and must be inherited from existing accreditations or otherwise addressed.

The initial iteration of this assessment delineates responsible parties. Incremental updates will add more details about implementation and updates to 800-53 Revision 5.

Name	Title	Responsible Party	Notes
AC-2(1)	Automated System Account Management	Customer	Implemented on customer identity store. The customer must employ automated mechanisms to support the management of information system accounts used to access their Tanzu Application Platform installation.
AC-2(2)	Removal of Temporary / Emergency Accounts	Customer	Implemented on customer identity store. If the customer chooses to use temporary or emergency accounts, they must ensure that the system automatically deactivates or removes the account following an organization-defined time period.
AC-2(3)	Deactivate Inactive Accounts	Customer	Implemented on customer identity store. The customer must automatically deactivate inactive accounts used to access their Tanzu Application Platform installation following an organization-defined time period of inactivity.

Name	Title	Responsible Party	Notes
AC-2(4)	Automated Audit Actions	Customer	Implemented on customer identity store. The customer must automatically audit account creation, modification, enabling, deactivating, and removal actions associated with accounts used to access their Tanzu Application Platform installation and must notify an organization-defined personnel or role.
AC-3	Access Enforcement	Customer	The customer must federate their IdP with Tanzu Application Platform to enforce approved access authorizations to their Tanzu Application Platform installation.
AC-4	Information Flow Enforcement	Customer	The customer is responsible for enforcing approved authorizations for controlling the flow of information between Tanzu Application Platform and interconnected systems, based on organization-defined information flow control policies, for example, a SIEM. Tanzu Application Platform does not restrict intra-service or inter-system communication. Future versions of Tanzu Application Platform will include this feature using service mesh architecture or similar methods.
AC-6	Least Privilege	Shared	The customer is responsible for enforcing least privilege by ensuring Tanzu Application Platform users have the minimum permissions necessary to perform their job function. Tanzu Application Platform is responsible for providing RBAC functionality to enforce least privilege.
AC-6(1)	Authorize Access to Security Functions	Shared	The customer is responsible for explicitly authorizing access to organization-defined security functions and security-relevant information as it relates to their Tanzu Application Platform installation. Tanzu Application Platform is responsible for providing the RBAC functionality necessary to restrict which users can access security functions and security-related information.
AC-6(5)	Privileged Accounts	Shared	The customer must restrict privileged Tanzu Application Platform accounts to organization-defined personnel or roles. Tanzu Application Platform is responsible for providing the RBAC functionality for customers to restrict privileged Tanzu Application Platform accounts to organization-defined personnel or roles.
AC-6(9)	Auditing Use of Privileged Functions	Shared	The customer is responsible for configuring Tanzu Application Platform and underlying Kubernetes to send log streams to their SIEM tool for log analysis to be capable of auditing the execution of privileged functions. Tanzu Application Platform is responsible for generating logs pertaining to the execution of privileged functions that can be ingested by the customer SIEM tool for analysis.
AC-6(10)	Prohibit Non-Privileged Users from Executing Privileged Functions	Tanzu Application Platform	This functionality is inherent to Tanzu Application Platform/Kubernetes RBAC and can't be configured otherwise.
AC-7 AC-7a AC-7b	Unsuccessful Logon Attempts	Customer	Implemented on customer identity provider. The customer is responsible for configuring their IdP to enforce a limit of consecutive invalid logon attempts by a user during an organization-defined time period which locks the user's account for an organization-defined time period, or until released by an administrator.

Name	Title	Responsible Party	Notes
AC-8 AC-8a AC-8a.1 AC-8a.2 AC-8a.3 AC-8a.4 AC-8b AC-8c AC-8c.1 AC-8c.2 AC-8c.3	System Use Notification	Customer	Implemented on customer identity provider. Customer must configure their IdP to display the system use notification banner before login.
AC-11 AC-11a AC-11b	Session Lock	Customer	The customer must configure sessions locks on user workstations used to access their Tanzu Application Platform installation. Tanzu Application Platform does not have a concept of session locks and relies on sessions locks applied by the user's workstation. Tanzu Application Platform provides logout functionality in place of session locking.
AC-11(1)	Pattern-Hiding Displays	Customer	The customer must configure sessions locks on user workstations used to access their Tanzu Application Platform installation. This includes hiding the user's private session with a publicly available image. Tanzu Application Platform does not have a concept of session locks and relies on sessions locks applied by the user's workstation. Tanzu Application Platform provides logout functionality in place of session locking.
AC-12	Session Termination	Shared	Implemented on customer identity provider. The customer is responsible for configuring IdP token TTL and refresh policies that apply to Tanzu Application Platform sessions. Tanzu Application Platform enforces token policies and cannot be configured otherwise.
AC-14 AC-14a	Permitted Actions Without Identification or Authentication	Shared	The customer is responsible for identifying organization-defined user actions that can be performed on the information system without identification or authentication consistent with organizational missions/business functions. For production installations, Tanzu Developer Portal must be configured with OIDC authentication and guest access deactivated.
AC-17(1)	Automated Monitoring / Control	Customer	"Remote Access" is defined as outside-the-org endpoints like remote workers over VPN. This is outside the scope of Tanzu Application Platform. The customer is responsible for all aspects regarding "remote access" to Tanzu Application Platform.
AC-17(2)	Protection of Confidentiality / Integrity Using Encryption	Customer	"Remote Access" is defined as outside-the-org endpoints like remote workers over VPN. This is outside the scope of Tanzu Application Platform. The customer is responsible for implementing cryptographic mechanisms to protect the confidentiality and integrity of "remote access" sessions to Tanzu Application Platform.
AC-17(3)	Managed Access Control Points	Customer	"Remote Access" is defined as outside-the-org endpoints like remote workers over VPN. This is outside the scope of Tanzu Application Platform. The customer is responsible for routing all "remote accesses" to Tanzu Application Platform through an organization-defined number of managed network access control points.
AC-19 AC-19 AC-19b	Access Control for Mobile Devices	Customer	The customer is responsible for all aspects regarding mobile devices which grant access to Tanzu Application Platform.
AU-3	Content of Audit Records	Tanzu Application Platform	The Tanzu Application Platform application must be capable of generating audit logs that contain the minimum content required by the customer consuming the application.

Name	Title	Responsible Party	Notes
AU-3(1)	Additional Audit Information	Customer	Implemented on customer SIEM. The customer is responsible for parsing Tanzu Application Platform logs on their SIEM to extract organization-defined extra information.
AU-4	Audit Storage Capacity	Customer	Implemented on customer Kubernetes. Tanzu Application Platform logs are all captured by Kubernetes logging. The customer is responsible for configuring their Kubernetes hosts with record storage capacity to ensure that there is adequate storage of logs generated by Tanzu Application Platform clusters.
AU-5 AU-5a AU-5b	Response to Audit Processing Failures	Customer	Implemented on customer Kubernetes. Tanzu Application Platform audit records are collected and managed by Kubernetes and are out of Tanzu Application Platform scope. The customer is responsible for configuring their Kubernetes hosts to account for audit processing failures and to alert the appropriate personnel responsible to take appropriate action.
AU-7 AU-7a AU-7b	Audit Reduction and Report Generation	Customer	Implemented on customer Kubernetes and SIEM Tanzu Application Platform audit records are collected and managed by Kubernetes. The customer is responsible for ensuring that Kubernetes ships Tanzu Application Platform audit records to a central SIEM for review and analysis.
AU-7(1)	Automatic Processing	Customer	Implemented on customer Kubernetes and SIEM Tanzu Application Platform audit records are collected and managed by Kubernetes. The customer is responsible for ensuring that Kubernetes ships Tanzu Application Platform audit records to a central SIEM for review and analysis.
AU-8 AU-8a AU-8b	Time Stamps	Tanzu Application Platform	Tanzu Application Platform components pull their system time from the container OS and the Kubernetes host and cannot be configured otherwise. Tanzu Application Platform components log statements include UTC timestamps and cannot be configured otherwise.
AU-8(1) AU-8(1)(a) AU-8(1)(b)	Synchronization With Authoritative Time Source	Customer	The customer is responsible for configuring authoritative time sources on K8 clusters.
AU-9	Protection of Audit Information	Customer	Tanzu Application Platform audit records are collected and managed by Kubernetes. The customer is responsible for protecting Kubernetes and Kubernetes logging configurations from unauthorized access, modification, and deletion.
AU-12 AU-12a AU-12b AU-12c	Audit Generation	Shared	Tanzu Application Platform audit records are collected and managed by Kubernetes. The customer is responsible for ensuring that Kubernetes ships Tanzu Application Platform audit records to a central SIEM for review and analysis. Tanzu Application Platform cannot be configured to audit specific information. Tanzu Application Platform logs verbosely and lets the customer filter out what is relevant to them using their SIEM. Tanzu Application Platform logging cannot be deactivated.
CM-7 CM-7a CM-7b	Least Functionality	Shared	The customer is responsible for configuring Tanzu Application Platform to provide only essential capabilities. Tanzu Application Platform is responsible for providing customers with the capability to deactivate non-essential features not required by the customer. The customer must restrict the use of functions, ports, protocols, and services for the Tanzu Application Platform installation. Tanzu Application Platform is responsible for ensuring that functions, ports, protocols, and services are limited to those explicitly required for the application to operate.

Name	Title	Responsible Party	Notes
CM-7(2)	Prevent Program Execution	Tanzu Application Platform	As an extension of CM-7, Least Functionality, this control is a responsibility of Tanzu Application Platform. Tanzu Application Platform only consists of containers with purposeful services with no extra programs running or bloat. This cannot be configured by the customer.
CM-7(4) (b)	Unauthorized Software/Denylisting	Tanzu Application Platform	Tanzu Application Platform service containers do not implement a deny-by-exception policy to prohibit the execution of unauthorized software programs. Tanzu Application Platform service containers are built to provide stripped-down services and do not include extra programs or bloat. Tanzu Application Platform can provide a SBOM to compare against customer organization policies on disallowed software.
IA-2	Identification and Authentication (Organizational Users)	Shared	The customer is responsible for configuring Tanzu Application Platform to use their IdP which is capable of uniquely identifying and authenticating organizational users. Tanzu Application Platform is responsible for providing customers with the capability to integrate their IdP to allow Tanzu Application Platform to uniquely identify organizational users.
IA-2(1)	Network Access to Privileged Accounts	Customer	Implemented on customer identity provider. The customer is responsible for implementing multifactor authentication on their IdP for network access to privileged accounts.
IA-2(2)	Network Access to Non-Privileged Accounts	Customer	Implemented on customer identity provider. The customer is responsible for implementing multifactor authentication on their IdP for network access to non-privileged accounts.
IA-2(3)	Local Access to Privileged Accounts	N/A	Tanzu Application Platform does not use local accounts. All access occurs over a network connection.
IA-2(8)	Network Access to Privileged Accounts - Replay Resistant	Tanzu Application Platform	Tanzu Application Platform is responsible for ensuring that all connections to the customer IdP are over TLS 1.2+.
IA-2(11)	Remote Access - Separate Device	Customer	The customer is responsible for all aspects of MFA and MFA devices used to authenticate to their Tanzu Application Platform installation, including using remote access.
IA-2(12)	Acceptance of Piv Credentials	Customer	Implemented on customer identity provider. The customer is responsible for implementing CAC/PIV credentials with their IdP.
IA-3	Device Identification and Authentication	Customer	The customer is responsible for uniquely identifying and authenticating organization-defined specific and/or types of devices before establishing a local, remote, or network connection.
IA-4e	Identifier Management	Customer	Implemented on customer identity provider. The customer is responsible for configuring IdP token TTL and refresh policies that apply to Tanzu Application Platform sessions. Tanzu Application Platform enforces token policies and cannot be configured otherwise.

Name	Title	Responsible Party	Notes
IA-5(1) IA-5(1)(a) IA-5(1)(b) IA-5(1)(c) IA-5(1)(d) IA-5(1)(e) IA-5(1)(f)	Password-Based Authentication	Customer	Implemented on customer identity store. The customer is responsible for all aspects of password-based authentication to their IdP, using their identity store. Tanzu Application Platform does not employ password-based authentication itself.
IA-5(2) IA-5(2)(a) IA-5(2)(b) IA-5(2)(c) IA-5(2)(d)	PKI-Based Authentication	Customer	Implemented on customer identity provider. The customer is responsible for all aspects of PKI-based authentication on the IdP used to access their Tanzu Application Platform installation.
IA-5(11)	Hardware Token-Based Authentication	Customer	The customer is responsible for ensuring hardware token-based authentication employs mechanisms that satisfy organization-defined token quality requirements.
IA-6	Authenticator Feedback	Customer	Implemented on customer identity provider. The customer is responsible for ensuring their IdP obscures feedback of authentication information during the authentication process.
IA-7	Cryptographic Module Authentication	Customer	Implemented on customer identity provider. The customer is responsible for ensuring their IdP implements FIPS 140-2 validated cryptographic modules.
IA-8	Identification and Authentication(Non-Organizational Users)	Customer	Implemented on customer identity provider. The customer is responsible for ensuring that their IdP uniquely identifies and authenticates non-organizational Tanzu Application Platform users, or processes acting on behalf of non-organizational users.
IA-8(1)	Acceptance of Piv Credentials from Other Agencies	Customer	Implemented on customer identity provider. The customer is responsible for configuring their IdP to accept and electronically verify Personal Identity Verification(PIV) credentials from other federal agencies.
IA-8(2)	Acceptance of Third-Party Credentials	Customer	Implemented on customer identity provider. The customer is responsible for configuring their IdP to accept only FICAM-approved third-party credentials.
IA-8(3)	Use of FICAM-Approved Products	Customer	Implemented on customer identity provider. The customer is responsible for employing only FICAM-approved information system components on their IdP to accept third-party credentials.
IA-8(4)	Use of FICAM-Issued Profiles	Customer	Implemented on customer identity provider. The customer is responsible for ensuring their IdP conforms to FICAM-issued profiles.
SC-2	Application Partitioning	Tanzu Application Platform	Tanzu Application Platform does not isolate user and management functionality on separate network interfaces, instances, CPUs, or similar. Tanzu Application Platform relies on different roles and Kubernetes RBAC to keep user and management functionality distinct.
SC-4	Information in Shared Resources	Tanzu Application Platform	Tanzu Application Platform creates dedicated Kubernetes namespaces upon deployment. Kubernetes namespaces prevent unauthorized and unintended information transfer using shared system resources.

Name	Title	Responsible Party	Notes
SC-5	Denial of Service Protection	Customer	The customer is responsible for ensuring that organizational DoS protections at the network layer include the Tanzu Application Platform installation.
SC-7 SC-7a SC-7b SC-7c	Boundary Protection	Customer	The customer is responsible for the configuration and management of boundary protection devices.
SC-7(4)(c)	External Telecommunications Services	Customer	The customer is responsible for external telecommunication services used to establish connections to their Tanzu Application Platform installation.
SC-7(5)	Deny by Default / Allow by Exception	Shared	Tanzu Application Platform does not implement “deny by default” network policies. This might be mitigated by network-level access controls configured by the customer.
SC-7(7)	Prevent Split Tunneling for Remote Devices	Customer	The customer is responsible for all configuration of remote devices used to access Tanzu Application Platform.
SC-8	Transmission Confidentiality and Integrity	Tanzu Application Platform	Tanzu Application Platform is responsible for ensuring all communications occur over TLS 1.2+.
SC-8(1)	Cryptographic or Alternate Physical Protection	Tanzu Application Platform	Tanzu Application Platform is responsible for ensuring all communications occur over TLS 1.2+.
SC-10	Network Disconnect	Tanzu Application Platform	Tanzu Application Platform tears down TCP connections and deallocates system resources following the expiration of a session token and cannot be configured otherwise.
SC-12	Cryptographic Key Establishment and Management	Tanzu Application Platform	Tanzu Application Platform is responsible for providing customers with the ability to manage trust stores.
SC-13	Cryptographic Protection	Tanzu Application Platform	Tanzu Application Platform is responsible for implementing FIPS 140 validated cryptographic modules and providing the customer with a means to enable “FIPS Mode”.
SC-21	Secure Name / Address Resolution Service (Recursive or Caching Resolver)	Customer	Tanzu Application Platform inherits the DNSSEC capabilities of the organization resolvers it is configured to use. The customer is responsible for configuring the Tanzu Application Platform and Kubernetes infrastructure to use DNSSEC-capable resolvers.
SC-23	Session Authenticity	Tanzu Application Platform	Tanzu Application Platform is responsible for ensuring all communications occur over TLS 1.2+.

Name	Title	Responsible Party	Notes
SC-28	Protection of Information at Rest	Customer	Tanzu Application Platform does not natively provide encryption for data at rest, but instead relies on the underlying Kubernetes persistent volumes for appropriate cryptographic protections. The customer is responsible for deploying Tanzu Application Platform to Kubernetes with persistent volumes for appropriate cryptographic protections.
SC-39	Process Isolation	Tanzu Application Platform	Tanzu Application Platform container OS enforces the use of separate execution domains for each executing process and cannot be configured otherwise. The underlying Kubernetes host isolates each container from the other.
SI-2c	Flaw Remediation	Tanzu Application Platform	The customer is responsible for keeping the Tanzu Application Platform installation up to date, to within org-defined standards. Tanzu Application Platform does not automatically update itself.
SI-3(2)	Automatic Updates	N/A	Tanzu Application Platform does not include malicious code protection mechanisms therefore automatic update to such mechanisms does not apply.
SI-7(1)	Integrity Checks	Tanzu Application Platform	Tanzu Application Platform performs a hash check when images are downloaded, and a cryptographic signature validation at runtime. This cannot be configured otherwise.
SI-10	Information Input Validation	Tanzu Application Platform	Tanzu Application Platform is responsible for performing input validation of user-supplied input to Tanzu Application Platform.
SI-11 SI-11a SI-11b	Error Handling	Tanzu Application Platform	Tanzu Application Platform limits error message verbosity but does display errors to users. Given the development/coding nature of Tanzu Application Platform, deployment errors and similar must be raised to the user so they can be corrected.
SI-16	Memory Protection	Tanzu Application Platform	Tanzu Application Platform container OS protects its memory from unauthorized code execution and cannot be configured otherwise. The underlying Kubernetes host also isolates container memory pages.

Harden Tanzu Application Platform

This topic provides you with VMware recommendations on how to install and configure Tanzu Application Platform (commonly known as TAP) so that it to complies with NIST Publication 800-53.

Configuring your Tanzu Application Platform installation to this standard does not ensure approval as there are multiple organizational requirements and deviations that a platform team can make during installation and configuration.

Tanzu Application Platform is deployed on Kubernetes and relies on the Kubernetes platform being hardened in a shared responsibility model. For information about hardening Kubernetes, see:

- NSA/CISA Cybersecurity Technical Report
- NIST Kubernetes STIG Checklist
- CIS Kubernetes Benchmark

Identity and access management

To provide an audit trail of what a user does in a system, it is important to configure Tanzu Application Platform so that the identity of a user is known. When installing and configuring Tanzu Application Platform, there are several areas where user identity configuration must be considered. Tanzu Application Platform has three different areas where users have identities.

1. Tanzu Developer Portal
2. Tanzu Developer Portal authentication to remote clusters
3. The Kubernetes cluster where Tanzu Application Platform components are installed

VMware recommends using the same identity provider for each of these so that a common identity is shared across Tanzu Application Platform. To facilitate this, components can use common OIDC providers.

Tanzu Developer Portal

Tanzu Developer Portal is based on the Backstage open source project and has a variety of OIDC providers that you can configure as an identity provider.

To configure authentication for the Tanzu Developer Portal, VMware recommends the following:

1. Enable user authentication using one of the supported providers. For more information, see Set up authentication for Tanzu Developer Portal.

   Note

   Due to the limitations of the Backstage authentication implementation, enabling authentication does not ensure full end-to-end security as Backstage doesn't currently support per-API authentication. VMware recommends implementing additional security either using an inbound proxy or by leveraging networking using a firewall or VPN. For more information, see Authentication in Backstage.

2. Disable guest access in the `tap_gui` section in the `tap-values.yaml` file:

   ```
   tap_gui:
    app_config:
      auth:
        allowGuestAccess: false
   ```

Tanzu Developer Portal authentication to remote clusters

Several plug-ins within the Tanzu Developer Portal, such as the Runtime Resource Viewer, Supply Chain Visualization, and Security Analysis GUI require authentication to remote Kubernetes clusters to query Kubernetes resources.

To do so, the plug-ins must authenticate to the Kubernetes API on remote clusters. Configure authentication in two ways: a shared Kubernetes service account that all users use to authenticate to remote clusters, and by setting up an authentication provider for the remote cluster. As best security practice, VMware recommends setting up a remote authentication provider for the Kubernetes cluster.

For more information, see Update Tanzu Developer Portal to view resources on multiple clusters.

As best practice, the users on the Kubernetes clusters that are used for remote authentication must be assigned to Kubernetes roles that limit access in a least privilege model.

The Kubernetes cluster

VMware recommends enabling authentication to the Kubernetes clusters where the Tanzu Application Platform components are installed, using the same identity provider that other components are using.

While there are many options on how to enable OIDC providers for authentication with the Kubernetes API, VMware supports the Pinniped project. For information on using Pinniped in Tanzu Application Platform, see Set up authentication for your Tanzu Application Platform deployment and Install Pinniped on Tanzu Application Platform.

By configuring this to use the same identify provider as the Tanzu Developer Portal, users can have a common identity across the Kubernetes clusters and the Tanzu Developer Portal. Because the Tanzu CLI is making Kubernetes API calls, this configuration is also enabled for the Tanzu CLI.

Using Pinniped provides authentication for Kubernetes clusters but still requires the users to be bound to Kubernetes roles. To provide a starting point, the Tanzu Application Platform provides six Kubernetes Roles as part of the installation that users can be bound to. For more information about the roles used for authorization, see Default roles for Tanzu Application Platform.

Artifact Metadata Repository (AMR) Observer, CloudEvent Handler, and GraphQL

AMR Observer deploys on Full, Build and Run Tanzu Application Platform profiles.

AMR CloudEvent Handler and GraphQL deploy on Full and View Tanzu Application Platform profiles.

AMR Observer, AMR CloudEvent Handler, and GraphQL create Kubernetes service accounts, cluster roles, cluster role bindings, and secrets required for communication between these components internally and externally between Kubernetes clusters. For more information, see Kubernetes service account automatic configuration.

Deactivate the automatic configuration for each of these components for the respective profiles and manually create them as known resources.

To deactivate AMR Observer automatic configuration, update the `tap-values.yaml` as follows:

```
amr:
  observer:
    auth:
      kubernetes_service_accounts:
        autoconfigure: false
```

To deactivate AMR CloudEvent Handler and GraphQL automatic configuration, update the `tap-values.yaml` as follows:

```
amr:
  cloudevent_handler:
    auth:
      kubernetes_service_accounts:
        autoconfigure: false

  graphql:
    auth:
      kubernetes_service_accounts:
        autoconfigure: false
```

For best practice, users on the Kubernetes clusters must create resources with Kubernetes roles that limit access in a least privilege model. For more information, see User-defined Kubernetes Service Account Configuration.

Encryption

Encryption of data prevents unauthorized access to data. In Tanzu Application Platform, there are two states where data should be encrypted:

1. Encryption of data in transit

2. Encryption of data at rest

Encryption of data in transit

This section describes how to encryption of internal communication between services that originate within the cluster and data at rest, and how to secure exposed ingress endpoints.

Internal Communication of data in transit configuration

Communication between services that originate and terminate within the cluster is referred to as internal communication. Tanzu Application Platform is in the process of enabling TLS on internal communication for components.

If you require encrypted internal communication, there are three remediating options:

1. Enable Tanzu Service Mesh, which provides mutual TLS between components. For more information, see Set up Tanzu Service Mesh.
2. Configure Kubernetes to encrypt all communication with a Container Networking Interface (CNI) that supports traffic encryption, for example, Antrea.
3. Use the underlying network infrastructure running Kubernetes which has encryption on all network traffic.

External communication of data in transit configuration

TLS enables encryption of communication from end-users to the cluster. Because Contour is the edge gateway for all the traffic ingressing into the cluster, it is suitable to set up TLS and ensure that all communications between users and the cluster are encrypted.

It also allows cluster owners to satisfy compliance requirements such as NIST 800-53 ControlSC-8 to protect the confidentiality of transmitted information.

It might be required that certain cipher suites or TLS versions are used when encrypting communications. NIST 800-53 requires that all government-only applications use TLS v1.2 and they also must be configured to use TLS v1.3.

Configuring TLS for Contour

To configure Contour to use TLS, create a new section in `tap-values.yaml`:

```
...
contour:
  * existing stuff, probably already there if you're following tap docs
  envoy:
    service:
      type: LoadBalancer # This is set by default, but can be overridden by setting a
different value.
  * new stuff
  contour:
    configFileContents:
      tls:
        minimum-protocol-version: "1.2"
        cipher-suites:
        - '[ECDHE-ECDSA-AES128-GCM-SHA256|ECDHE-ECDSA-CHACHA20-POLY1305]'
        - '[ECDHE-RSA-AES128-GCM-SHA256|ECDHE-RSA-CHACHA20-POLY1305]'
        - 'ECDHE-ECDSA-AES256-GCM-SHA384'
        - 'ECDHE-RSA-AES256-GCM-SHA384'
```

After adding this section, apply the `tap-values.yaml` file that will change the configuration of TLS to match the requirements.

For more information, see TLS Configuration in the Contour documentation.

Ingress Certificates

For information about how to configure TLS for a Tanzu Application Platform installation's ingress endpoints, see Secure exposed ingress endpoints in Tanzu Application Platform.

Encryption of Data At Rest

All data must be encrypted at rest. The Tanzu Application Platform runs on Kubernetes and verifies the default storage class configured on the Kubernetes cluster. If you require Encryption of Data at Rest (DARE), you must provide a PersistentVolume Provisioner that supports encryption to the Kubernetes infrastructure.

Ports and protocols

Ports are used in TCP and UDP protocols for identification of applications. While some applications use common port numbers, such as 80 for HTTP, or 443 for HTTPS, some applications use dynamic ports. An open port refers to a port on which a system is accepting communication. An open port does not always mean that there is a security issue, but it can provide a pathway for attackers listening on that port. To help understand the traffic flows in Tanzu Application Platform, VMware provides a list of Tanzu Application Platform ports and protocols on request. For more information, see the TAP Architecture Overview.

Networking

Ensure that workloads only expose internal-only routes.

All traffic must go through Contour and LoadBalancer without using NodePort services.

Tanzu Application Platform is supported by Tanzu Service Mesh.

You must configure proper affinity rules on Knative deployed services. For more information, see Install Tanzu Application Platform in an air-gapped environment.

Key management

Key management is the foundation of all data security. Data is encrypted and decrypted with encryption keys or secrets that must be safely stored to prevent the loss or compromise of infrastructure, systems, and applications. Tanzu Application Platform values are secrets and must be protected to ensure that the security and integrity of the platform is maintained.

Tanzu Application Platform stores all sensitive values as Kubernetes Secrets.

Encryption of secrets at rest are Kubernetes distribution dependent.

Use External Secrets Operator (beta) to automate the lifecycle management of Secrets stored in a Secret management service, such as, Hashicorp Vault, Google Secrets Manager, Amazon Secrets Manager, or Microsoft Azure Key Vault use.

For more information related to safeguarding sensitive information from exploitation, such as, Tanzu Application Platform values, see the AC-23 section in the SP 800-53 publication.

Logging

Log files provide an audit trail to monitor activity within infrastructure. Use log files to identify policy violations, unusual activity, and security incidents. It is important that logs are captured and retained according to the policies set forth by your organization's security team or governing body.

Tanzu Application Platform components run as pods on the Kubernetes infrastructure and all components output is captured as part of the pod logs.

All Tanzu Application Platform components follow Kubernetes logging best practices. Log aggregation must be implemented following the best practices of the organization log retention process. For more information, see the AU-4 Audit Log Storage Capacity section in the SP 800-53 publication.

Deployment architecture

Tanzu Application Platform provides a reference architecture that depicts separate components based on function. VMware recommends multiple Kubernetes clusters for the iterate, build, view, and run functions. This separation enables Kubernetes administrators to manage each function independently and therefore, protect the availability and performance of the platform during high usage periods, for example, building or scanning.

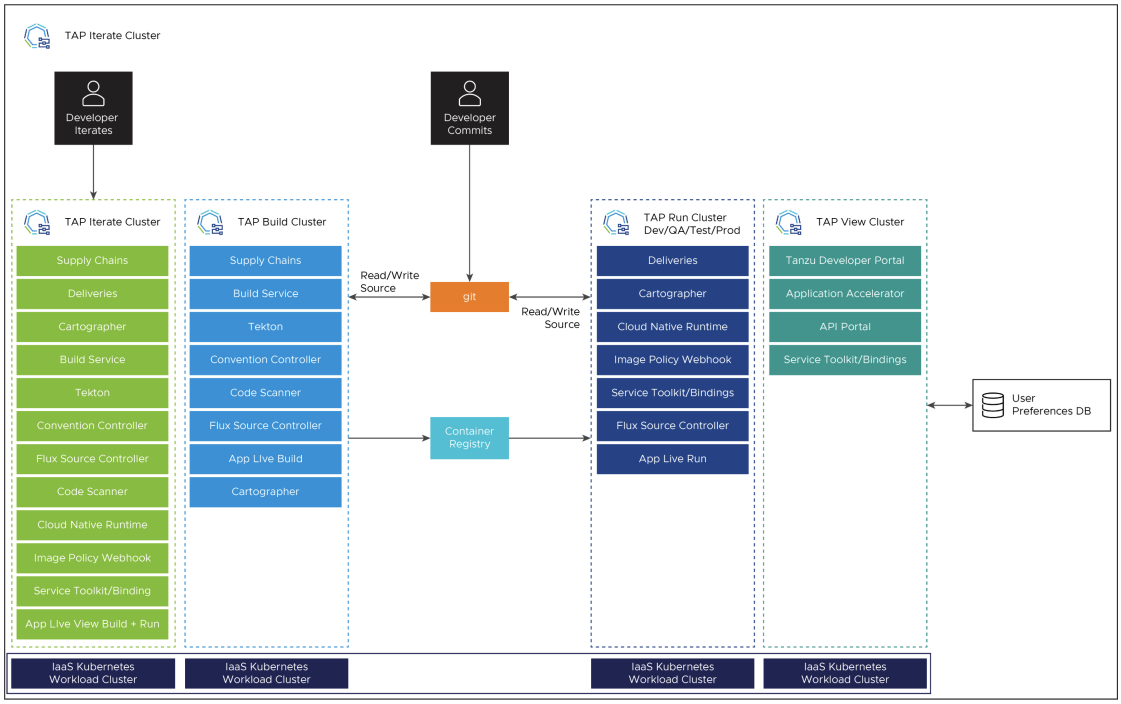
Overview of multicluster Tanzu Application Platform

You can install Tanzu Application Platform (commonly known as TAP) in various topologies to reflect your existing landscape. VMware has tested and recommends a multicluster topology for production use. Because flexibility and choice are core to Tanzu Application Platform's design, none of the implementation recommendations are set in stone.

The multicluster topology uses Tanzu Application Platform profiles. Each cluster adopts one of following multicluster-aligned profiles:

- **Iterate:** Intended for inner-loop iterative application development.
- **Build:** Transforms source revisions to workload revisions; specifically, hosting workloads and supply chains.
- **Run:** Transforms workload revisions to running pods; specifically, hosting deliveries and deliverables.
- **View:** For applications related to centralized developer experiences; specifically, Tanzu Developer Portal and metadata store.

The following diagram illustrates this topology.



Next steps

To get started with installing a multicluster topology, see Install multicluster Tanzu Application Platform profiles.

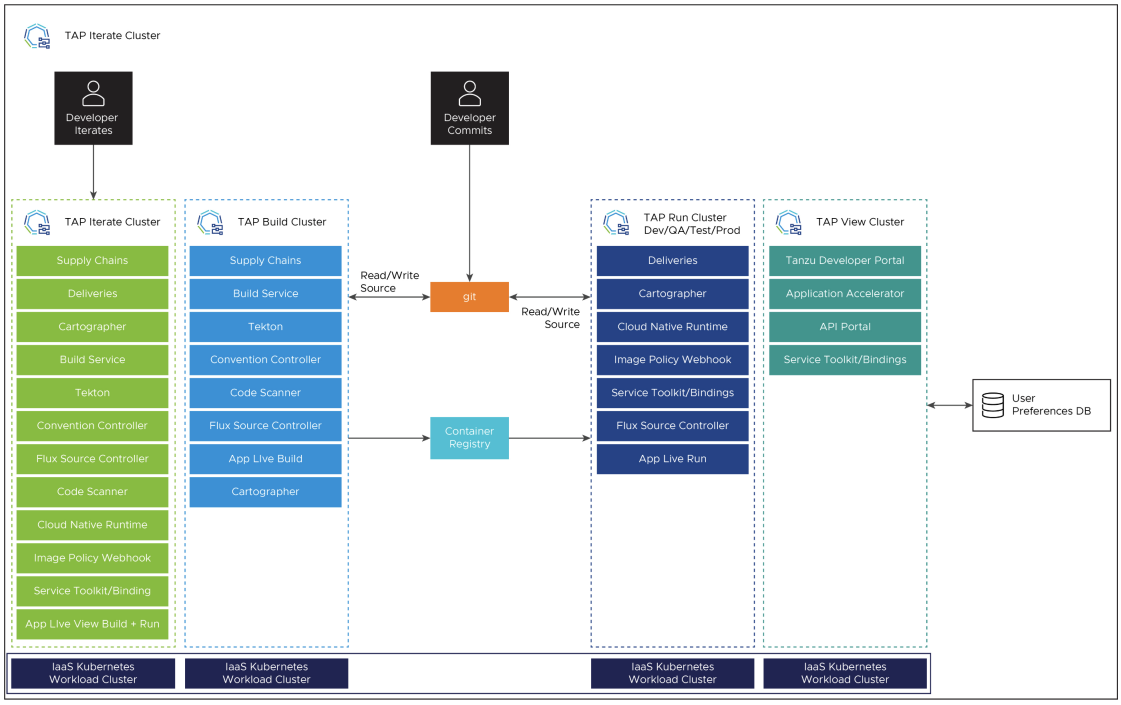
Overview of multicluster Tanzu Application Platform

You can install Tanzu Application Platform (commonly known as TAP) in various topologies to reflect your existing landscape. VMware has tested and recommends a multicluster topology for production use. Because flexibility and choice are core to Tanzu Application Platform's design, none of the implementation recommendations are set in stone.

The multicluster topology uses Tanzu Application Platform profiles. Each cluster adopts one of following multicluster-aligned profiles:

- **Iterate:** Intended for inner-loop iterative application development.
- **Build:** Transforms source revisions to workload revisions; specifically, hosting workloads and supply chains.
- **Run:** Transforms workload revisions to running pods; specifically, hosting deliveries and deliverables.
- **View:** For applications related to centralized developer experiences; specifically, Tanzu Developer Portal and metadata store.

The following diagram illustrates this topology.



Next steps

To get started with installing a multicluster topology, see Install multicluster Tanzu Application Platform profiles.

Install multicluster Tanzu Application Platform profiles

This topic tells you how to install a multicluster topology for your Tanzu Application Platform (commonly known as TAP).

Prerequisites

Before installing multicluster Tanzu Application Platform profiles, you must meet the following prerequisites:

- All clusters must satisfy all the requirements to install Tanzu Application Platform. See Prerequisites.
- Accept Tanzu Application Platform EULA and install Tanzu CLI with any required plug-ins.
- Install Tanzu Cluster Essentials on all clusters. For more information, see Deploy Cluster Essentials.

Multicluster Installation Order of Operations

The installation order is flexible given the ability to update the installation with a modified values file using the `tanzu package installed update` command. The following is an example of the order of operations to be used:

1. Install View profile cluster.
2. Install Build profile cluster.
3. Install Run profile cluster.
4. Install Iterate profile cluster.
5. Add Build, Run and Iterate clusters to Tanzu Developer Portal.
6. Update the View cluster's installation values file with the previous information and run the following command to pass the updated config values to Tanzu Developer Portal:

   ```
   tanzu package installed update tap -p tap.tanzu.vmware.com -v TAP-VERSION --val
   ues-file tap-values.yaml -n tap-install
   ```

   Where `TAP-VERSION` is the Tanzu Application Platform version you've installed.

Install View cluster

Install the View profile cluster first, because some components must exist before installing the Run clusters. For example, the Application Live View back end must be present before installing the Run clusters. For more information about profiles, see About Tanzu Application Platform package profiles.

To install the View cluster:

1. Follow the steps described in Installing the Tanzu Application Platform package and profiles by using a reduced values file as shown in View profile.
2. Verify that you can access Tanzu Developer Portal by using the ingress that you set up. The address must follow this format: `https://tap-gui.INGRESS-DOMAIN`, where `INGRESS-DOMAIN` is the DNS domain you set in `shared.ingress_domain` which points to the shared Contour installation in the `tanzu-system-ingress` namespace with the service `envoy`.
3. Follow the steps to configure Supply Chain Security Tools - Store described in Set up multicluster Supply Chain Security Tools (SCST) - Store.

Install Build clusters

To install the Build profile cluster, follow the steps described in Installing the Tanzu Application Platform package and profiles by using a reduced values file as shown in Build profile.

Install Run clusters

To install the Run profile cluster:

1. Follow the steps described in Install the Tanzu Application Platform package and profiles by using a reduced values file as shown in Run profile.
2. To use Application Live View, set the `INGRESS-DOMAIN` for `appliveview_connector` to match the value you set on the View profile for the `appliveview` in the values file.

> **Note**
>
> The default configuration of `shared.ingress_domain` points to the local Run cluster, rather than the View cluster, as a result, `shared.ingress_domain` must be set explicitly.

Install Iterate clusters

To install the Iterate profile cluster, follow the steps described in Install the Tanzu Application Platform package and profiles by using a reduced values file as shown in Iterate profile.

Add Build, Run and Iterate clusters to Tanzu Developer Portal

After installing the Build, Run and Iterate clusters, follow the steps in View resources on multiple clusters in Tanzu Developer Portal to:

1. Create the `Service Accounts` that Tanzu Developer Portal uses to read objects from the clusters.
2. Add a remote cluster.

These steps create the necessary RBAC elements allowing you to pull the URL and token from the Build, Run and Iterate clusters that allows them come back and add to the View cluster's values file.

You must add the Build, Run and Iterate clusters to the View cluster for all plug-ins to function as expected.

Next steps

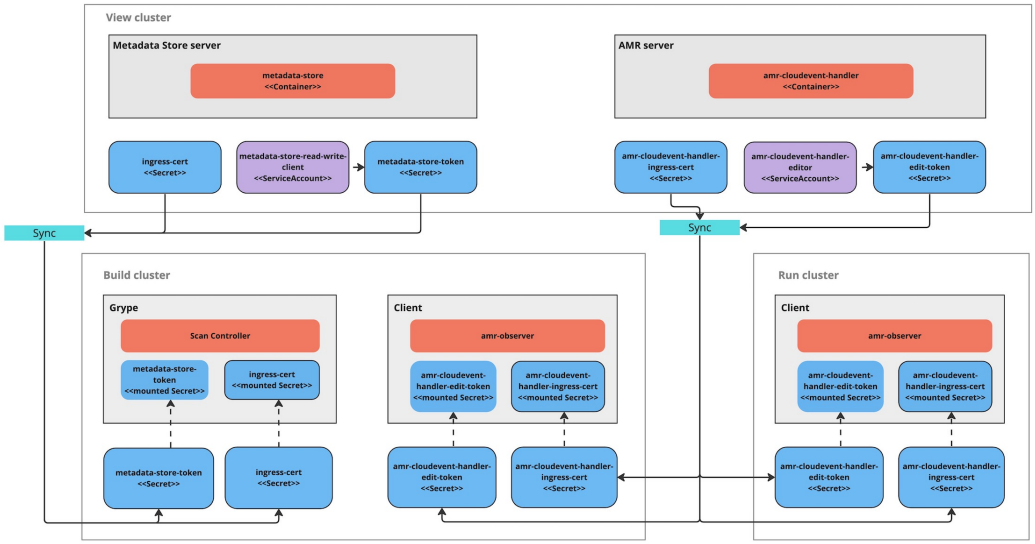
After setting up the four profiles, you're ready to run a workload by using the supply chain. See Get started with multicluster Tanzu Application Platform.

Multicluster setup for Supply Chain Security Tools - Store

This topic describes how you can deploy Supply Chain Security Tools (SCST) - Store in a multicluster setup, including installing multiple profiles such as, View, Build, Run, and Iterate.

Overview

After installing the View profile, but before installing the Build profile and Run profile, you must copy certain configurations from the View cluster to the Build and Run Kubernetes clusters. This topic explains how to add these configurations which allows components in the Build and Run clusters to communicate with SCST - Store in the View cluster.



Prerequisites

You must first install the View profile. See Install View profile. This installation automatically creates the CA certificates and tokens necessary to talk to the CloudEvent Handler.

Procedure summary

To deploy Supply Chain Security Tools (SCST) - Store in a multicluster setup:

1. Copy SCST - Store CA certificates from the View cluster.
    1. Copy the Metadata Store CA certificate from the View cluster.
    2. Copy the AMR CloudEvent Handler CA certificate from the View cluster.
2. Copy SCST - Store tokens from the View cluster.
    1. Copy the Metadata Store authentication token from the View cluster.
    2. Copy the AMR CloudEvent Handler edit token from the View cluster.
3. Apply the SCST - Store CA certificates and SCST - Store tokens to the Build and Run clusters.
    1. Apply the Metadata Store CA certificate and authentication token to the Build cluster.
    2. Apply the AMR CloudEvent Handler CA certificate and edit token for the Build and Run cluster.
4. Install the Build and Run profiles.

Copy SCST - Store CA certificates from the View cluster

To copy SCST - Store CA certificates from the View cluster, you must copy the Metadata Store CA certificate and the AMR CloudEvent Handler CA certificate from the View cluster.

Copy Metadata Store CA certificate from the View cluster

With your kubectl targeted at the View cluster, you can get Metadata Store's TLS CA certificate.

```
MDS_CA_CERT=$(kubectl get secret -n metadata-store ingress-cert -o json | jq -r ".dat
```

```
a.\"ca.crt\"" | base64 -d)
```

Copy AMR CloudEvent Handler CA certificate data from the View cluster

With your kubectl targeted at the View cluster, you can get AMR CloudEvent Handler's TLS CA certificate's data.

```
CEH_CA_CERT_DATA=$(kubectl get secret -n metadata-store amr-cloudevent-handler-ingress
-cert -o json | jq -r ".data.\"ca.crt\"" | base64 -d)
```

Copy SCST - Store authentication tokens from the View cluster

To copy SCST - Store tokens from the View cluster, you must copy the Metadata Store authentication token and the AMR CloudEvent Handler edit token from the View cluster.

Copy Metadata Store authentication token from the View cluster

Copy the Metadata Store authentication token into an environment variable:

```
MDS_AUTH_TOKEN=$(kubectl get secrets metadata-store-read-write-client -n metadata-stor
e -o jsonpath="{.data.token}" | base64 -d)
```

You use this environment variable in the next step.

Copy AMR CloudEvent Handler edit token from the View cluster

Copy the AMR CloudEvent Handler token into an environment variable:

```
CEH_EDIT_TOKEN=$(kubectl get secrets amr-cloudevent-handler-edit-token -n metadata-sto
re -o jsonpath="{.data.token}" | base64 -d)
```

You use this environment variable in the next step.

Apply the SCST - Store CA certificates and SCST - Store tokens to the Build and Run clusters

After you copy the certificate and tokens, apply them to the Build and Run clusters before deploying the profiles.

Build cluster:

- Metadata Store CA certificate
- Metadata Store authentication token
- CloudEvent Handler CA certificate
- CloudEvent Handler edit token

Run cluster:

- CloudEvent Handler CA certificate
- CloudEvent Handler edit token

Configure SCST - Scan with the Metadata Store CA certificate and authentication token on the Build cluster

Update the Build profile `values.yaml` file to add the following snippet using the contents of $MDS_CA_CERT and $MDS_AUTH_TOKEN copied in an earlier step. It configures SCST - Scan with the Metadata Store CA certificate and authentication token.

```
```console
scanning:
  metadataStore:
    exports:
      ca:
        pem: |
          <CONTENTS OF $MDS_CA_CERT>
      auth:
        token: <CONTENTS OF $MDS_AUTH_TOKEN>
```
```

Apply the CloudEvent Handler CA certificate data and edit token to the Build and Run clusters

You can apply the CloudEvent Handler CA certificate and edit the token to the Build and Run clusters. These values must be accessible during the Build and Run profile deployments.

1. Update your kubectl to target the Build cluster.
2. If you already installed Build Cluster you can skip this step. Create a namespace for the CloudEvent Handler CA certificate and edit token.

   ```
   kubectl create ns amr-observer-system
   ```

3. Update the Build profile `values.yaml` file to add the following snippet. It configures the CA certificate and endpoint. In `amr.observer.cloudevent_handler.endpoint` you specify the location of the CloudEvent Handler which was deployed to the View cluster. In `amr.observer.ca_cert_data` you paste the contents of `$CEH_CA_CERT_DATA` which you copied earlier.

   ```
   amr:
     observer:
       auth:
         kubernetes_service_accounts:
           enable: true
       cloudevent_handler:
         endpoint: https://amr-cloudevent-handler.<VIEW-CLUSTER-INGRESS-DOMAIN>
       ca_cert_data: |
           <CONTENTS OF $CEH_CA_CERT_DATA>
   ```

4. Create a secret to store the CloudEvent Handler edit token. This uses the `CEH_EDIT_TOKEN` environment variable.

   ```
   kubectl create secret generic amr-observer-edit-token \
     --from-literal=token=$CEH_EDIT_TOKEN -n amr-observer-system
   ```

5. Repeat the earlier steps, but configure kubectl to target the Run cluster instead of the Build cluster.

After all the steps are done, both the Build and Run clusters each have a CloudEvent Handler CA certificate and edit token named `amr-observer-edit-token` in the namespaces `metadata-store-secrets` and `amr-observer-system`. Now you are ready to deploy the Build and Run profiles.

Install the Build and Run profiles

If you came to this topic from Install multicluster Tanzu Application Platform profiles after installing the View profile, return to that topic to install the Build profile and install the Run profile.

How to configure Grype in the Build profile values file

The Build profile `values.yaml` uses the secrets you created to configure the Grype scanner which talks to SCST - Store. After performing a vulnerabilities scan, the Grype scanner sends the results to SCST - Store.

For example:

```
...
grype:
  targetImagePullSecret: "TARGET-REGISTRY-CREDENTIALS-SECRET"
  metadataStore:
    url: METADATA-STORE-URL-ON-VIEW-CLUSTER # Url with http / https
    caSecret:
        name: store-ca-cert
        importFromNamespace: metadata-store-secrets # Must match with `ingress-cert.da
ta."ca.crt"` of store on view cluster
    authSecret:
        name: store-auth-token # Must match with valid store token of metadata-store o
n view cluster
        importFromNamespace: metadata-store-secrets
...
```

Where:

- `METADATA-STORE-URL-ON-VIEW-CLUSTER` is the ingress URL of SCST - Store deployed to the View cluster. For example, `https://metadata-store.example.com`. See Ingress support.
- `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the secret that contains the credentials to pull an image from the registry for scanning.

Configure developer namespaces

After you finish installing Tanzu Application Platform, configure developer namespaces. To prepare developer namespaces, you must export the Metadata Store secrets you created earlier to those namespaces.

Exporting SCST - Store secrets to a developer namespace in a Tanzu Application Platform multicluster deployment

Export secrets to a developer namespace by creating `SecretExport` resources on the developer namespace. You must have created and populated the `metadata-store-secrets` namespace. To create the `SecretExport` resources, run:

```
cat <<EOF | kubectl apply -f -
---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretExport
metadata:
  name: store-ca-cert
  namespace: metadata-store-secrets
spec:
  toNamespaces: [DEV-NAMESPACES]
---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretExport
metadata:
  name: store-auth-token
```

```
  namespace: metadata-store-secrets
spec:
  toNamespaces: [DEV-NAMESPACES]
EOF
```

Where `DEV-NAMESPACES` is an array of developer namespaces where the Metadata Store secrets are exported.

For information about `metadata` configuration, see Cluster Specific Store Configuration.

Additional resources

- Ingress support
- Custom certificate configuration

Get started with multicluster Tanzu Application Platform

This topic tells you how to validate the implementation of a multicluster topology by taking a sample workload and passing it by using the supply chains on the Build and Run clusters.

Use this topic to build an application on the Build profile clusters and run the application on the Run profile clusters.

You can view the workload and associated objects from Tanzu Developer Portal interface on the View profile cluster.

You can take various approaches to configuring the supply chain in this topology, but the following procedures validate the most basic capabilities.

Prerequisites

Before implementing a multicluster topology, complete the following:

1. Complete all installation steps for the four profiles: Build, Run, View and Iterate.
2. For the sample workload, VMware uses the same Application Accelerator - Tanzu Java Web App in the non-multicluster Get Started guide. You can download this accelerator to your own Git infrastructure of choice. You might need to configure additional permissions. Alternatively, you can also use the application-accelerator-samples GitHub repository.
3. The two supply chains are `ootb-supply-chain-basic` on the Build/Iterate profile and `ootb-delivery-basic` on the Run profile. For the Build/Iterate and Run profiled clusters, perform the steps described in Setup Developer Namespace. This guide assumes that you use the `default` namespace.
4. To set the value of `DEVELOPER_NAMESPACE` to the namespace you setup in the previous step, run:

   ```
   export DEVELOPER_NAMESPACE=YOUR-DEVELOPER-NAMESPACE
   ```

   Where:

   - `YOUR-DEVELOPER-NAMESPACE` is the namespace you set up in Set up developer namespaces to use your installed packages. `default` is used in this example.

Start the workload on the Build profile cluster

The Build cluster starts by building the necessary bundle for the workload that is delivered to the Run cluster.

1. Use the Tanzu CLI to start the workload down the first supply chain:

```
tanzu apps workload create tanzu-java-web-app \
--git-repo https://github.com/vmware-tanzu/application-accelerator-samples \
--sub-path tanzu-java-web-app \
--git-branch main \
--type web \
--label app.kubernetes.io/part-of=tanzu-java-web-app \
--yes \
--namespace ${DEVELOPER_NAMESPACE}
```

2. To monitor the progress of this process, run:

```
tanzu apps workload tail tanzu-java-web-app --since 10m --timestamp --namespace
${DEVELOPER_NAMESPACE}
```

3. To exit the monitoring session, press **CTRL** + **C**.

4. Verify that your supply chain has produced the necessary `ConfigMap` containing `Deliverable` content produced by the `Workload`:

```
kubectl get configmap tanzu-java-web-app-deliverable --namespace ${DEVELOPER_NA
MESPACE} -o go-template='{{.data.deliverable}}'
```

   The output resembles the following:

```
apiVersion: carto.run/v1alpha1
kind: Deliverable
metadata:
  name: tanzu-java-web-app-deliverable
  labels:
    apis.apps.tanzu.vmware.com/register-api: "true"
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/workload-type: web
    app.kubernetes.io/component: deliverable
    app.tanzu.vmware.com/deliverable-type: web
spec:
  params:
  - name: source_credentials_secret
    value: ""
  source:
    git:
      url: http://git-server.default.svc.cluster.local/app-namespace/tanzu-java
-web-app
      ref:
        branch: main
```

5. Store the `Deliverable` content, which you can take to the Run profile clusters from the `ConfigMap` by running:

```
kubectl get configmap tanzu-java-web-app-deliverable -n ${DEVELOPER_NAMESPACE}
-o go-template='{{.data.deliverable}}' > deliverable.yaml
```

6. Take this `Deliverable` file to the Run profile clusters by running:

```
kubectl apply -f deliverable.yaml --namespace ${DEVELOPER_NAMESPACE}
```

7. Verify that this `Deliverable` is started and `Ready` by running:

```
kubectl get deliverables --namespace ${DEVELOPER_NAMESPACE}
```

   The output resembles the following:

```
kubectl get deliverables --namespace default
NAME                SOURCE
DELIVERY         READY   REASON   AGE
tanzu-java-web-app   tapmulticloud.azurecr.io/tap-multi-build-dev/tanzu-java-we
b-app-default-bundle:xxxx-xxxx-xxxx-xxxx-1a7beafd6389   delivery-basic   True
Ready    7m2s
```

8. To test the application, query the URL for the application. Look for the `httpProxy` by running:

   ```
   kubectl get httpproxy --namespace ${DEVELOPER_NAMESPACE}
   ```

   The output resembles the following:

   ```
   kubectl get httpproxy --namespace default
   NAME                                                                FQDN
   TLS SECRET   STATUS   STATUS DESCRIPTION
   tanzu-java-web-app-contour-a98df54e3629c5ae9c82a395501ee1fdtanz   tanzu-java-we
   b-app.default.svc.cluster.local                                  valid    Valid HTTPP
   roxy
   tanzu-java-web-app-contour-e1d997a9ff9e7dfb6c22087e0ce6fd7ftanz   tanzu-java-we
   b-app.default.apps.run.multi.kapplegate.com                      valid    Valid HTTPP
   roxy
   tanzu-java-web-app-contour-tanzu-java-web-app.default             tanzu-java-we
   b-app.default                                                    valid    Valid HTTPP
   roxy
   tanzu-java-web-app-contour-tanzu-java-web-app.default.svc         tanzu-java-we
   b-app.default.svc                                                valid    Valid HTTPP
   roxy
   ```

   Select the URL that corresponds to the domain you specified in your Run cluster's profile and enter it into a browser. Expect to see the message "Greetings from Spring Boot + Tanzu!".

9. View the component in Tanzu Developer Portal, by following these steps and using the catalog file from the sample accelerator in GitHub.

Install Tanzu Application Platform Build profile

This topic tells you how to install Build profile cluster by using a reduced values file.

Prerequisites

Before installing the Build profile, follow all the steps in Install View cluster.

Example values.yaml

The following is the YAML file sample for the build-profile:

```
profile: build
ceip_policy_disclosed: FALSE-OR-TRUE-VALUE # Installation fails if this is not set to
true. Not a string.

shared:
  ingress_domain: "INGRESS-DOMAIN"
  kubernetes_distribution: "openshift" # To be passed only for OpenShift. Defaults to
"".
  kubernetes_version: "K8S-VERSION"
  image_registry:
    project_path: "SERVER-NAME/REPO-NAME" # To be used by Build Service by appending
```

```
"/buildservice" and used by Supply chain by appending "/workloads".
    secret:
      name: "KP-DEFAULT-REPO-SECRET"
      namespace: "KP-DEFAULT-REPO-SECRET-NAMESPACE"
  ca_cert_data: | # To be passed if using custom certificates.
    -----BEGIN CERTIFICATE-----
    MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...
    -----END CERTIFICATE-----

# The above shared keys can be overridden in the below section.

buildservice:
# Takes the value from the shared section by default, but can be overridden by setting
a different value.
  kp_default_repository: "KP-DEFAULT-REPO"
  kp_default_repository_secret:
    name: "KP-DEFAULT-REPO-SECRET"
    namespace: "KP-DEFAULT-REPO-SECRET-NAMESPACE"
supply_chain: testing_scanning
ootb_supply_chain_testing_scanning: # Optional if the corresponding shared keys are pr
ovided.
  source:
    credentials_secret: "GIT-SOURCE-CREDENTIAL-SECRET-NAME" # (Optional) Defaults to
"".
  registry:
    server: "SERVER-NAME"
    repository: "REPO-NAME"
  gitops:
    credentials_secret: "GITOPS-CREDENTIAL-SECRET-NAME" # (Optional) Defaults to "".
tap_telemetry:
  customer_entitlement_account_number: "CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER" # (Option
al) Identify data for creating Tanzu Application Platform usage reports.

app_scanning:
  amr:
    url: https://amr-graphql.VIEW-CLUSTER-INGRESS-DOMAIN # AMR GraphQL location at the
View profile cluster.
    accessToken: "AMR-GRAPHQL-READ-ACCESS-TOKEN"

scanning:
  metadataStore:
    exports:
      ca:
        pem: |
          "METADATA STORE CA" # (Optional) Defaults to "".
      auth:
        token: METADATA STORE AUTHENTICATION TOKEN
amr:
  observer:
    auth:
      kubernetes_service_accounts:
        enable: true
    cloudevent_handler:
      endpoint: https://amr-cloudevent-handler.VIEW-CLUSTER-INGRESS-DOMAIN # AMR Cloud
Event Handler location at the View profile cluster.
    ca_cert_data: |
        "AMR-CLOUDEVENT-HANDLER-CA"
```

Important

Installing Grype by using `tap-values.yaml` as follows is deprecated in v1.6 and will be removed in v1.8:

```
grype:
  targetImagePullSecret: "TARGET-REGISTRY-CREDENTIALS-SECRET"
```

You can install Grype by using Namespace Provisioner instead.

Where:

- `K8S-VERSION` is the Kubernetes version used by your OpenShift cluster. It must be in the form of `1.23.x` or `1.24.x`, where `x` stands for the patch version. Examples:
  - Red Hat OpenShift Container Platform v4.10 uses the Kubernetes version `1.23.3`.
  - Red Hat OpenShift Container Platform v4.11 uses the Kubernetes version `1.24.1`.
- `KP-DEFAULT-REPO` is a writable repository in your registry. The Tanzu Build Service dependencies are written to this location. Examples:
  - Harbor has the form `kp_default_repository: "my-harbor.io/my-project/build-service"`
  - Docker Hub has the form `kp_default_repository: "my-dockerhub-user/build-service"` or `kp_default_repository: "index.docker.io/my-user/build-service"`
  - Google Cloud Registry has the form `kp_default_repository: "gcr.io/my-project/build-service"`
  - For Google Cloud Registry, use the contents of the service account JSON file.
- `KP-DEFAULT-REPO-SECRET` is the secret with user credentials that can write to `KP-DEFAULT-REPO`. You can `docker push` to this location with this credential.
  - For Google Cloud Registry, use `kp_default_repository_username: _json_key`.
  - You must create the secret before the installation. For example, you can use the `registry-credentials` secret created earlier.
- `KP-DEFAULT-REPO-SECRET-NAMESPACE` is the namespace where `KP-DEFAULT-REPO-SECRET` is created.
- `SERVER-NAME` is the host name of the registry server. Examples:
  - Harbor has the form `server: "my-harbor.io"`.
  - Docker Hub has the form `server: "index.docker.io"`.
  - Google Cloud Registry has the form `server: "gcr.io"`.
- `REPO-NAME` is where workload images are stored in the registry. Images are written to `SERVER-NAME/REPO-NAME/workload-name`. Examples:
  - Harbor has the form `repository: "my-project/supply-chain"`.
  - Docker Hub has the form `repository: "my-dockerhub-user"`.
  - Google Cloud Registry has the form `repository: "my-project/supply-chain"`.
- `GIT-SOURCE-CREDENTIAL-SECRET-NAME` is the name of the Kubernetes secret in the developer namespace that supplies the Git credentials for the supply chain to fetch source code from. See Git authentication for more information.
- `GITOPS-CREDENTIAL-SECRET-NAME` is the name of the Kubernetes secret in the developer namespace that supplies the Git credentials for the supply chain to push configuration to. See Git authentication for more information.
- `MY-DEV-NAMESPACE` is the name of the developer namespace. SCST - Scan deploys the `ScanTemplates` there. This allows the scanning feature to run in this namespace.

- `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the Secret that contains the credentials to pull an image from the registry for scanning.
- `CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER` (optional) refers to the Entitlement Account Number (EAN), which is a unique identifier VMware assigns to its customers. Tanzu Application Platform telemetry uses this number to identify data that belongs to a particular customers and prepare usage reports.
- `VIEW-CLUSTER-INGRESS-DOMAIN` is the subdomain you set up on the View profile cluster. This matches the `shared.ingress_domain` on the View profile cluster.
- `AMR-GRAPHQL-READ-ACCESS-TOKEN` is the read access token. For more information about how to obtain the token, see AMR GraphQL querying.
- `AMR-CLOUDEVENT-HANDLER-CA` contains the AMR CloudEvent Handler CA data. For more information about configuring the `amr` portion of the values file, see Set up multicluster Supply Chain Security Tools (SCST) - Store.

When you install Tanzu Application Platform, it is bootstrapped with the `lite` set of dependencies, including buildpacks and stacks, for application builds. For more information about buildpacks, see the VMware Tanzu Buildpacks Documentation. You can find the buildpack and stack artifacts installed with Tanzu Application Platform on Tanzu Network. You can update the dependencies by upgrading Tanzu Application Platform to the latest patch.

See Set up multicluster Supply Chain Security Tools (SCST) - Store for more information about the value settings of `grype.metadataStore`.

You must set the `scanning.metadatastore.url` to an empty string if you're installing Grype Scanner v1.2.0 and later or Snyk Scanner to deactivate the embedded SCST - Store integration.

If you use custom CA certificates, you must provide one or more PEM-encoded CA certificates under the `ca_cert_data` key. If you configured `shared.ca_cert_data`, Tanzu Application Platform component packages inherit that value by default.

Install Tanzu Application Platform Run profile

This topic tells you how to install Run profile cluster by using a reduced values file.

The following is the YAML file sample for the run-profile:

```
profile: run
ceip_policy_disclosed: FALSE-OR-TRUE-VALUE # Installation fails if this is not set to
true. Not a string.

shared:
  ingress_domain: INGRESS-DOMAIN
  kubernetes_distribution: "openshift" # To be passed only for OpenShift. Defaults to
"".
  kubernetes_version: "K8S-VERSION"
  ca_cert_data: | # To be passed if using custom certificates.
    -----BEGIN CERTIFICATE-----
    MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...
    -----END CERTIFICATE-----
supply_chain: basic

contour:
  envoy:
    service:
      type: LoadBalancer # NodePort can be used if your Kubernetes cluster doesn't sup
port LoadBalancing.

appliveview_connector:
  backend:
```

```
    sslDeactivated: TRUE-OR-FALSE-VALUE
    ingressEnabled: true
    host: appliveview.VIEW-CLUSTER-INGRESS-DOMAIN

tap_telemetry:
  customer_entitlement_account_number: "CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER" # (Option
al) Identify data for creating Tanzu Application Platform usage reports.

amr:
  observer:
    auth:
      kubernetes_service_accounts:
        enable: true
    cloudevent_handler:
      endpoint: https://amr-cloudevent-handler.VIEW-CLUSTER-INGRESS-DOMAIN # AMR Cloud
Event Handler location at the View profile cluster.
    ca_cert_data: |
        "AMR-CLOUDEVENT-HANDLER-CA"
```

Where:

- `INGRESS-DOMAIN` is the subdomain for the host name that you point at the `tanzu-shared-ingress` service's external IP address.
- `K8S-VERSION` is the Kubernetes version used by your OpenShift cluster. It must be in the form of `1.23.x` or `1.24.x`, where `x` stands for the patch version. Examples:
  - Red Hat OpenShift Container Platform v4.10 uses the Kubernetes version `1.23.3`.
  - Red Hat OpenShift Container Platform v4.11 uses the Kubernetes version `1.24.1`.
- `VIEW-CLUSTER-INGRESS-DOMAIN` is the subdomain you setup on the View profile cluster. This matches the value key `appliveview.ingressDomain` or `shared.ingress_domain` on the view cluster. Include the default host name `appliveview.` ahead of the domain.
- `CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER` (optional) refers to the Entitlement Account Number (EAN), which is a unique identifier VMware assigns to its customers. Tanzu Application Platform telemetry uses this number to identify data that belongs to a particular customers and prepare usage reports.
- `AMR-CLOUDEVENT-HANDLER-CA` contains the AMR CloudEvent Handler CA data. For more information about configuring the `amr` portion of the values file, see Set up multicluster Supply Chain Security Tools (SCST) - Store.

If you use custom CA certificates, you must provide one or more PEM-encoded CA certificates under the `ca_cert_data` key. If you configured `shared.ca_cert_data`, Tanzu Application Platform component packages inherit that value by default.

If you set `shared.ingress_domain` in run profile, the `appliveview_connector.backend.host` is automatically configured as `host: appliveview.INGRESS-DOMAIN`. To override the shared ingress for Application Live View to connect to the view cluster, set the `appliveview_connector.backend.host` key to `appliveview.VIEW-CLUSTER-INGRESS-DOMAIN`.

Install Tanzu Application Platform View profile

This topic tells you how to install View profile cluster by using a reduced values file.

The following is the YAML file sample for the view-profile:

```
profile: view
ceip_policy_disclosed: FALSE-OR-TRUE-VALUE # Installation fails if this is not set to
true. Not a string.

shared:
```

```
  ingress_domain: "INGRESS-DOMAIN"
  kubernetes_distribution: "openshift" # To be passed only for OpenShift. Defaults to
"".
  kubernetes_version: "K8S-VERSION"
  ca_cert_data: | # To be passed if using custom certificates.
    -----BEGIN CERTIFICATE-----
    MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...
    -----END CERTIFICATE-----

contour:
  envoy:
    service:
      type: LoadBalancer # NodePort can be used if your Kubernetes cluster doesn't sup
port LoadBalancing.

tap_gui:
  app_config:
    auth:
      allowGuestAccess: true  # This allows unauthenticated users to log in to your po
rtal. If you want to deactivate it, make sure you configure an alternative auth provid
er.
    catalog:
      locations:
        - type: url
          target: https://GIT-CATALOG-URL/catalog-info.yaml
    kubernetes:
      serviceLocatorMethod:
        type: 'multiTenant'
      clusterLocatorMethods:
        - type: 'config'
          clusters:
            - url: CLUSTER-URL
              name: CLUSTER-NAME # Build profile cluster can go here.
              authProvider: serviceAccount
              serviceAccountToken: CLUSTER-TOKEN
              skipTLSVerify: TRUE-OR-FALSE-VALUE
            - url: CLUSTER-URL
              name: CLUSTER-NAME # Run profile cluster can go here.
              authProvider: serviceAccount
              serviceAccountToken: CLUSTER-TOKEN
              skipTLSVerify: TRUE-OR-FALSE-VALUE

appliveview:
  ingressEnabled: true

tap_telemetry:
  customer_entitlement_account_number: "CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER" # (Option
al) Identify data for creating Tanzu Application Platform usage reports.
```

Where:

- `INGRESS-DOMAIN` is the subdomain for the host name that you point at the `tanzu-shared-ingress` service's external IP address.
- `K8S-VERSION` is the Kubernetes version used by your OpenShift cluster. It must be in the form of `1.23.x` or `1.24.x`, where `x` stands for the patch version. Examples:
    - Red Hat OpenShift Container Platform v4.10 uses the Kubernetes version `1.23.3`.
    - Red Hat OpenShift Container Platform v4.11 uses the Kubernetes version `1.24.1`.
- `GIT-CATALOG-URL` is the path to the `catalog-info.yaml` catalog definition file. You can download either a blank or populated catalog file from the Tanzu Application Platform product page. Otherwise, use a Backstage-compliant catalog you've already built and posted on the Git infrastructure in the Integration section.

- `CLUSTER-URL`, `CLUSTER-NAME` and `CLUSTER-TOKEN` are described in the View resources on multiple clusters in Tanzu Developer Portal. Observe the order of operations laid out in the previous steps.
- `CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER` (optional) refers to the Entitlement Account Number (EAN), which is a unique identifier VMware assigns to its customers. Tanzu Application Platform telemetry uses this number to identify data that belongs to a particular customers and prepare usage reports.

If you use custom CA certificates, you must provide one or more PEM-encoded CA certificates under the `ca_cert_data` key. If you configured `shared.ca_cert_data`, Tanzu Application Platform component packages inherit that value by default.

The `appliveview.ingressEnabled` key is set to `false` by default. In a multicluster setup, `ingressEnabled` key must be set to `true`. If the `shared.ingress_domain` key is set, the Application Live View back end is automatically exposed through the shared ingress.

Install Tanzu Application Platform Iterate profile

This topic tells you how to install Iterate profile cluster by using a reduced values file.

The following is the YAML file sample for the iterate-profile:

```
profile: iterate

shared:
  ingress_domain: "INGRESS-DOMAIN"
  kubernetes_distribution: "openshift" # To be passed only for OpenShift. Defaults to
"".
  kubernetes_version: "K8S-VERSION"
  image_registry:
    project_path: "SERVER-NAME/REPO-NAME" # To be used by Build Service by appending
"/buildservice" and used by Supply chain by appending "/workloads"
    username: "KP-DEFAULT-REPO-USERNAME"
    password: "KP-DEFAULT-REPO-PASSWORD"
  ca_cert_data: | # To be passed if using custom certificates
  -----BEGIN CERTIFICATE-----
  MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...
  -----END CERTIFICATE-----

ceip_policy_disclosed: FALSE-OR-TRUE-VALUE # Installation fails if this is not set to
true. Not a string.

# The above shared keys may be overridden in the below section.

buildservice: # Optional if the corresponding shared keys are provided.
  kp_default_repository: "KP-DEFAULT-REPO"
  kp_default_repository_username: "KP-DEFAULT-REPO-USERNAME"
  kp_default_repository_password: "KP-DEFAULT-REPO-PASSWORD"

supply_chain: basic
ootb_supply_chain_basic: # Optional if the shared above mentioned shared keys are prov
ided.
  source:
    credentials_secret: "GIT-SOURCE-CREDENTIAL-SECRET-NAME" # (Optional) Defaults to
"".
  registry:
    server: "SERVER-NAME"
    repository: "REPO-NAME"
  gitops:
    credentials_secret: "GITOPS-CREDENTIAL-SECRET-NAME" # (Optional) Defaults to "".

image_policy_webhook:
```

```
  allow_unmatched_tags: true

contour:
  envoy:
    service:
      type: LoadBalancer # (Optional) Defaults to LoadBalancer.

cnrs:
  domain_name: "TAP-ITERATE-CNR-DOMAIN" # Optional if the shared.ingress_domain is pro
vided.

appliveview_connector:
  backend:
    sslDeactivated: TRUE-OR-FALSE-VALUE
    ingressEnabled: true
    host: appliveview.VIEW-CLUSTER-INGRESS-DOMAIN

tap_telemetry:
  customer_entitlement_account_number: "CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER" # (Option
al) Identify data for creating Tanzu Application Platform usage reports.
```

Where:

- `K8S-VERSION` is the Kubernetes version used by your OpenShift cluster. It must be in the form of `1.23.x` or `1.24.x`, where `x` stands for the patch version. Examples:
  - Red Hat OpenShift Container Platform v4.10 uses the Kubernetes version `1.23.3`.
  - Red Hat OpenShift Container Platform v4.11 uses the Kubernetes version `1.24.1`.
- `KP-DEFAULT-REPO` is a writable repository in your registry. Tanzu Build Service dependencies are written to this location. Examples:
  - Harbor has the form `kp_default_repository: "my-harbor.io/my-project/build-service"`.
  - Docker Hub has the form `kp_default_repository: "my-dockerhub-user/build-service"` or `kp_default_repository: "index.docker.io/my-user/build-service"`.
  - Google Cloud Registry has the form `kp_default_repository: "gcr.io/my-project/build-service"`.
- `KP-DEFAULT-REPO-USERNAME` is the user name that can write to `KP-DEFAULT-REPO`. You can `docker push` to this location with this credential.
  - For Google Cloud Registry, use `kp_default_repository_username: _json_key`.
- `KP-DEFAULT-REPO-PASSWORD` is the password for the user that can write to `KP-DEFAULT-REPO`. You can `docker push` to this location with this credential. This credential can also be configured by using a Secret reference. For more information, see Install Tanzu Build Service for details.
  - For Google Cloud Registry, use the contents of the service account JSON file.
- `SERVER-NAME` is the host name of the registry server. Examples:
  - Harbor has the form `server: "my-harbor.io"`.
  - Docker Hub has the form `server: "index.docker.io"`.
  - Google Cloud Registry has the form `server: "gcr.io"`.
- `REPO-NAME` is where workload images are stored in the registry. Images are written to `SERVER-NAME/REPO-NAME/workload-name`. Examples:
  - Harbor has the form `repository: "my-project/supply-chain"`.
  - Docker Hub has the form `repository: "my-dockerhub-user"`.
  - Google Cloud Registry has the form `repository: "my-project/supply-chain"`.

- `GIT-SOURCE-CREDENTIAL-SECRET-NAME` is the name of the Kubernetes secret in the developer namespace that supplies the Git credentials for the supply chain to fetch source code from. See Git authentication for more information.
- `GITOPS-CREDENTIAL-SECRET-NAME` is the name of the Kubernetes secret in the developer namespace that supplies the Git credentials for the supply chain to push configuration to. See Git authentication for more information.
- `TAP-ITERATE-CNR-DOMAIN` is the iterate cluster Cloud Native Runtimes domain.
- `VIEW-CLUSTER-INGRESS-DOMAIN` is the subdomain you setup on the View profile cluster. This matches the value key `appliveview.ingressDomain` or `shared.ingress_domain` on the view cluster. Include the default host name `appliveview.` ahead of the domain.
- `CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER` (optional) refers to the Entitlement Account Number (EAN), which is a unique identifier VMware assigns to its customers. Tanzu Application Platform telemetry uses this number to identify data that belongs to a particular customers and prepare usage reports.

If you use custom CA certificates, you must provide one or more PEM-encoded CA certificates under the `ca_cert_data` key. If you configured `shared.ca_cert_data`, Tanzu Application Platform component packages inherit that value by default.

If you set `shared.ingress_domain` in the iterate profile, the `appliveview_connector.backend.host` is automatically configured as `host: appliveview.INGRESS-DOMAIN`. To override the shared ingress for Application Live View to connect to the view cluster, set the `appliveview_connector.backend.host` key to `appliveview.VIEW-CLUSTER-INGRESS-DOMAIN`.

Get started with Tanzu Application Platform

Welcome to Tanzu Application Platform (commonly known as TAP). The guides in this section provide hands-on instructions for developers and operators to help you get started on Tanzu Application Platform.

In addition to the resources provided in this Get started guide, VMware also provides developer and operator focused courses, and an experimental developer sandbox environment in Tanzu Academy.

Prerequisites

Before you start, verify you have successfully:

- **Installed Tanzu Application Platform**
  See Installing Tanzu Application Platform.
- **Installed Tanzu Application Platform on the target Kubernetes cluster**
  See Installing the Tanzu CLI and Installing the Tanzu Application Platform Package and Profiles.
- **Set the default kubeconfig context to the target Kubernetes cluster**
  See Changing clusters.
- **Installed Out of The Box (OOTB) Supply Chain Basic**
  See Install Out of The Box Supply Chain Basic. If you used the default profiles provided in Installing the Tanzu Application Platform Package and Profiles, you have already installed the Out of The Box (OOTB) Supply Chain Basic.
- **Installed Tekton Pipelines**
  See Install Tekton Pipelines. If you used the default profiles provided in Installing the Tanzu Application Platform Package and Profiles, you have already installed Tekton Pipelines.
- **Set up a developer namespace to accommodate the developer workload**
  See Set up developer namespaces to use your installed packages.
- **Installed Tanzu Developer Portal**
  See Install Tanzu Developer Portal. If you used the Full or View profiles provided in Installing the Tanzu Application Platform Package and Profiles, you have already installed Tanzu Developer Portal.
- **Installed the VS Code Tanzu Extension**
  See Install the Visual Studio Code Tanzu Extension for instructions.

When you have completed these prerequisites, you are ready to get started.

Next steps

For developers:

- Create an application accelerator

- Deploy an app on Tanzu Application Platform
- Deploy Spring Cloud Applications to Tanzu Application Platform

For operators:

- Add testing and scanning to your application
- Configure image signing

Get started with Tanzu Application Platform

Welcome to Tanzu Application Platform (commonly known as TAP). The guides in this section provide hands-on instructions for developers and operators to help you get started on Tanzu Application Platform.

In addition to the resources provided in this Get started guide, VMware also provides developer and operator focused courses, and an experimental developer sandbox environment in Tanzu Academy.

Prerequisites

Before you start, verify you have successfully:

- **Installed Tanzu Application Platform**
  See Installing Tanzu Application Platform.
- **Installed Tanzu Application Platform on the target Kubernetes cluster**
  See Installing the Tanzu CLI and Installing the Tanzu Application Platform Package and Profiles.
- **Set the default kubeconfig context to the target Kubernetes cluster**
  See Changing clusters.
- **Installed Out of The Box (OOTB) Supply Chain Basic**
  See Install Out of The Box Supply Chain Basic. If you used the default profiles provided in Installing the Tanzu Application Platform Package and Profiles, you have already installed the Out of The Box (OOTB) Supply Chain Basic.
- **Installed Tekton Pipelines**
  See Install Tekton Pipelines. If you used the default profiles provided in Installing the Tanzu Application Platform Package and Profiles, you have already installed Tekton Pipelines.
- **Set up a developer namespace to accommodate the developer workload**
  See Set up developer namespaces to use your installed packages.
- **Installed Tanzu Developer Portal**
  See Install Tanzu Developer Portal. If you used the Full or View profiles provided in Installing the Tanzu Application Platform Package and Profiles, you have already installed Tanzu Developer Portal.
- **Installed the VS Code Tanzu Extension**
  See Install the Visual Studio Code Tanzu Extension for instructions.

When you have completed these prerequisites, you are ready to get started.

Next steps

For developers:

- Create an application accelerator

- Deploy an app on Tanzu Application Platform
- Deploy Spring Cloud Applications to Tanzu Application Platform

For operators:

- Add testing and scanning to your application
- Configure image signing

Add testing and scanning to your application

This topic guides you through installing the optional OOTB Supply Chain with Testing and the optional OOTB Supply Chain with Testing and Scanning.

For more information about available supply chains, see Supply chains on Tanzu Application Platform.

What you will do

- Install OOTB Supply Chain with Testing.
- Add a Tekton pipeline to the cluster and update the workload to point to the pipeline and resolve errors.
- Install OOTB Supply Chain with Testing and Scanning.
- Update the workload to point to the Tekton pipeline and resolve errors.
- Query for vulnerabilities and dependencies.
- (Optional) Enable recurring scanning for post-build scans.

Overview

The default Out of the Box (OOTB) Supply Chain Basic and its dependencies were installed on your cluster during the Tanzu Application Platform install. As demonstrated in this guide, you can add testing and security scanning to your application. When you activate OOTB Supply Chain with Testing, it deactivates OOTB Supply Chain Basic.

The following installations also provide a sample Tekton pipeline that tests your sample application. The pipeline is configurable. Therefore, you can customize the steps to perform either additional testing or other tasks with Tekton Pipelines.

Install OOTB Supply Chain with Testing

To install OOTB Supply Chain with Testing:

1. You can activate the OOTB Supply Chain with Testing by updating your profile to use `testing` rather than `basic` as the selected supply chain for workloads in this cluster. The `tap-values.yaml` is the file used to customize the profile in `Tanzu package install tap --values-file=...`. Update `tap-values.yaml` with the following changes:

```
- supply_chain: basic
+ supply_chain: testing

- ootb_supply_chain_basic:
+ ootb_supply_chain_testing:
    registry:
      server: "SERVER-NAME"
      repository: "REPO-NAME"
```

Where:

- `SERVER-NAME` is the name of your server.
- `REPO-NAME` is the name of the image repository that hosts the container images.

2. Update the installed profile by running:

```
tanzu package installed update tap -p tap.tanzu.vmware.com -v VERSION-NUMBER --
values-file tap-values.yaml -n tap-install
```

Where `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.9.1`.

Tekton pipeline config example

In this section, a Tekton pipeline is added to the cluster. In the next section, the workload is updated to point to the pipeline and resolve any current errors.

Note

Developers can perform this step because they know how their application must be tested. The operator can also add the Tekton pipeline to a cluster before the developer gets access.

To add the Tekton pipeline to the cluster, apply the following YAML to the cluster:

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: developer-defined-tekton-pipeline
  namespace: DEVELOPER-NAMESPACE
  labels:
    apps.tanzu.vmware.com/pipeline: test     # (!) required
spec:
  params:
    - name: source-url                       # (!) required
    - name: source-revision                  # (!) required
  tasks:
    - name: test
      params:
        - name: source-url
          value: $(params.source-url)
        - name: source-revision
          value: $(params.source-revision)
      taskSpec:
        params:
          - name: source-url
          - name: source-revision
        steps:
          - name: test
            image: gradle
            script: |-
              cd `mktemp -d`

              wget -qO- $(params.source-url) | tar xvz -m
              ./mvnw test
```

Where `DEVELOPER-NAMESPACE` is the name of your developer namespace.

The preceding YAML puts a Tekton pipeline in the developer namespace you specify. It defines the Tekton pipeline with a single step. The step contained in the `steps` pulls the code from the repository indicated in the developers `workload` and runs the tests within the repository. The steps

of the Tekton pipeline are configurable and allow the developer to add additional items needed to test their code.

There are many steps in the supply chain. In this case, the next step is an image build. Any additional steps the developer adds to the Tekton pipeline to test their code are independent of the image being built and of any subsequent steps executed in the supply chain. This independence gives the developer freedom to focus on testing their code.

The `params` are templated by Supply Chain Choreographer. Additionally, Tekton pipelines require a Tekton `pipelineRun` to execute on the cluster. Supply Chain Choreographer handles creating the `pipelineRun` dynamically each time that step of the supply chain requires execution.

Workload update

To connect the new supply chain to the workload, the workload must be updated to point at your Tekton pipeline.

1. Update the workload by running the following with the Tanzu CLI:

   ```
   tanzu apps workload apply tanzu-java-web-app \
     --git-repo https://github.com/vmware-tanzu/application-accelerator-samples \
     --sub-path tanzu-java-web-app \
     --git-branch main \
     --type web \
     --label apps.tanzu.vmware.com/has-tests=true \
     --label app.kubernetes.io/part-of=tanzu-java-web-app \
     --yes
   ```

2. After accepting the workload creation, monitor the creation of new resources by the workload by running:

   ```
   kubectl get workload,gitrepository,pipelinerun,images.kpack,podintent,app,servi
   ces.serving
   ```

   The result is output similar to the following example that shows the objects created by Supply Chain Choreographer:

   ```
   NAME                                      AGE
   workload.carto.run/tanzu-java-web-app   109s

   NAME                                                              URL
   READY   STATUS                                                                AGE
   gitrepository.source.toolkit.fluxcd.io/tanzu-java-web-app   https://github.com/
   vmware-tanzu/application-accelerator-samples   True    Fetched revision: main/8
   72ff44c8866b7805fb2425130edb69a9853bfdf   109s

   NAME                                                SUCCEEDED   REASON      START
   TIME   COMPLETIONTIME
   pipelinerun.tekton.dev/tanzu-java-web-app-4ftlb   True        Succeeded   104s
   77s

   NAME                                  LATESTIMAGE
   READY
   image.kpack.io/tanzu-java-web-app   10.188.0.3:5000/foo/tanzu-java-web-app@sha2
   56:1d5bc4d3d1ffeb8629fbb721fcd1c4d28b896546e005f1efd98fbc4e79b7552c   True

   NAME                                                                READY   REASON
   AGE
   podintent.conventions.carto.run/tanzu-java-web-app   True             7s

   NAME                                        DESCRIPTION           SINCE-DEPLOY
   AGE
   app.kappctrl.k14s.io/tanzu-java-web-app   Reconcile succeeded   1s
   ```

```
2s

NAME                                                URL
LATESTCREATED              LATESTREADY                READY     REASON
service.serving.knative.dev/tanzu-java-web-app   http://tanzu-java-web-app.deve
loper.example.com   tanzu-java-web-app-00001   tanzu-java-web-app-00001   Unkno
wn   IngressNotConfigured
```

Install OOTB Supply Chain with Testing and Scanning

To install OOTB Supply Chain with Testing and Scanning, you must install the Scan Controller and the Grype scanner.

Leveraging both Grype and Tanzu Build Service in your Tanzu Application Platform supply chain enables enhanced scanning coverage for languages and frameworks. For the languages and frameworks supported, see the *Extended Scanning Coverage using Anchore Grype* column in the Language and Framework Support Table.

To install OOTB Supply Chain with Testing and Scanning:

Note

The OOTB Supply Chain with Testing and Scanning capability has been changed to opt-in and is skipped by default starting in Tanzu Application Platform v1.6. For more information, see Scan Types.

1. Supply Chain Security Tools (SCST) - Scan is installed as part of the Tanzu Application Platform profiles. Verify that Scan Controller is installed by running:

   ```
   tanzu package installed get scanning -n tap-install
   ```

   If the package is not already installed, follow the steps in Supply Chain Security Tools - Scan to install the required scanning components.

2. Apply a ScanPolicy in the required namespace. The sample ScanPolicy provided in this step is configured with `notAllowedSeverities := ["Critical","High","UnknownSeverity"]`. This blocks a supply chain when the scanner finds CVEs with critical, high, and unknown ratings.

   You can also configure the supply chain to use your own custom policies and apply exceptions when you want to ignore certain CVEs. See Out of the Box Supply Chain with Testing and Scanning. To apply the sample ScanPolicy, you can either add the namespace flag to the kubectl command or add the namespace field to the template by running:

   ```
   kubectl apply -f - -o yaml << EOF
   ---
   apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
   kind: ScanPolicy
   metadata:
     name: scan-policy
     labels:
       'app.kubernetes.io/part-of': 'enable-in-gui'
   spec:
     regoFile: |
       package main

       # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "Unkn
   ownSeverity"
       notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]
       ignoreCves := []
   ```

```
    contains(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
      severities := { e | e := match.ratings.rating.severity } | { e | e := mat
ch.ratings.rating[_].severity }
      some i
      fails := contains(notAllowedSeverities, severities[i])
      not fails
    }

    isSafe(match) {
      ignore := contains(ignoreCves, match.id)
      ignore
    }

    deny[msg] {
      comps := { e | e := input.bom.components.component } | { e | e := input.b
om.components.component[_] }
      some i
      comp := comps[i]
      vulns := { e | e := comp.vulnerabilities.vulnerability } | { e | e := com
p.vulnerabilities.vulnerability[_] }
      some j
      vuln := vulns[j]
      ratings := { e | e := vuln.ratings.rating.severity } | { e | e := vuln.ra
tings.rating[_].severity }
      not isSafe(vuln)
      msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
    }
EOF
```

3. (optional) The Tanzu Application Platform profiles install the Supply Chain Security Tools - Store package by default. To persist and query the vulnerability results post-scan, confirm it is installed by running:

   ```
   tanzu package installed get metadata-store -n tap-install
   ```

   If the package is not installed, follow the installation instructions at Install Supply Chain Security Tools - Store independent from Tanzu Application Platform profiles.

4. Update the profile to use the supply chain with testing and scanning by updating `tap-values.yaml` (the file used to customize the profile in `tanzu package install tap --values-file=...`) with the following changes:

   ```
   - supply_chain: testing
   + supply_chain: testing_scanning

   - ootb_supply_chain_testing:
   + ootb_supply_chain_testing_scanning:
       registry:
         server: "<SERVER-NAME>"
         repository: "<REPO-NAME>"
   ```

5. Update the `tap` package:

   ```
   tanzu package installed update tap -p tap.tanzu.vmware.com -v VERSION-NUMBER --
   values-file tap-values.yaml -n tap-install
   ```

   Where `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.9.1`.

6. Because you set `supply_chain: testing_scanning` in your `tap-values.yaml` earlier, Grype Scanner for SCST - Scan is installed through Namespace Provisioner. Verify that Grype Scanner is present on your workload namespace:

   ```
   tanzu package installed get grype-scanner-YOUR-DEV-NAMESPACE -n tap-install
   ```

   To install Grype Scanner to multiple namespaces, VMware recommends using Namespace Provisioner. ScanTemplates with Grype scanner are automatically installed into namespaces created by Namespace Provisioner.

   Grype scanner must be present in the namespace that the workload is deployed to for scanning.

Workload update

To connect the new supply chain to the workload, update the workload to point to your Tekton pipeline:

1. Update the workload by running the following using the Tanzu CLI:

   ```
   tanzu apps workload apply tanzu-java-web-app \
     --git-repo https://github.com/vmware-tanzu/application-accelerator-samples \
     --sub-path tanzu-java-web-app \
     --git-branch main \
     --type web \
     --label apps.tanzu.vmware.com/has-tests=true \
     --label app.kubernetes.io/part-of=tanzu-java-web-app \
     --yes
   ```

2. After accepting the workload creation, view the new resources that the workload created by running:

   ```
   kubectl get workload,gitrepository,sourcescan,pipelinerun,images.kpack,imagesca
   n,podintent,app,services.serving
   ```

   The following is an example output, which shows the objects that Supply Chain Choreographer created:

   ```
   NAME                                     AGE
   workload.carto.run/tanzu-java-web-app   109s

   NAME                                                              URL
   READY   STATUS                                                                AGE
   gitrepository.source.toolkit.fluxcd.io/tanzu-java-web-app   https://github.com/
   vmware-tanzu/application-accelerator-samples   True    Fetched revision: main/8
   72ff44c8866b7805fb2425130edb69a9853bfdf   109s

   NAME                                                          PHASE       SCAN
   NEDREVISION                              SCANNEDREPOSITORY
   AGE    CRITICAL   HIGH   MEDIUM   LOW   UNKNOWN   CVETOTAL
   sourcescan.scanning.apps.tanzu.vmware.com/tanzu-java-web-app   Completed   1878
   50b39b754e4256213407879327 59a0838795   https://github.com/vmware-tanzu/applicat
   ion-accelerator-samples   90s

   NAME                                              SUCCEEDED   REASON      START
   TIME   COMPLETIONTIME
   pipelinerun.tekton.dev/tanzu-java-web-app-4ftlb   True        Succeeded   104s
   77s

   NAME                                LATESTIMAGE
   READY
   image.kpack.io/tanzu-java-web-app   10.188.0.3:5000/foo/tanzu-java-web-app@sha2
   ```

```
56:1d5bc4d3d1ffeb8629fbb721fcd1c4d28b896546e005f1efd98fbc4e79b7552c   True

NAME                                                                  PHASE       SCANN
EDIMAGE
AGE   CRITICAL   HIGH   MEDIUM   LOW   UNKNOWN   CVETOTAL
imagescan.scanning.apps.tanzu.vmware.com/tanzu-java-web-app   Completed   10.18
8.0.3:5000/foo/tanzu-java-web-app@sha256:1d5bc4d3d1ffeb8629fbb721fcd1c4d28b8965
46e005f1efd98fbc4e79b7552c   14s

NAME                                                                    READY   REASON
AGE
podintent.conventions.carto.run/tanzu-java-web-app   True             7s

NAME                                        DESCRIPTION           SINCE-DEPLOY
AGE
app.kappctrl.k14s.io/tanzu-java-web-app   Reconcile succeeded   1s
2s

NAME                                                URL
LATESTCREATED              LATESTREADY                READY     REASON
service.serving.knative.dev/tanzu-java-web-app   http://tanzu-java-web-app.deve
loper.example.com   tanzu-java-web-app-00001   tanzu-java-web-app-00001   Unkno
wn   IngressNotConfigured
```

Important

If the source or image scan has a “Failed” phase this means that the scan failed due to a scan policy violation and the supply chain stops. For information about the CVE triage workflow, see Out of the Box Supply Chain with Testing and Scanning.

Query for vulnerabilities

Scan reports are automatically saved to the Supply Chain Security Tools - Store, and you can query them for vulnerabilities and dependencies. For example, related to open-source software (OSS) or third-party packages.

Query the tanzu-java-web-app image dependencies and vulnerabilities by running:

```
tanzu insight image get --digest DIGEST
tanzu insight image vulnerabilities --digest  DIGEST
```

Where `DIGEST` is the component version or image digest printed in the `KUBECTL GET` command.

For additional information and examples, see Tanzu Insight plug-in overview.

Congratulations! You have successfully added testing and security scanning to your application on the Tanzu Application Platform.

Take the next steps to learn about recommended supply chain security best practices and gain a powerful services journey experience on the Tanzu Application Platform by enabling several advanced use cases.

(Optional) Scan for vulnerabilities after build time

When you add the capability to scan image containers to your supply chain, you can detect vulnerabilities at build time. However, as new vulnerabilities are discovered and reported daily, it is important to frequently scan the container images to detect newly reported vulnerabilities.

You can enable recurring scanning to schedule periodic scans of images built by workloads and containers running in Tanzu Application Platform clusters. When you enable recurring scanning, an

option to **View Latest Scan Results** becomes available on the Supply Chain Choreographer plug-in. For how to enable recurring scanning, see Set up recurring scanning.

Next steps

- Configure image signing and verification in your supply chain

Add testing and scanning to your application

This topic guides you through installing the optional OOTB Supply Chain with Testing and the optional OOTB Supply Chain with Testing and Scanning.

For more information about available supply chains, see Supply chains on Tanzu Application Platform.

What you will do

- Install OOTB Supply Chain with Testing.
- Add a Tekton pipeline to the cluster and update the workload to point to the pipeline and resolve errors.
- Install OOTB Supply Chain with Testing and Scanning.
- Update the workload to point to the Tekton pipeline and resolve errors.
- Query for vulnerabilities and dependencies.
- (Optional) Enable recurring scanning for post-build scans.

Overview

The default Out of the Box (OOTB) Supply Chain Basic and its dependencies were installed on your cluster during the Tanzu Application Platform install. As demonstrated in this guide, you can add testing and security scanning to your application. When you activate OOTB Supply Chain with Testing, it deactivates OOTB Supply Chain Basic.

The following installations also provide a sample Tekton pipeline that tests your sample application. The pipeline is configurable. Therefore, you can customize the steps to perform either additional testing or other tasks with Tekton Pipelines.

Install OOTB Supply Chain with Testing

To install OOTB Supply Chain with Testing:

1. You can activate the OOTB Supply Chain with Testing by updating your profile to use `testing` rather than `basic` as the selected supply chain for workloads in this cluster. The `tap-values.yaml` is the file used to customize the profile in `Tanzu package install tap --values-file=...`. Update `tap-values.yaml` with the following changes:

```
- supply_chain: basic
+ supply_chain: testing

- ootb_supply_chain_basic:
+ ootb_supply_chain_testing:
    registry:
      server: "SERVER-NAME"
      repository: "REPO-NAME"
```

Where:

- `SERVER-NAME` is the name of your server.
- `REPO-NAME` is the name of the image repository that hosts the container images.

2. Update the installed profile by running:

```
tanzu package installed update tap -p tap.tanzu.vmware.com -v VERSION-NUMBER --
values-file tap-values.yaml -n tap-install
```

Where `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.9.1`.

Tekton pipeline config example

In this section, a Tekton pipeline is added to the cluster. In the next section, the workload is updated to point to the pipeline and resolve any current errors.

Note

Developers can perform this step because they know how their application must be tested. The operator can also add the Tekton pipeline to a cluster before the developer gets access.

To add the Tekton pipeline to the cluster, apply the following YAML to the cluster:

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: developer-defined-tekton-pipeline
  namespace: DEVELOPER-NAMESPACE
  labels:
    apps.tanzu.vmware.com/pipeline: test     # (!) required
spec:
  params:
    - name: source-url                       # (!) required
    - name: source-revision                  # (!) required
  tasks:
    - name: test
      params:
        - name: source-url
          value: $(params.source-url)
        - name: source-revision
          value: $(params.source-revision)
      taskSpec:
        params:
          - name: source-url
          - name: source-revision
        steps:
          - name: test
            image: gradle
            script: |-
              cd `mktemp -d`

              wget -qO- $(params.source-url) | tar xvz -m
              ./mvnw test
```

Where `DEVELOPER-NAMESPACE` is the name of your developer namespace.

The preceding YAML puts a Tekton pipeline in the developer namespace you specify. It defines the Tekton pipeline with a single step. The step contained in the `steps` pulls the code from the repository indicated in the developers `workload` and runs the tests within the repository. The steps

of the Tekton pipeline are configurable and allow the developer to add additional items needed to test their code.

There are many steps in the supply chain. In this case, the next step is an image build. Any additional steps the developer adds to the Tekton pipeline to test their code are independent of the image being built and of any subsequent steps executed in the supply chain. This independence gives the developer freedom to focus on testing their code.

The `params` are templated by Supply Chain Choreographer. Additionally, Tekton pipelines require a Tekton `pipelineRun` to execute on the cluster. Supply Chain Choreographer handles creating the `pipelineRun` dynamically each time that step of the supply chain requires execution.

Workload update

To connect the new supply chain to the workload, the workload must be updated to point at your Tekton pipeline.

1. Update the workload by running the following with the Tanzu CLI:

   ```
   tanzu apps workload apply tanzu-java-web-app \
     --git-repo https://github.com/vmware-tanzu/application-accelerator-samples \
     --sub-path tanzu-java-web-app \
     --git-branch main \
     --type web \
     --label apps.tanzu.vmware.com/has-tests=true \
     --label app.kubernetes.io/part-of=tanzu-java-web-app \
     --yes
   ```

2. After accepting the workload creation, monitor the creation of new resources by the workload by running:

   ```
   kubectl get workload,gitrepository,pipelinerun,images.kpack,podintent,app,servi
   ces.serving
   ```

   The result is output similar to the following example that shows the objects created by Supply Chain Choreographer:

   ```
   NAME                                      AGE
   workload.carto.run/tanzu-java-web-app   109s

   NAME                                                              URL
   READY   STATUS                                                                AGE
   gitrepository.source.toolkit.fluxcd.io/tanzu-java-web-app   https://github.com/
   vmware-tanzu/application-accelerator-samples   True    Fetched revision: main/8
   72ff44c8866b7805fb2425130edb69a9853bfdf   109s

   NAME                                                SUCCEEDED   REASON      START
   TIME   COMPLETIONTIME
   pipelinerun.tekton.dev/tanzu-java-web-app-4ftlb   True        Succeeded   104s
   77s

   NAME                                  LATESTIMAGE
   READY
   image.kpack.io/tanzu-java-web-app   10.188.0.3:5000/foo/tanzu-java-web-app@sha2
   56:1d5bc4d3d1ffeb8629fbb721fcd1c4d28b896546e005f1efd98fbc4e79b7552c   True

   NAME                                                                READY   REASON
   AGE
   podintent.conventions.carto.run/tanzu-java-web-app   True             7s

   NAME                                        DESCRIPTION           SINCE-DEPLOY
   AGE
   app.kappctrl.k14s.io/tanzu-java-web-app   Reconcile succeeded   1s
   ```

```
2s

NAME                                                URL
LATESTCREATED              LATESTREADY                READY     REASON
service.serving.knative.dev/tanzu-java-web-app   http://tanzu-java-web-app.deve
loper.example.com   tanzu-java-web-app-00001   tanzu-java-web-app-00001   Unkno
wn   IngressNotConfigured
```

Install OOTB Supply Chain with Testing and Scanning

To install OOTB Supply Chain with Testing and Scanning, you must install the Scan Controller and the Grype scanner.

Leveraging both Grype and Tanzu Build Service in your Tanzu Application Platform supply chain enables enhanced scanning coverage for languages and frameworks. For the languages and frameworks supported, see the *Extended Scanning Coverage using Anchore Grype* column in the Language and Framework Support Table.

To install OOTB Supply Chain with Testing and Scanning:

Note

The OOTB Supply Chain with Testing and Scanning capability has been changed to opt-in and is skipped by default starting in Tanzu Application Platform v1.6. For more information, see Scan Types.

1. Supply Chain Security Tools (SCST) - Scan is installed as part of the Tanzu Application Platform profiles. Verify that Scan Controller is installed by running:

   ```
   tanzu package installed get scanning -n tap-install
   ```

   If the package is not already installed, follow the steps in Supply Chain Security Tools - Scan to install the required scanning components.

2. Apply a ScanPolicy in the required namespace. The sample ScanPolicy provided in this step is configured with `notAllowedSeverities := ["Critical","High","UnknownSeverity"]`. This blocks a supply chain when the scanner finds CVEs with critical, high, and unknown ratings.

   You can also configure the supply chain to use your own custom policies and apply exceptions when you want to ignore certain CVEs. See Out of the Box Supply Chain with Testing and Scanning. To apply the sample ScanPolicy, you can either add the namespace flag to the kubectl command or add the namespace field to the template by running:

   ```
   kubectl apply -f - -o yaml << EOF
   ---
   apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
   kind: ScanPolicy
   metadata:
     name: scan-policy
     labels:
       'app.kubernetes.io/part-of': 'enable-in-gui'
   spec:
     regoFile: |
       package main

       # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "Unkn
   ownSeverity"
       notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]
       ignoreCves := []
   ```

```
    contains(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
      severities := { e | e := match.ratings.rating.severity } | { e | e := mat
ch.ratings.rating[_].severity }
      some i
      fails := contains(notAllowedSeverities, severities[i])
      not fails
    }

    isSafe(match) {
      ignore := contains(ignoreCves, match.id)
      ignore
    }

    deny[msg] {
      comps := { e | e := input.bom.components.component } | { e | e := input.b
om.components.component[_] }
      some i
      comp := comps[i]
      vulns := { e | e := comp.vulnerabilities.vulnerability } | { e | e := com
p.vulnerabilities.vulnerability[_] }
      some j
      vuln := vulns[j]
      ratings := { e | e := vuln.ratings.rating.severity } | { e | e := vuln.ra
tings.rating[_].severity }
      not isSafe(vuln)
      msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
    }
EOF
```

3. (optional) The Tanzu Application Platform profiles install the Supply Chain Security Tools - Store package by default. To persist and query the vulnerability results post-scan, confirm it is installed by running:

   ```
   tanzu package installed get metadata-store -n tap-install
   ```

   If the package is not installed, follow the installation instructions at Install Supply Chain Security Tools - Store independent from Tanzu Application Platform profiles.

4. Update the profile to use the supply chain with testing and scanning by updating `tap-values.yaml` (the file used to customize the profile in `tanzu package install tap --values-file=...`) with the following changes:

   ```
   - supply_chain: testing
   + supply_chain: testing_scanning

   - ootb_supply_chain_testing:
   + ootb_supply_chain_testing_scanning:
       registry:
         server: "<SERVER-NAME>"
         repository: "<REPO-NAME>"
   ```

5. Update the `tap` package:

   ```
   tanzu package installed update tap -p tap.tanzu.vmware.com -v VERSION-NUMBER --
   values-file tap-values.yaml -n tap-install
   ```

   Where `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.9.1`.

6. Because you set `supply_chain: testing_scanning` in your `tap-values.yaml` earlier, Grype Scanner for SCST - Scan is installed through Namespace Provisioner. Verify that Grype Scanner is present on your workload namespace:

   ```
   tanzu package installed get grype-scanner-YOUR-DEV-NAMESPACE -n tap-install
   ```

   To install Grype Scanner to multiple namespaces, VMware recommends using Namespace Provisioner. ScanTemplates with Grype scanner are automatically installed into namespaces created by Namespace Provisioner.

   Grype scanner must be present in the namespace that the workload is deployed to for scanning.

Workload update

To connect the new supply chain to the workload, update the workload to point to your Tekton pipeline:

1. Update the workload by running the following using the Tanzu CLI:

   ```
   tanzu apps workload apply tanzu-java-web-app \
     --git-repo https://github.com/vmware-tanzu/application-accelerator-samples \
     --sub-path tanzu-java-web-app \
     --git-branch main \
     --type web \
     --label apps.tanzu.vmware.com/has-tests=true \
     --label app.kubernetes.io/part-of=tanzu-java-web-app \
     --yes
   ```

2. After accepting the workload creation, view the new resources that the workload created by running:

   ```
   kubectl get workload,gitrepository,sourcescan,pipelinerun,images.kpack,imagesca
   n,podintent,app,services.serving
   ```

   The following is an example output, which shows the objects that Supply Chain Choreographer created:

   ```
   NAME                                     AGE
   workload.carto.run/tanzu-java-web-app    109s

   NAME                                                                URL
   READY   STATUS                                                                  AGE
   gitrepository.source.toolkit.fluxcd.io/tanzu-java-web-app   https://github.com/
   vmware-tanzu/application-accelerator-samples   True    Fetched revision: main/8
   72ff44c8866b7805fb2425130edb69a9853bfdf   109s

   NAME                                                              PHASE       SCAN
   NEDREVISION                              SCANNEDREPOSITORY
   AGE    CRITICAL   HIGH   MEDIUM   LOW   UNKNOWN   CVETOTAL
   sourcescan.scanning.apps.tanzu.vmware.com/tanzu-java-web-app   Completed   1878
   50b39b754e425621340787932759a0838795   https://github.com/vmware-tanzu/applicat
   ion-accelerator-samples   90s

   NAME                                              SUCCEEDED   REASON      START
   TIME   COMPLETIONTIME
   pipelinerun.tekton.dev/tanzu-java-web-app-4ftlb   True        Succeeded   104s
   77s

   NAME                                LATESTIMAGE
   READY
   image.kpack.io/tanzu-java-web-app   10.188.0.3:5000/foo/tanzu-java-web-app@sha2
   ```

```
56:1d5bc4d3d1ffeb8629fbb721fcd1c4d28b896546e005f1efd98fbc4e79b7552c   True

NAME                                                                  PHASE       SCANN
EDIMAGE
AGE   CRITICAL   HIGH   MEDIUM   LOW   UNKNOWN   CVETOTAL
imagescan.scanning.apps.tanzu.vmware.com/tanzu-java-web-app   Completed   10.18
8.0.3:5000/foo/tanzu-java-web-app@sha256:1d5bc4d3d1ffeb8629fbb721fcd1c4d28b8965
46e005f1efd98fbc4e79b7552c   14s

NAME                                                                    READY   REASON
AGE
podintent.conventions.carto.run/tanzu-java-web-app   True             7s

NAME                                        DESCRIPTION           SINCE-DEPLOY
AGE
app.kappctrl.k14s.io/tanzu-java-web-app   Reconcile succeeded   1s
2s

NAME                                              URL
LATESTCREATED              LATESTREADY                READY     REASON
service.serving.knative.dev/tanzu-java-web-app   http://tanzu-java-web-app.deve
loper.example.com   tanzu-java-web-app-00001   tanzu-java-web-app-00001   Unkno
wn   IngressNotConfigured
```

Important

If the source or image scan has a “Failed” phase this means that the scan failed due to a scan policy violation and the supply chain stops. For information about the CVE triage workflow, see Out of the Box Supply Chain with Testing and Scanning.

Query for vulnerabilities

Scan reports are automatically saved to the Supply Chain Security Tools - Store, and you can query them for vulnerabilities and dependencies. For example, related to open-source software (OSS) or third-party packages.

Query the tanzu-java-web-app image dependencies and vulnerabilities by running:

```
tanzu insight image get --digest DIGEST
tanzu insight image vulnerabilities --digest  DIGEST
```

Where `DIGEST` is the component version or image digest printed in the `KUBECTL GET` command.

For additional information and examples, see Tanzu Insight plug-in overview.

Congratulations! You have successfully added testing and security scanning to your application on the Tanzu Application Platform.

Take the next steps to learn about recommended supply chain security best practices and gain a powerful services journey experience on the Tanzu Application Platform by enabling several advanced use cases.

(Optional) Scan for vulnerabilities after build time

When you add the capability to scan image containers to your supply chain, you can detect vulnerabilities at build time. However, as new vulnerabilities are discovered and reported daily, it is important to frequently scan the container images to detect newly reported vulnerabilities.

You can enable recurring scanning to schedule periodic scans of images built by workloads and containers running in Tanzu Application Platform clusters. When you enable recurring scanning, an

option to **View Latest Scan Results** becomes available on the Supply Chain Choreographer plug-in. For how to enable recurring scanning, see Set up recurring scanning.

Next steps

- Configure image signing and verification in your supply chain

Configure image signing and verification in your supply chain

This topic guides you through configuring your Tanzu Application Platform (commonly known as TAP) supply chain to sign and verify your image builds.

What you will do

- Configure your supply chain to sign your image builds.
- Configure an admission control policy to verify image signatures before admitting pods to the cluster.

Configure your supply chain to sign and verify your image builds

1. Use Cosign to configure Tanzu Build Service to sign your container image builds. For instructions, see Configure Tanzu Build Service to sign your image builds.
2. Create a `values.yaml` file, and install the Supply Chain Security Tools - Policy Controller. For instructions, see Install Supply Chain Security Tools - Policy Controller.
3. Create a `ClusterImagePolicy` that passes Tanzu Application Platform images. It is planned for a future release for these to be signed and verifiable, but currently we recommend creating a policy to pass them:

   For example:

   ```
   kubectl apply -f - -o yaml << EOF
   ---
   apiVersion: policy.sigstore.dev/v1beta1
   kind: ClusterImagePolicy
   metadata:
     name: image-policy-exceptions
   spec:
     images:
     - glob: registry.example.org/myproject/*
     - glob: REPO-NAME*
     authorities:
     - static:
         action: pass
   EOF
   ```

   Where:

   - `REPO-NAME` is the repository in your registry where Tanzu Build Service dependencies are stored. This is the exact same value configured in the `kp_default_repository` inside your `tap-values.yaml` or `tbs-values.yaml` files. Examples:
     - Harbor has the form `"my-harbor.io/my-project/build-service"`.

        - Docker Hub has the form `"my-dockerhub-user/build-service"` or `"index.docker.io/my-user/build-service"`.
        - Google Cloud Registry has the form `"gcr.io/my-project/build-service"`.
    - Add any unsigned image that must run in your namespace to the previous policy. For example, if you add a Tekton pipeline that runs a Gradle image for testing, you need to add `glob: index.docker.io/library/gradle*` to `spec.images.glob` in the preceding code.
    - Replace `registry.example.org/myproject/*` with your target registry for your Tanzu Application Platform images. If you did not relocate the Tanzu Application Platform images to your own registry during installation, use `registry.tanzu.vmware.com/tanzu-application-platform/tap-packages*`.

4. Configure and apply a `ClusterImagePolicy` resource to the cluster to verify image signatures when deploying resources. For instructions, see Create a ClusterImagePolicy resource.

   For example:

   ```
   kubectl apply -f - -o yaml << EOF
   ---
   apiVersion: policy.sigstore.dev/v1beta1
   kind: ClusterImagePolicy
   metadata:
     name: example-policy
   spec:
     images:
     - glob: registry.example.org/myproject/*
     authorities:
     - key:
         data: |
           -----BEGIN PUBLIC KEY-----
           <content ...>
           -----END PUBLIC KEY-----
   EOF
   ```

5. Enable the policy controller verification in your namespace by adding the label `policy.sigstore.dev/include: "true"` to the namespace resource.

   For example:

   ```
   kubectl label namespace YOUR-NAMESPACE policy.sigstore.dev/include=true
   ```

   Where `YOUR-NAMESPACE` is the name of your secure namespace.

   > **Note**
   >
   > Supply Chain Security Tools - Policy Controller only validates resources in namespaces that have chosen to opt in.

When you apply the `ClusterImagePolicy` resource, your cluster requires valid signatures for all images that match the `spec.images.glob[]` you define in the configuration. For more information about configuring an image policy, see Configuring Supply Chain Security Tools - Policy.

Next steps

- Consume services on Tanzu Application Platform

Or learn more about Supply Chain Security Tools:

- Overview for Supply Chain Security Tools - Policy
- Configuring Supply Chain Security Tools - Policy
- Supply Chain Security Tools - Policy known issues

Integrate a plug-in into your Tanzu Developer Portal

This topic guides you through integrating the hello-world plug-in into your Tanzu Developer Portal by using the Configurator tool. The hello-world plug-in is a demonstration of the Configurator tool's capabilities.

For general information about the Tanzu Developer Portal Configurator and its basic concepts, see Overview of Configurator.

What you will do

- Create a customized Tanzu Developer Portal containing the hello-world plug-in
- Observe the customized Tanzu Developer Portal instance

Prerequisites

Before you start, you must have:

- All the Prerequisites for the getting started guide.
- The Tanzu Developer Portal Configurator bundle available (`tpb.tanzu.vmware.com`).

  To verify it's present, run:

  ```
  tanzu package available list --namespace tap-install | grep tpb.tanzu.vmware.com
  ```

- An instance of the canonical Tanzu Developer Portal.

  To get the fully-qualified domain name for the portal, run:

  ```
  kubectl get httpproxy tap-gui -n tap-gui
  ```

  Note

  By default, Tanzu Developer Portal uses a self-signed certificate and might cause a security error in the browser. To address this error, see Configure a TLS certificate by using an existing certificate.

Customize your Tanzu Developer Portal by adding the hello-world plug-in

To integrate the hello-world plug-in into your Tanzu Developer Portal:

1. Create the **tpb-config.yaml** file specifying the list of additional plug-ins that you want to integrate. In this example, you only specify the hello-world plug-in that has both front end and back end parts:

   ```
   app:
     plugins:
       - name: '@tpb/plugin-hello-world'
   ```

```
        version: '^1.6.0-release-1.6.x.1'
    backend:
      plugins:
        - name: '@tpb/plugin-hello-world-backend'
          version: '^1.6.0-release-1.6.x.1'
```

Note

If the version of the plug-in is not specified, the workload that uses the config file as input is likely to fail.

2. Encode the file in base64, to later embed `tpb-config.yaml` in the workload definition file, by running:

```
base64 -i tdp-config.yaml
```

Example output:

```
YXBwOgogIHBsdWdpbnM6CiAgICAtIG5hbWU6ICdAdHBiL3BsdWdpbi1oZWxsby13b3JsZCcKICAgICA
gdmVyc2lvbjogJ14xLjYuMC1yZWxlYXNlLTEuNi54LjEnIApiYWNrZW5kOgogIHBsdWdpbnM6CiAgIC
AtIG5hbWU6ICdAdHBiL3BsdWdpbi1oZWxsby13b3JsZC1iYWNrZW5kJwogICAgICB2ZXJzaW9uOiAnX
jEuNi4wLXJlbGVhc2UtMS42LnguMScK
```

You use this value later as `ENCODED-TDP-CONFIG-VALUE`.

3. Identify the location of the Configurator's image by running these commands:

```
export OUTPUT_IMAGE=$(kubectl -n tap-install get package tpb.tanzu.vmware.com.
0.1.2 -o "jsonpath={.spec.template.spec.fetch[0].imgpkgBundle.image}")
```

```
imgpkg pull -b ${OUTPUT_IMAGE} -o tpb-package
```

```
yq -r ".images[0].image" <tpb-package/.imgpkg/images.yml
```

You use this value later as `TDP-IMAGE-LOCATION`.

4. Prepare your workload definition file. Create a file called `tdp-workload.yaml` with the following content:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: tdp-configurator
  namespace: DEVELOPER-NAMESPACE
  labels:
    apps.tanzu.vmware.com/workload-type: web
    app.kubernetes.io/part-of: tdp-configurator
spec:
  build:
    env:
      - name: BP_NODE_RUN_SCRIPTS
        value: 'set-tpb-config,portal:pack'
      - name: TPB_CONFIG
        value: /tmp/tpb-config.yaml
      - name: TPB_CONFIG_STRING
        value: ENCODED-TDP-CONFIG-VALUE

  source:
    image: TDP-IMAGE-LOCATION
    subPath: builder
```

Where:

- `DEVELOPER-NAMESPACE` is the developer namespace configured on the cluster.
- `ENCODED-TDP-CONFIG-VALUE` is the base64-encoded **tpb-config.yaml** that you encoded earlier.
- `TDP-IMAGE-LOCATION` is the location of the Configurator image that you identified earlier.

Important

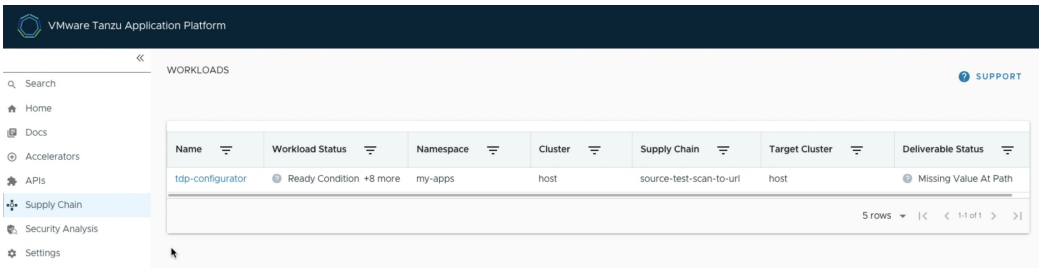
Depending on which supply chain you're using and how you configured it, you might have to add extra sections to your workload definition file to accommodate activities such as testing.

5. Deploy your workload on your Tanzu Application Platform cluster by running:

```
tanzu apps workload create -f tdp-workload.yaml -n DEVELOPER-NAMESPACE
```

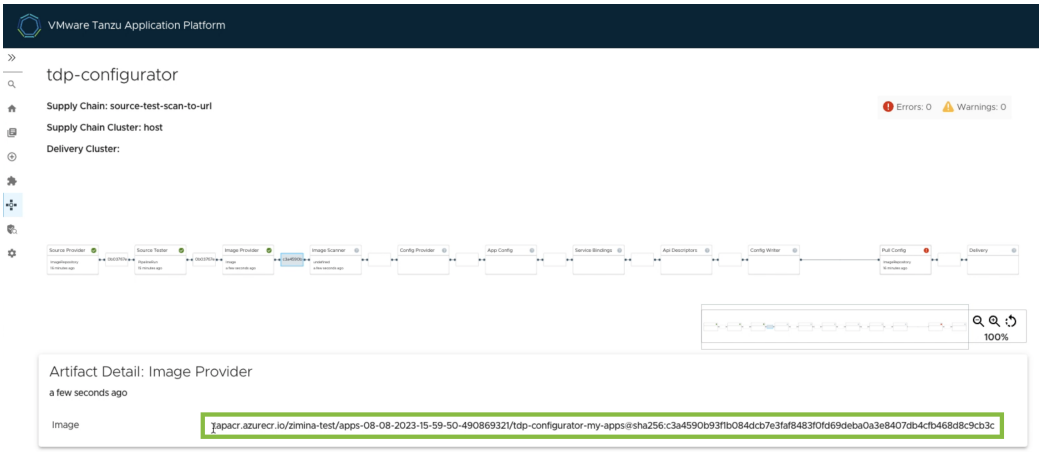
Where `DEVELOPER-NAMESPACE` is the developer namespace configured on the cluster.

6. After the workload has deployed, view it in the **Supply Chains** tab of the Tanzu Developer Portal.



Wait for the **Image Provider** step of the supply chain to complete.

7. From the **Supply Chains** tab of the Tanzu Developer Portal, click the box that follows the **Image Provider** step and copy the address to the customized Tanzu Developer Portal's image from the **Artifact Detail: Image Provider** section.



You use this value as `IMAGE-REFERENCE` in the next step.

8. Create the **tdp-overlay-secret.yaml** to insert the new image into the Tanzu Developer Portal instance.

```
apiVersion: v1
kind: Secret
metadata:
  name: tpb-app-image-overlay-secret
  namespace: tap-install
stringData:
  tpb-app-image-overlay.yaml: |
    #@ load("@ytt:overlay", "overlay")

    #! makes an assumption that tap-gui is deployed in the namespace: "tap-gui"
    #@overlay/match by=overlay.subset({"kind": "Deployment", "metadata": {"nam
e": "server", "namespace": "tap-gui"}}), expects="1+"
    ---
    spec:
      template:
        spec:
          containers:
            #@overlay/match by=overlay.subset({"name": "backstage"}),expects="1
+"
            #@overlay/match-child-defaults missing_ok=True
            - image: IMAGE-REFERENCE
            #@overlay/replace
              args:
              - -c
              - |
                export KUBERNETES_SERVICE_ACCOUNT_TOKEN="$(cat /var/run/secret
s/kubernetes.io/serviceaccount/token)"
                exec /layers/tanzu-buildpacks_node-engine-lite/node/bin/node po
rtal/dist/packages/backend  \
                --config=portal/app-config.yaml \
                --config=portal/runtime-config.yaml \
                --config=/etc/app-config/app-config.yaml
```

Where `IMAGE-REFERENCE` is address of the customized Tanzu Developer Portal image identified in the previous step.

Important

Any changes in the overlay of the YAML file will cause an error.

9. Deploy the secret to the Tanzu Application Platform cluster by running:

```
kubectl apply -f tdp-overlay-secret.yaml
```

10. Amend the `tap-values.yaml` file to include the overlay secret by adding the following lines:

```
    profile: full
    tap_gui:
      ...
    package_overlays:
    - name: tap-gui
      secrets:
      - name: tpb-app-image-overlay-secret
```

11. Apply the new `tap-values.yaml` to your cluster. The exact steps vary depending on your installation method such as GitOps, online install, or offline install.

    For how to do so for an online installation, see Install your Tanzu Application Platform package.

    For example, you can use the following Tanzu CLI command:

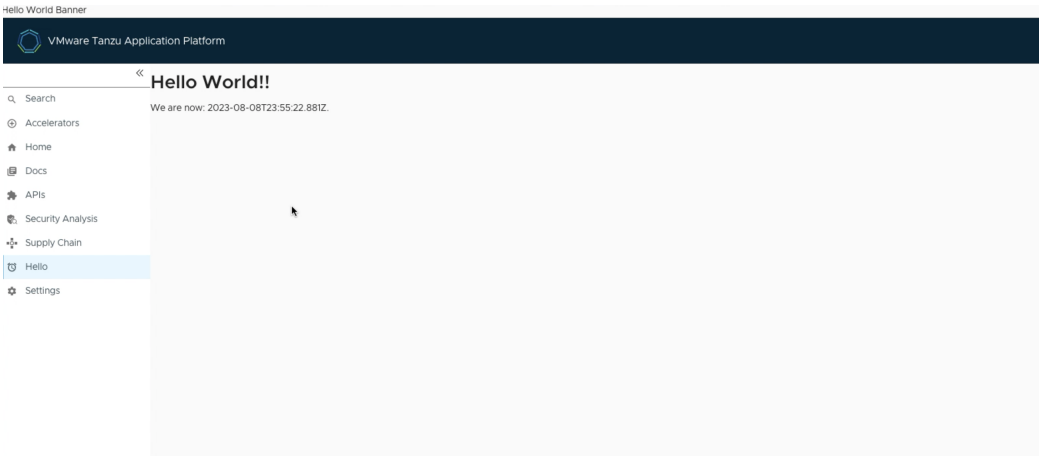
```
tanzu package installed update tap -f tap-values.yaml -n tap-install
```

12. After all packages have reconciled, the Tanzu Developer Portal pod restarts and your customized portal takes the place of the default one.

> **Note**
>
> You might have to clear your browser cache or hard refresh to see the updated Tanzu Developer Portal.

If the old Tanzu Developer Portal pod does not automatically delete, manually delete it by running:

```
kubectl delete pod POD-NAME -n tap-gui
```



Next steps

Now that you have your hello-world plug-in integrated into your Tanzu Developer Portal, explore how to Add external plug-ins to Tanzu Developer Portal.

Generate an application with Application Accelerator

This topic guides you through how to generate a new project using Application Accelerator and how to deploy the project onto a Tanzu Application Platform (commonly known as TAP) cluster. For background information, see Application Accelerator.

Prerequisites

Before you start, complete all Getting Started prerequisites.

Generate a project using an Application Accelerator

There are multiple interfaces that you can use to generate a new project. The options are:

- Application Accelerator extension for VS Code
- Application Accelerator plug-in for IntelliJ
- Tanzu Developer Portal

Choose one of the following tabs for how to generate and deploy applications using your selected interface. If you have already generated a project and want to skip this step, you can go to Deploying your application with Tanzu Application Platform.

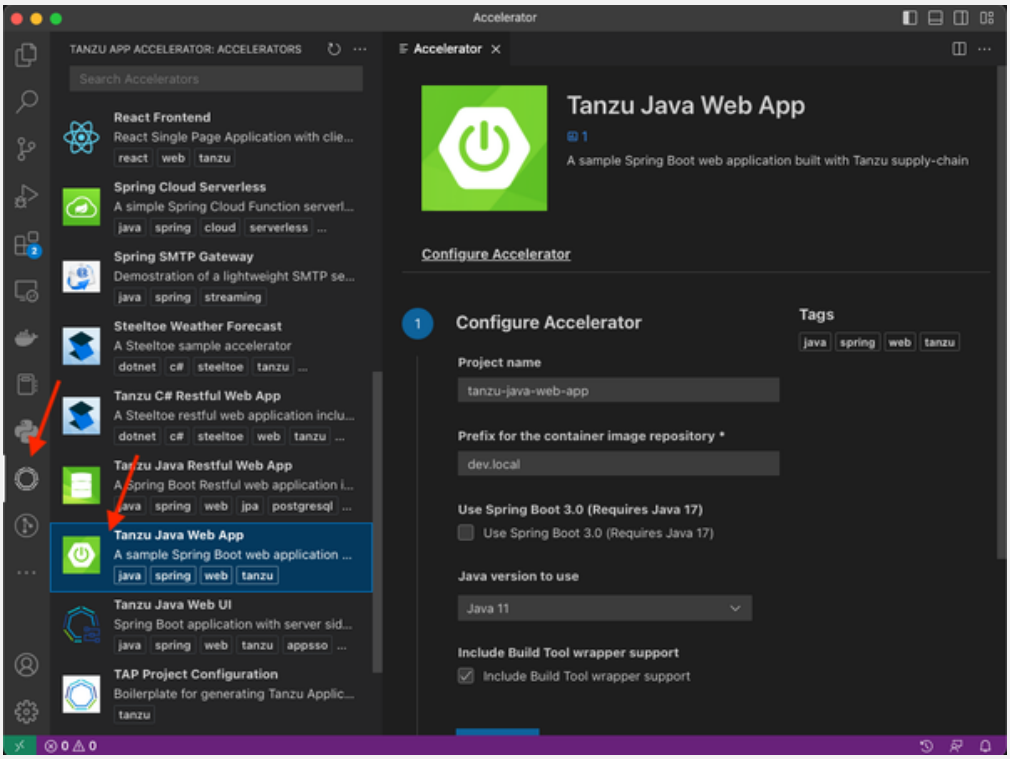
VS Code

What you will do:

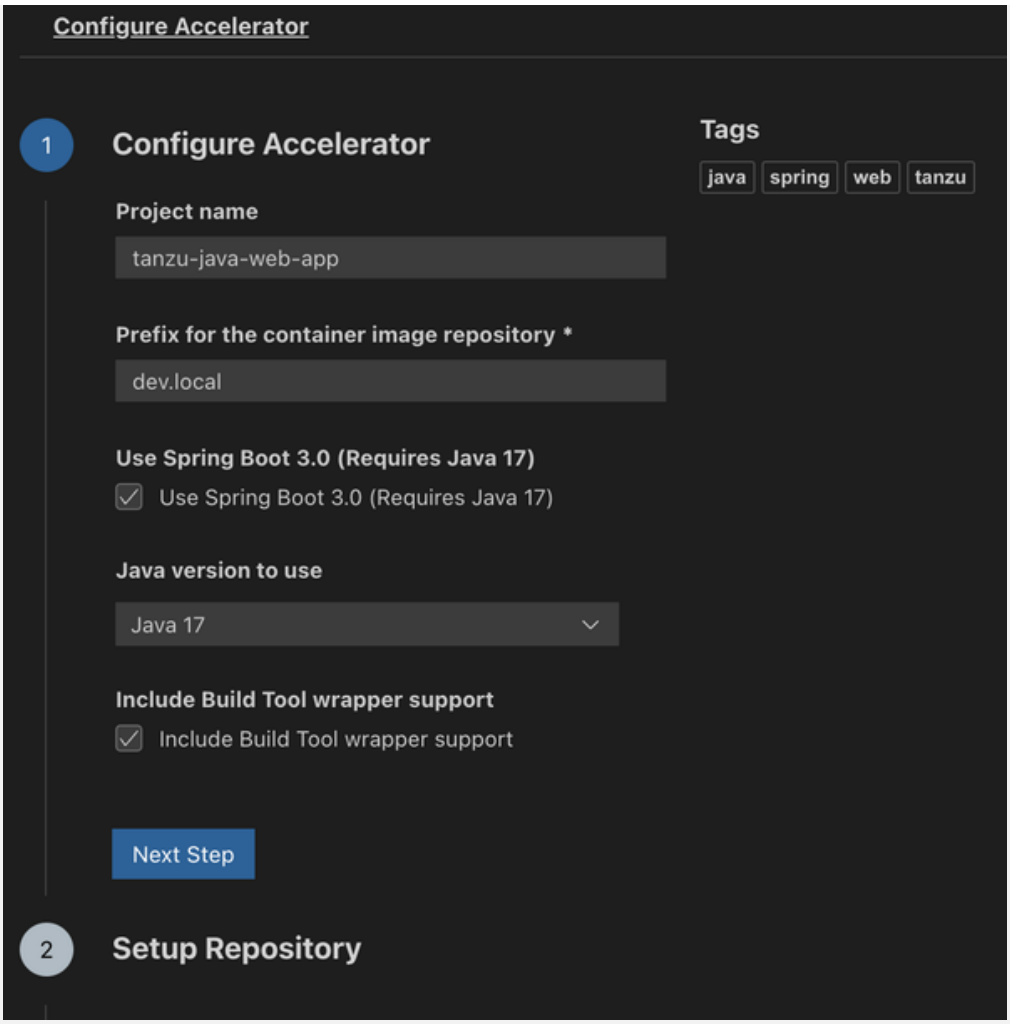
- Install the Application Accelerator extension for VS Code.
- (Optional) Provision a new GitHub repository and upload the project to the repository.
- Generate a project using an Application Accelerator.

To generate a new project using an Application Accelerator:

1. Install and configure the Application Accelerator extension for VS Code, see Application Accelerator Visual Studio Code extension.
2. Select an accelerator from the catalog. This example uses `Tanzu Java Web App`.



3. In **Configure Accelerator**, configure the accelerator as defined by your project's requirements. This example configures the project to use Spring Boot v3.0 and Java v17.



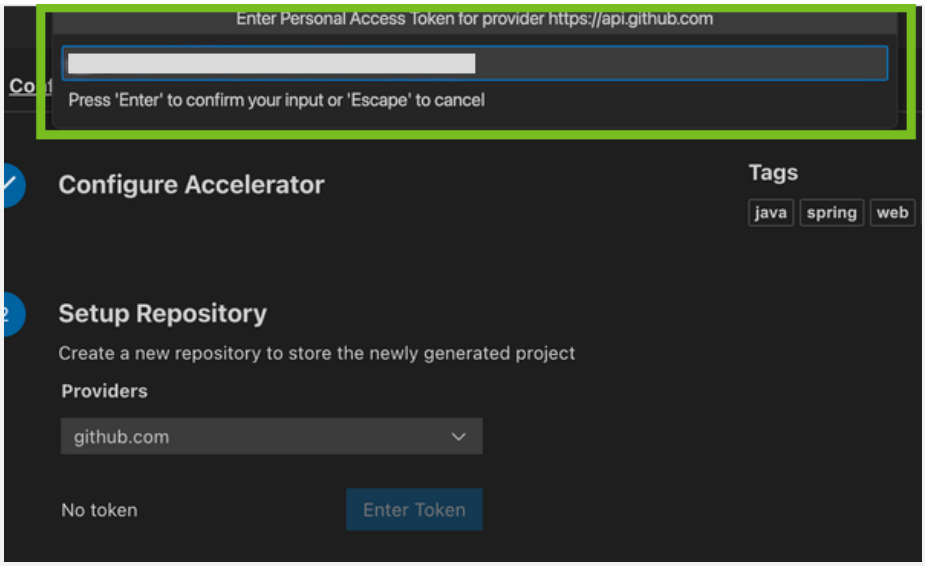
4. Click **Next Step**.
5. If your organization's Tanzu Application Platform is configured for Git repository creation, configure the **Setup Repository** step using the following sub-steps. If not, click **Skip** and go to step 5.

   Note

   For information about configuring optional Git repository creation and supported repositories, see Create an Application Accelerator Git repository during project creation.

   1. Using the **Providers** drop-down menu, select your Git provider. For example, `github.com`.
   2. After you select the provider, a dialog box appears for you to enter an API token for your Git provider. Populate the text box with your provider's API token and press Enter.

      This API key must be able to create new repositories for an organization or user. For information about how to create an API token for Git repository creation, see Creating a personal access token in the GitHub documentation.



3. In the **Owner** text box, enter the name of either the GitHub organization or user name to create the repository under.
4. In the **Repository Name** text box, enter the name of the project repository.
5. In the **Repository Branch** text box, enter the name of the default branch for the project repository. Typically, this is set to `main`.
6. Click **Next Step**.

6. In the **Review and Generate** step, verify that all the information you provided is accurate, then click **Generate Project**.
7. A dialog box appears for you to choose a location for the project to be stored on the local file system. Choose a directory or create a new one.
8. After the project has generated, a second dialog box appears for you to open the new project in a new window. Click **Yes**.
9. When opened, the project is ready for development.

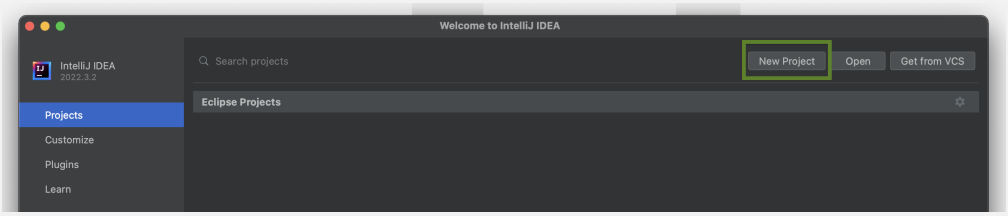
IntelliJ

What you will do:

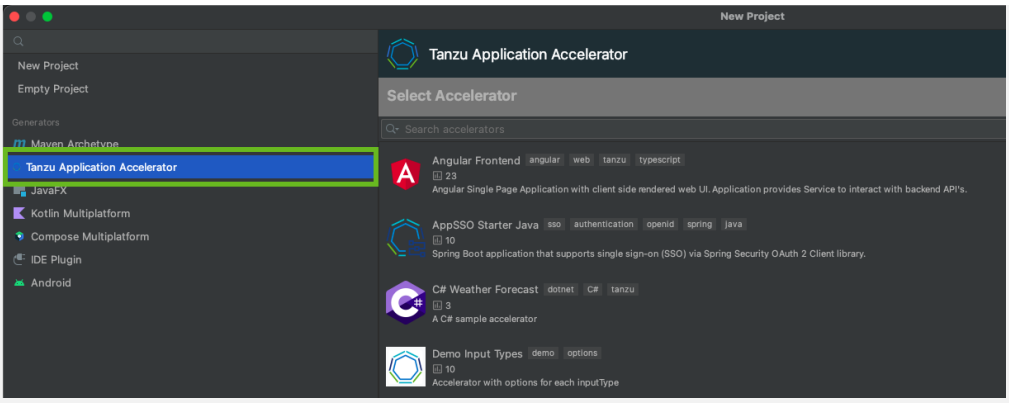
- Install the Application Accelerator plug-in for IntelliJ.
- Generate a project using an Application Accelerator.

To generate a new project using an Application Accelerator:

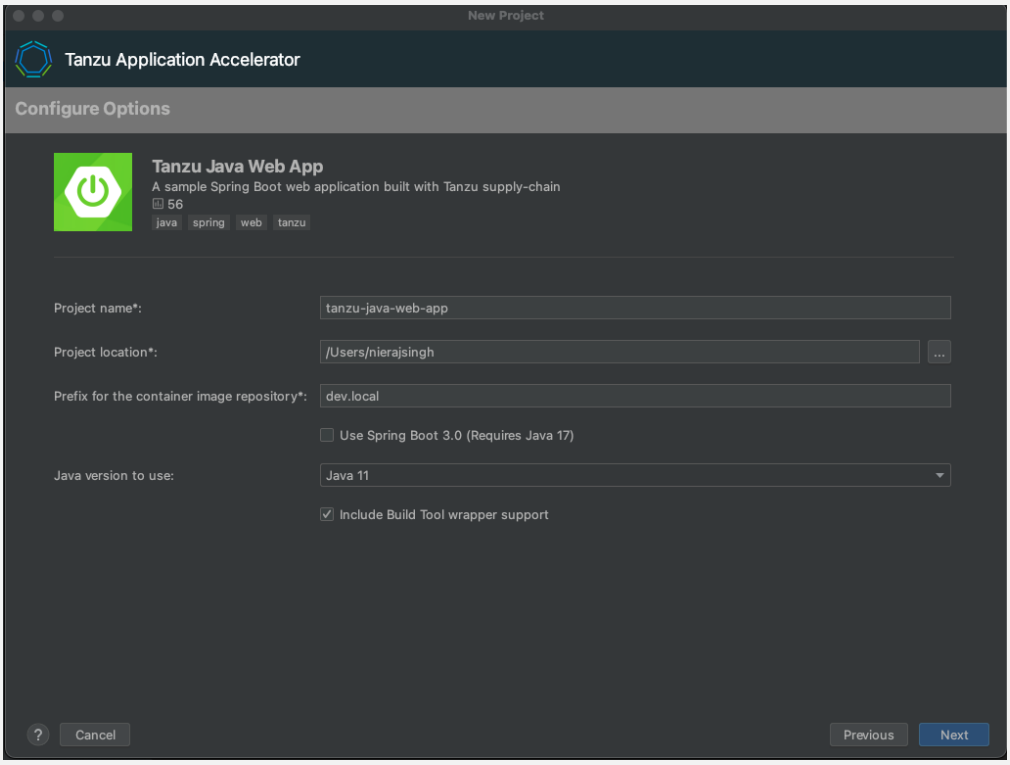
1. Install and configure the Application Accelerator plug-in for IntelliJ, see Application Accelerator plugin for IntelliJ.
2. On the Welcome to IntelliJ IDEA page, click **New Project**.



3. Click **Tanzu Application Accelerator** in the left side panel.



4. Select an accelerator from the catalog. This example uses `Tanzu Java Web App`.
5. Click **Next**.
6. In the **Configure Options** step, configure the accelerator as defined by your project's requirements.



7. Click **Next**.
8. In the **Review and Generate** step, verify that all the information provided is accurate then click **Next**.
9. After the project has generated, click **Create** to open the new project in IntelliJ.
10. When opened, the project is ready for development.

Tanzu Developer Portal

In this example, you use the `Tanzu-Java-Web-App` accelerator. You also use Tanzu Developer Portal. For information about connecting to Tanzu Developer Portal, see Access Tanzu Developer Portal.

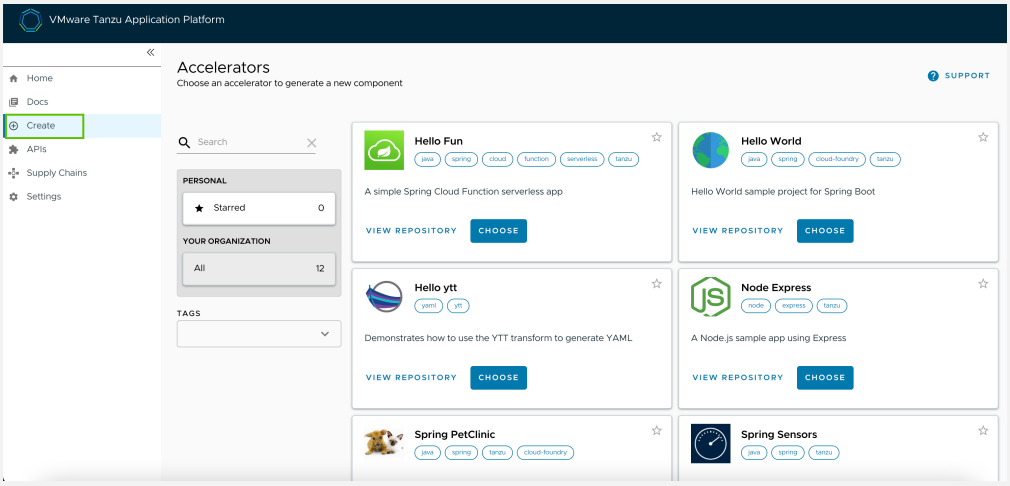
What you will do:

- Generate a project from an Application Accelerator.
- (Optional) Provision a new Git repository for the project.

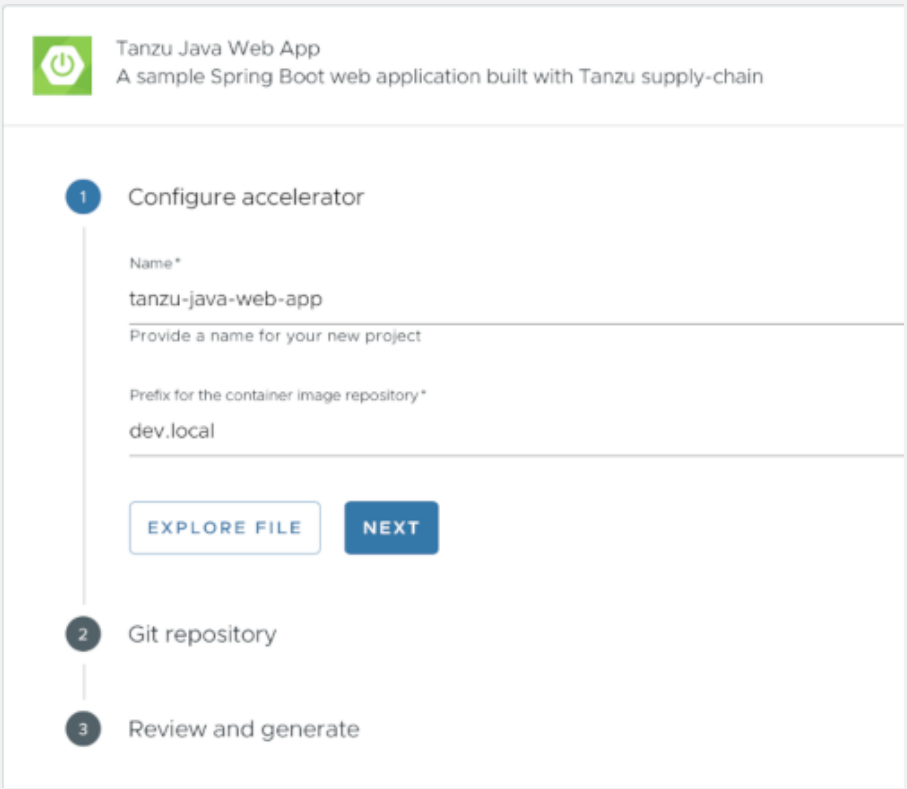
- Upload it to your Git repository of choice.

To generate a new project using an Application Accelerator:

1. From Tanzu Developer Portal, click **Create** located on the left side of the navigation pane to see the list of available accelerators.



2. Locate the Tanzu Java Web App accelerator and click **CHOOSE**.
3. In the **Generate Accelerators** dialog box, replace the default value `dev.local` in the **prefix for container image registry** text box with the registry in the form of `SERVER-NAME/REPO-NAME`. The `SERVER-NAME/REPO-NAME` must match what was specified for `registry` as part of the installation values for `ootb_supply_chain_basic`. See the Full Profile section on Installing Tanzu Application Platform package and../install-online/install.hbs.md profiles.

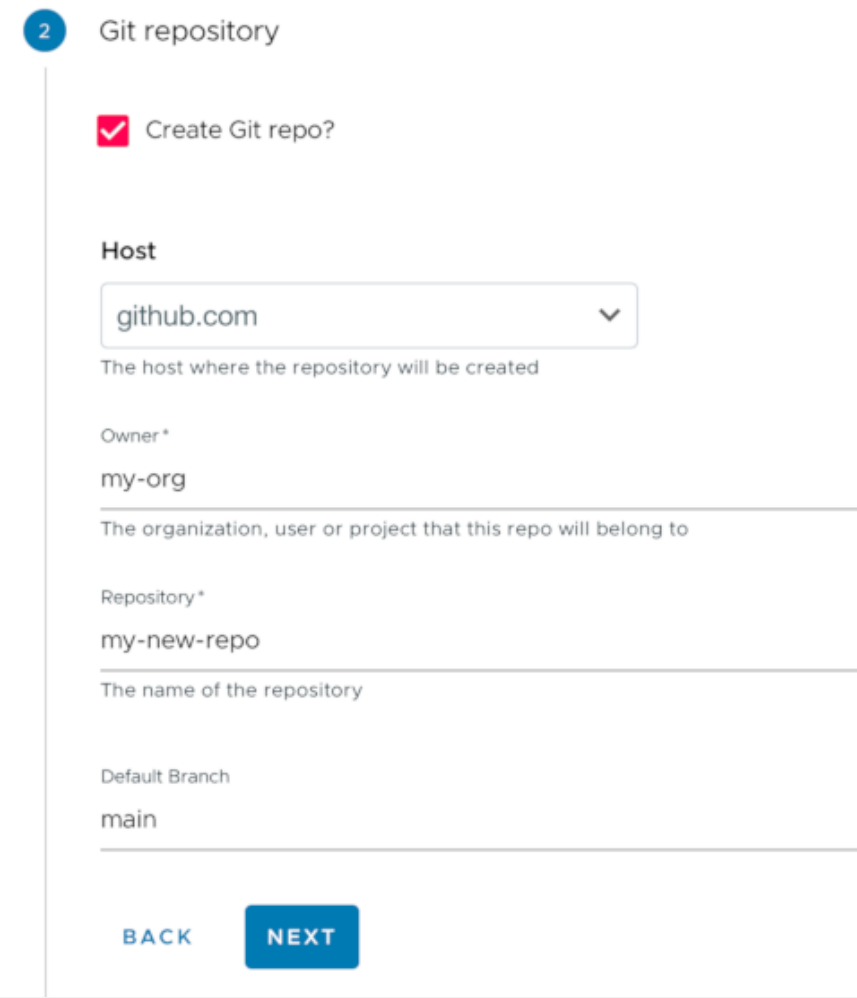


4. Click **NEXT**.
5. If your instance has optional Git repository support enabled, continue with the following sub-steps. If your instance does not support this, skip to step 5.

Note

For information about configuring optional Git repository creation and supported repositories, see Create an Application Accelerator Git repository during project creation.

1. Select the **Create Git repo?** check box.
2. Select the host Git repository provider from the **Host** drop-down menu. For example, `github.com`.
3. Populate the **Owner** and **Repository** text boxes.



4. While you are populating the form, a dialog box appears asking for permission to provision Git repositories. Follow the prompts and continue.
5. Click **NEXT**.
6. Verify the provided information, and click **GENERATE ACCELERATOR**.
7. After the Task Activity processes complete, click **DOWNLOAD ZIP FILE**.
8. After downloading the ZIP file, expand it in a workspace directory. If you did not create a Git repository in the preceding steps, follow your preferred procedure for uploading the generated project files to a Git repository for your new project.

Learn more about Application Accelerator

- For information about how to configure optional Git repository creation, see Configure in *Create an Application Accelerator Git repository during project creation*.
- For information about Application Accelerator configurations, see Configure Application Accelerator.
- For information about installing the Application Accelerator extension for Visual Studio Code, see Application Accelerator Visual Studio Code extension.
- For general accelerator troubleshooting, see Troubleshooting Application Accelerator for VMware Tanzu.

Next Steps

Now that you have generated a project that is ready for Tanzu Application Platform, learn how to quickly deploy the application on a Tanzu Application Platform cluster in Deploy an app on Tanzu Application Platform.

Generate an application with Application Accelerator

This topic guides you through how to generate a new project using Application Accelerator and how to deploy the project onto a Tanzu Application Platform (commonly known as TAP) cluster. For background information, see Application Accelerator.

Prerequisites

Before you start, complete all Getting Started prerequisites.

Generate a project using an Application Accelerator

There are multiple interfaces that you can use to generate a new project. The options are:

- Application Accelerator extension for VS Code
- Application Accelerator plug-in for IntelliJ
- Tanzu Developer Portal

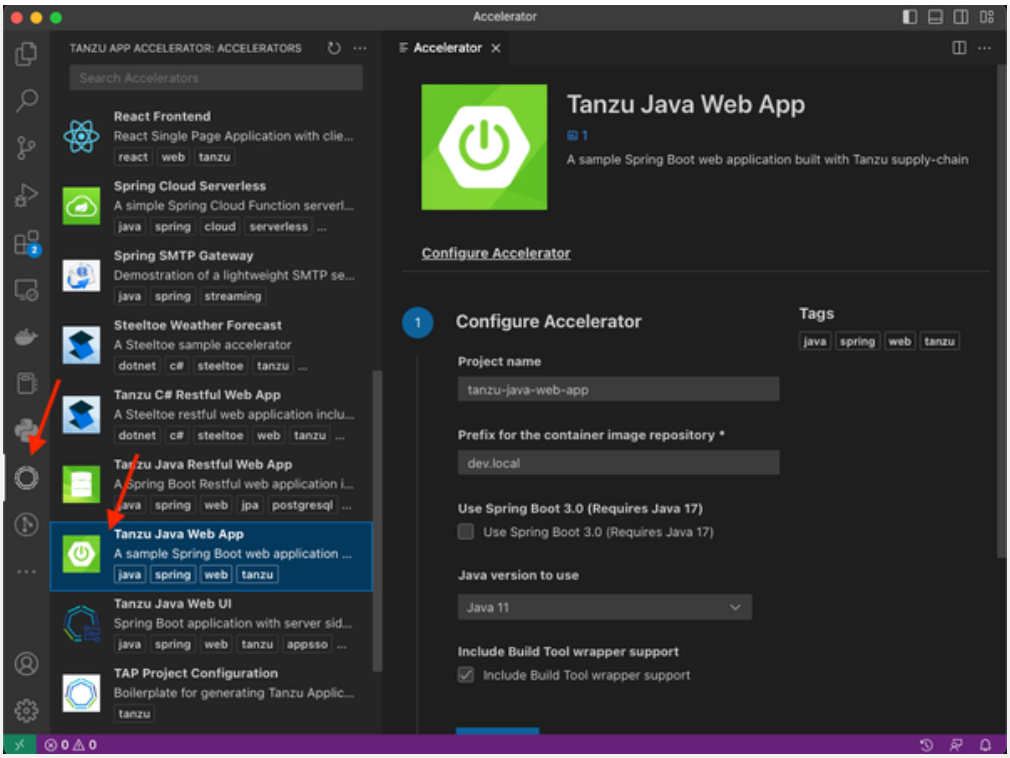
Choose one of the following tabs for how to generate and deploy applications using your selected interface. If you have already generated a project and want to skip this step, you can go to Deploying your application with Tanzu Application Platform.

VS Code
What you will do:

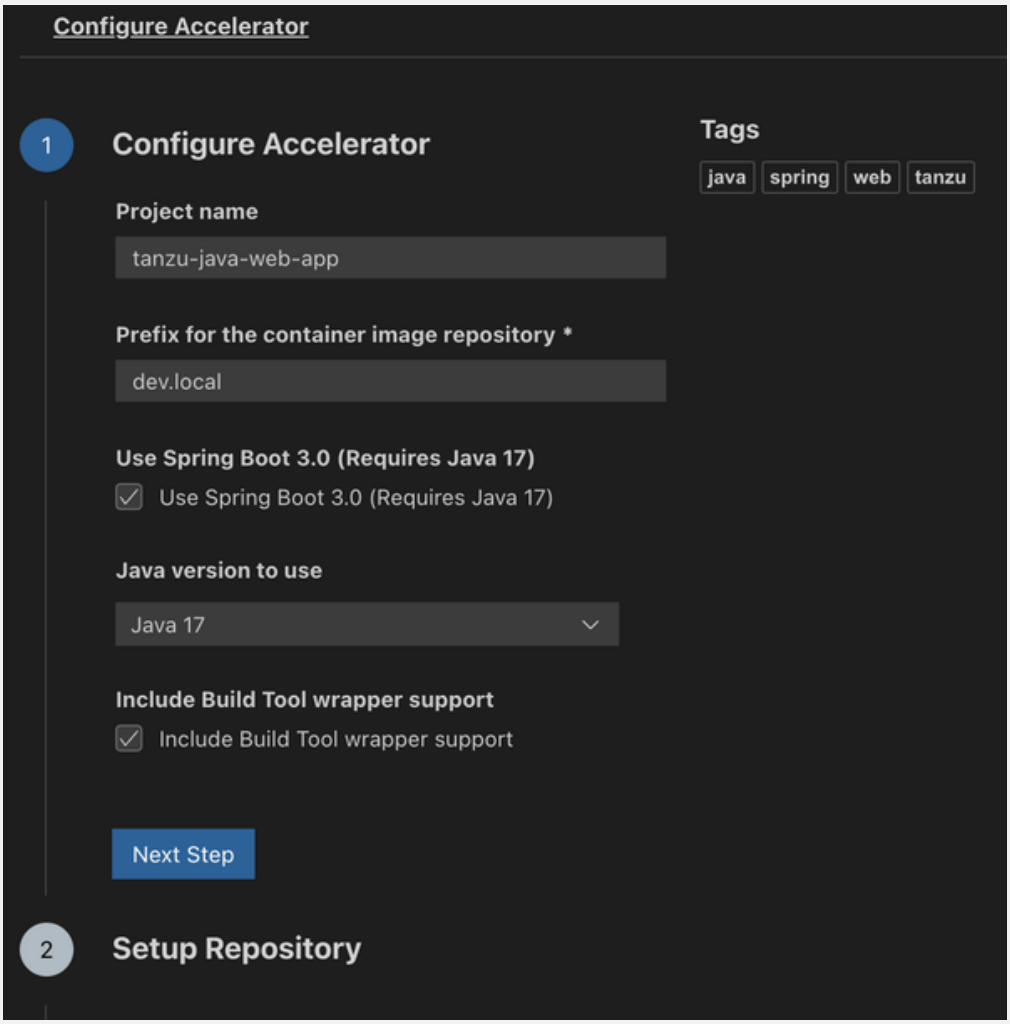
- Install the Application Accelerator extension for VS Code.
- (Optional) Provision a new GitHub repository and upload the project to the repository.
- Generate a project using an Application Accelerator.

To generate a new project using an Application Accelerator:

1. Install and configure the Application Accelerator extension for VS Code, see Application Accelerator Visual Studio Code extension.
2. Select an accelerator from the catalog. This example uses `Tanzu Java Web App`.



3. In **Configure Accelerator**, configure the accelerator as defined by your project's requirements. This example configures the project to use Spring Boot v3.0 and Java v17.

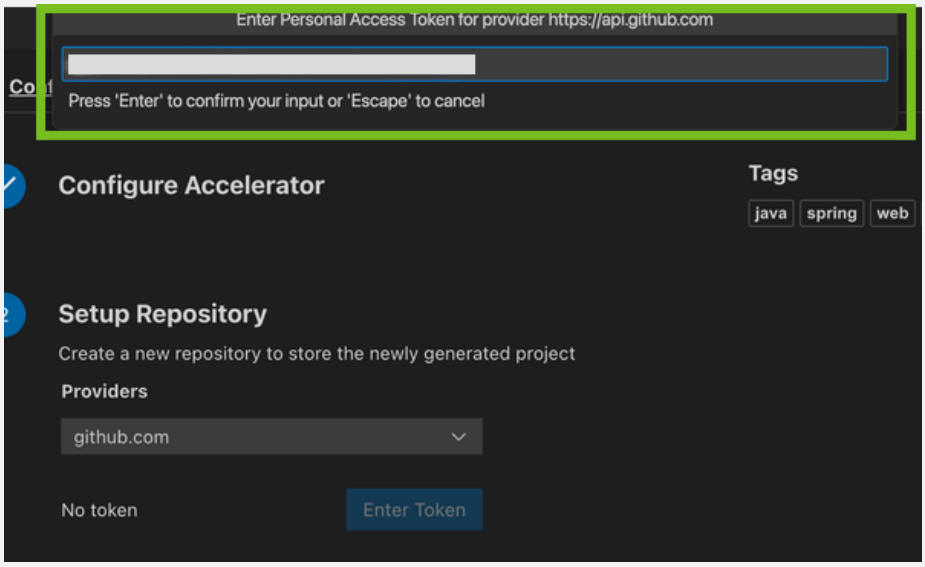


4. Click **Next Step**.
5. If your organization's Tanzu Application Platform is configured for Git repository creation, configure the **Setup Repository** step using the following sub-steps. If not, click **Skip** and go to step 5.

   > **Note**
   >
   > For information about configuring optional Git repository creation and supported repositories, see Create an Application Accelerator Git repository during project creation.

   1. Using the **Providers** drop-down menu, select your Git provider. For example, `github.com`.
   2. After you select the provider, a dialog box appears for you to enter an API token for your Git provider. Populate the text box with your provider's API token and press Enter.

      This API key must be able to create new repositories for an organization or user. For information about how to create an API token for Git repository creation, see Creating a personal access token in the GitHub documentation.

   3. In the **Owner** text box, enter the name of either the GitHub organization or user name to create the repository under.
   4. In the **Repository Name** text box, enter the name of the project repository.
   5. In the **Repository Branch** text box, enter the name of the default branch for the project repository. Typically, this is set to `main`.
   6. Click **Next Step**.
6. In the **Review and Generate** step, verify that all the information you provided is accurate, then click **Generate Project**.
7. A dialog box appears for you to choose a location for the project to be stored on the local file system. Choose a directory or create a new one.
8. After the project has generated, a second dialog box appears for you to open the new project in a new window. Click **Yes**.
9. When opened, the project is ready for development.

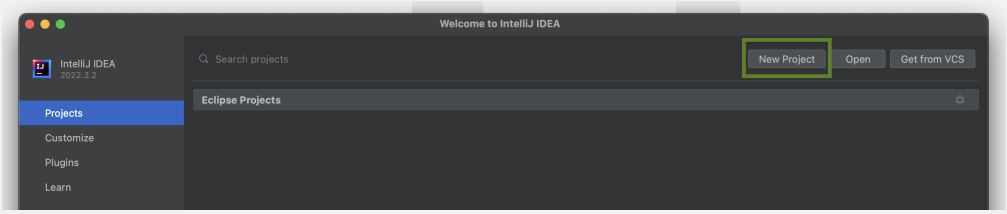
IntelliJ

What you will do:

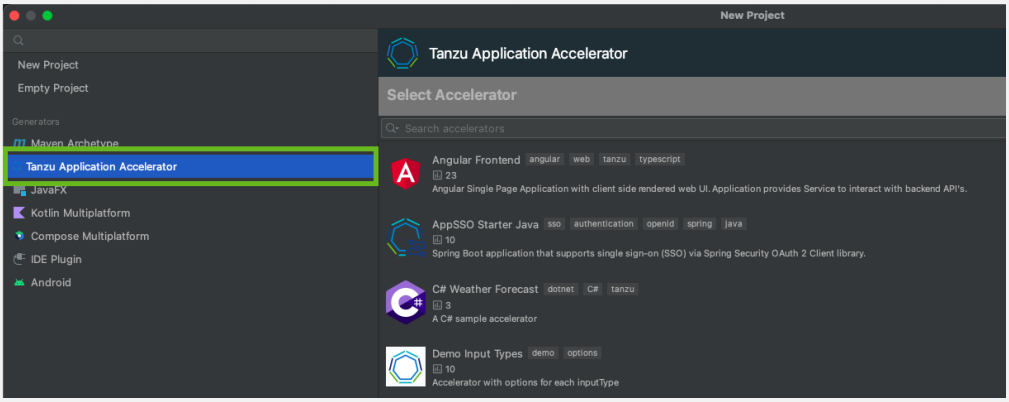
- Install the Application Accelerator plug-in for IntelliJ.
- Generate a project using an Application Accelerator.

To generate a new project using an Application Accelerator:

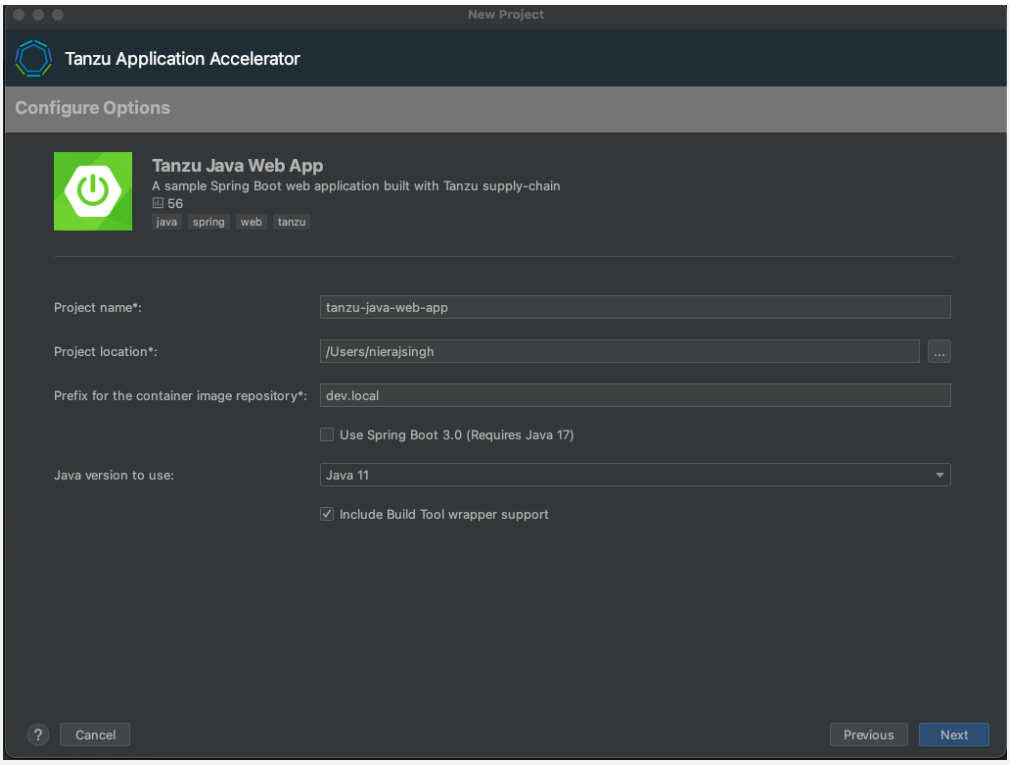
1. Install and configure the Application Accelerator plug-in for IntelliJ, see Application Accelerator plugin for IntelliJ.
2. On the Welcome to IntelliJ IDEA page, click **New Project**.



3. Click **Tanzu Application Accelerator** in the left side panel.



4. Select an accelerator from the catalog. This example uses `Tanzu Java Web App`.
5. Click **Next**.
6. In the **Configure Options** step, configure the accelerator as defined by your project's requirements.



7. Click **Next**.
8. In the **Review and Generate** step, verify that all the information provided is accurate then click **Next**.
9. After the project has generated, click **Create** to open the new project in IntelliJ.
10. When opened, the project is ready for development.

Tanzu Developer Portal

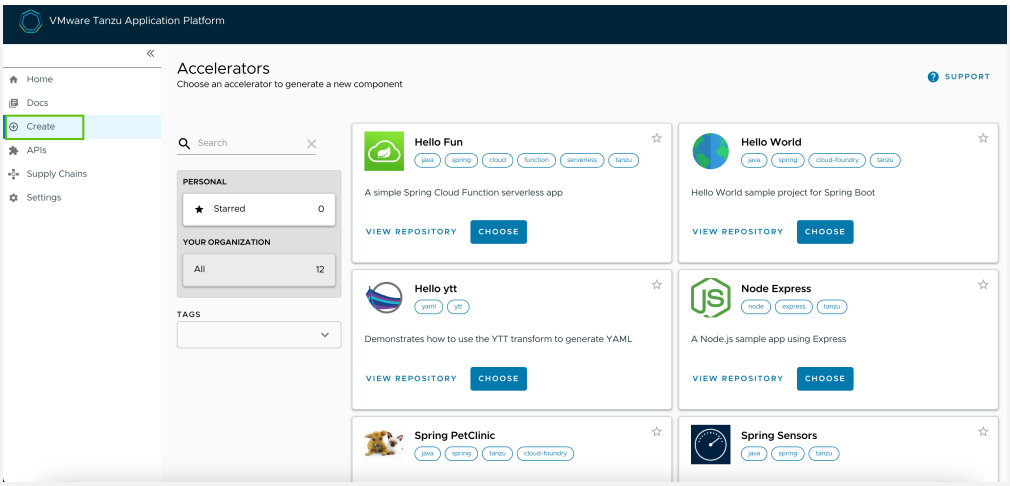
In this example, you use the `Tanzu-Java-Web-App` accelerator. You also use Tanzu Developer Portal. For information about connecting to Tanzu Developer Portal, see Access Tanzu Developer Portal.

What you will do:

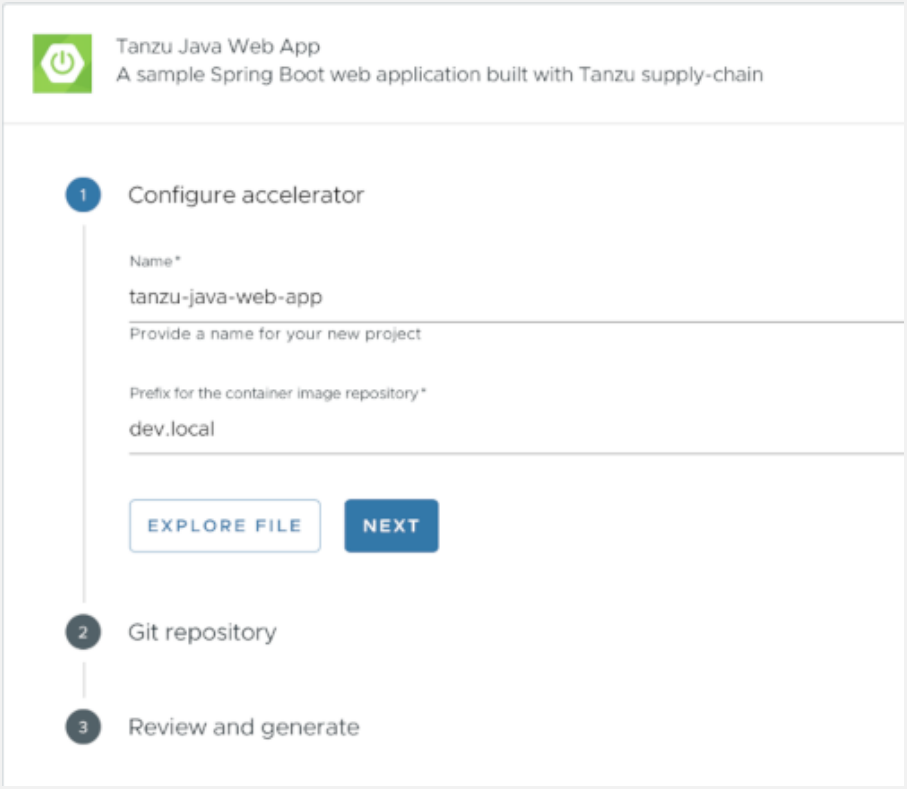
- Generate a project from an Application Accelerator.
- (Optional) Provision a new Git repository for the project.
- Upload it to your Git repository of choice.

To generate a new project using an Application Accelerator:

1. From Tanzu Developer Portal, click **Create** located on the left side of the navigation pane to see the list of available accelerators.



2. Locate the Tanzu Java Web App accelerator and click **CHOOSE**.
3. In the **Generate Accelerators** dialog box, replace the default value `dev.local` in the **prefix for container image registry** text box with the registry in the form of `SERVER-NAME/REPO-NAME`. The `SERVER-NAME/REPO-NAME` must match what was specified for `registry` as part of the installation values for `ootb_supply_chain_basic`. See the Full Profile section on Installing Tanzu Application Platform package and../install-online/install.hbs.md profiles.

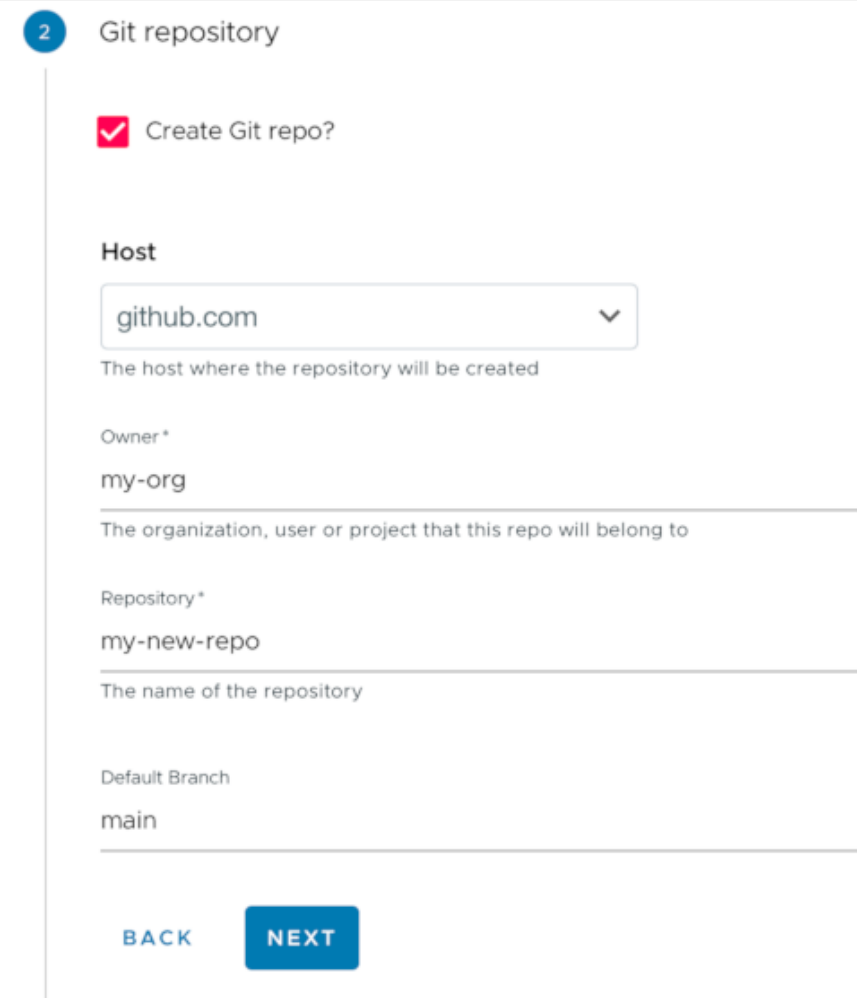


4. Click **NEXT**.
5. If your instance has optional Git repository support enabled, continue with the following sub-steps. If your instance does not support this, skip to step 5.

Note

For information about configuring optional Git repository creation and supported repositories, see Create an Application Accelerator Git repository during project creation.

1. Select the **Create Git repo?** check box.
2. Select the host Git repository provider from the **Host** drop-down menu. For example, `github.com`.
3. Populate the **Owner** and **Repository** text boxes.



4. While you are populating the form, a dialog box appears asking for permission to provision Git repositories. Follow the prompts and continue.
5. Click **NEXT**.
6. Verify the provided information, and click **GENERATE ACCELERATOR**.
7. After the Task Activity processes complete, click **DOWNLOAD ZIP FILE**.
8. After downloading the ZIP file, expand it in a workspace directory. If you did not create a Git repository in the preceding steps, follow your preferred procedure for uploading the generated project files to a Git repository for your new project.

Learn more about Application Accelerator

- For information about how to configure optional Git repository creation, see Configure in *Create an Application Accelerator Git repository during project creation*.
- For information about Application Accelerator configurations, see Configure Application Accelerator.
- For information about installing the Application Accelerator extension for Visual Studio Code, see Application Accelerator Visual Studio Code extension.
- For general accelerator troubleshooting, see Troubleshooting Application Accelerator for VMware Tanzu.

Next Steps

Now that you have generated a project that is ready for Tanzu Application Platform, learn how to quickly deploy the application on a Tanzu Application Platform cluster in Deploy an app on Tanzu Application Platform.

Deploy an app on Tanzu Application Platform

This topic guides you through deploying your first application on Tanzu Application Platform (commonly known as TAP) by using the Tanzu CLI, and optionally adding your application to the Tanzu Developer Portal software catalog.

This guide is a continuation from the previous step, Generate an application with Application Accelerator.

What you will do

- Deploy an app using the Tanzu CLI.
- View the build and runtime logs for your app.
- View the web app in your browser.
- (Optional) Add your application to Tanzu Developer Portal software catalog.

Prerequisites

Before you start, you must have:

- Completed all Getting Started prerequisites.
- Created a project. To do so, you can follow the steps in Generate an application with Application Accelerator.
- Created a Git repository during the project creation stage. If the project does not have an associated Git repository, create a repository and update the `workload.yaml` the repository URL and branch.

Deploy your application using the Tanzu CLI

Complete the following steps to deploy your application using the Tanzu CLI.

Prerequisites

Ensure that you meet the following prerequisites:

- Before you deploy your application using the Tanzu CLI, ensure that you have created a Git repository during the project creation stage.

- If the project does not have an associated Git repository, you must create one, and then update the `workload.yaml` with the repository URL and branch.

Procedure

1. Deploy the Tanzu Java Web App project that you generated in Generate an application with Application Accelerator by running the `tanzu apps workload create` command:

   ```
   tanzu apps workload create --file config/workload.yaml --namespace YOUR-NAMESPA
   CE
   ```

   Alternatively, you can create a workload using the command line:

   ```
   tanzu apps workload create tanzu-java-web-app \
   --git-repo GIT-REPO-URL \
   --git-branch main \
   --type web \
   --label app.kubernetes.io/part-of=tanzu-java-web-app \
   --label apps.tanzu.vmware.com/has-tests="true" \
   --yes \
   --namespace YOUR-NAMESPACE
   ```

   Where:

   - `GIT-REPO-URL` is the Git repository URL for where your project is stored. For example, `https://github.com/vmware-tanzu/my-tanzu-java-web-app-project`.
   - `YOUR-NAMESPACE` is the namespace where workloads are deployed. For example, `my-app-dev-namespace`. This depends on your organization's Tanzu Application Platform configuration. For more information, consult with your Tanzu Application Platform administrators.

   For more information, see Create or update a workload.

2. View the build and runtime logs for your app by running the `get` command:

   ```
   tanzu apps workload get tanzu-java-web-app --namespace YOUR-DEVELOPER-NAMESPACE
   ```

   Where `YOUR-DEVELOPER-NAMESPACE` is the namespace configured earlier.

   Note

   To watch updates in real time, prepend `watch -n1` to the `tanzu apps workload get` command to see the result update every second.

   An example of the output from an early-stage deployment looks like the following:

   ```
   Overview
       name:         tanzu-java-web-app
       type:         web
       namespace:    dev-namespace

   Source
       type:      git
       url:       https://github.com/my-organization/tanzu-java-web-app
       branch:    main

   Supply Chain
       name:    source-to-url

       NAME                READY     HEALTHY    UPDATED    RESOURCE
   ```

```
    source-provider    True      True      5s        gitrepositories.source.too
lkit.fluxcd.io/tanzu-java-web-app
    image-provider     Unknown   Unknown   5s        images.kpack.io/tanzu-java
-web-app
    config-provider    False     Unknown   8s        not found
    app-config         False     Unknown   8s        not found
    service-bindings   False     Unknown   8s        not found
    api-descriptors    False     Unknown   8s        not found
    config-writer      False     Unknown   8s        not found

Delivery
    name:   delivery-basic

    NAME              READY   HEALTHY   UPDATED   RESOURCE
    source-provider   False   False     2s        imagerepositories.source.app
s.tanzu.vmware.com/tanzu-java-web-app-delivery
    deployer          False   Unknown   5s        not found

Messages
    Workload [MissingValueAtPath]:   waiting to read value [.status.latestImag
e] from resource [images.kpack.io/tanzu-java-web-app] in namespace [dev-namespa
ce]
    Deliverable [HealthyConditionRule]:   Unable to resolve image with tag "my-
instance.azurecr.io/tap/tanzu-java-web-app-dev-namespace-bundle:0da415bc-5d79-4
d80-8ff1-0d27f42f871c" to a digest: HEAD https://my-instance.azurecr.io/v2/
    tap/tanzu-java-web-app-dev-namespace-bundle/manifests/0da415bc-5d79-4d80-8f
f1-0d27f42f871c: unexpected status code 404 Not Found (HEAD responses have no b
ody, use GET for details)

Pods
    NAME                                    READY   STATUS     RESTARTS   AGE
    tanzu-java-web-app-build-1-build-pod    0/1     Init:0/6   0          5s
```

After the workload is deployed, text similar to the following is displayed:

```
Overview
    name:        tanzu-java-web-app
    type:        web
    namespace:   dev-namespace

Source
    type:     git
    url:      https://github.com/my-organization/tanzu-java-web-app
    branch:   main

Supply Chain
    name:   source-to-url

    NAME               READY   HEALTHY   UPDATED   RESOURCE
    source-provider    True    True      5m26s     gitrepositories.source.toolk
it.fluxcd.io/tanzu-java-web-app
    image-provider     True    True      4m30s     images.kpack.io/tanzu-java-w
eb-app
    config-provider    True    True      4m24s     podintents.conventions.cart
o.run/tanzu-java-web-app
    app-config         True    True      4m24s     configmaps/tanzu-java-web-ap
p
    service-bindings   True    True      4m24s     configmaps/tanzu-java-web-ap
p-with-claims
    api-descriptors    True    True      4m24s     configmaps/tanzu-java-web-ap
p-with-api-descriptors
    config-writer      True    True      4m12s     runnables.carto.run/tanzu-ja
va-web-app-config-writer

Delivery
    name:   delivery-basic
```

```
    NAME              READY   HEALTHY   UPDATED   RESOURCE
    source-provider   True    True      3m23s     imagerepositories.source.app
s.tanzu.vmware.com/tanzu-java-web-app-delivery
    deployer          True    True      3m17s     apps.kappctrl.k14s.io/tanzu-j
ava-web-app

Messages
    No messages found.

Pods
    NAME                                          READY   STATUS      RESTARTS
AGE
    tanzu-java-web-app-build-1-build-pod          0/1     Completed   0
5m25s
    tanzu-java-web-app-config-writer-p47cg-pod    0/1     Completed   0
4m24s

Knative Services
    NAME                READY   URL
    tanzu-java-web-app  Ready   https://tanzu-java-web-app.dev-namespace.apps.
my-organization.com
```

3. After the workload is built and deployed, fetch the URL of the deployed app. The URL of the web app is in the `Knative Services` section at the bottom of the output of the `tanzu apps workload get` command:

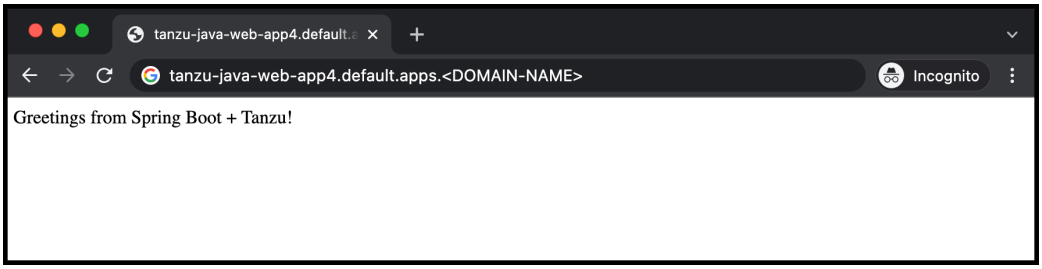
   ```
   tanzu apps workload get tanzu-java-web-app --namespace YOUR-DEVELOPER-NAMESPACE
   ```

   Where `YOUR-DEVELOPER-NAMESPACE` is the namespace configured earlier.

   The output looks similar to the following:

   ```
   Knative Services
       NAME                READY   URL
       tanzu-java-web-app  Ready   https://tanzu-java-web-app.dev-namespace.apps.
   my-organization.com
   ```

4. View the web app in your browser.

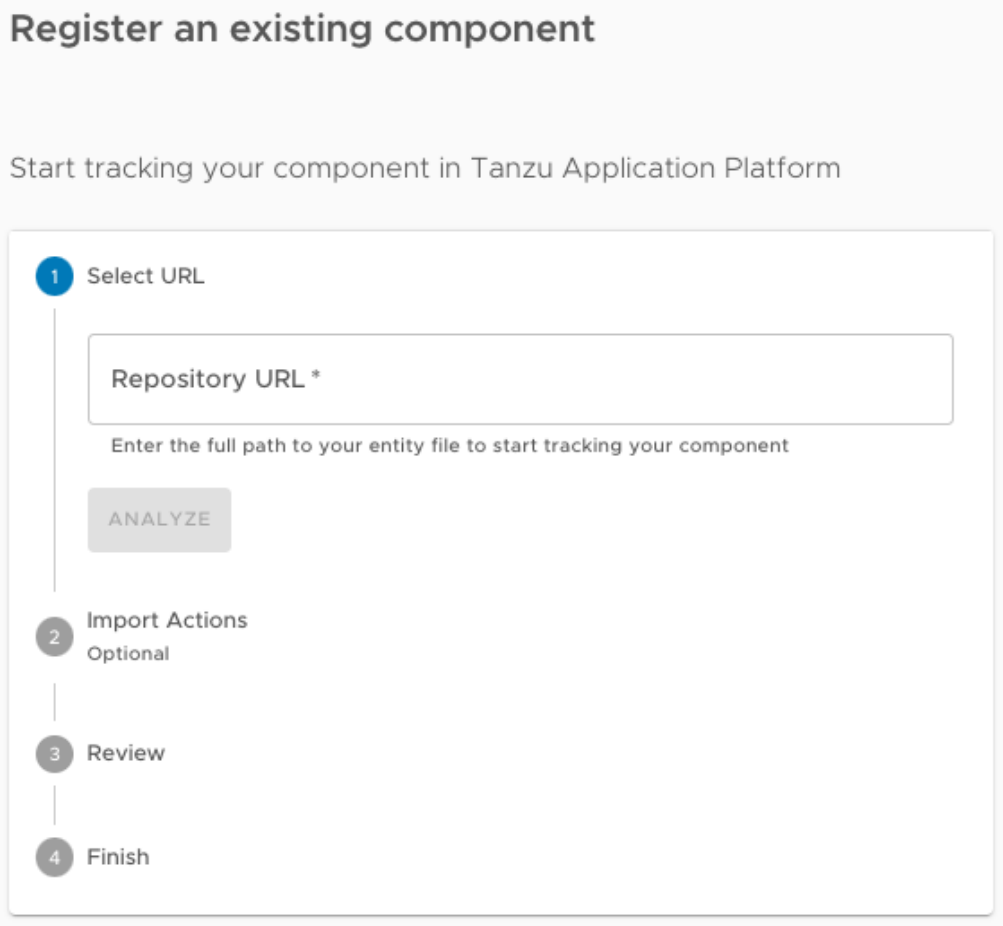


Add your application to Tanzu Developer Portal software catalog

1. Navigate to the home page of Tanzu Developer Portal and click **Home**, located on the left navigation pane.
2. Click **REGISTER ENTITY**.

   Alternatively, you can add a link for the `catalog-info.yaml` to the `tap-values.yaml` configuration file in the `tap_gui.app_config.catalog.locations` section. For more information, see Installing the Tanzu Application Platform Package and Profiles.

3. **Register an existing component** prompts you to type a repository URL. Type the link to the `catalog-info.yaml` file of the `tanzu-java-web-app` in the Git repository text box. For example, `https://github.com/USERNAME/PROJECTNAME/blob/main/catalog-info.yaml`.



4. Click **ANALYZE**.
5. Review the catalog entities to be added and click **IMPORT**.
6. Navigate back to the home page. The catalog changes and entries are visible for further inspection.

Note

If your Tanzu Developer Portal instance does not have a PostgreSQL database configured, you must re-register the `catalog-info.yaml` location after the instance is restarted or upgraded.

Next steps

Now that you have your application deployed on your Tanzu Application Platform cluster, the next step is to iterate on your application.

- If you are an IntelliJ user, see the Iterate on your new app using IntelliJ guide.
- If you are a Visual Studio user, see the Iterate on your new app using Visual Studio guide.
- If you are a VS Code user, see the Iterate on your new app using VS Code guide.

Iterate on your new app using Tanzu Developer Tools for IntelliJ

This topic guides you through starting to iterate on your first application on Tanzu Application Platform (commonly known as TAP). You deployed the app in the previous how-to Deploy your first application.

What you will do

- Prepare to iterate on your application.
    - Prepare your project to support Live Update.
    - Prepare your IDE to iterate on your application.
- Apply your application to the cluster.
- Live update your application to view code changes updating live on the cluster.
- Debug your application.
- Delete your application from the cluster.

Prepare to iterate on your application

In the previous Getting started how-to topic, Deploy your first application, you deployed your first application on Tanzu Application Platform. Now that you have developed a skeleton workload, you are ready to begin to iterate on your new application and test code changes on the cluster.

Tanzu Developer Tools for IntelliJ is VMware Tanzu's official IDE extension for IntelliJ. It helps you develop and receive fast feedback on your workloads running on the Tanzu Application Platform.

The IntelliJ extension enables live updates of your application while running on the cluster and allows you to debug your application directly on the cluster.

For information about installing the prerequisites and the Tanzu Developer Tools for IntelliJ extension, see Install Tanzu Developer Tools for IntelliJ.

Important

Use Tilt v0.30.12 or later for the sample application.

To prepare to iterate on your application, you must:

1. Prepare your project to support Live Update
2. Set up the IDE

Prepare your project to support Live Update

Tanzu Live Update uses Tilt. This requires a suitable `Tiltfile` to exist at the root of your project. Both Gradle and Maven projects are supported, but each requires a `Tiltfile` specific to that type of project.

The Tanzu Java Web App accelerator allows you to choose between Maven and Gradle and includes a `Tiltfile`. If you used the accelerator, your project is already set up correctly.

To verify your project is set up correctly, review the following requirements depending on your chosen build system.

Maven Spring Boot project requirements

If you are using Maven, your `Tiltfile` must be similar to the following:

```
SOURCE_IMAGE = os.getenv("SOURCE_IMAGE", default='your-registry.io/project/tanzu-java-
web-app-source')
LOCAL_PATH = os.getenv("LOCAL_PATH", default='.')
NAMESPACE = os.getenv("NAMESPACE", default='default')

k8s_custom_deploy(
    'tanzu-java-web-app',
    apply_cmd="tanzu apps workload apply -f config/workload.yaml --update-strategy rep
lace --debug --live-update" +
                " --local-path " + LOCAL_PATH +
                " --source-image " + SOURCE_IMAGE +
                " --namespace " + NAMESPACE +
                " --yes --output yaml",
    delete_cmd="tanzu apps workload delete -f config/workload.yaml --namespace " + NAM
ESPACE + " --yes",
    container_selector='workload',
    deps=['pom.xml', './target/classes'],
    live_update=[
      sync('./target/classes', '/workspace/BOOT-INF/classes')
    ]
)

k8s_resource('tanzu-java-web-app', port_forwards=["8080:8080"],
             extra_pod_selectors=[{'carto.run/workload-name': 'tanzu-java-web-app', 'ap
p.kubernetes.io/component': 'run'}])
```

Gradle Spring Boot project requirements

If you are using Gradle, review the following requirements:

- The `Tiltfile` looks like a Maven Tiltfile except for some key differences in the `deps=` and `live-update=` sections:

  ```
      ...
      deps=['build.gradle.kts', './build/classes/java/main', './build/resources/m
  ain'],
      live_update=[
         sync('./build/classes/java/main', '/workspace/BOOT-INF/classes'),
         sync('./build/resources/main', '/workspace/BOOT-INF/classes')
      ]
      ...
  ```

- The project must be built as an exploded JAR. This is not the default behavior for a Gradle-based build. For a typical Spring Boot Gradle project, you must deactivate the `jar` task in the `build.gradle.kts` file as follows:

  ```
  ...
  tasks.named<Jar>("jar") {
      enabled = false
  }
  ```

Set up the IDE

After verifying your project has the required `Tiltfile` and Maven or Gradle build support, you are ready to set up your development environment.

1. Open the Tanzu Java Web App as a project within your IntelliJ IDE by selecting **File** > **Open**, then selecting the Tanzu Java Web App folder and clicking **Open**. If you don't have the Tanzu Java Web App you can obtain it by following the instructions in Generate a new

project using an Application Accelerator, or from the Application Accelerator Samples GitHub page.

2. Confirm that your current Kubernetes context contains a default namespace. The Tanzu Panel, found by clicking **Tanzu Panel** at the bottom-left of the IntelliJ window, uses the default namespace associated with your current Kubernetes context to populate the workloads from the cluster.

    1. Open the Terminal by clicking **View** > **Terminal**.
    2. Ensure that your current context has a default namespace by running:

        ```
        kubectl config get-contexts
        ```

        This command returns a list of all of your Kubernetes contexts with an asterisk (`*`) in front of your current context. Verify that your current context has a namespace in the namespace column.

    3. If your current context does not have a namespace in the namespace column, run:

        ```
        kubectl config set-context --current --namespace=YOUR-DEVELOPER-NAMESPACE
        ```

        Where `YOUR-DEVELOPER-NAMESPACE` is the namespace value you want to assign to your current Kubernetes context.

You are now ready to iterate on your application.

Apply your application to the cluster

Apply the workload to see your application running on the cluster:

1. In the **Project** tab in IntelliJ, right-click any file under the application name `tanzu-java-web-app` and click **Tanzu** > **Apply Workload**.
2. In the dialog box enter your **Source Image**, **Local Path**, and optionally a **Namespace**.

    1. Customize the installation of Local Source Proxy.

        If you don't have Local Source Proxy configured, you can use the source image parameter instead. The source image value tells the Tanzu Developer Tools for IntelliJ extension where to publish the container image with your non-compiled source code, and what to name that image. The image must be published to a container image registry where you have write access. For example, `gcr.io/myteam/tanzu-java-web-app-source`.

        Note

        See the documentation for the registry you're using to find out which steps are necessary to authenticate and gain push access.

        For example, if you use Docker, see the Docker documentation, or if you use Harbor, see the Harbor documentation.

    2. In the **Local Path** text box, provide the path to the directory containing the Tanzu Java Web App. The current directory is the default.

        The local path value tells the Tanzu Developer Tools for IntelliJ extension which directory on your local file system to bring into the source image. For example, dot (`.`) uses the working directory, or you can specify a full file path.

3. (Optional) In the **Namespace** text box, provide the namespace to be associated with the workload on the cluster. If you followed the steps to Prepare your IDE to iterate on your application earlier, you do not need to enter a namespace because IntelliJ uses the namespace you associated with your context.
4. Click the **OK** button.

The `apply workload` command runs, which opens a terminal and shows you the output of the command. The `apply workload` command can take a few minutes to deploy your application onto the cluster.

You can also use the **Tanzu Panel** to monitor your application as it's being deployed to the cluster. The **Tanzu Panel** shows information about the workloads in the namespace associated with your current Kubernetes context. On the left side, it shows the workloads in the namespace. In the center, it shows the details of the Kubernetes resources for the running workloads.

Enable Live Update for your application

Live Update allows you to save changes to your code and see those changes reflected within seconds in the workload running on the cluster.

To enable Live Update for your application:

1. Create a Run Configuration.
    1. In IntelliJ, select the **Edit Run/Debug configurations** drop-down menu at the top-right corner. Alternatively, navigate to **Run** > **Edit Configurations**.
    2. Select **Tanzu Live Update**.
    3. Select **Add new run configuration**, or click the plus icon at the top of the list.
    4. Give your new run configuration a name, for example, `Tanzu Live Update - tanzu-java-web-app`.
    5. In the **Tiltfile Path** text box, provide the path to the `Tiltfile` in the Tanzu Java Web App project directory.
    6. Select the folder icon on the right-side of the text box, go to the `Tanzu Java Web App` directory, select the `Tiltfile`, and click **Open**. The `Tiltfile` facilitates Live Update using Tilt.
    7. In the **Local Path** text box, provide the path to the directory containing the Tanzu Java Web App.

        The local path value tells the Tanzu Developer Tools for IntelliJ extension which directory on your local file system to bring into the source image.

        For example, `/Users/developer/Documents/tanzu-java-web-app`.
    8. Customize the installation of Local Source Proxy.

        If you don't have Local Source Proxy configured, you can use the source image parameter instead. The source image value tells the Tanzu Developer Tools for IntelliJ extension where to publish the container image with your non-compiled source code, and what to name that image. The image must be published to a container image registry where you have write access. For example, `gcr.io/myteam/tanzu-java-web-app-source`.

        Note

        See the documentation for the registry you're using to find out which steps are necessary to authenticate and gain push access.

For example, if you use Docker, see the Docker documentation, or if you use Harbor, see the Harbor documentation.

9. Click **Apply**, and then click the **OK** button.

2. Begin Live Updating the application on the cluster by doing one of the following:
    - In the Project tab of IntelliJ, right-click the `Tiltfile` file under the application name `tanzu-java-web-app` and click **Run 'Tanzu Live Update - tanzu-java-web-app'**.
    - Alternatively, click the **Edit Run/Debug configurations** drop-down menu in the top-right corner, select **Tanzu Live Update - tanzu-java-web-app**, and then click the green play button to the right of the **Edit Run/Debug configurations** drop-down menu.

    The **Run** tab opens and displays the output from Tanzu Application Platform and from Tilt indicating that the container is being built and deployed.

    On the **Tanzu Panel** tab, the status of Live Update is reflected under the `tanzu-java-web-app` workload entry. Live update can take up to three minutes while the workload deploys and the Knative service becomes available.

    Note

    Depending on the type of cluster you use, you might see an error similar to the following:

    ```
    ERROR: Stop! cluster-name might be production. If you're sure you want to deploy there, add: allow_k8s_contexts('cluster-name') to your Tiltfile. Otherwise, switch k8scontexts and restart Tilt.
    ```

    Follow the instructions and add the line, `allow_k8s_contexts('cluster-name')` to your `Tiltfile`.

3. When the Live Update task in the **Run** tab is successful, it resolves to `Live Update Started`. Use the hyperlink at the top of the Run output following the words **Tilt started on** to view your application in your browser.
4. In the IDE, make a change to the source code. For example, in `HelloController.java`, edit the string returned to say `Hello!` and save.
5. (Optional) Build your project by clicking **Build** > **Build Project** if you do not have **Build project automatically** activated under **Preferences** > **Build, Execution, Deployment** > **Compiler**.
6. The container is updated when the logs stop streaming. Navigate to your browser and refresh the page.
7. View the changes to your workload running on the cluster.
8. Either continue making changes, or stop the Live Update process when finished. To stop Live Update navigate to the **Run** tab at the bottom left of the IntelliJ window and click the red stop icon on the left side of the screen.

Debug your application

Debug the cluster either on the application or in your local environment.

To debug the cluster:

1. Set a breakpoint in your code. For example, in `HelloController.java`, set a breakpoint on the line returning text.
2. Create a Run Configuration.
    1. In IntelliJ, select the **Edit Run/Debug configurations** drop-down menu at the top-right corner. Alternatively, navigate to **Run** > **Edit Configurations**.
    2. Select **Tanzu Debug Workload**.
    3. Select **Add new run configuration**, or click the plus icon at the top of the list.
    4. Give your new run configuration a name, for example, `Tanzu Debug Workload - tanzu-java-web-app`.
    5. In the **Workload File Path** text box, provide the path to the `workload.yaml` file in the Tanzu Java Web App project directory located at **Config** > **workload.yaml**.
    6. Select the folder icon on the right-side of the text box, navigate to the Tanzu Java Web App directory, select the `workload.yaml` file and click the **Open** button. The `workload.yaml` provides configuration instructions about your application to the Tanzu Application Platform.
    7. In the **Local Path** text box, provide the path to the directory containing the Tanzu Java Web App.

        The local path value tells the Tanzu Developer Tools for IntelliJ extension which directory on your local file system to bring into the source image. For example, `/Users/developer/Documents/tanzu-java-web-app`.
    8. Customize the installation of Local Source Proxy.

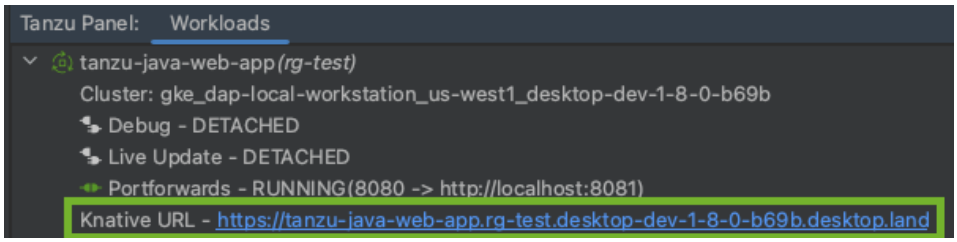
        If you don't have Local Source Proxy configured, you can use the source image parameter instead. The source image value tells the Tanzu Developer Tools for IntelliJ extension where to publish the container image with your non-compiled source code, and what to name that image. The image must be published to a container image registry where you have write access. For example, `gcr.io/myteam/tanzu-java-web-app-source`.

        Note

        See the documentation for the registry you're using to find out which steps are necessary to authenticate and gain push access.

        For example, if you use Docker, see the Docker documentation, or if you use Harbor, see the Harbor documentation.
    9. (Optional) In the **Namespace** text box, provide the namespace to be associated with the workload on the cluster. If you followed the steps to Prepare your IDE to iterate on your application, you do not need to enter a namespace because IntelliJ uses the namespace you associated with your context.
    10. Click **Apply**, and then click **OK**.
3. Apply your application to the cluster.
4. Obtain the URL for your workload by doing one of the following:
    - **If your app deploys a Knative URL:** Click the URL from the Workloads panel.



    - **If your app does not deploy a Knative URL but exposes an app port:** Access your app through a `portforward`. For instructions, see Use a portforward to access an application locally.

5. In the Project tab of IntelliJ, right-click the `workload.yaml` file under the application name `tanzu-java-web-app` and select **Run 'Tanzu Debug Workload - tanzu-java-web-app'** to begin debugging the application on the cluster.
    1. Alternatively, select the **Edit Run/Debug configurations** drop-down menu in the top-right corner, select **Tanzu Debug Workload - tanzu-java-web-app**, and then click the green debug button to the right of the **Edit Run/Debug configurations** drop-down menu.
6. The Debug tab opens and displays a message that it has connected.
7. In your web browser, reload your workload. IntelliJ opens to show your breakpoint.
8. You can now use the resume program action, or stop debugging, in the **Debug** tab.

Delete your application from the cluster

You can use the delete action to remove your application from the cluster as follows:

1. In the **Project** tab, right-click any file under the application name `tanzu-java-web-app` and select **Tanzu** > **Delete Workload**.
2. Alternatively, right-click `tanzu-java-web-app` in the **TANZU WORKLOADS** panel and select **Delete Workload**.
3. In the confirmation dialog box that appears, click **OK** to delete the application from the cluster.

Next steps

- Consume services on Tanzu Application Platform

Iterate on your new app using Tanzu Developer Tools for Visual Studio

This topic guides you through starting to iterate on your first application on Tanzu Application Platform (commonly known as TAP). You deployed the app in the previous how-to Deploy your first application.

What you will do

- Prepare to iterate on your application.
    - Prepare your project to support Live Update.
    - Prepare your IDE to iterate on your application.
- Apply your application to the cluster.

- Live update your application to view code changes updating live on the cluster.
- Debug your application.
- Monitor your running application on the Application Live View UI.
- Delete your application from the cluster.

Prepare to iterate on your application

In the previous Getting started how-to topic, Deploy your first application, you deployed your first application on Tanzu Application Platform. Now that you have developed a skeleton workload, you are ready to begin to iterate on your new application and test code changes on the cluster.

Tanzu Developer Tools for Visual Studio is VMware Tanzu's official IDE extension for Visual Studio. It helps you develop and receive fast feedback on your workloads running on the Tanzu Application Platform.

The Visual Studio extension enables live updates of your application while running on the cluster and allows you to debug your application directly on the cluster.

For information about installing the prerequisites and the Tanzu Developer Tools for Visual Studio extension, see Install Tanzu Developer Tools for Visual Studio.

Important

Use Tilt v0.30.12 or later for the sample application.

To prepare to iterate on your application, you must:

1. Prepare your project to support Live Update
2. Set up the IDE

Prepare your project to support Live Update

Tanzu Live Update uses Tilt. This requires a suitable `Tiltfile` to exist at the root of your project.

Your `Tiltfile` must be similar to the following:

```
SOURCE_IMAGE = os.getenv("SOURCE_IMAGE", default='your-registry.io/project/csharp-weat
herforecast-source')
LOCAL_PATH = os.getenv("LOCAL_PATH", default='.')
NAMESPACE = os.getenv("NAMESPACE", default='default')
NAME = os.getenv("NAME", default='sample-app')

k8s_custom_deploy(
    NAME,
    apply_cmd="tanzu apps workload apply -f config/workload.yaml --update-strategy rep
lace --debug --live-update" +
                " --local-path " + LOCAL_PATH +
                " --namespace " + NAMESPACE +
                " --yes --output yaml",
    delete_cmd="tanzu apps workload delete " + NAME + " --namespace " + NAMESPACE + "
--yes",
    deps=['./bin'],
    container_selector='workload',
    live_update=[
      sync('./bin/Debug/net6.0', '/workspace')
    ]
)

k8s_resource('tanzu-java-web-app', port_forwards=["8080:8080"],
```

```
            extra_pod_selectors=[{'carto.run/workload-name': 'sample-app', 'app.kubern
etes.io/component': 'run'}])
```

Set up the IDE

After verifying your project has the required `Tiltfile`, you are ready to set up your development environment.

1. Open the Weather Forecast solution in Visual Studio by selecting **File** > **Open** > **Project/Solution...**. If you don't have the Weather Forecast app you can obtain it by following the instructions in Generate an application with Application Accelerator, or from the Application Accelerator Samples GitHub page.

You are now ready to iterate on your application.

Apply your application to the cluster

Apply the workload to see your application running on the cluster:

1. In **Solution Explorer**, right-click any file under the application name and click **Tanzu** > **Apply Workload**.
2. In the dialog box, enter the following:
    1. In the **Local Path** text box, provide the path to the directory containing the Weather Forecast app. The current directory is the default.

        The local path value tells the Tanzu Developer Tools for Visual Studio extension which directory on your local file system to bring into the source image. For example, dot (`.`) uses the working directory, or you can specify a full file path.
    2. In the **Namespace** text box, provide the namespace to be associated with the workload on the cluster.
    3. (Optional) In the **Source Image** text box, provide the destination image repository to publish the image containing your workload source code.

        The source image value tells the Tanzu Developer Tools for Visual Studio extension where to publish the container image with your uncompiled source code, and what to name that image. The image must be published to a container image registry where you have write (push) access. For example, `gcr.io/myteam/weather-forecast-source`.

        Note

        See the documentation for the registry you're using to find out which steps are necessary to authenticate and gain push access.

        For example, if you use Docker, see the Docker documentation, or if you use Harbor, see the Harbor documentation.
    4. Click the **OK** button.

The `apply workload` command runs and opens a an output window in which you can monitor the output of the command. The `apply workload` command can take a few minutes to deploy your application onto the cluster.

Enable Live Update for your application

Live Update allows you to save changes to your code and see those changes reflected within seconds in the workload running on the cluster.

To enable Live Update for your application:

1. In **Solution Explorer**, right-click any file under the application name and click **Tanzu** > **Start Live Update**.
2. Live update can take up to three minutes while the workload deploys and the Knative service becomes available.

   > **Note**
   >
   > Depending on the type of cluster you use, you might see an error similar to the following:
   >
   > ```
   > ERROR: Stop! cluster-name might be production. If you're sure you want to deploy there, add: allow_k8s_contexts('cluster-name') to your Tiltfile. Otherwise, switch k8scontexts and restart Tilt.
   > ```
   >
   > Follow the instructions and add the line, `allow_k8s_contexts('cluster-name')` to your `Tiltfile`.

3. In the IDE, make a change to the source code.
4. Build your project.
5. The container is updated when the logs stop streaming. Go to your browser and refresh the page.
6. View the changes to your workload running on the cluster.
7. Either continue making changes, or stop the Live Update process when finished. To stop Live Update, in **Solution Explorer**, right-click any file under the application name and click **Tanzu** > **Stop Live Update**.

Debug your application

Debug the cluster either on the application or in your local environment.

To start debugging the cluster:

1. Set a breakpoint in your code.
2. Apply your application to the cluster.
3. In **Solution Explorer**, right-click any file under the application name and click **Tanzu** > **Debug Workload**.

To stop debugging the cluster:

1. In main, click **Debug** > **Detach All**

Delete your application from the cluster

You can use the delete action to remove your application from the cluster as follows:

1. In **Solution Explorer**, right-click any file under the application name and click **Tanzu** > **Delete Workload**.
2. In the confirmation dialog box that appears, click **OK** to delete the application from the cluster.

Next steps

- Consume services on Tanzu Application Platform

Iterate on your new app using Tanzu Developer Tools for VS Code

This topic guides you through starting to iterate on your first application on Tanzu Application Platform (commonly known as TAP). You deployed the app in the previous how-to Deploy your first application.

What you will do

- Prepare to iterate on your application.
  - Prepare your project to support Live Update.
  - Prepare your IDE to iterate on your application.
- Apply your application to the cluster.
- Live update your application to view code changes updating live on the cluster.
- Debug your application.
- Monitor your running application on the Application Live View UI.
- Delete your application from the cluster.

Prepare to iterate on your application

In the previous Getting started how-to topic, Deploy your first application, you deployed your first application on Tanzu Application Platform. Now that you have developed a skeleton workload, you are ready to begin to iterate on your new application and test code changes on the cluster.

Tanzu Developer Tools for VS Code is VMware Tanzu's official IDE extension for VS Code. It helps you develop and receive fast feedback on your workloads running on the Tanzu Application Platform.

The VS Code extension enables live updates of your application while running on the cluster and allows you to debug your application directly on the cluster. For information about installing the prerequisites and the Tanzu Developer Tools for VS Code extension, see Install Tanzu Developer Tools for your VS Code.

Important

Use Tilt v0.30.12 or a later version for the sample application.

To prepare to iterate on your application, you must:

1. Prepare your project to support Live Update
2. Set up the IDE

Prepare your project to support Live Update

Tanzu Live Update uses Tilt. This requires a suitable `Tiltfile` to exist at the root of your project. Both Gradle and Maven projects are supported, but each requires a `Tiltfile` specific to that type of project.

The Tanzu Java Web App accelerator allows you to choose between Maven and Gradle and includes a `Tiltfile`. If you used the accelerator, your project is already set up correctly.

To verify your project is set up correctly, review the following requirements depending on your chosen build system.

Maven Spring Boot project requirements

If you are using Maven, your `Tiltfile` must be similar to the following:

```
SOURCE_IMAGE = os.getenv("SOURCE_IMAGE", default='your-registry.io/project/tanzu-java-
web-app-source')
LOCAL_PATH = os.getenv("LOCAL_PATH", default='.')
NAMESPACE = os.getenv("NAMESPACE", default='default')

k8s_custom_deploy(
    'tanzu-java-web-app',
    apply_cmd="tanzu apps workload apply -f config/workload.yaml --update-strategy rep
lace --debug --live-update" +
                " --local-path " + LOCAL_PATH +
                " --source-image " + SOURCE_IMAGE +
                " --namespace " + NAMESPACE +
                " --yes --output yaml",
    delete_cmd="tanzu apps workload delete -f config/workload.yaml --namespace " + NAM
ESPACE + " --yes",
    container_selector='workload',
    deps=['pom.xml', './target/classes'],
    live_update=[
      sync('./target/classes', '/workspace/BOOT-INF/classes')
    ]
)

k8s_resource('tanzu-java-web-app', port_forwards=["8080:8080"],
             extra_pod_selectors=[{'carto.run/workload-name': 'tanzu-java-web-app', 'ap
p.kubernetes.io/component': 'run'}])
```

Gradle Spring Boot project requirements

If you are using Gradle, review the following requirements:

- The `Tiltfile` looks like a Maven Tiltfile except for some key differences in the `deps=` and `live-update=` sections:

  ```
      ...
      deps=['build.gradle.kts', './bin/main'],
      live_update=[
          sync('./bin/main', '/workspace/BOOT-INF/classes')
      ]
      ...
  ```

- The project must be built as an exploded JAR. This is not the default behavior for a Gradle-based build. For a typical Spring Boot Gradle project, you must deactivate the `jar` task in the `build.gradle.kts` file as follows:

  ```
  ...
  tasks.named<Jar>("jar") {
      enabled = false
  }
  ```

Set up the IDE

After verifying your project has the required `Tiltfile` and Maven or Gradle build support, you are ready to set up your development environment.

1. Open the Tanzu Java Web App as a project within your VS Code IDE by clicking **File** > **Open Folder**, select the Tanzu Java Web App folder and click **Open**.

   If you don't have the Tanzu Java Web App you can obtain it by following the instructions in Generate a new project using an Application Accelerator, or from the Application Accelerator Samples GitHub page.

2. To ensure that your extension assists you with iterating on the correct project, configure its settings as follows:

   1. In Visual Studio Code, navigate to **Preferences** > **Settings** > **Extensions** > **Tanzu Developer Tools**.
   2. In the **Local Path** text box, provide the path to the directory containing the Tanzu Java Web App. The current directory is the default.

      The local path value tells the Tanzu Developer Tools for VS Code extension which directory on your local file system to bring into the source image. For example, dot (`.`) uses the working directory, or you can specify a full file path.

   3. Customize the installation of Local Source Proxy.

      If you don't have Local Source Proxy configured, you can use the source image parameter instead. The source image value tells the Tanzu Developer Tools for VS Code extension where to publish the container image with your non-compiled source code, and what to name that image. The image must be published to a container image registry where you have write access. For example, `gcr.io/myteam/tanzu-java-web-app-source`.

      Note

      See the documentation for the registry you're using to find out which steps are necessary to authenticate and gain push access.

      For example, if you use Docker, see the Docker documentation, or if you use Harbor, see the Harbor documentation.

      For troubleshooting failed registry authentication, see Troubleshoot using Tanzu Application Platform

3. Confirm that your current Kubernetes context has a namespace associated with it. The **TANZU WORKLOADS** section of the **Explorer** view in the left Side Bar uses the namespace associated with your current Kubernetes context to populate the workloads from the cluster.

   1. Open the Terminal by clicking **View** > **Terminal**.
   2. Ensure your current context has a default namespace by running:

      ```
      kubectl config get-contexts
      ```

      This command returns a list of all of your Kubernetes contexts with an asterisk (`*`) in front of your current context. Verify that your current context has a namespace in the namespace column.

   3. If your current context does not have a namespace in the namespace column, run:

```
kubectl config set-context --current --namespace=YOUR-DEVELOPER-NAMESPACE
```

Where `YOUR-DEVELOPER-NAMESPACE` is the namespace value you want to assign to your current Kubernetes context.

You are now ready to iterate on your application.

Apply your application to the cluster

Apply the workload to see your application running on the cluster by doing one of the following:

- In the **Explorer** view in the left Side Bar, right-click any file under the application name `tanzu-java-web-app` and click **Tanzu: Apply Workload** to begin applying the workload to the cluster.
- Alternatively, use the Command Palette, ⇧⌘P on Mac and Ctrl+Shift+P on Windows or **View** > **Command Palette**, to run the `Tanzu: Apply Workload` command.

The `apply workload` command runs, which opens a terminal and shows you the output of the workload apply.

You can also monitor your application as it's being deployed to the cluster using the **TANZU ACTIVITY** tab in the Panel at the bottom of VS Code. The **TANZU ACTIVITY** tab shows the details of the Kubernetes resources for the workloads running in the namespace associated with your current Kubernetes context.

To view the **TANZU ACTIVITY** tab, open the Panel at the bottom of VS Code (**View** > **Appearance** > **Panel**) and then click the **TANZU ACTIVITY** tab. The apply workload command can take a few minutes to deploy your application onto the cluster. After complete, you can see the workload running in the **TANZU WORKLOADS** section of the **Explorer** view in the left Side Bar.

Enable Live Update for your application

Live Update allows you to save changes to your code and see those changes reflected within seconds in the workload running on the cluster.

To enable Live Update for your application:

1. To begin Live Updating the workload on the cluster, do one of the following:
    - In the **Explorer** view in the left Side Bar, right-click any file under the application name `tanzu-java-web-app` and click `Tanzu: Live Update Start`.
    - Right-click the `tanzu-java-web-app` in the **TANZU WORKLOADS** section of the **Explorer** view and click `Tanzu: Live Update Start`.
    - From the Command Palette, ⇧⌘P on Mac and Ctrl+Shift+P on Windows, type in and select `Tanzu: Live Update Start`.

    You can view output from Tanzu Application Platform indicating that the container is being built and deployed.

    The status of Live Update is reflected in the **TANZU WORKLOADS** view under the `tanzu-java-web-app` workload entry. You can also see `Live Update starting...` in the status bar at the bottom right. Live update can take up to three minutes while the workload deploys and the Knative service becomes available.

    Note

    Depending on the type of cluster you use, you might see an error similar to the following:

> `ERROR: Stop! cluster-name might be production. If you're sure you want to deploy there, add: allow_k8s_contexts('cluster-name') to your Tiltfile. Otherwise, switch k8scontexts and restart Tilt.`
>
> Follow the instructions and add the line, `allow_k8s_contexts('cluster-name')` to your `Tiltfile`.

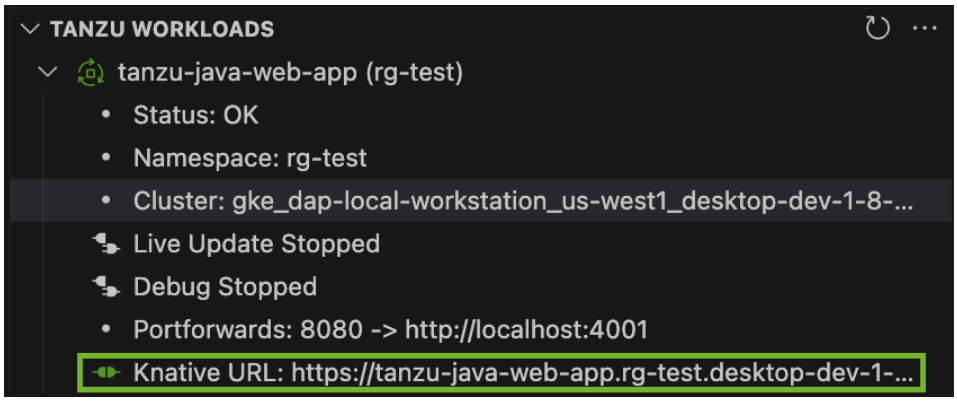
2. When the Live Update status in the **TANZU WORKLOADS** view changes from `Live Update Stopped` to `Live Update Running`, navigate to `http://localhost:8080` in your browser to view your running application.
3. In the IDE, make a change to the source code. For example, in `HelloController.java`, edit the string returned to say `Hello!`, and save.
4. The container is updated when the logs stop streaming. Go to your browser and refresh the page.
5. View the changes to the workload running on the cluster.
6. Either continue making changes, or stop the Live Update process when finished. To stop Live Update, open the Terminal by navigating to **View** > **Terminal**, and click the trash can icon that appears when you place your hover over the **tilt: up - tanzu-java-web-app** process, or select the process and use hot key ⌘+Backspace.

Debug your application

Debug your application in a production-like environment by debugging on your Kubernetes cluster.

To debug the cluster:

1. Set a breakpoint in your code. For example, in `HelloController.java`, set a breakpoint on the line returning text.
2. Apply your application to the cluster.
3. Access your app in your browser by doing one of the following:
   - **If your app deploys a Knative URL:** Click the URL from the Workloads panel.



   - **If your app does not deploy a Knative URL but exposes an app port:** Access your app through a `portforward`. For instructions, see Use a `portforward` to access an application locally.
4. Begin debugging the workload on the cluster by doing one of the following:
   - In the **Explorer** view in the left Side Bar, right-click any file under the application name `tanzu-java-web-app` and click **Tanzu: Java Debug Start**.
   - Alternatively, right-click the `tanzu-java-web-app` in the **TANZU WORKLOADS** view and click **Tanzu: Java Debug Start**.

5. In a few moments, debugging is enabled on the workload. The **Deploy and Connect** task completes and the debug actions are made available to you in the debug overlay, indicating that the debugger has attached.

   The **TANZU WORKLOADS** view shows **Debug Running** under the `tanzu-java-web-app` workload.

6. In your web browser, reload your workload. VS Code opens to show your breakpoint.
7. You can now continue the program, or stop debugging, using the debug controls overlay.

Monitor your running application

Inspect the runtime characteristics of your running application using the Application Live View UI to monitor:

- Resource consumption
- Java Virtual Machine (JVM) status
- Incoming traffic
- Change log level

You can also troubleshoot environment variables and fine-tune the running application.

Use the following steps to diagnose Spring Boot-based applications by using Application Live View:

1. Confirm that the Application Live View components are installed. For instructions, see Install Application Live View.
2. Access the Application Live View UI plug-in in Tanzu Developer Portal. For instructions, see Entry point to Application Live View plug-in.
3. Select your running application to view the diagnostic options and inside the application. For more information, see Application Live View features.

Delete your application from the cluster

You can use the delete action to remove your application from the cluster by doing one of the following:

- In the **Explorer** view in the left Side Bar, right-click any file under the application name `tanzu-java-web-app` and click **Tanzu: Delete Workload** to delete the workload from the cluster.
- Alternatively, right-click the `tanzu-java-web-app` in the **TANZU WORKLOADS** view and click **Tanzu: Delete Workload**.

Next steps

- Consume services on Tanzu Application Platform

Claim services on Tanzu Application Platform

This topic for application operators guides you through claiming a service instance and therefore making credentials available to workloads within your namespace. The topic uses RabbitMQ as an example, but the process is the same regardless of the service you want to consume.

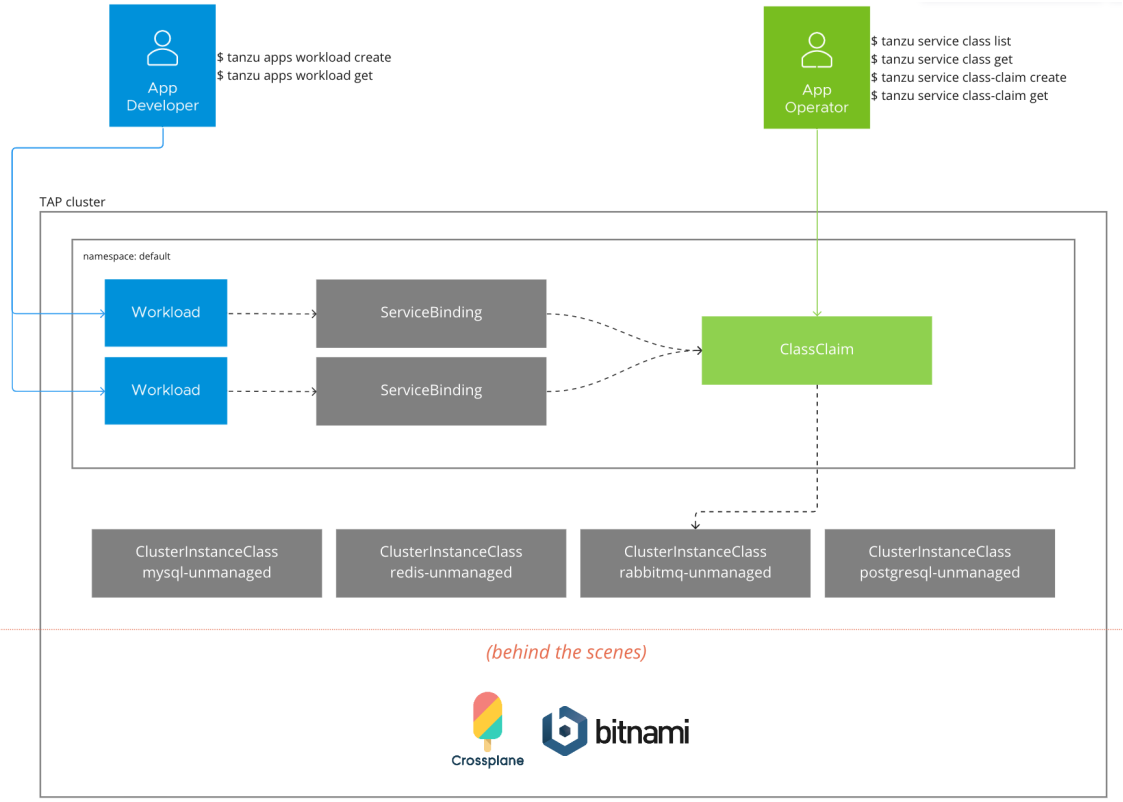
You will use the `tanzu service` CLI plug-in and will learn about classes, claims, and bindings.

What you will do

- Discover the range of services available to you
- Create a claim for an instance of one of the services

Overview

The following diagram depicts a summary of what this tutorial covers.



Bear the following observations in mind as you work through this guide:

1. There are a set of four service classes preinstalled on the cluster.
2. Service operators do not need to configure or setup these four services.
3. The life cycle of a service binding is implicitly tied to the life cycle of a workload, and is managed by the application developer.
4. The life cycles of claims are explicitly managed by the application operator.
5. The diagram and tutorial in this guide are predominantly focused on the application operator, therefore the inner workings of how service instances are provisioned are not in the diagram and are labeled as “behind the scenes”.

Prerequisites

Before following this tutorial, an application operator must:

1. Have access to a cluster with Tanzu Application Platform installed.
2. Have the Tanzu CLI and the corresponding plug-ins.
3. Have access to the `default` namespace which has been set up to use installed packages. For more information, see Set up developer namespaces to use your installed packages.

Discover available services

This section covers using `tanzu service class list` and `tanzu service class get` to find information about the classes of services.

- To discover the range of available services, run the `tanzu service class list` command:

  ```
  tanzu service class list
  ```

  Expected output:

  ```
    NAME                  DESCRIPTION
    mysql-unmanaged       MySQL by Bitnami
    postgresql-unmanaged  PostgreSQL by Bitnami
    rabbitmq-unmanaged    RabbitMQ by Bitnami
    redis-unmanaged       Redis by Bitnami
  ```

  The output lists four classes that cover a range of services: MySQL, PostgreSQL, RabbitMQ and Redis. This is the default set of services that come preconfigured with Tanzu Application Platform. They are backed by Bitnami Helm charts that run on the Tanzu Application Platform cluster. You can consider these to be unmanaged services with no guarantees of service provided.

- To see more detailed information for a class, run the `tanzu service class get` command:

  ```
  tanzu service class get rabbitmq-unmanaged
  ```

  Expected output:

  ```
    NAME:           rabbitmq-unmanaged
    DESCRIPTION:    RabbitMQ by Bitnami
    READY:          true

    PARAMETERS:
      KEY         DESCRIPTION
  TYPE     DEFAULT  REQUIRED
      replicas    The desired number of replicas forming the cluster
  integer  1        false
      storageGB   The desired storage capacity of a single replica, in Gigabytes.
  integer  1        false
  ```

  The `PARAMETERS` section is of particular interest because it lists the range of configuration options available to you when creating a claim for the given class.

Create a claim for a service instance

This section covers using `tanzu service class-claim create` to create a claim for an instance of a class and using `tanzu service class-claim get` to get detailed information about the status of the claim.

- To create a claim for an instance of a class, run the `tanzu service class-claim create` command:

  ```
  tanzu service class-claim create rabbitmq-1 --class rabbitmq-unmanaged --parame
  ter storageGB=3
  ```

  In this example, you create a claim for the `rabbitmq-unmanaged` class and pass a parameter to the command to set the storage capacity of the resulting instance to 3 Gigabytes, rather than using the default 1 Gigabyte.

Expected output:

```
  Creating claim 'rabbitmq-1' in namespace 'default'.
```

- To get detailed information about the claim, run the `tanzu service class-claim get` command:

  ```
  tanzu service class-claim get rabbitmq-1
  ```

  Expected output:

  ```
    Name: rabbitmq-1
    Namespace: default
    Claim Reference: services.apps.tanzu.vmware.com/v1alpha1:ClassClaim:rabbitmq-
  1
    Class Reference:
      Name: rabbitmq-unmanaged
    Parameters:
      storageGB: 3
    Status:
      Ready: True
      Claimed Resource:
        Name: b5982046-a1e9-40cf-8282-00fe67a2f868
        Namespace: default
        Group:
        Version: v1
        Kind: Secret
  ```

  It might take a moment or two for the claim to report `Ready: True`.

In the background, the creation of the claim triggers the on-demand creation of a Helm release of the Bitnami RabbitMQ Helm chart. Credentials and connectivity information required to connect to the RabbitMQ cluster are formatted according to the Service Binding Specification for Kubernetes and stored in a `Secret` in your namespace.

As an application operator you don't need to know what's happening in the background. Tanzu Application Platform promotes a strong separation of concerns between service operators, who are responsible for managing service instances for the platform, and application operators, who want to use those service instances with their application workloads. The class and claims abstractions enable that separation of concerns. Application operators create claims and service operators help to fulfil them.

Now that you have a claim for a RabbitMQ service instance, you can now follow instructions to Consume services on Tanzu Application Platform.

Learn more

To learn more about working with services on Tanzu Application Platform, see the Services Toolkit component documentation:

- Tutorials
- How-to guides
- Explanations
- Reference material

Next steps

Now that you completed the Getting started guides, learn about:

- Multicluster Tanzu Application Platform

Consume services on Tanzu Application Platform

This topic for application developers guides you through deploying two application workloads and configuring them to communicate using a service instance. The topic uses RabbitMQ as an example, but the process is the same regardless of the service you want to consume.

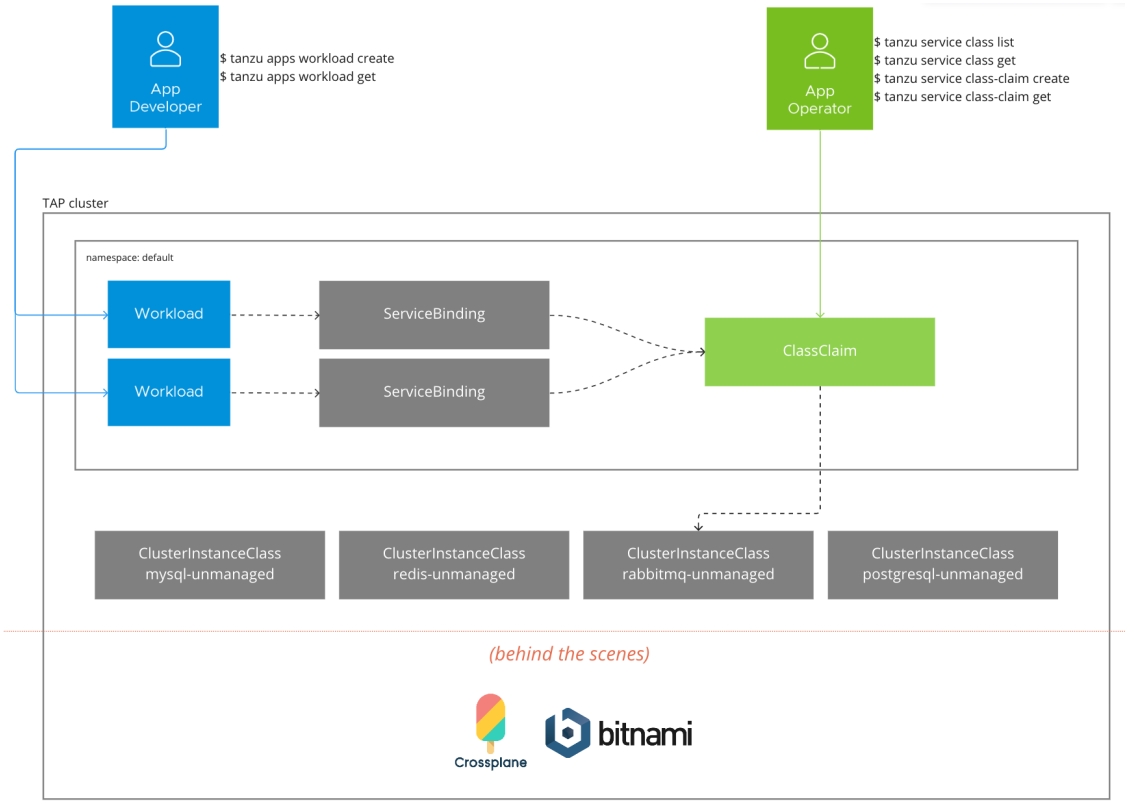
You will use the Tanzu Service CLI plug-in and will learn about classes, claims, and bindings.

What you will do

- Discover existing claims on service instances within your namespace
- Create two application workloads and bind them to an existing claim so that the workloads use the service instance.

Overview

The following diagram depicts a summary of what this tutorial covers.



Bear the following observations in mind as you work through this guide:

1. There is a set of four service classes preinstalled on the cluster.
2. Service operators do not need to configure or setup these four services.
3. The life cycle of a service binding is implicitly tied to the life cycle of a workload, and is managed by the application developer.
4. The life cycles of claims are explicitly managed by the application operator.
5. The diagram and tutorial in this guide are predominantly focused on the application operator and developer user roles, as such the inner workings of how service instances are provisioned are not in the diagram and are labeled as “behind the scenes”.

Prerequisites

Before following this tutorial, an application developer must:

1. Have access to a cluster with Tanzu Application Platform installed.
2. Have the Tanzu CLI and the corresponding plug-ins.
3. Have access to the `default` namespace which has been set up to use installed packages. For more information, see Set up developer namespaces to use your installed packages.
4. Have a Tanzu Application Platform cluster that can pull source code from GitHub.

Discovering existing claims

This section covers using `tanzu service class-claim list` and `tanzu service class-class get` to discover existing claims within your namespace and obtaining information needed to bind your workload to them.

1. To get the list of claims within your namespace, run:

   ```
   tanzu service class-claim list
   ```

   Expected output:

   ```
   NAME        CLASS                READY  REASON
   rabbitmq-1  rabbitmq-unmanaged   True   Ready
   ```

2. To get detailed information about the claim, run:

   ```
   tanzu service class-claim get rabbitmq-1
   ```

   Expected output:

   ```
     Name: rabbitmq-1
     Namespace: default
     Claim Reference: services.apps.tanzu.vmware.com/v1alpha1:ClassClaim:rabbitmq-
   1
     Class Reference:
       Name: rabbitmq-unmanaged
     Parameters:
       storageGB: 3
     Status:
       Ready: True
       Claimed Resource:
         Name: b5982046-a1e9-40cf-8282-00fe67a2f868
         Namespace: default
         Group:
         Version: v1
         Kind: Secret
   ```

Binding application workloads to the service instance

This section covers using `tanzu apps workload create` with the `--service-ref` flag to create workloads and to bind them to the service instance through the claim.

In Tanzu Application Platform, service bindings are created when you create application workloads using the `--service-ref` flag of the `tanzu apps workload create` command.

To create an application workload:

1. Review the output of the `tanzu service class-claim get` command you ran in Discovering existing claims earlier, and note the value of the `Claim Reference`. This is the value to pass to `--service-ref` when creating the application workloads.
2. Create the application workload by running:

```
tanzu apps workload create spring-sensors-consumer-web \
  --git-repo https://github.com/tanzu-end-to-end/spring-sensors \
  --git-branch rabbit \
  --type web \
  --label app.kubernetes.io/part-of=spring-sensors \
  --annotation autoscaling.knative.dev/minScale=1 \
  --service-ref="rmq=services.apps.tanzu.vmware.com/v1alpha1:ClassClaim:rabbitm
q-1"

tanzu apps workload create \
  spring-sensors-producer \
  --git-repo https://github.com/tanzu-end-to-end/spring-sensors-sensor \
  --git-branch main \
  --type web \
  --label app.kubernetes.io/part-of=spring-sensors \
  --annotation autoscaling.knative.dev/minScale=1 \
  --service-ref="rmq=services.apps.tanzu.vmware.com/v1alpha1:ClassClaim:rabbitm
q-1"
```

3. After the workloads are ready, visit the URL of the `spring-sensors-consumer-web` app. Confirm that sensor data is passing from the `spring-sensors-producer` workload to the `spring-sensors-consumer-web` workload using the `RabbitmqCluster` service instance.

Learn more

To learn more about working with services on Tanzu Application Platform, see the Services Toolkit component documentation:

- Tutorials
- How-to guides
- Concepts
- Reference material

Next steps

Now that you completed the Getting started guides, learn about:

- Multicluster Tanzu Application Platform

Deploy an air-gapped workload on Tanzu Application Platform

This topic for developers guides you through deploying your first workload on Tanzu Application Platform (commonly known as TAP) in an air-gapped environment.

For information about installing Tanzu Application Platform in an air-gapped environment, see Install Tanzu Application Platform in an air-gapped environment.

What you will do

- Create a workload from Git.

- Create a basic supply chain workload.
- Create a testing supply chain workload.
- Create a testing scanning supply chain workload.

Prerequisites

Before a developer can deploy an air-gapped workload, a platform operator must:

- Configure the air-gapped environment using Namespace Provisioner. For instructions, see Work with Git repositories in air-gapped environments with Namespace Provisioner.
- For the testing supply chain workload: Follow the steps in Install OOTB Supply Chain with Testing.
- For the testing scanning supply chain workload:
  - Follow the steps in Install OOTB Supply Chain with Testing and Scanning.
  - Set up vulnerability scanning. For instructions, see Use vulnerability scanning in offline and air-gapped environments.

Create a workload from Git

To create a workload from Git through HTTPS, follow these steps:

1. (Optional) To pass in login credentials for a Git repository with the certificate authority (CA) certificate, create a file called `git-credentials.yaml`. For example:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: git-ca
     # namespace: default
   type: Opaque
   stringData:
     username: USERNAME
     password: PASSWORD
     caFile: |
       CADATA
   ```

   Where:

   - `USERNAME` is the user name.
   - `PASSWORD` is the password.
   - `CADATA` is the PEM-encoded CA certificate for the Git repository.

2. To pass in a custom `settings.xml` for Java or NuGet:

   - For Java, create a file called `settings-xml.yaml`. For example:

     ```
     apiVersion: v1
     kind: Secret
     metadata:
       name: settings-xml
     type: service.binding/maven
     stringData:
       type: maven
       provider: sample
       settings.xml: |
         <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0" xmlns:xsi="h
     ttp://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 https://
     ```

```
maven.apache.org/xsd/settings-1.0.0.xsd">
        <mirrors>
            <mirror>
                <id>reposilite</id>
                <name>Tanzu seal Internal Repo</name>
                <url>https://reposilite.tap-trust.cf-app.com/releases</ur
l>
                <mirrorOf>*</mirrorOf>
            </mirror>
        </mirrors>
        <servers>
            <server>
                <id>reposilite</id>
                <username>USERNAME</username>
                <password>PASSWORD</password>
            </server>
        </servers>
    </settings>
```

- For NuGet, create a file called `settings-xml.yaml`. For example:

```
apiVersion: v1
kind: Secret
metadata:
  name: settings-xml
type: service.binding/nugetconfig
stringData:
  type: nugetconfig
  provider: sample
  nuget.config: |
    <?xml version="1.0" encoding="utf-8"?>
      <configuration>
        <packageSources>
          <clear />
          <add key="nuget-proxy" value=https://internal_nuget-proxy_fqdn/
repository/nuget.org-proxy/index.json />
        </packageSources>
      </configuration>
```

3. Apply the file:

```
kubectl create -f settings-xml.yaml -n DEVELOPER-NAMESPACE
```

Create a basic supply chain workload

Next, create your basic supply chain workload.

To pass the CA certificate in when you create the workload, run:

```
tanzu apps workload create APP-NAME --git-repo  https://GITREPO --git-branch BRANCH --
type web --label app.kubernetes.io/part-of=CATALOGNAME --yes --param-yaml buildService
Bindings='[{"name": "settings-xml", "kind": "Secret"}]' --param "source_credentials_se
cret=git-ca" --param "gitops_credentials_secret=git-ca"
```

Create a testing supply chain workload

To add the Tekton supply chain to the cluster, apply the following YAML to the cluster:

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: developer-defined-tekton-pipeline
```

```
  labels:
    apps.tanzu.vmware.com/pipeline: test     # (!) required
spec:
  params:
    - name: source-url                       # (!) required
    - name: source-revision                  # (!) required
  tasks:
    - name: test
      params:
        - name: source-url
          value: $(params.source-url)
        - name: source-revision
          value: $(params.source-revision)
      taskSpec:
        params:
          - name: source-url
          - name: source-revision
        steps:
          - name: test
            image: MY-REGISTRY/gradle
            script: |-
              cd `mktemp -d`
```

Where `MY-REGISTRY` is your container image registry. Relocate all the images given in the pipeline YAML to your private container registry.

Create the workload by running:

```
tanzu apps workload create APP-NAME --git-repo  https://GITURL --git-branch BRANCH --t
ype web --label app.kubernetes.io/part-of=CATALOGNAME --yes --param-yaml --label apps.
tanzu.vmware.com/has-tests=true buildServiceBindings='[{"name": "settings-xml", "kin
d": "Secret"}]'
```

To instead pass the CA certificate when you create the workload, run:

```
tanzu apps workload create APP-NAME --git-repo  https://GITREPO --git-branch BRANCH --
type web --label app.kubernetes.io/part-of=CATALOGNAME --yes --param-yaml --label app
s.tanzu.vmware.com/has-tests=true buildServiceBindings='[{"name": "settings-xml", "kin
d": "Secret"}]' --param "source_credentials_secret=git-ca" --param "gitops_credentials
_secret=git-ca"
```

Create a testing scanning supply chain workload

Create workload by running:

```
tanzu apps workload create APP-NAME --git-repo  https://GITURL --git-branch BRANCH --t
ype web --label app.kubernetes.io/part-of=CATALOGNAME --yes --param-yaml --label apps.
tanzu.vmware.com/has-tests=true buildServiceBindings='[{"name": "settings-xml", "kin
d": "Secret"}]'
```

To instead pass the CA certificate when you create the workload, run:

```
tanzu apps workload create APP-NAME --git-repo  https://GITREPO --git-branch BRANCH --
type web --label app.kubernetes.io/part-of=CATALOGNAME --yes --param-yaml --label app
s.tanzu.vmware.com/has-tests=true buildServiceBindings='[{"name": "settings-xml", "kin
d": "Secret"}]' --param "source_credentials_secret=git-ca" --param "gitops_credentials
_secret=git-ca"
```

Deploy Spring Cloud applications to Tanzu Application Platform

This sub-section tells you how to run Spring applications that rely on various Spring Cloud services as workloads on Tanzu Application Platform (commonly known as TAP).

In this sub-section:

- Deploy Spring Cloud Config applications
- Deploy Spring Cloud DiscoveryClient applications
- Use Spring Cloud Gateway for Kubernetes

Deploy Spring Cloud applications to Tanzu Application Platform

This sub-section tells you how to run Spring applications that rely on various Spring Cloud services as workloads on Tanzu Application Platform (commonly known as TAP).

In this sub-section:

- Deploy Spring Cloud Config applications
- Deploy Spring Cloud DiscoveryClient applications
- Use Spring Cloud Gateway for Kubernetes

Deploy Spring Cloud Config applications to Tanzu Application Platform

This topic tells you how to run Spring applications that depend on Spring Cloud Config Server as workloads on Tanzu Application Platform (commonly known as TAP).

Identify Spring Cloud Config applications

The Spring Cloud Config project is used within many common configuration services for Spring applications, including the following:

- The Config Server in the managed service tile Spring Cloud Services for VMware Tanzu that is supported by VMware Tanzu Application Service for VMs.
- Application Configuration Service for Tanzu in Azure Spring Apps. For more information about Azure Spring Apps, see the Microsoft Azure documentation.

Spring applications that use these configuration services often include a client dependency that interacts with the Spring Cloud Config Server:

- Applications that use the Spring Cloud Services Config Server on Tanzu Application Service typically include the `spring-cloud-services-starter-config-client` dependency from the `io.pivotal.spring.cloud` group. For more information, see the Config Server in the Spring Cloud Services documentation.
- Applications that use the open-source Spring Cloud Config Server typically include the `spring-cloud-starter-config` dependency from the `org.springframework.cloud` group. For more information, see the Spring Cloud Config documentation.

Prerequisites

Before you can deploy Spring Cloud Config applications, you must Install Application Configuration Service for VMware Tanzu.

The Application Configuration Service for VMware Tanzu component in Tanzu Application Platform distributes configuration information to applications through Kubernetes Secrets that contain Spring

properties.

Configure workloads

For instructions for how to run existing Spring applications that rely on the Spring Cloud Config Server as workloads in Tanzu Application Platform, see Configuring Workloads in Tanzu Application Platform using Application Configuration Service in the Application Configuration Service for VMware Tanzu documentation.

Deploy Spring Cloud DiscoveryClient applications to Tanzu Application Platform

This topic tells you how to run Spring applications that use the Spring Cloud DiscoveryClient as workloads on Tanzu Application Platform (commonly known as TAP).

Identify Spring Cloud DiscoveryClient applications

The Spring Cloud DiscoveryClient abstraction underlies several common libraries and services for Spring applications to register themselves as services for other applications and to look up connection details of registered applications. These services include the following:

- The Service Registry in the managed service tile Spring Cloud Services for VMware Tanzu supported by VMware Tanzu Application Service for VMs.
- The Tanzu Service Registry in Azure Spring Apps. For more information about Azure Spring Apps, see the Microsoft Azure documentation.
- The Spring Cloud Netflix project, which includes the Eureka client library and the Eureka server.

Spring applications that use these discovery services include a client dependency that implements the Spring Cloud DiscoveryClient:

- Applications that use the Spring Cloud Services Service Registry on Tanzu Application Service typically include the `spring-cloud-services-starter-service-registry` dependency from the `io.pivotal.spring.cloud` group. For more information, see Service Registry in the Spring Cloud Services documentation.
- Applications that use the Tanzu Service Registry in Azure Spring Apps or that use the Spring Cloud Netflix libraries typically include the `spring-cloud-starter-netflix-eureka-client` dependency from the `org.springframework.cloud` group. For more information about how to use the Tanzu Service Registry, see the Microsoft Azure documentation. For more information about how to include Eureka Client, see the Spring documentation.

Each of these client dependencies includes the Spring Cloud SimpleDiscoveryClient from the Spring Cloud Commons project as a base dependency. The approach in this topic uses this common dependency to configure service resolution for client applications.

Prerequisites

Before you can continue with the example in this topic, you must Install Application Configuration Service for VMware Tanzu.

In this example, the Application Configuration Service for VMware Tanzu component in Tanzu Application Platform distributes service discovery information to client applications as Spring properties.

Example: The Greeting application

The following sections show how to run the Greeting sample application as a pair of workloads on Tanzu Application Platform.

Create a properties file in your configuration repository

In a Git repository that is reachable from your Run cluster, create a `greeter-dev.yaml` file as follows:

```
eureka:
  client:
    # this disables the Eureka Spring Cloud discovery client
    enabled: false
spring:
  cloud:
    discovery:
      client:
        simple:
          instances:
            greeter-messages:
            - uri: http://greeter-messages.my-apps.svc.cluster.local
```

The values under `spring.cloud.discovery.client.simple.instances` list all the services that your application requires. The example `greeter-dev.yaml` file shows how to connect to another workload running on the same cluster.

In the example in Create application workload resources, the `greeter-messages` microservice is deployed as a workload of type `web`, so the discovery client configuration must use the fully qualified domain name for the service within the Kubernetes cluster. If you instead choose to run the `greeter-messages` microservice as a workload of type `server`, this address still works, but the `greeter` microservice can also connect using the shorter URI `http://greeter-messages`.

Create Application Configuration Service resources

On your Run cluster, create the `ConfigurationSource` and `ConfigurationSlice` resources that tell Application Configuration Service (ACS) how to fetch the discovery configuration from the Git repository you are using.

The following example uses a public repository and no encryption. For more information about how to connect to private repositories, encrypt configuration, and load properties in other formats, see the ACS documentation.

```
---
apiVersion: "config.apps.tanzu.vmware.com/v1alpha4"
kind: ConfigurationSource
metadata:
  name: greeter-config-source
  namespace: my-apps
spec:
  backends:
    - type: git
      uri: https://github.com/your-org/your-config-repo
---
apiVersion: config.apps.tanzu.vmware.com/v1alpha4
kind: ConfigurationSlice
metadata:
  name: greeter-config
  namespace: my-apps
spec:
  configurationSource: greeter-config-source
  content:
  - greeter/dev
```

```
  secretStrategy: applicationProperties
  interval: 10m
```

A Kubernetes secret is created in the `my-apps` namespace with a name starting with `greeter-config-`.

Create application workload resources

The `ConfigurationSlice` object you created in the previous section is a Provisioned Service. You can use a `ResourceClaim` to claim it within the `my-apps` namespace. You then supply the resource claim in the `serviceClaims` list in the `Workload` object to provide the configuration inside the runtime environment of the workload.

The `SPRING_CONFIG_IMPORT` variable passes this configuration to Spring. If your application already uses that variable to apply other Spring configuration, use the `SPRING_CONFIG_ADDITIONAL_LOCATION` variable instead.

In the following example, one workload is created for the `greeter-messages` microservice, and a second workload is created for the greeter microservice. Both apps bind to the `ConfigurationSlice` to add Spring configuration:

```
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: greeter-messages
  namespace: my-apps
  labels:
    apps.tanzu.vmware.com/workload-type: web
    apps.tanzu.vmware.com/has-tests: "true"
    app.kubernetes.io/part-of: greeter
spec:
  build:
    env:
    - name: BP_JVM_VERSION
      value: "17"
    # this tells the Gradle buildpack which module to build
    - name: BP_GRADLE_BUILT_MODULE
      value: "greeter-messages"
  env:
  # the Greeting app enables basic authentication unless the
  # development profile is used
  - name: SPRING_PROFILES_ACTIVE
    value: "development"
  - name: SPRING_CONFIG_IMPORT
    value: "${SERVICE_BINDING_ROOT}/spring-properties/"
  serviceClaims:
  - name: spring-properties
    ref:
      apiVersion: services.apps.tanzu.vmware.com/v1alpha1
      kind: ResourceClaim
      name: greeter-config-claim
  source:
    git:
      url: https://github.com/spring-cloud-services-samples/greeting
      ref:
        branch: main
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: greeter
  namespace: my-apps
  labels:
```

```
    apps.tanzu.vmware.com/workload-type: web
    apps.tanzu.vmware.com/has-tests: "true"
    app.kubernetes.io/part-of: greeter
spec:
  build:
    env:
    - name: BP_JVM_VERSION
      value: "17"
    - name: BP_GRADLE_BUILT_MODULE
      value: "greeter"
  env:
  - name: SPRING_PROFILES_ACTIVE
    value: "development"
  - name: SPRING_CONFIG_IMPORT
    value: "${SERVICE_BINDING_ROOT}/spring-properties/"
  serviceClaims:
  - name: spring-properties
    ref:
      apiVersion: services.apps.tanzu.vmware.com/v1alpha1
      kind: ResourceClaim
      name: greeter-config-claim
  source:
    git:
      url: https://github.com/spring-cloud-services-samples/greeting
      ref:
        branch: main
---
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ResourceClaim
metadata:
  name: greeter-config-claim
  namespace: my-apps
spec:
  ref:
    apiVersion: config.apps.tanzu.vmware.com/v1alpha4
    kind: ConfigurationSlice
    name: greeter-config
```

The greeter application builds, starts up, and finds the `greeter-messages` URI using the SimpleDiscoveryClient.

Using Spring Cloud Gateway for Kubernetes with Tanzu Application Platform

This topic tells you how to use Spring Cloud Gateway for Kubernetes as an API gateway for workloads running on Tanzu Application Platform (commonly known as TAP).

Spring Cloud Gateway is a popular project library for creating an API Gateway that is built on top of the Spring ecosystem. The open source library is a foundational component of VMware Spring Cloud Gateway for Kubernetes and Spring Cloud Gateway for VMware Tanzu Application Service commercial offerings with commercial-only capabilities and platform-integrated operator experiences. You can use the open source and commercial offerings as a reverse proxy with extra API Gateway functions to handle request and response to upstream application services.

Spring Cloud Gateway for Kubernetes is included with Tanzu Application Platform v1.5 and later. You can migrate upstream applications that expose API routes on Spring Cloud Gateway from Tanzu Application Service and custom open source implementations to Tanzu Application Platform. For how to do so, see the VMware Spring Cloud Gateway for Kubernetes documentation.

Run Spring Boot apps on Tanzu Application Platform as GraalVM native images

This topic guides you through how to run Spring Boot applications on Tanzu Application Platform as native images using GraalVM, including the available options.

What you will do

- Configure your Spring Boot application and `workload.yaml` file to run on Tanzu Application Plaform as a native image.
- Configure your app and `workload.yaml` file to enable Application Live View for Spring Boot native applications.

Introduction

Running a native image has several advantages:

- **Lower memory footprint:** This translates into lower resource use or increased scalability, such as deploying more replicas for the same memory cost.
- **Faster start times:** Component initialization is reduced to milliseconds, which makes pod recovery time and horizontal scalability faster.

Requirements for your Spring Boot application

To compile your Spring Boot applications into native images using GraalVM on Tanzu Application Platform, they must use Spring Boot 3 and Java 17 as a minimum. VMware recommends that you use the latest version of Spring Boot 3. To learn more about turning your Spring Boot applications into GraalVM native images in general, see the Spring Boot documentation.

Running Spring Boot applications as native images using GraalVM independent of Tanzu Application Platform might require some manual work to make your application work correctly when compiled to a native image. Changes to your application include avoiding the use of patterns such as reflection and implementing compiler hints. Therefore, VMware recommends that you make your application ready to run as a native image from the beginning and run extensive tests before trying to run them as native images on the platform. To confirm Spring Boot support for native testing, see the Spring Boot Documentation.

If your application can run as native images in general, Tanzu Application Platform allows you to run your applications as native executable files on the platform as well and tries to make it as easy as possible to manage your applications over time.

Run your Spring Boot workload as a native image

This section explains how to configure Tanzu Application Platform to compile your Spring Boot application into a native executable file and how to run that native executable file on the platform.

Configure the application side

For your application, use the usual process for developing and testing workloads that support native images. For general information about native image support, see the Spring documentation.

You must include the GraalVM native plug-in in your Maven POM file or Gradle build file.

- Example for Maven:

```
<build>
  <plugins>
    <plugin>
      <groupId>org.graalvm.buildtools</groupId>
```

```
      <artifactId>native-maven-plugin</artifactId>
    </plugin>
    <plugin>
    .
    .
    .
</build>
```

- Example for Gradle:

```
plugins {
  id("org.graalvm.buildtools.native") version "0.9.20"
  .
  .
  .
}
```

Configure the workload

To enable a native build, configure the buildpack to perform the native compilation step. You might also need to enable the native profile, mainly for Maven/Spring Boot 3.x projects.

You configure these settings using the `spec.build` environment parameters in the workload.

The following example works for all use cases. If you are using Gradle, remove the additional Maven build arguments. This example has the full `spec.build` configuration:

```
spec:
  build:
  env:
  - name: BP_JVM_VERSION
    value: "17"
  - name: BP_NATIVE_IMAGE
    value: "true"
  - name: BP_MAVEN_BUILD_ARGUMENTS
    value: '-Pnative -Dmaven.test.skip=true --no-transfer-progress package -Dspring-bo
ot.aot.jvmArguments=''-Dmanagement.endpoint.health.probes.add-additional-paths=''tru
e''
      -Dmanagement.endpoint.health.show-details=''always'' -Dmanagement.endpoints.web.
base-path=''/actuator''
      -Dmanagement.endpoints.web.exposure.include=''*'' -Dmanagement.health.probes.ena
bled=''true''
      -Dmanagement.server.port=8081 -Dserver.port=8080'' '
```

Because native images implement a closed world philosophy, you cannot set configuration at runtime deterministically. As a consequence, the Spring Boot Convention cannot automatically set options for native images to optimize running the application on the platform. However, the `tanzu-java-web-app` Tanzu Application Platform accelerator provides an example that you can refer to.

The most important elements from the configuration are `BP_JVM_VERSION`, `BP_NATIVE_IMAGE` and `BP_MAVEN_BUILD_ARGUMENTS`. They provide the instructions for buildpacks to `init` the native compilation. Without these, a normal Java virtual machine (JVM) compilation to generate a regular JAR file occurs.

The `BP_JVM_VERSION` configures buildpacks to use a JDK 17, because Spring Boot 3.x requires a JDK 17 as a minimum.

Note

Values used at build time do not specify the values that will be used at runtime. If you provide build time instrumentation, it only enables configuration to be set at

runtime. You must provide runtime values separately.

Depending on the app type, such as web, server, or worker, different methods for monitoring the health of the application are employed that might require configuring certain runtime parameters. For applications that do not need automatically configured actuators, you can generically achieve this using the following `build.spec` environment parameters:

```
 - name: MANAGEMENT_ENDPOINT_HEALTH_PROBES_ADD_ADDITIONAL_PATHS
         value: "true"
       - name: MANAGEMENT_HEALTH_PROBES_ENABLED
         value: "true"
       - name: SERVER_PORT
         value: "8080"
```

To set the actual runtime values, you can use the preceding settings in the `spec.env` parameters in the workload.

Example workload enabling native images

The following is an example of a full `workload.yaml` file that enables native image support:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
 name: myworkload
 labels:
   apps.tanzu.vmware.com/workload-type: web
   apps.tanzu.vmware.com/has-tests: "true"
   app.kubernetes.io/part-of: myworkload
spec:
 build:
   env:
   - name: BP_JVM_VERSION
     value: "17"
   - name: BP_NATIVE_IMAGE
     value: "true"
   - name: BP_MAVEN_ADDITIONAL_BUILD_ARGUMENTS
     value: -Pnative
   - name: MANAGEMENT_HEALTH_PROBES_ENABLED
     value: "true"
   - name: MANAGEMENT_ENDPOINT_HEALTH_PROBES_ADD_ADDITIONAL_PATHS
     value: "true"
   - name: SERVER_PORT
     value: "8080"
 env:
 - name: MANAGEMENT_HEALTH_PROBES_ENABLED
   value: "true"
 - name: MANAGEMENT_ENDPOINT_HEALTH_PROBES_ADD_ADDITIONAL_PATHS
   value: "true"
 - name: MANAGEMENT_ENDPOINT_HEALTH_SHOW_DETAILS
   value: always
 - name: SERVER_PORT
   value: "8080"
 source:
   git:
     url: https://github.com/myorg/myworkload
     ref:
       branch: main
```

What about the Spring Boot Convention?

When Spring Boot Convention detects a Spring Boot application, it automatically configures the workload for the platform, such as configuring the server port. The Spring Boot Convention does this by setting the `JAVA_TOOL_OPTIONS` environment variable and including various system property settings.

Because the setting `JAVA_TOOL_OPTIONS` doesn't work for native compiled Spring Boot applications, many of the configurations that the Spring Boot Convention applies don't have any effect. This is why you must configure this manually using separate environment variables as described earlier.

The goal for future versions of the Spring Boot Convention is to have it automatically do much of the current manual work.

Using Application Live View with Spring Boot native images

The Application Live View component of Tanzu Application Platform is designed around the Spring Boot Actuator extension for Spring Boot and therefore also works for Spring Boot apps that are compiled to native executable files.

However, due to current limits to automation, to enable Application Live View for your application you must configure specific options when building and running your Spring Boot apps as native images on Tanzu Application Platform.

Configure the application side

Application Live View relies on information obtained from the actuator endpoints of an application and therefore requires the actuator library to be present.

Configure your application to include the actuator library:

- Example for Maven:

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
  </dependency>
  .
  .
  .
<dependencies>
```

- Example for Gradle:

```
dependencies {
  implementation("org.springframework.boot:spring-boot-starter-actuator")
  .
  .
  .
}
```

Configure the workload

For an application to integrate with Application Live View:

1. The application must declare the integration by either:
    - In the `workload.yaml` file, set the `apps.tanzu.vmware.com/auto-configure-actuators` label. For more information, see Workload-level configuration.

- In the `tap-values.yaml` file, enable platform level configuration for actuator automatic configuration in the Spring Boot convention. For more information, see Platform-level configuration.

2. Configure additional runtime properties for Application Live View, which requires additional AOT (ahead-of-time) instrumentation at build time.

   Similar to general native image workloads, supply additional AOT configuration in the `workload.yaml` using `spec.build` environment parameters. Application Live View requires the following build environment parameters:

   ```
   - name: MANAGEMENT_HEALTH_PROBES_ENABLED
     value: "true"
   - name: MANAGEMENT_ENDPOINT_HEALTH_PROBES_ADD_ADDITIONAL_PATHS
     value: "true"
   - name: MANAGEMENT_ENDPOINT_HEALTH_SHOW_DETAILS
     value: always
   - name: MANAGEMENT_ENDPOINTS_WEB_BASE_PATH
     value: /actuator
   - name: MANAGEMENT_ENDPOINTS_WEB_EXPOSURE_INCLUDE
     value: '*'
   - name: MANAGEMENT_SERVER_PORT
     value: "8081"
   - name: SERVER_PORT
     value: "8080"
   ```

   Also similar to general native image workloads, runtime configuration is also provided through the use of environment variables.

Example workload enabling Application Live View

The following is an example of a full `workload.yaml` file that enables both native image compilation and integration with Application Live View:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
 name: myworkload
 labels:
   apps.tanzu.vmware.com/workload-type: web
   apps.tanzu.vmware.com/has-tests: "true"
   app.kubernetes.io/part-of: myworkload
   apps.tanzu.vmware.com/auto-configure-actuators: "true"
spec:
 build:
   env:
   - name: BP_JVM_VERSION
     value: "17"
   - name: BP_NATIVE_IMAGE
     value: "true"
   - name: BP_MAVEN_ADDITIONAL_BUILD_ARGUMENTS
     value: -Pnative
   - name: MANAGEMENT_HEALTH_PROBES_ENABLED
     value: "true"
   - name: MANAGEMENT_ENDPOINT_HEALTH_PROBES_ADD_ADDITIONAL_PATHS
     value: "true"
   - name: MANAGEMENT_ENDPOINT_HEALTH_SHOW_DETAILS
     value: always
   - name: MANAGEMENT_ENDPOINTS_WEB_BASE_PATH
     value: /actuator
   - name: MANAGEMENT_ENDPOINTS_WEB_EXPOSURE_INCLUDE
     value: '*'
   - name: MANAGEMENT_SERVER_PORT
```

```
    value: "8081"
  - name: SERVER_PORT
    value: "8080"
env:
- name: MANAGEMENT_HEALTH_PROBES_ENABLED
  value: "true"
- name: MANAGEMENT_ENDPOINT_HEALTH_PROBES_ADD_ADDITIONAL_PATHS
  value: "true"
- name: MANAGEMENT_ENDPOINT_HEALTH_SHOW_DETAILS
  value: always
- name: MANAGEMENT_ENDPOINTS_WEB_BASE_PATH
  value: /actuator
- name: MANAGEMENT_ENDPOINTS_WEB_EXPOSURE_INCLUDE
  value: '*'
- name: MANAGEMENT_SERVER_PORT
  value: "8081"
- name: SERVER_PORT
  value: "8080"
source:
  git:
    url: https://github.com/myorg/myworkload
    ref:
      branch: main
```

Usually, this configuration is automatically applied by the Spring Boot Convention when it detects a Spring Boot application. In the case of a natively compiled Spring Boot application, this configuration has to be done manually.

The Application Live View specific part of the Spring Boot Convention automatically flags the workload with a specific label to enable the Application Live View feature for the workload.

Register the app in the UI

The previous steps help Application Live View to detect your app, but it still won't show in the UI until you register the app.

To register the app in the UI, you must:

1. Navigate to **Tanzu Developer Portal**.
2. Click **REGISTER ENTITY**.
3. Follow the instructions onscreen.

For further instructions, see Deploy an app on Tanzu Application Platform.

Important

Not all the usual information will be available. For example, JVM memory information won't be available because native images have a slightly different mode, and there are other elements that won't be available in the actuator endpoints.

Create a new application accelerator

This topic guides you through creating an accelerator and registering it in a Tanzu Application Platform (commonly known as TAP) instance.

Tanzu Application Platform offers a selection of built-in accelerators to streamline your development process. However, if these accelerators don't meet your needs, you can create a new accelerator. By creating an accelerator, you can ensure that your technology stacks and organizational best practices are adhered to.

Note

This guide follows a quick start format. See the Application Accelerator documentation for advanced features.

What you will do

- Create a new accelerator project that contains an `accelerator.yaml` file and `README.md` file.
- Configure the `accelerator.yaml` file to alter the project's `README.md`.
- Test your accelerator locally using the Tanzu CLI `generate-from-local` command.
- Create a new Git repository for the project and push the project to it.
- Register the accelerator in a Tanzu Application Platform instance.
- Verify project generation with the new accelerator by using Tanzu Developer Portal.

Set up Visual Studio Code

1. To simplify accelerator authoring, code assist capabilities are available. To install the extension, navigate to the Marketplace page for the YAML plug-in and click **Install**.

   Note

   Code assist for authoring accelerators is also available in the IntelliJ IDE. You can enable this by selecting **Application Accelerator** in the schema mapping drop-down menu. For more information about how to enable this, see the IntelliJ Using schemas from JSON Schema Store documentation.

2. After you install the plug-in, editing files entitled `accelerator.yaml` automatically uses the code assist capabilities.

Create a simple project

To create your project, follow these instructions to set up the project directory, prepare the `README.md` and `accelerator.yaml`, and test your accelerator.

Set up the project directory

1. Create a new directory for the project named `myProject` and change to the newly created directory.

   ```
   mkdir myProject
   cd myProject
   ```

2. Create two new files in the `myProject` directory named `README.md` and `accelerator.yaml`.

   ```
   touch README.MD accelerator.yaml
   ```

Prepare the `README.md` and `accelerator.yaml`

The following instructions require using Visual Studio Code to edit the files.

1. Using Visual Studio Code, open the `README.md`, copy and paste the following code block into it, and save the file. `CONFIGURABLE_PARAMETER_#` is targeted to be transformed during project generation in the upcoming `accelerator.yaml` definition.

```
## Tanzu Application Accelerator Sample Project

This is some very important placeholder text that should describe what this pro
ject can do and how to use it.

Here are some configurable parameters:

* CONFIGURABLE_PARAMETER_1
* CONFIGURABLE_PARAMETER_2
```

2. Open `accelerator.yaml` and begin populating the file section using the snippet below. This section contains important information, such as the accelerator's display name, description, tags, and more.

   For all possible parameters available in this section, see Creating accelerator.yaml.

```
accelerator:
  displayName: Simple Accelerator
  description: Contains just a README
  iconUrl: https://blogs.vmware.com/wp-content/uploads/2022/02/tap.png
  tags:
    - simple
    - getting-started
```

3. Add the configuration parameters using the following code snippet. This configures what parameters are displayed in the accelerator form during project creation.

   In this example snippet, the field `firstConfigurableParameter` takes in text the user provides. The `secondConfigurableParameter` does the same, except it is only displayed if the user checks `secondConfigurableParameterCheckbox` because of the `dependsOn` parameter.

   For more information about possible options, see Creating accelerator.yaml.

```
# Place this after the 'tags' section from the previous step
  options:
    - name: firstConfigurableParameter
      inputType: text
      label: The text used to replace the first placeholder text in the README.
md. Converted to lowercase.
      defaultValue: Configurable Parameter 1
      required: true
    - name: secondConfigurableParameterCheckbox
      inputType: checkbox
      dataType: boolean
      label: Enable to configure the second configurable parameter, otherwise u
se the default value.
    - name: secondConfigurableParameter
      inputType: text
      label: The text used to replace the second placeholder text in the READM
E.md. Converted to lowercase.
      defaultValue: Configurable Parameter 2
      dependsOn:
        name: secondConfigurableParameterCheckbox
```

4. Add the `engine` configuration by using the following code snippet and save the file.

The `engine` configuration tells the `accelerator engine` behind the scenes what must be done to the project files during project creation. In this example, this instructs the engine to replace `CONFIGURABLE_PARAMETER_1` and, if the check box is checked, `CONFIGURABLE_PARAMETER_2` with the parameters that the user passes in during project creation.

This also leverages Spring Expression Language (SpEL) syntax to convert the text input to all lowercase.

For more information about the possible parameters for use within the `engine` section, see Creating accelerator.yaml.

```
# Place this after the `options` section from the previous step
engine:
  merge:
    - include: [ "README.md" ]
      chain:
        - type: ReplaceText
          substitutions:
            - text: "CONFIGURABLE_PARAMETER_1"
              with: "#firstConfigurableParameter.toLowerCase()"
        - condition: "#secondConfigurableParameterCheckbox"
          chain:
          - type: ReplaceText
            substitutions:
              - text: "CONFIGURABLE_PARAMETER_2"
                with: "#secondConfigurableParameter.toLowerCase()"
```

Test the accelerator

It is important to quickly test and iterate on accelerators as they are being developed to ensure that the resulting project is generated as expected.

1. Using the terminal of your choice with access to the `tanzu` command, run the following command to test the accelerator created earlier.

   This step takes the local `accelerator.yaml` and project files, configures the project using the parameters passed in through the `--options` field, and outputs the project to a specified directory.

   Important

   This step requires that the `TANZU-APPLICATION-ACCELERATOR-URL` endpoint is exposed and accessible. For more information, see Server API connections for operators and developers.

   ```
   tanzu accelerator generate-from-local \
       --accelerator-path simple-accelerator="$(pwd)" `# The path to new accelerat
   or` \
       --server-url TANZU-APPLICATION-ACCELERATOR-URL `# Example: https://accelera
   tor.mytapcluster.myorg.com` \
       --options '{"firstConfigurableParameter": "Parameter 1", "secondConfigurabl
   eParameterCheckbox": true, "secondConfigurableParameter":"Parameter 2"}' \
       -o "${HOME}/simple-accelerator/" `# Change this path to change where the pr
   oject folder gets generated`
   ```

2. After the project is generated, a status message is displayed.

   ```
   generated project simple-accelerator
   ```

3. Navigate to the output directory and verify that the `README.md` is updated based on the `--options` specified in the preceding `generate-from-local` command.

```
## Tanzu Application Accelerator Sample Project

This is some very important placeholder text that should describe what this pro
ject can do and how to use it.

Here are some configurable parameters:

- parameter 1
- parameter 2
```

Upload the project to a Git repository

The Application Accelerator system and Tanzu Developer Portal depend on an accelerator project residing inside a Git repository. For this example, GitHub is used.

1. Create a new repository in GitHub and ensure that **Visibility** is set to **Public**. Click **Create Repository**.
2. To push your accelerator project (**not** the generated project from `generate-from-local`) to GitHub, follow the instructions that GitHub provides for the *…or create a new repository on the command line* that is shown after clicking **Create Repository**. Instructions can also be found in the GitHub documentation.
3. Verify that the project is pushed to the target repository.

Register the accelerator to the Tanzu Application Platform and verify project generation output

Now that the accelerator is committed to its own repository, you can register the accelerator to Tanzu Developer Portal for developers to generate projects from the newly created accelerator.

To do so, use the URL of the Git repository and branch name created earlier and run the following command using the Tanzu CLI to register the accelerator to Tanzu Developer Portal.

Note

`tanzu accelerator create` works with monorepos as well. Add the `--git-sub-path` parameter with the desired subpath to fetch the accelerator project in that directory. For more information, see the [Tanzu CLI Command Reference] (https://docs.vmware.> com/en/VMware-Tanzu-CLI/1.3/tanzu-cli/command-ref.html) documentation.

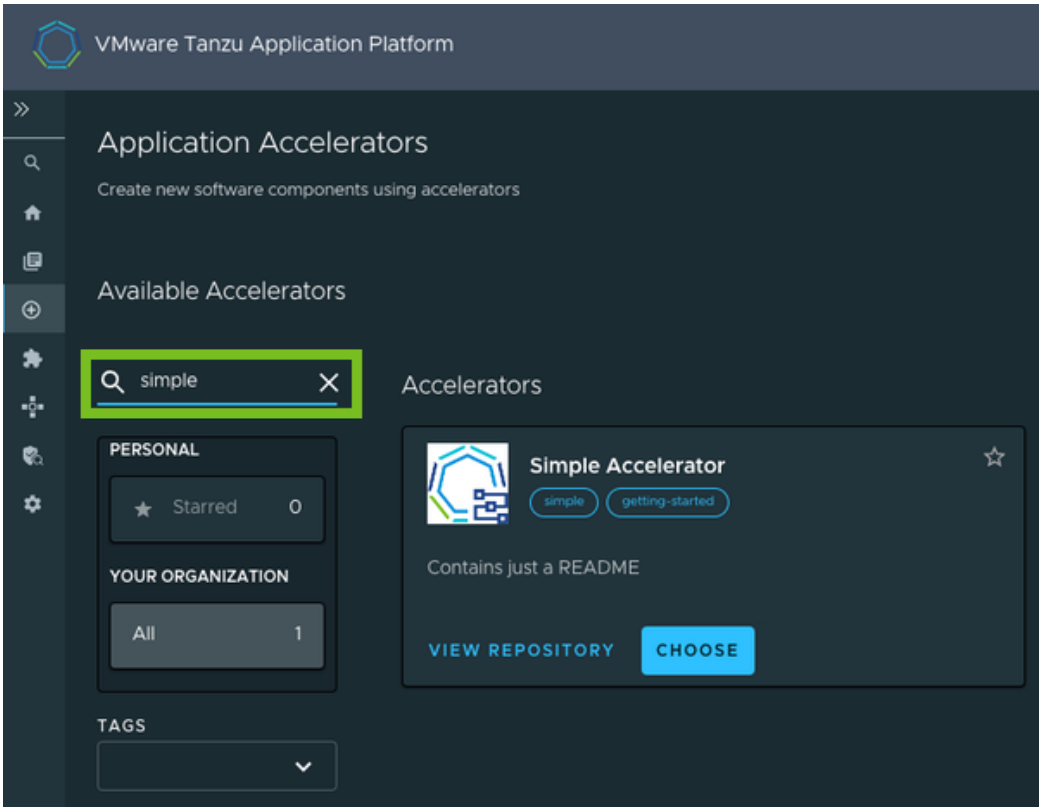
```
tanzu accelerator create simple-accelerator --git-repository https://github.com/myuser
name/myprojectrepository --git-branch main
```

The accelerator can take time to reconcile. After it has reconciled, it is available for use in Tanzu Developer Portal and the Application Accelerator extension for Visual Studio Code.

Verify project generation output by using Tanzu Developer Portal

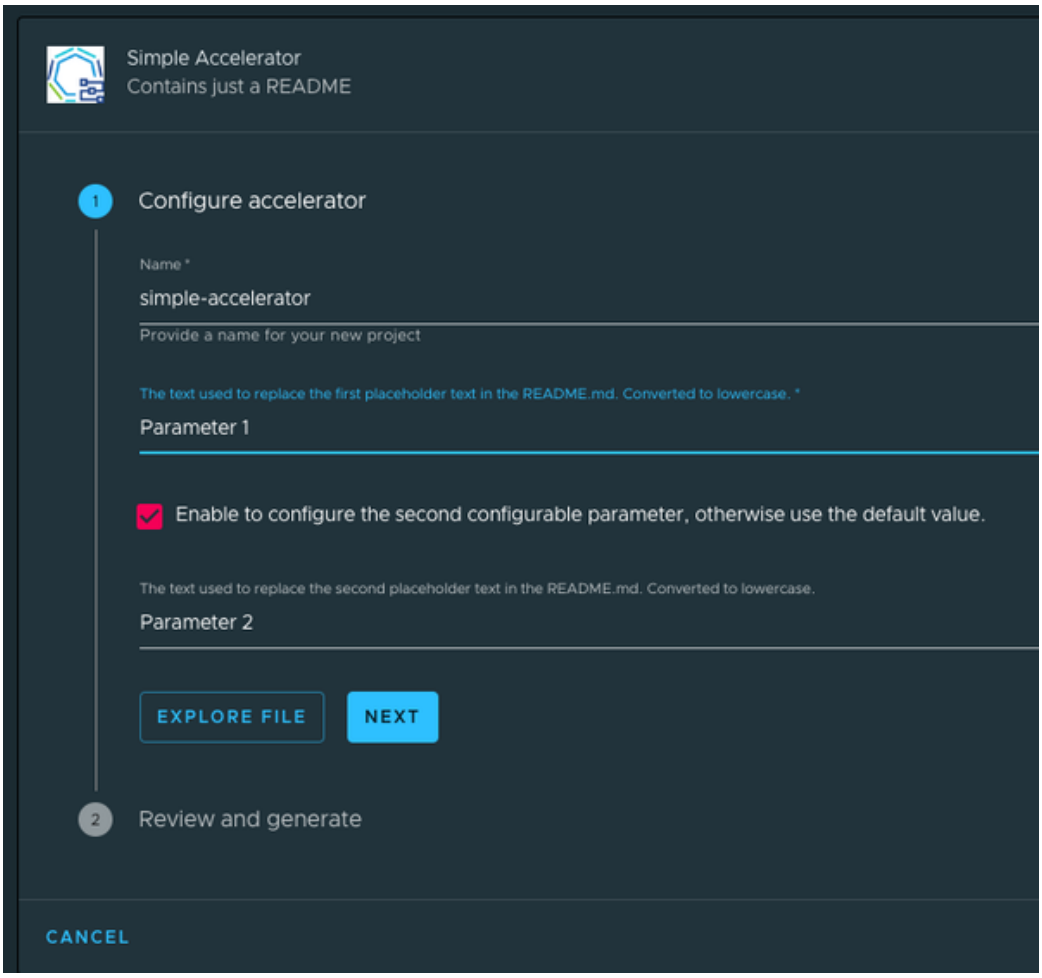
1. Navigate to your organization's instance of Tanzu Developer Portal.

2. On the left navigation pane, click **Create**.
3. Using the search bar near the left side of the page, search for `simple accelerator`. After you've found it, click **Choose** on the accelerator card.



4. Configure the project by filling in the parameters in the form.

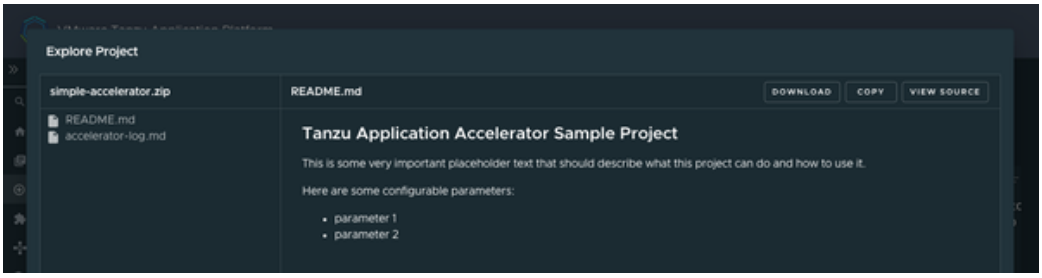
   The options you defined in `accelerator.yaml` are now displayed for you to configure. The `secondConfigurableParameter dependsOn secondConfigurableParameterCheckbox` might be hidden depending on whether the check box is selected.



5. After configuration is complete, click **Next**.

> **Note**
>
> Depending on your organization's Tanzu Application Platform configuration, you might be presented with an option to create a Git repository. In this guide, this is skipped and is covered in Deploy an app on Tanzu Application Platform.

6. On the **Review and generate** step, review the parameters and click **Generate Accelerator**.
7. Explore the ZIP file of the configured project and verify that the project is generated with the parameters you provided during configuration.



Learn more about Application Accelerator

- For advanced functionality when creating accelerators, such as accelerator best practices, accelerator fragments, engine transforms, and more, see the Application Accelerator documentation.
- For more information about Application Accelerator configurations, see the Configure Application Accelerator documentation.
- For information about installing the Application Accelerator extension for Visual Studio Code, see the Application Accelerator Visual Studio Code extension documentation.
- For general accelerator troubleshooting, see Troubleshooting Application Accelerator for VMware Tanzu.

Learn about Tanzu Application Platform

The topics in this section explain concepts important to getting started with Tanzu Application Platform (commonly known as TAP).

In this section:

- Application Accelerator
- Supply chains on Tanzu Application Platform
- Vulnerability scanning and metadata storage for your supply chain
- Consume services on Tanzu Application Platform

Application accelerators on Tanzu Application Platform

This topic describes the key concepts you need to know about application accelerators on Tanzu Application Platform (commonly known as TAP).

What are application accelerators

Application accelerators are templates that not only codify best practices but also provide important configuration and structures ready and available for use. Developers can create applications and get started with feature development immediately with the help of application accelerators.

Enterprise Architects use Application Accelerator to create application accelerators, which provide developers and admins in their organization with ready-made, enterprise-conforming code and configurations. Accelerators contain complete and runnable application code and deployment configurations. They also contain metadata for altering the code and deployment configurations based on input values provided for specific options defined in the accelerator metadata.

Working with accelerators

The Application Accelerator plug-in for Tanzu Developer Portal helps you to discover accelerators and to enter extra information used for processing the files before downloading. As of Tanzu Application Platform v1.2, developers can also discover and work on accelerators right in Visual Studio Code with the Tanzu Application Accelerator for VS Code extension. Developers can use the `list`, `get`, and `generate` commands to use accelerators available in an Application Accelerator server.

Admins use the `create`, `update`, and `delete` commands for managing accelerators in a Kubernetes context. When admins want to use the `get` and `list` commands, they can specify the `--from-context` flag to access accelerators in a Kubernetes context.

Next steps

Apply what you have learned:

Developers:

- Deploy an app on Tanzu Application Platform

Operators:

- Create an application accelerator

Supply chains on Tanzu Application Platform

This topic describes the key concepts you need to know about supply chains and Continuous Integration/Continuous Delivery (CI/CD) on Tanzu Application Platform (commonly known as TAP).

What are supply chains

Supply chains provide a way of codifying all of the steps of your path to production, more commonly known as CI/CD. CI/CD is a method to frequently deliver applications by introducing automation into the stages of application development. The main concepts attributed to CI/CD are continuous integration, continuous delivery, and continuous deployment.

CI/CD is the method used by supply chains to deliver applications through automation. Tanzu Application Platform supply chains allow you to use CI/CD and add any other steps necessary for an application to reach production or a different environment, such as staging.

A path to production

A path to production allows you to create a unified access point for all of the tools required for your applications to reach a customer-facing environment. Instead of having four tools that are loosely coupled to each other, a path to production defines all four tools in a single, unified layer of abstraction. The path to production can be automated and repeatable between teams for applications at scale.

Typically tools cannot integrate with one another without scripting or webhooks. Whereas with a path to production, there is a unified automation tool to codify all the interactions between each of the tools. Supply chains that are used to codify the path to production for an organization are configurable. This allows their authors to add all of the steps of the path to production for their applications.

Available supply chains

Tanzu Application Platform provides three out of the box (OOTB) supply chains to work with the Tanzu Application Platform components. They include:

- OOTB Supply Chain Basic (default)
- OOTB Supply Chain with Testing (optional)
- OOTB Supply Chain with Testing+Scanning (optional)

1: OOTB Basic (default)

The default **OOTB Basic** supply chain and its dependencies were installed on your cluster during the Tanzu Application Platform install. The following diagram and table provide a description of the supply chain and dependencies provided with Tanzu Application Platform.

Name	Package Name	Description	Dependencies
Out of the Box Basic (Default - Installed during Installing Part 2)	`ootb-supply-chain-basic.tanzu.vmware.com`	This supply chain monitors a repository that is identified in the developer's `workload.yaml` file. When any new commits are made to the application, the supply chain: • Creates a new image. • Applies any predefined conventions. • Deploys the application to the cluster.	• Flux/Source Controller • Tanzu Build Service • Cartographer Conventions • Tekton • Cloud Native Runtimes • If using Service References: ◦ Service Bindings ◦ Services Toolkit

2: OOTB Testing

OOTB Testing supply chain runs a Tekton pipeline within the supply chain. The following diagram and table provide a description of the supply chain and dependencies provided with Tanzu Application Platform.

Name	Package Name	Description	Dependencies
Out of the Box Testing	`ootb-supply-chain-testing.tanzu.vmware.com`	Out of the Box Testing contains all of the same elements as the Source to URL. It allows developers to specify a Tekton pipeline that runs as part of the CI step of the supply chain. • The application tests using the Tekton pipeline. • A new image is created. • Any predefined conventions are applied. • The application is deployed to the cluster.	All of the Source to URL dependencies

3: OOTB Testing+Scanning

OOTB Testing+Scanning supply chain includes integrations for secure scanning tools. The following diagram and table provide a description of the supply chain and dependencies provided with Tanzu Application Platform.

Name	Package Name	Description	Dependencies
Out of the Box Testing and Scanning	`ootb-supply-chain-testing-scanning.tanzu.vmware.com`	Out of the Box Testing and Scanning contains all of the same elements as the Out of the Box Testing supply chain, and it also includes integrations with the secure scanning components of Tanzu Application Platform. • The application is tested using the provided Tekton pipeline. • (Optional) The application source code is scanned for vulnerabilities. For how to opt in, see Adding Source Scan to the Test and Scan Supply Chain. • A new image is created. • The image is scanned for vulnerabilities. • Any predefined conventions are applied. • The application deploys to the cluster.	All of the Source to URL dependencies, and: • The secure scanning components included with Tanzu Application Platform

Next steps

Apply what you have learned:

- Add testing and scanning to your application

Or learn about:

- Vulnerability scanning and metadata storage for your supply chain

Vulnerability scanning, storing, and viewing for your supply chain

This topic describes the vulnerability scanning features you can use with Tanzu Application Platform (commonly known as TAP).

This feature set allows an application operator to introduce source code and image vulnerability scanning, storing, and viewing to their Tanzu Application Platform supply chain. It also allows for the creation of scan-time rules that prevent critical vulnerabilities from flowing to the supply chain unresolved.

Features

Features include:

- Scan source code repositories and images for known common vulnerabilities and exposures (CVEs) before deploying to a cluster.
- Identify CVEs by scanning continuously on each new code commit or each new image built.
- Analyze scan results against user-defined policies by using Open Policy Agent. Create scan policy to prevent vulnerable components from going into production.
- Produce vulnerability scan results and post them to the SCST - Store where they can be queried.
- Query the store for such use cases as:
    - What images and packages are affected by a specific vulnerability?
    - What source code repositories are affected by a specific vulnerability?
    - What packages and vulnerabilities does a particular image have?

- Visualize the supply chain and its packages and vulnerabilities of your supply chain.

Components

- Supply Chain Security Tools (SCST) - Scan scans source code and images for their packages and vulnerabilities.
- SCST - Store takes the vulnerability scanning results and stores them.
- Tanzu Insight plug-in provides a CLI to query for packages and vulnerabilities.
- Supply Chain Choreographer in Tanzu Developer Portal visualizes the supply chain, including scans, packages, and vulnerabilities.

Next steps

Apply what you have learned:

- Add testing and scanning to your application

Or learn about:

- Supply chains on Tanzu Application Platform

Or go deeper into scanning on Tanzu Application Platform:

- Scan samples to try the scan and store features as individual one-off scans
- Configure Code Repositories and Image Artifacts to be Scanned
- Code and Image Compliance Policy Enforcement Using Open Policy Agent (OPA)
- How to Create a ScanTemplate
- Viewing and Understanding Scan Status Conditions
- Observing and Troubleshooting
- Tanzu Insight plug-in overview

Troubleshooting

- SCST Scan - Observing and Troubleshooting
- SCST Store - Troubleshooting
- Tanzu Developer Portal - Troubleshooting

About consuming services on Tanzu Application Platform

This topic describes the key concepts and terms you need to know about consuming services on Tanzu Application Platform (commonly known as TAP).

As part of Tanzu Application Platform, you can work with backing services such as RabbitMQ, PostgreSQL, and MySQL among others. The most common use of services is binding an application workload to a service instance.

Key concepts

When working with services on Tanzu Application Platform, you must be familiar with service instances, service bindings, and resource claims. This section provides a brief overview of each of these key concepts.

Service instances

A **service instance** is a logical grouping of one or more Kubernetes resources that together expose a known capability through a well-defined interface. For example, a theoretical MySQL service instance might consist of a `MySQLDatabase` and a `MySQLUser` resource. When considering compatibility of service instances for Tanzu Application Platform, one of the resources of a service instance must adhere to the Service Binding for Kubernetes specification.

Service bindings

Service binding refers to a mechanism in which connectivity information, such as service instance credentials, and connectivity information, such as host and port, are automatically communicated to application workloads. Tanzu Application Platform uses a standard named Service Binding for Kubernetes to implement this mechanism. See this standard to fully understand the services aspect of Tanzu Application Platform.

Resource claims

Resource claims are inspired in part by Persistent Volume Claims. For more information, see the Kubernetes documentation. Resource claims provide a mechanism for users to claim service instances on a cluster, while also decoupling the life cycle of application workloads and service instances.

Services you can use with Tanzu Application Platform

The following list of Kubernetes operators expose APIs that integrate well with Tanzu Application Platform:

1. VMware RabbitMQ for Kubernetes.
2. VMware SQL with Postgres for Kubernetes.
3. VMware SQL with MySQL for Kubernetes.

Compatibility of a service with Tanzu Application Platform ranges on a scale between fully compatible and incompatible. The minimum requirement for compatibility is that there must be a declarative, Kubernetes-based API on which at least one API resource type adheres to the Provisioned Service duck type defined by the Service Binding Specification for Kubernetes in GitHub. This duck type includes any resource type with the following schema:

```
status:
  binding:
    name: # string
```

The value of `.status.binding.name` must point to a `Secret` in the same namespace. The `Secret` contains required credentials and connectivity information for the resource.

Typically, APIs that include these resource types are installed onto the Tanzu Application Platform cluster as Kubernetes operators. These Kubernetes operators provide custom resource definitions (CRDs) and corresponding controllers to reconcile the resources of the CRDs, as is the case with the three Kubernetes operators listed earlier.

For services that do not provide a resource adhering to the Service Binding Specification for Kubernetes, you might still be able to provide configurations allowing such services to integrate with Tanzu Application Platform. For an example of how to do this for Amazon AWS RDS, see the tutorial Integrating cloud services into Tanzu Application Platform.

User roles and responsibilities

It is important to understand the user roles for services on Tanzu Application Platform and the responsibilities assumed by each. The following table describes each user role.

User role	Exists as a default role in Tanzu Application Platform?	Responsibilities
Service operator	Yes - service-operator	• Life cycle management (CRUD) of service instances • Life cycle management (CRUD) of service instance classes • Life cycle management (CRUD) of resource claim policies • View and query for resource claims across namespaces
Application operator	Yes - app-operator	Life cycle management (CRUD) of resource claims
Application developer	Yes - app-editor and app-viewer	Binding service instances to application workloads

Next steps

Apply what you've learned:

- Claim services on Tanzu Application Platform
- Consume services on Tanzu Application Platform

Migrate applications from Tanzu Application Service

The topics in this section tell you how to migrate applications from Tanzu Application Service (commonly known as TAS) to Tanzu Application Platform (commonly known as TAP).

In this section:

Migrate applications from Tanzu Application Service

The topics in this section tell you how to migrate applications from Tanzu Application Service (commonly known as TAS) to Tanzu Application Platform (commonly known as TAP).

In this section:

Compare service management on Tanzu Application Platform and Tanzu Application Service

This topic compares how you manage services on Tanzu Application Platform (commonly known as TAP) with how you manage services on Tanzu Application Service (commonly known as TAS).

Overview

Tanzu Application Service and Tanzu Application Platform both provide service binding capabilities to declaratively connect applications to their backing services. However, the catalog of available services, the user experience, and the mechanics of this capability differ.

For Tanzu Application Platform, the enabling package for services is called Services Toolkit. Tanzu Application Platform is usually installed using package profiles. Services Toolkit is installed by default in the Run and Iterate profiles. For more information, see Components and installation profiles for Tanzu Application Platform.

In Tanzu Application Platform, services are deployed alongside workloads in the same Kubernetes cluster. By contrast, Tanzu Application Service registers service brokers with the Cloud Foundry Marketplace which can run anywhere, but are commonly run as BOSH or Cloud Foundry deployments.

Service offerings

The following table compares the services offering capabilities in Tanzu Application Service and Tanzu Application Platform.

Feature	Tanzu Application Service	Tanzu Application Platform
Deploy an API for service instance CRUD	Services are packaged in a Tile which provision an OSBAPI-compliant API (“service broker”)	Services are packaged as Crossplane Composition and XRD, and Services Toolkit ClusterInstanceClass.
Publish a service offering	When a tile is installed, it can then be registered in the Services Marketplace. Service brokers include catalog metadata that is used by services Marketplace.	A ClusterInstanceClass Kubernetes resource is created to reference a Crossplane Composition.
List available service offerings	CLI: `cf marketplace`	CLI: `tanzu service class list`
Service Plans	Defined in the service broker implementation	There is no concept of plans. Parameters can be exposed to the user and wired through Crossplane composition, XRD, and ClusterInstanceClass. Multiple compositions can be created to offer different variants of a service, for example `{small, large}`
Get Service offering details	CLI: `cf marketplace -s SERVICE_NAME`	CLI: `tanzu service class get SERVICE_NAME`

Available out-of-the-box services

Services offered out of the box are very similar to offerings in `cf marketplace`. The following services are available on Tanzu Application Platform out of the box as provisioner-based classes that and require little to no configuration for a user to self-provision:

- Dev Backing Services by Bitnami: Redis, RabbitMQ, MySQL, PostgreSQL, Kafka, MongoDB
- AWS Services: RDS (PostgreSQL and MySQL) and AWS MQ (RabbitMQ)

For services not offered out of the box, Tanzu Application Platform also has a workflow for user-provided service instances called Direct Secret Reference.

Service compatibility and creating new service offerings

You can integrate most services with Tanzu Application Platform workloads. There are four levels of services consumption on Tanzu Application Platform. On one end of the spectrum, Tanzu Application Platform has a concept similar to Tanzu Application Service’s user provided service instances called direct secret references. User-provided service instances and direct secret

references allow you to use services that are not available in the Marketplace by providing the credentials and network endpoint. On the other end of the spectrum, a platform operator can write Crossplane compositions to offer a provisioner-based class. With each level comes more complexity but provides a more integrated self-service offering. Use the highest-level abstraction you can to make life cycle management of the services as easy to use as possible.

A key difference between Tanzu Application Service and Tanzu Application Platform is that Tanzu Application Service requires a much more rigid implementation that requires the build and release of a new versioned Tile. Whereas on Tanzu Application Platform the approach is much more modular and can be updated by modifying YAML to change settings or exposed parameters.

Tanzu Application Platform also offers a layer of abstraction called pooled classes where pre-provisioned services can be grouped together by service type, for example, MySQL, and those instances can be claimed without having to specify a particular service instance. This is a great option for still being able to offer developer self-service in organizations that have advanced service configuration requirements managed by a central services team

- Using Direct Secret references (Level 1): See Using direct secret references
- Create a Tanzu RabbitMQ service offering (Level 4): See Set up dynamic provisioning of service instances
- Create a Tanzu PostgreSQL service offering (Level 4): See Configure dynamic provisioning of VMware SQL with Postgres for Kubernetes service instances
- Create a service offering for a cloud service (Level 4): See Integrating cloud services into Tanzu Application Platform

Service binding workflow

This section compares the service binding workflow for Tanzu Application Service to Tanzu Application Platform.

Out of the box services

The following table compares the binding workflow for out of the box services.

Feature	Tanzu Application Service	Tanzu Application Platform
List available service offerings	`cf marketplace`	`tanzu service class list`
Provision a service instance	`cf create-service SERVICE-CLASS PLAN SERVICE-INSTANCE`	`tanzu service class-claim create SERVICE-INSTANCE -class SERVICE-CLASS`
Bind a service to app	`cf bind-service APP-NAME SERVICE-INSTANCE`	`tanzu apps workload update APP-NAME --service-ref SERVICE-INSTANCE`
Unbind a service to app	`cf unbind-service APP-NAME SERVICE-INSTANCE`	`tanzu apps workload update my-app --service-ref=SERVICE-INSTANCE:-`
Delete a service instance	`cf delete-service SERVICE-INSTANCE`	`tanzu service class-claim delete SERVICE-INSTANCE`

User-provided services

The following table compares the binding workflow for for user provided services.

Feature	Tanzu Application Service	Tanzu Application Platform
Create service credentials	`cf cups SERVICE-INSTANCE -p '{"username":"USERNAME", "password":"PASSWORD"}'`	Create a Kubernetes secret containing key/value pairs
Bind a service to app	`cf bind-service APP-NAME SERVICE-INSTANCE`	`tanzu apps workload update APP-NAME --service-ref SERVICE-INSTANCE`
Unbind a service to app	`cf unbind-service APP-NAME SERVICE-INSTANCE`	`tanzu apps workload update my-app --service-ref=SERVICE-INSTANCE:-`

Common operations

This section compares how Tanzu Application Service and Tanzu Application Platform meets the needs of different roles.

Service author perspective

The following table compares Tanzu Application Service and Tanzu Application Platform for service authors.

Feature	Tanzu Application Service	Tanzu Application Platform
Integrate a backing service with the platform	• Develop and maintain an OSBAPI Service Broker • Develop and maintain a Tanzu Application Service Tile	• Develop and maintain a Kubernetes Operator • Develop and maintain a set of Crossplane and Services Toolkit configurations (XRD, Composition, ClusterInstanceClass)

Service operator perspective

The following table compares Tanzu Application Service and Tanzu Application Platform for service operators.

Feature	Tanzu Application Service	Tanzu Application Platform
Offer a backing service for self-serve consumption to application developers on the platform	Install the Tanzu Application Service Tile	Install the Kubernetes operator
Ensure that all service instances on the platform meet all compliance and security requirements	Configure settings in the Tile using Ops Manager	Configure the Crossplane XRD or Composition and Services Toolkit ClusterInstanceClass accordingly

Application developer perspective

The following table compares Tanzu Application Service and Tanzu Application Platform for app developers.

Feature	Tanzu Application Service	Tanzu Application Platform
Discover the list of backing services that are available	CLI: `cf marketplace`	CLI: `tanzu service class list`
See detailed information about a particular backing service	CLI: `cf marketplace`	CLI: `tanzu service class get CLASS-NAME`

Feature	Tanzu Application Service	Tanzu Application Platform
Create a new service instance of one of the available backing services	CLI: `cf create-service SERVICE-CLASS PLAN SERVICE-INSTANCE`	CLI: `tanzu service class-claim create INSTANCE-NAME --class CLASS-NAME --parameter foo=bar`
Bind a service instance to an application workload	CLI: `cf bind-service APP-NAME SERVICE-INSTANCE`	CLI: `tanzu apps workload update my-app --service-ref CLASS-CLAIM-REF`
Bind a User Provided Service to an application workload	CLI: `cf cups SERVICE-INSTANCE -p '{"username":"admin","password":"pa55woRD"}` then run `cf bind-service`	Create a Kubernetes Secret containing the required credentials then run `tanzu apps workload update my-app --service-ref KUBERNETES-SECRET-REF`
Bind two application workloads to the same service instance in the same Tanzu Application Service space or Tanzu Application Platform namespace	Run the `cf bind-service` command twice, once for each app	Run the `tanzu apps workload update` command twice, once for each app
Share a service instance with another Tanzu Application Service space or Tanzu Application Platform namespace	CLI: `cf share-service SERVICE-INSTANCE -s OTHER_SPACE [-o OTHER_ORG]`	Not currently supported

Key differences

- Tanzu Application Platform does not use OSBAPI to provide backing service integrations. Tanzu Application Platform uses the Service Binding Specification for Kubernetes instead.
- Tanzu Application Platform does not have a concept of Service Plans. However, you can configure the ClusterInstanceClass in Tanzu Application Platform to expose zero or more properties that enable developers to configure parameters for service instances to meet their needs. You can also set defaults for those parameters.

  This approach is arguably more flexible because service operators can configure or update these properties when needed without requiring code changes and version updates to the corresponding OSBAPI brokers.
- Tanzu Application Platform does not currently have a GUI element for services.
- In Tanzu Application Service when you bind a service instance to an application workload, each separate binding is guaranteed to receive a unique set of credentials for access to the service instance. This is not guaranteed in Tanzu Application Platform.

  The behavior in Tanzu Application Platform depends on the backing Kubernetes operator.

End-to-end example deploying an app

for a step-by-step workflow of deploying an app with a service, see Migrate Spring Data Services apps to Tanzu Application Platform.

Migrate from Single Sign-On for VMware Tanzu Application Service

This topic tells you how to migrate single-sign on services from Tanzu Application Service (TAS) to Tanzu Application Platform (TAP).

Application Single Sign-On for VMware Tanzu provides single sign-on services for VMware Tanzu Application Platform. There is a comparable offering available for VMwate Tanzu Application

Service. For more information, see the Single Sign-On for VMware Tanzu Application Service documentation.

Concept comparison

The following table outlines the essential concepts, highlighting the key differences between Single Sign-On for VMware Tanzu Application Service (SSO for TAS) and Application Single Sign-On for VMware Tanzu Application Platform (AppSSO for TAP).

SSO for TAS	AppSSO for TAP	Notes
Service Plan	`AuthServer` and `ClusterWorkloadRegistrationClass`	In TAS User Account and Authentication (UAA), this corresponds to an Identity Zone. For more information, see Curate a service offering.
System Plan	n/a	This represents the special tenant in the UAA for platform access.
Internal user store	n/a	AppSSO does not support internal users. For more information, see Unsupported features. `AuthServer.spec.identityProviders.internalUnsafe` is for testing purposes.
External Identity Provider	`AuthServer.spec.identityProviders`	For more information, see Identity providers for AppSSO.
Operator dashboard	`AuthServer` Custom Resource	Operators or Service Operators interact with AppSSO by using the `AuthServer` custom resource.
Developer dashboard	`ClientRegistration` Custom Resource	Developers or App Operators interact with AppSSO by using the `ClientRegistration` resource. For more information, see ClientRegistration API for AppSSO.
Resources	n/a	Resources map to OAuth2 scopes and are tied to a Cloud Foundry space. AppSSO does not support the concept of resource. For more information, see Unsupported features.
Application	Workload	This is not AppSSO specific but applicable to both Tanzu Application Service and Tanzu Application Platform.
Service Instance	n/a	In SSO for TAS, this is the way of making a Service Plan visible in a Cloud Foundry Space. There are no direct equivalents in AppSSO.
Service Binding	Service Claim by using the `ClassClaim` resource	Service Binding binds an app to a service, creating credentials for this app and injecting those credentials. In Tanzu Application Service, this is done by using the `VCAP_SERVICES` environment variable. In Tanzu Application Platform, this is done by using Kubernetes Service Bindings. AppSSO creates a `ClientRegistration` in the background and a `Secret` holding the credentials.
Service Key	`ClientRegistration` resource	AppSSO generates a `Secret` for the given `ClientRegistration`.
UAA (one per Cloud Foundry foundation)	The running `AuthServer` Deployment. One per `AuthServer` resource means one per “Service Plan” or one per Identity Zone.	The UAA is multi-tenant. The `AuthServer` is single-tenant, and one `AuthServer` is used per UAA tenant.

Unsupported features

The following features available in Single Sign-On for VMware Tanzu Application Service (SSO for TAS) are not supported in Application Single Sign-On for VMware Tanzu Application Platform

(AppSSO for TAP):

- **UAA Internal user store:** AppSSO does not support creating and managing users in an internal user database.
- **Security Assertion Markup Language (SAML) identity providers:** AppSSO does not support SAML external identity providers. It only supports OpenID Connect and Lightweight Directory Access Protocol (LDAP).
- **User management:** AppSSO does not support managing individual users. In SSO for TAS, users from an external Identity Provider are synchronized to the user database when they first log in. Permissions or scopes can be assigned to each user. This is not supported in AppSSO, all scopes come from roles to scopes mapping. For more information, see Mapping individual roles into authorization scopes.

  Important: Users cannot manage their own accounts in AppSSO. When allowed, they can manage their accounts in the External Identity Provider.
- **Single logout:** AppSSO does not support single logout.
- **OAuth2 grant types:** According to OAuth 2 Security best practices, AppSSO does not support the `password` or `implicit` grant types. You can migrate your application to the `authorization_code` grant type.
- **Selective auto-approval of scopes:** Consent is optional for all scopes except `openid`.
- **Resource management:** AppSSO operates solely on OAuth2 scopes, without using the concepts of permissions or resources.
  - In SSO for TAS, Operators control the availability of scopes for a specific Service Plan in a space using the Developer Dashboard. For more information, see the Single Sign-On for VMware Tanzu Application Service documentation.

    When an app is bound by using a Service Binding, an OAuth2 Client is created for that app in User Account and Authentication (UAA). The client can only request scopes that match the permissions pre-configured by operators.
  - In AppSSO, Developers can specify OAuth2 scopes when creating a `ClientRegistration` or a `ClassClaim`, with role management aggregated in the well-known roles of the Tanzu Application Platform. For more information, see RBAC for AppSSO.
- **Unsecured Lightweight Directory Access Protocol (LDAP):** LDAP traffic must be secured with TLS by using the `ldaps://` protocol.

Migration prerequisites

The migration scripts use:

- ytt for templating Kubernetes resources.
- uaac for extracting data from the UAA.
- python v3.9 and later for migrating data.
- several utilities, such as kubectl and jq.

If you do not already have an admin client for your UAA Identity Zone, you must create an UAA Admin client first. For more information, see the Single Sign-On for VMware Tanzu Application Service documentation

Migrate Service Plans to AuthServer custom resources

List all service plans by subdomain:

```
uaac curl "/identity-zones" -b | jq -r ".[].subdomain"
export SUBDOMAIN=<your-subdomain>
```

Identity-zone level configuration

The `AuthServer` specification only includes equivalents for the following configurable entries:

- Token expiration times.
- Cross-origin resource sharing (CORS) configuration.

By default, they are configured in the UAA Service Plan, and can be overridden in each individual plan.

The AuthServer definition includes default values. You can skip this section if you don't want to replicate the exact settings.

Cross-origin resource sharing (CORS) configuration

Cross-origin resource sharing (CORS) configuration is available under `config.corsPolicy` in the UAA as follows:

```
uaac curl "/identity-zones" -b | jq ".[] | select(.subdomain == \"$SUBDOMAIN\") | .con
fig.corsPolicy"
```

You can port the values into AppSSO's `AuthServer` by using `AuthServer.spec.cors`. For more information, see Public clients and CORS for AppSSO, or the `kubectl explain AuthServer.spec.cors` API documentation.

Token configuration

Token configuration is available under `config.tokenPolicy` in the UAA as follows:

```
uaac curl "/identity-zones" -b | jq ".[] | select(.subdomain == \"$SUBDOMAIN\") | .con
fig.tokenPolicy"
```

Fields can be `null` or equal to `-1`, indicating configuration at the UAA level. In such cases, you can retrieve the configuration by using:

```
uaac curl "/identity-zones" -b | jq ".[] | select(.subdomain == \"\") | .config.tokenP
olicy"
```

You can port the values into AppSSO's `AuthServer` by using `AuthServer.spec.token`. For more information, see Token settings for Application Single Sign-On, or the `kubectl explain AuthServer.spec.token` API documentation.

Migrate Identity Providers

Most of the configuration for OpenID Connect (OIDC) or LDAP-type identity providers can be ported automatically.

1. Log into the UAA and get a token for the admin client by using `uaac`.
2. Copy the UAA URL and the access token from `uaac context`:

   ```
   $ uaac context

   [1]*[➡ UAA_URL]
   ```

```
  [0]*[...]
      client_id: ...
      access_token: ➡ ACCESS_TOKEN
      token_type: ...
      expires_in: ...
      scope: ...
      jti: ...
```

3. Save the following script as `migration.py`.

```
#!/usr/bin/env python3

import getopt
import json
import sys
from urllib.request import Request, urlopen


def transform_ldap_providers(providers, external_group_mappings=[]):
    def to_ldap(provider):
        config = provider["config"]
        result = {
            "name": "ldap",
            "ldap": {
                "url": config["baseUrl"],
                "bind": {
                    "dn": config["bindUserDn"],
                    "passwordRef": {"name": "ldap-password"},
                },
                "user": {
                    "searchBase": config["userSearchBase"],
                    "searchFilter": config["userSearchFilter"],
                },
            },
        }

        roles_config = __extract_ldap_roles_config(config)
        if roles_config:
            result["ldap"]["roles"] = roles_config

        id_token_config = __extract_id_token_config(config)
        if id_token_config:
            result["ldap"]["idToken"] = id_token_config
        access_token_config = __extract_access_token_config(
            provider["originKey"], external_group_mappings
        )
        if access_token_config:
            result["ldap"]["accessToken"] = access_token_config

        return result

    return [to_ldap(provider) for provider in providers if provider["type"] ==
"ldap"]


def transform_openid_providers(providers, external_group_mappings=[]):
    """
    Transform UAA openID providers identity providers into
    AuthServer.spec.identityProviders.openID
    """

    def to_openid(provider):
        """
        Map an individual provider
        """
        config = provider["config"]
```

```
        result = {
            "name": provider["originKey"],
            "openID": {
                "clientID": config["relyingPartyId"],
                "scopes": config["scopes"],
                "displayName": config["providerDescription"],
                "clientSecretRef": {"name": provider["originKey"] + "-credentia
ls"},
            },
        }
        openId = result["openID"]
        if config["discoveryUrl"] is not None:
            openId["configurationURI"] = config["discoveryUrl"]
        if config["authUrl"] is not None:
            openId["authorizationUri"] = config["authUrl"]
        if config["tokenUrl"] is not None:
            openId["tokenUri"] = config["tokenUrl"]
        if config["userInfoUrl"] is not None:
            openId["userinfoUri"] = config["userInfoUrl"]
        if config["tokenKeyUrl"] is not None:
            openId["jwksUri"] = config["tokenKeyUrl"]

        roles_config = __extract_openid_roles_config(config)
        if roles_config:
            openId["roles"] = roles_config

        id_token_config = __extract_id_token_config(config)
        if id_token_config:
            openId["idToken"] = id_token_config

        access_token_config = __extract_access_token_config(
            provider["originKey"], external_group_mappings
        )
        if access_token_config:
            openId["accessToken"] = access_token_config

        return result

    return [to_openid(p) for p in providers if p["type"] == "oidc1.0"]


def __extract_group_filters(config):
    def to_group_filter(group):
        """
        Helper function for filters
        """
        if "*" in group:
            return {"regex": group.replace("*", ".*")}
        else:
            return {"exactMatch": group}

    if (
        config["externalGroupsWhitelist"] == ["*"]
        or not config["externalGroupsWhitelist"]
    ):
        return None
    return [to_group_filter(group) for group in config["externalGroupsWhitelis
t"]]


def __extract_openid_roles_config(config):
    """
    Extracts the UAA external groups and filtering rules,
    and modify them to match AuthServer.spec.identityProviders.openID.roles
    """
    if "external_groups" not in config["attributeMappings"]:
```

```
        return None
    result = {"fromUpstream": {"claim": config["attributeMappings"]["external_g
roups"]}}
    group_filter = __extract_group_filters(config)
    if group_filter:
        result["filterBy"] = group_filter

    return result


def __extract_ldap_roles_config(config):
    if not config["groupSearchBase"]:
        return None
    if config["groupSearchBase"] == "memberOf":
        return {"fromUpstream": {"attribute": "memberOf"}}

    result = {
        "fromUpstream": {
            "search": {
                "base": config["groupSearchBase"],
                "filter": config["groupSearchFilter"],
            },
        }
    }
    group_filter = __extract_group_filters(config)
    if group_filter:
        result["filterBy"] = group_filter

    return result


def __extract_id_token_config(config):
    """
    Extracts the UAA attributes mapping, and modify them
    to match AuthServer.spec.identityProviders.*.idToken
    """
    claims_mapping = [
        {"toClaim": key.replace("user.attribute.", ""), "fromUpstream": value}
        for key, value in config["attributeMappings"].items()
        if key.startswith("user.attribute")
    ]
    if len(claims_mapping) == 0:
        return None

    return {"claims": claims_mapping}


def __extract_access_token_config(originKey, group_mappings):
    """
    Filter the UAA /Groups/External by Identity Provider "originKey",
    group them by "external group name", and merge them into
    AuthServer.spec.identityProviders.*.accessToken
    """
    roles_to_groups = {}
    for mapping in group_mappings:
        if mapping["origin"] != originKey:
            continue
        external_group = mapping["externalGroup"]
        if not external_group in roles_to_groups:
            roles_to_groups[external_group] = {
                "fromRole": external_group,
                "toScopes": [],
            }
        roles_to_groups[external_group]["toScopes"].append(mapping["displayNam
e"])
    if len(roles_to_groups) == 0:
```

```
        return None
    return {"scope": {"rolesToScopes": list(roles_to_groups.values())}}


class UaaClient:
    def __init__(self, url, token, subdomain):
        self.url = url
        self.token = token
        self.subdomain = subdomain

    def get_idp(self):
        r = Request(
            f"{self.url}/identity-providers?rawConfig=true",
            headers={
                "Authorization": f"Bearer {self.token}",
                "X-Identity-Zone-Subdomain": self.subdomain,
            },
        )
        with urlopen(r) as response:
            return json.loads(response.read())

    def get_groups(self):
        r = Request(
            f"{self.url}/Groups/External?count=500",
            headers={
                "Authorization": f"Bearer {self.token}",
                "X-Identity-Zone-Subdomain": self.subdomain,
            },
        )
        with urlopen(r) as response:
            return json.loads(response.read())["resources"]


def extract_cli_args(argv):
    subdomain = None
    token = None
    url = None
    try:
        opts, _ = getopt.getopt(argv[1:], "hu:s:t:", ["url=", "subdomain=", "to
ken="])
    except getopt.GetoptError:
        print("idzone-to-authserver.py -u <uaa url> -t <uaa token> -s <subdomai
n>")
        sys.exit(2)
    for opt, arg in opts:
        if opt == "-h":
            print("idzone-to-authserver.py -u <uaa url> -t <uaa token> -s <subd
omain>")
            sys.exit()
        elif opt in ("-s", "--subdomain"):
            subdomain = arg
        elif opt in ("-t", "--token"):
            token = arg
        elif opt in ("-u", "--url"):
            url = arg
    return url, token, subdomain


if __name__ == "__main__":
    uaa = UaaClient(*extract_cli_args(sys.argv))
    uaa_idps = uaa.get_idp()
    uaa_groups = uaa.get_groups()
    idps = [
        *transform_openid_providers(uaa_idps, uaa_groups),
        *transform_ldap_providers(uaa_idps, uaa_groups),
```

```
    ]
    print(json.dumps(idps))
```

4. Run the script with the UAA URL, access token, and the previously captured `SUBDOMAIN`.

```
python3 migration.py -u $UAA_URL -t $TOKEN -s $SUBDOMAIN
```

The following JSON output can be templated into an `AuthServer` by using tools such as YTT. This structure is nearly complete for establishing a functioning authorization server with migrated identity providers.

```
#@ load("@ytt:data", "data")
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: MY-AUTH-SERVER
  namespace: MY-NAMESPACE
  labels:
    name: MY-AUTH-SERVER
spec:
  #! Token signature keys are managed in the UAA Config and not accessible thro
ugh
  #! the API. These are UAA-level, not id-zone level config options.
  #! The signing key is required.
  tokenSignature:
    signAndVerifyKeyRef:
      name: my-token-signing-key
  identityProviders: #@ data.values.identityProviders

  #! All other properties are omitted, because they are optional. You can updat
e them as needed.

---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: my-token-signing-key
  namespace: MY-NAMESPACE
spec:
  secretTemplate:
    type: Opaque
    stringData:
      key.pem: $(privateKey)
      pub.pem: $(publicKey)
```

5. Chain the migration script into YTT and obtain an `AuthServer` configuration.

```
ytt \
  --data-value-yaml identityProviders="$(python3 migration.py -u "$UAA_URL" -s
"$SUBDOMAIN" -t "$ACCESS_TOKEN")" \
  --file authserver-template.yaml
```

You must manually input the credentials for connecting to the upstream identity providers, including the bind password for LDAP and the client secret for OIDC, because the UAA does not expose such secrets. You can either create new OIDC clients or edit existing ones to include the `AuthServer` redirect URIs. For more information about configuring properties for AppSSO identity providers, see Identity providers for AppSSO. The secrets adhere to the following format:

```
#! For LDAP-type identity provider:
---
```

```
apiVersion: v1
kind: Secret
metadata:
  name: ldap-password
  namespace: MY-NAMESPACE
stringData:
  password: MY-PASSWORD

#! One secret per openID identity provider and it must match clientSecretRef.na
me:
---
apiVersion: v1
kind: Secret
metadata:
  name: MY-OIDC-IDENTITY-PROVIDER
  namespace: MY-NAMESPACE
stringData:
  clientSecret: MY-CLIENT-SECRET
```

Migrate OAuth2 Clients

Migrating OAuth2 Clients, including Service Bindings, Service Keys, Applications created in the SSO dashboard, are more difficult to automate, because the configuration model is very different. It is likely that the domains in the clients `redirect_uri`s differ from platform to platform.

The recommended approach to use Service Bindings in Tanzu Application Platform is to use an AppSSO `ClassClaim`. For more information, see Claim credentials for an Application Single Sign-On service offering. The `ClassClaim` is bound to a workload, but a UAA client lacks a direct binding to the UAA. The connection is established solely through a layer in Cloud Foundry.

For Service Keys or Applications created in the dashboard, the common use case is to copy the credentials returned in the create service key call, and then use them either in Cloud Foundry apps or off-platform. For those use cases, you can use a `ClientRegistration` instead of a `ClassClaim`. For more information, see ClientRegistration API for AppSSO.

Automation is possible to a certain extent, but manual intervention is required. The following script retrieves a list of all clients for your Identity Zone. You can save the script as `get-clients.sh`:

```
uaac curl "/oauth/clients" -H"X-Identity-Zone-Subdomain: $SUBDOMAIN" -b |
  jq ".resources | map(
{
  apiVersion: \"services.apps.tanzu.vmware.com/v1alpha1\",
  kind: \"ClientRegistration\",
  metadata: {
    name: .name,
    namespace: \"PLACEHOLDER\",
  },
  spec: {
    authServerSelector: {
      matchLabels: {
        PLACEHOLDER: \"PLACEHOLDER\",
      },
    },
    redirectUris: .redirect_uri,
    scopes: .scope | map({name: .}),
    authorization_grant_types: .authorized_grant_types,
  }
})
"
```

Store the following content in `client-registration-template.yaml`:

```
#@ load("@ytt:data", "data")
#@ load("@ytt:template", "template")

#@ for client in data.values.clients:
---
_: #@ template.replace(client)
#@ end
```

Use YTT to convert clients into placeholders for `ClientRegistration`s:

```
ytt -f client-registration-template.yaml --data-value-yaml clients="$(./get-clients.s
h)"
```

Service-to-Service flows

In SSO for TAS, you can register some applications to use a Service-to-Service flow. This corresponds to the OAuth2 `client_credentials` grant, where no human interaction is required to obtain tokens.

In this case, you must use a `ClientRegistration` described in Migrate OAuth2 Clients, with `ClientRegistration.spec.authorizationGrantTypes=["client_credentials"]`. You can use the resulting client id and client secret such as a Service-to-Service app uses them in Tanzu Application Platform.

When creating a `ClientRegistration` name “example-cr”, a corresponding Kubernetes `Secret` is created in the same namespace, with the name “example-cr”. The credentials are available in this secret under the fields `client-id` and `client-secret`.

Spring Boot application

You have the option to deploy your Spring Boot application with or without Spring Cloud Bindings

With Spring Cloud Bindings
Spring Boot applications deployed to Tanzu Application Service likely use the java-cfenv library to read service bindings data from `VCAP_SERVICES`. In Tanzu Application Platform, `java-cfenv` is not applicable. You can remove it from your dependencies.

If you build your Spring Boot app by using buildpacks, such as Tanzu Build Service, Tanzu Application Platform Supply Chains, Spring Boot maven, or gradle plugins, Spring Cloud Bindings are added to your path and perform the binding automatically. For more information, see Secure a Spring Boot workload. VMware recommends that your application uses Spring Boot v2.7, so that `ClientRegistration.spec.client_authentication_method` maps to a value known to Spring Boot, such as `client_secret_basic` or `client_secret_post`. Older versions of Spring Boot expect the value to be either `basic` or `post`.

Spring Cloud Bindings is not intended to work with existing configuration values in `spring.security.oauth2.client`. Instead, it serves as an alternative mechanism for automatic population of these values. The recommended approach is to create a `local` or similar profile for local development, tailored to the authorization server used in the development environment. In your standard application.yaml, refrain from populating `spring.security.oauth2.client`. Including default values in your `application.yaml` might cause inconsistent behavior, because Spring Cloud Bindings can override some of the values. For more information, see the appsso-starter-java accelerator documentation in GitHub.

Without Spring Cloud Bindings
If you are unable or prefer not to depend on Spring Cloud Bindings, see Secure a workload with Application Single Sign-On, which explains how credentials are exposed to the underlying application.

To use multiple `ClientRegistration`s in a single app, or a `ClientRegistration` with multiple grant types, such as `authorization_code` and `client_credentials`, you can turn off the Spring Cloud Bindings integration by setting the following configuration property: `org.springframework.cloud.bindings.boot.oauth2.enable=false`. To access the bindings, you can either load the Secret values in environment variables, or read the files populated by the Secret in the pod. You can still use the Spring Cloud Bindings library to access the files as follows:

```
class Example {

   public void exampleMethod() {
       // ...
       var binding = new Bindings().findBinding("MY-BINDING-NAME");
       String issuerUri = binding.get("issuer-uri");
       String clientId = binding.get("client-id");

       // or
       var binding = new Bindings().filterBindings("oauth2");
       // ...
   }

}
```

Migrate Spring Cloud Gateway apps to Tanzu Application Platform

This topic tells you how to migrate to Spring Cloud Gateway (SCG) for Kubernetes from a Tanzu Application Service (commonly known as TAS) offering of SCG or an open-source implementation of SCG. An example `animal-rescue` app is used to illustrate the high-level steps.

Deploy the `animal-rescue` app to Tanzu Application Service

This section describes, at a high level, the steps for deploying an example `animal-rescue` app and configuring SCG on TAS. For more detailed instructions, see the SCG documentation.

Create an SCG instance

Create an instance of SCG from Marketplace by running `cf create-service`.

```
cf create-service  p.gateway standard GATEWAY-NAME -c GATEWAY-CONFIG-FILE
```

Where:

- `GATEWAY-NAME` is the name of the SCG instance
- `GATEWAY-CONFIG-FILE` is the SCG instance configuration file

For more information, see app-gateway-config.json on GitHub.

Bind the SCG service to your app

You can directly bind the SCG service to the app or you can update the route configurations with the Service Broker API. For more information about configuring routes, see the SCG documentation.

Binding

To directly bind the SCG service to your app:

1. Bind the SCG service to your app by running:

   ```
   cf bind-service APP-NAME GATEWAY-NAME -c ROUTE-CONFIG-FILE
   ```

   Where:

   - `APP-NAME` is the app name.
   - `GATEWAY-NAME` is the name of the SCG instance created in the previous step
   - `ROUTE-CONFIG-FILE` is the route configuration file used for configuring routes of the app

2. Restart your app by running:

   ```
   cf restart $APP_NAME
   ```

For reference here and here are the configuration files used for configuring front-end and back-end routes with SCG service in the `animal-rescue` app.

Update the route config

To use curl to dynamically update route configurations with the Service Broker API:

1. Retrieve the GUID representing the `animal-rescue` app by running:

   ```
   cf app animals --guid
   6e41a492-a17c-4b27-83b0-78635baa54b4
   ```

2. Obtain the URL of a service instance's backing app for `my-gateway` by running:

   ```
   cf service my-gateway
   ```

   Example:

   ```
   $ cf service my-gateway
   Showing info of service my-gateway in org myorg / space dev as user...

   name:             my-gateway
   service:          p.gateway
   tags:
   plan:             standard
   description:      Spring Cloud Gateway for VMware Tanzu
   documentation:
   dashboard:        https://my-gateway.apps.example.com/scg-dashboard
   ```

3. Copy the URL given for the dashboard and remove the `/scg-dashboard` path. This is the URL of the service instance's backing app. For example, `https://my-gateway.apps.example.com`.

4. Update route configuration for the `animal-rescue` app through the routes actuator endpoint on the `my-gateway` service instance by running:

   ```
   curl -k -X PUT https://my-gateway.apps.example.com/actuator/bound-apps/6e41a4
   92-a17c-4b27-83b0-78635baa54b4/routes \
   -d "@./gateway-route-config.json" -H "Authorization: $(cf oauth-token)" -H "C
   ontent-Type: application/json"
   ...
   < HTTP/1.1 204 No Content
   ...
   ```

Deploy the `animal-rescue` app to Tanzu Application Platform

This section describes, at a high level, the steps for deploying an `animal-rescue` app and configuring the SCG on Tanzu Application Platform. For detailed instructions, see the `animal-rescue` app README file in GitHub.

Create an SCG instance

To create an SCG instance:

1. Create a file called `gateway-config.yaml` containing the following YAML definition:

```
---
apiVersion: "tanzu.vmware.com/v1"
kind: SpringCloudGateway
metadata:
  name: my-gateway
spec:
  count: 1
```

2. Apply this definition to your Kubernetes cluster by running:

```
kubectl apply --filename gateway-config.yaml
```

Example:

```
$ kubectl get springcloudgateway my-gateway
NAME               READY   REASON
my-gateway         True    Created
```

Create a route configuration

The SCG tile service has a JSON-based API route configuration approach. There are many similarities in terms of the configuration options available to those in Spring Cloud Gateway for Kubernetes.

1. Create a file called `animal-rescue-backend-route-config.yaml` with the following definition:

```
---
apiVersion: "tanzu.vmware.com/v1"
kind: SpringCloudGatewayRouteConfig
metadata:
  name: animal-rescue-backend-route-config
spec:
  service:
    name: animal-rescue-backend
  routes:
    - predicates:
        - Path=/api/**
      filters:
        - StripPrefix=1
```

2. Apply this definition to your Kubernetes cluster by running:

```
kubectl apply --filename animal-rescue-backend-route-config.yaml
```

Create an SCG mapping

In TAS, the SCG tile services use the Service Broker API and service-binding approach for apps to set up container-network routing of requests to a service instance.

Spring Cloud Gateway for Kubernetes provides a `SpringCloudGatewayMapping` resource definition to expose API routes on to an API SCG instance. For more information, see the Spring Cloud Gateway for Kubernetes documentation.

To create an SCG mapping:

1. Create another file called `animal-rescue-backend-mapping.yaml` with the following definition:

   ```
   ---
   apiVersion: "tanzu.vmware.com/v1"
   kind: SpringCloudGatewayMapping
   metadata:
     name: animal-rescue-backend-mapping
   spec:
     gatewayRef:
       name: my-gateway
     routeConfigRef:
       name: animal-rescue-backend-route-config
   ```

2. Apply `animal-rescue-backend-mapping.yaml` to your Kubernetes cluster by running:

   ```
   kubectl apply --filename animal-rescue-backend-mapping.yaml
   ```

   If your cluster has an ingress configured then, if `my-gateway` is accessible through the ingress with a fully-qualified domain name of `my-gateway.my-example-domain.com`, the `animal-rescue` back-end API is available in the path `my-gateway.my-example-domain.com/api/...`.

   One of the endpoints available in the sample app is `GET /api/animals`, which lists all of the animals available for adoption.

3. From curl, verify that this endpoint is accessible by running:

   ```
   curl my-gateway.my-example-domain.com/api/animals
   ```

Migrate Spring Config Server apps to Tanzu Application Platform

Tanzu Application Service (commonly known as TAS) and Tanzu Application Platform (commonly known as TAP) both enable you to load Java properties into an app from a configuration repository.

You can migrate an app running on TAS with Spring Configuration Service (SCS) to Tanzu Application Platform with Application Configuration Service (ACS).

This migration does not require changes to the app code or the configuration repository where properties are stored. However, the mechanism for loading properties into a Tanzu Application Platform workload is quite different and requires a substantially different setup.

At a high level SCS and ACS operate as follows:

- SCS instances run as web apps serving configuration by using REST requests. TAS apps bound to SCS instances fetch configuration using an autowired `ConfigServerConfigDataLoader` bean and add it to the app properties.
- ACS `ConfigurationServer` instances are Kubernetes custom resources, reconciled by a controller manager. These resources are wired into Tanzu Application Platform workloads using a `ConfigurationSlice` object. Each `ConfigurationSlice` creates a bindable secret

object containing properties, which is mounted to the app file system and loaded into the Spring app from there.

This topic uses the cook sample app to demonstrate how to migrate.

Deploy the `cook` app to Tanzu Application Service

This section describes, at a high level, the steps for deploying an example `cook` app on Tanzu Application Service.

To deploy the app:

1. Create an SCS instance that points to the cook-config GitHub repository by running:

```
cf create-service -c '{ "git": { "uri": "https://github.com/spring-cloud-servic
es-samples/cook-config", \
"label": "main"  } }' ${service_name} standard cook-config-server
```

2. Run `cf push` to push the app to TAS with an app manifest that binds to the service instance:

```
---
applications:
  - name: cook
    host: cookie
    instances: 1
    memory: 1G
    services:
      - cook-config-server
    env:
      SPRING_PROFILES_ACTIVE: development
      JBP_CONFIG_OPEN_JDK_JRE: '{ jre: { version: 17.+ }}'
```

For more detailed instructions, see the README file.

Migrate from TAS to Tanzu Application Platform

This section describes the areas to translate when creating a Tanzu Application Platform workload that will deploy the same app.

Create a Config Server instance

TAS/SCS

Create a service instance by running `cf create-service`, specifying the URI and label (branch) of the `cook-config` repository in the configuration JSON:

```
cf create-service -c '{ "git": { "uri": "https://github.com/spring-cloud-services-sa
mples/cook-config", \
"label": "main"  } }' p.config-server standard cook-config-server
```

Tanzu Application Platform/ACS

Create a `ConfigurationSource` resource, specifying the location of the `cook-config` repository in the `spec.backends` property. For detailed options, see the ACS documentation.

Specify the app name and Spring profiles

TAS/SCS

To specify the app name and Spring profiles:

1. Specify the app name is specified in the `spring.application.name` property. For the `cook` app this is found in application.yml.
2. Set the `SPRING_PROFILES_ACTIVE` environment variable in the `env` key in the app manifest to designate profiles. Multiple entries must be comma-separated.

Tanzu Application Platform/ACS

To specify the app name and Spring profiles:

1. Create a `ConfigurationSlice` resource, specifying the name of the `ConfigurationSource` resource in the `spec.configurationSource` property.
2. Create one or more entries in the `spec.content` array property of the `ConfigurationSlice` resource with the format `APP-NAME/PROFILE-NAME/OPTIONAL-LABEL` where:
    - `APP-NAME` is the app name.
    - `PROFILE-NAME` is the profile name.
    - `OPTIONAL-LABEL` is an optional comma-separated list of labels from which to retrieve properties.

For more information, see the ACS documentation.

Bind the Config Server to the app

TAS/SCS

To bind the Config Server to the app, either add the service-instance name in the services key in the app manifest, or run `cf bind-service` to bind the service to the app.

Tanzu Application Platform/ACS

To bind the Config Server to the app:

1. Create a `ResourceClaim` resource, specifying the details of the `ConfigurationSlice` resource in the `spec.ref` section of the configuration.
2. Specify the `ResourceClaim` resource in the `spec.serviceClaims` section of the workload. This mounts the properties to the running pod's file system.
3. Specify a `SPRING_CONFIG_IMPORT` property in the `spec.env` section of the workload with the value `optional:configtree:${SERVICE_BINDING_ROOT}/CLAIM-NAME/`, where `CLAIM-NAME` is the name you specified in the `spec.serviceClaims` list.

    This environment variable tells the Spring runtime to pull properties from the file system at the path where the `serviceClaims` mechanism mounted them. For more information, see ACS documentation.

Deploy the `cook` app to Tanzu Application Platform

Apply the following resources to Tanzu Application Platform to run the `cook` sample:

```
apiVersion: "config.apps.tanzu.vmware.com/v1alpha4"
kind: ConfigurationSource
metadata:
  name: cook-config-source
spec:
  backends:
    - type: git
      uri: https://github.com/spring-cloud-services-samples/cook-config
---
```

```
apiVersion: "config.apps.tanzu.vmware.com/v1alpha4"
kind: ConfigurationSlice
metadata:
  name: cook-config-slice
spec:
  configurationSource: cook-config-source
  content:
  - cook/production/tap
  - cook/default
---
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ResourceClaim
metadata:
  name: cook-config-claim
spec:
  ref:
    apiVersion: config.apps.tanzu.vmware.com/v1alpha4
    kind: ConfigurationSlice
    name: cook-config-slice
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: cook
  labels:
    app.kubernetes.io/part-of: cook-app
    apps.tanzu.vmware.com/has-tests: "true"
    apps.tanzu.vmware.com/workload-type: web
spec:
  serviceClaims:
  - name: spring-properties
    ref:
      apiVersion: services.apps.tanzu.vmware.com/v1alpha1
      kind: ResourceClaim
      name: cook-config
  build:
    env:
      - name: BP_JVM_VERSION
        value: "17"
  env:
    - name: SPRING_CONFIG_IMPORT
      value: "optional:configtree:${SERVICE_BINDING_ROOT}/spring-properties/"
    - name: SPRING_CLOUD_CONFIG_ENABLED
      value: "false"
    - name: SPRING_PROFILES_ACTIVE
      value: "development"
  source:
    git:
      url: https://github.com/spring-cloud-services-samples/cook.git
      ref:
        branch: tap
```

Configuration samples for deploying the `cook` app on Tanzu Application Platform are in the tap directory in GitHub.

Migrate Spring Service Registry apps to Tanzu Application Platform

This topic tells you how to migrate Spring Service Registry apps from Tanzu Application Service (commonly known as TAS) to Tanzu Application Platform (commonly known as TAP). This topic uses the sample Greeter app in the procedures to illustrate how migration works.

Tanzu Application Service and Tanzu Application Platform can both provide apps with an implementation of the Service Discovery pattern. You can migrate an app running on TAS with Spring Service Registry (SSR) to Tanzu Application Platform without changing the app code.

At a high level, SSR on Tanzu Application Service and SSR on Tanzu Application Platform operate as follows:

- **For Tanzu Application Service:**

  Creating a `service-registry` instance in a TAS space creates a `cf` app running a Eureka server.

  TAS apps bound to this service instance are autowired with a Spring Cloud Netflix Eureka client configuration bean. The presence of the `VCAP_APPLICATION` environment variable triggers the bean to examine the `VCAP_SERVICES` environment variable to inject `eureka.client.*` properties into the app by using connection information in `VCAP_SERVICES`.

- **For Tanzu Application Platform:**

  `EurekaServer` instances are Kubernetes custom resources. Each `EurekaServer` instance is reconciled into a Kubernetes deployment running a Eureka server with a bindable `Secret` object containing connection information for that Eureka server.

  When a `EurekaServer` is wired into Tanzu Application Platform workloads using a `ResourceClaim` object, it mounts the bindable `Secret` object into the application container's file system at a standard location.

  The spring-cloud-bindings library in GitHub finds any service bindings in this standard location and verifies that the binding is a Eureka server, based on a binding `type` key in the binding secret, and injects `eureka.client.*` properties into the app by using connection information in the mounted secret.

Deploy the `greeter` app to Tanzu Application Service

This section describes, at a high level, the steps for deploying an example `greeter` app on Tanzu Application Service.

To deploy the app:

1. Create an instance of the service registry product from Marketplace by running:

   ```
   cf create-service p.service-registry standard greeter-service-registry
   ```

2. Run `cf push` to push the apps to TAS with an app manifest that binds the service instance. Example:

   ```
   ---
   applications:
   - name: greeter-cs
     instances: 1
     memory: 1G
     services:
     - greeter-service-registry
     env:
       SPRING_PROFILES_ACTIVE: development
       JBP_CONFIG_OPEN_JDK_JRE: '{ jre: { version: 17.+ }}'

   - name: greeter-messages-cs
     instances: 1
     memory: 1G
     services:
     - greeter-service-registry
     env:
   ```

```
    SPRING_PROFILES_ACTIVE: development
    JBP_CONFIG_OPEN_JDK_JRE: '{ jre: { version: 17.+ }}'
```

For more information about the Greeter app, see its README file in GitHub.

For more information about Service Registry, see the TAS documentation.

Migrate from Tanzu Application Service to Tanzu Application Platform

This section describes the areas to translate when creating a Tanzu Application Platform workload that deploys the same app.

Create a database instance

Tanzu Application Service
Create a service instance by running:

```
cf create-service
```

Tanzu Application Platform
For Tanzu Application Platform, you must create a `EurekaServer` resource.

Bind the database instance to the app

Tanzu Application Service
Include the name of the service instance in the `services` key in the app manifest. Alternatively, use `cf bind-service` to bind the service to the app.

Tanzu Application Platform
Create a `ResourceClaim` resource, specifying the details of the `EurekaServer` resource in the `spec.ref` section of the configuration. Specify the `ResourceClaim` resource in the `spec.serviceClaims` section of the `Workload` resource.

Deploy the `greeter` app to Tanzu Application Platform

This section describes, at a high level, the steps for deploying an example `greeter` app on Tanzu Application Platform. For more information about how to deploy a service registry instance and configure workloads, see Overview of Service Registry.

To deploy the app:

1. Install the Eureka Service Registry package by running:

   ```
   tanzu package available list service-registry.spring.apps.tanzu.vmware.com --na
   mespace tap-install

   tanzu package install service-registry \
   --package service-registry.spring.apps.tanzu.vmware.com \
   --version VERSION -n tap-install

   tanzu package installed get service-registry -n tap-install
   ```

2. Create a `EurekaServer` resource by applying the following YAML to your Kubernetes cluster:

```
---
apiVersion: service-registry.spring.apps.tanzu.vmware.com/v1alpha1
kind: EurekaServer
metadata:
  name: eurekaserver-sample
  namespace: my-apps
spec:
  replicas: 2
```

A successful `EurekaServer` resource has a `Ready` condition set to `true` and a `status.binding.name` field pointing to a secret containing connection information.

3. Claim credentials by using `ResourceClaim`.

```
---
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ResourceClaim
metadata:
  name: eurekaserver-sample
  namespace: my-apps
spec:
  ref:
    apiVersion: service-registry.spring.apps.tanzu.vmware.com/v1alpha1
    kind: EurekaServer
    name: eurekaserver-sample
    namespace: my-apps
```

4. Configure workloads. The greeting app is a sample workload in GitHub. Deploy both `greeter` and `greeter-messages` by using the following YAML. The YAML claims `EurekaServer` credentials by adding a `spec.serviceClaims` section to each workload.

```
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: greeter-messages
  namespace: my-apps
  labels:
    apps.tanzu.vmware.com/workload-type: server
    apps.tanzu.vmware.com/has-tests: "true"
    app.kubernetes.io/part-of: greeter
spec:
  build:
    env:
      - name: BP_JVM_VERSION
        value: "17"
      - name: BP_GRADLE_BUILT_MODULE
        value: "greeter-messages"
      - name: BP_GRADLE_BUILD_ARGUMENTS
        value: "--no-daemon clean bootJar"
  env:
    - name: SPRING_PROFILES_ACTIVE
      value: "development"
  serviceClaims:
    - name: eureka
      ref:
        apiVersion: services.apps.tanzu.vmware.com/v1alpha1
        kind: ResourceClaim
        name: eurekaserver-sample
  source:
    git:
      url: https://github.com/spring-cloud-services-samples/greeting
      ref:
        branch: main
```

```
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: greeter
  namespace: my-apps
  labels:
    apps.tanzu.vmware.com/workload-type: web
    apps.tanzu.vmware.com/has-tests: "true"
    app.kubernetes.io/part-of: greeter
spec:
  build:
    env:
      - name: BP_JVM_VERSION
        value: "17"
      - name: BP_GRADLE_BUILT_MODULE
        value: "greeter"
      - name: BP_GRADLE_BUILD_ARGUMENTS
        value: "--no-daemon clean bootJar"
  env:
    - name: SPRING_PROFILES_ACTIVE
      value: "development"
  serviceClaims:
    - name: eureka
      ref:
        apiVersion: services.apps.tanzu.vmware.com/v1alpha1
        kind: ResourceClaim
        name: eurekaserver-sample
  source:
    git:
      url: https://github.com/spring-cloud-services-samples/greeting
      ref:
        branch: main
```

Use Service Registry with an executable JAR file app

In the `greeting` app example, `BP_GRADLE_BUILD_ARGUMENTS` is set to include the `bootJar` task in addition to the default Gradle build arguments. This setting is necessary for this example base because the `build.gradle` file contains a `jar` section and the Spring Boot buildpack does not inject the `spring-cloud-bindings` library into the app if it is an executable JAR file.

`spring-cloud-bindings` is required to process the `serviceClaim` into properties that tell the discovery client how to find the Eureka server.

To use Service Registry with an executable JAR file app, explicitly include spring-cloud-bindings v1.13.0 or later and set the `org.springframework.cloud.bindings.boot.enable=true` system property as described in the library README file in GitHub.

Migrate Spring Data Services apps to Tanzu Application Platform

This topic tells you how to migrate apps bound to Spring Data Services from Tanzu Application Service (commonly known as TAS) to Tanzu Application Platform (commonly known as TAP).

Both Tanzu Application Service and Tanzu Application Platform support applications with an implementation of the Service Discovery pattern. You can migrate an application using Spring Data Services from Tanzu Application Service to Tanzu Application Platform without changing the application code.

Deploy the `fortune-teller` app to Tanzu Application Service

This section describes, at a high level, the steps for deploying the example `fortune-teller` app on Tanzu Application Service. For more detailed instructions, see the fortune-teller README in GitHub.

To deploy the app:

1. Create an instance of MySQL database from the Marketplace by running:

   ```
   cf create-service  p.mysql db-small fortunes-db
   ```

2. Create an instance of Spring Config Server that points to the config JSON by running:

   ```
   cf create-service -c '{ "git": { "uri": "https://github.com/spring-cloud-servic
   es-samples/fortune-teller", "searchPaths": "configuration"  } }'p.config-server
   standard config-server
   ```

3. Create an instance of the Service Registry product from the Marketplace by running:

   ```
   cf create-service s p.service-registry standard service-registry
   ```

4. Push the applications to Tanzu Application Service using `cf push` with an application manifest that binds the service instance:

   ```
   ---
   applications:
   - name: fortune-service
     memory: 1024M
     host: fortunes
     path: fortune-teller-fortune-service/target/fortune-teller-fortune-service-0.
   0.1-SNAPSHOT.jar
     services:
     - fortunes-db
     - config-server
     - service-registry
     env:
       # JBP adds deprecated connectors when using mysql service
       JBP_CONFIG_SPRING_AUTO_RECONFIGURATION: '{"enabled":false}'
       # Replace with API URI of target PCF environment
       #CF_TARGET: https://api.yourpcfenvironment.local
   - name: fortune-ui
     memory: 1024M
     host: fortunes-ui
     path: fortune-teller-ui/target/fortune-teller-ui-0.0.1-SNAPSHOT.jar
     services:
     - config-server
     - service-registry
     env:
       # Replace with API URI of target PCF environment
       #CF_TARGET: https://api.yourpcfenvironment.local
   ```

Migrate from Tanzu Application Service to Tanzu Application Platform

This section describes the areas to translate when creating a Tanzu Application Platform workload that deploys the same app.

Create a database instance

Tanzu Application Service
Create a service instance using `cf create-service`.

Tanzu Application Platform
There are multiple options to create an instance. For more information, see Services Toolkit.

The easiest way to get started is to create a service instance using Bitnami Services. For more information, see Working with Bitnami Services.

Bind the database instance to the app

Tanzu Application Service

- Include the name of the service instance in the `services` key in the application manifest.
- Alternatively, use `cf bind-service` to bind the service to the application.

Tanzu Application Platform
If you are using Bitnami Services, Specify the `ClassClaim` resource in the `spec.serviceClaims` section of the `Workload`.

Deploy the `fortune-teller` application to Tanzu Application Platform

This section describes, at a high level, the steps for deploying an example `fortune-teller` app on Tanzu Application Platform.

For information about installing and configuring data services on Tanzu Application Platform, see the Services Toolkit documentation.

Prerequisites: To follow this example, you must first deploy Service Registry and Spring Config Server instances. To do so, apply the following YAML files in the fortune-teller repo:

- Install Service Registry: Apply the file eureka-server.yaml.
- Install Spring Config Server: Apply the file configuration-source.yaml.
- ConfigureSpring Config Server: Apply the file configuration-slice.yaml.
- Setup the `forture-teller` app to use Service Registry and Spring Config Server: Apply the file resource-claim.yaml.

Note

This prerequisite is not required for all Spring Data Services apps.

To deploy the app:

1. Claim MySQL service from Bitnami services by running:

```
tanzu service class-claim create fortune-database --class mysql-unmanaged -n my
-apps
tanzu services class-claims get fortune-database --namespace my-apps
```

2. Bind the service to your workload. Bind the MySQL service by adding `ClassClaim` under the `spec.serviceClaims` section as follows:

```
---
apiVersion: carto.run/v1alpha1
```

```
kind: Workload
metadata:
name: fortune-teller-fortune-service
namespace: my-apps
labels:
  apps.tanzu.vmware.com/workload-type: web
  apps.tanzu.vmware.com/has-tests: "true"
  app.kubernetes.io/part-of: fortune
spec:
params:
  - name: annotations
    value:
      autoscaling.knative.dev/minScale: "1"
build:
  env:
  - name: BP_JVM_VERSION
    value: "8"
  - name: BP_MAVEN_BUILT_MODULE
    value: "fortune-teller-fortune-service"
env:
  - name: SPRING_CONFIG_IMPORT
    value: "optional:configtree:${SERVICE_BINDING_ROOT}/spring-properties/"
serviceClaims:
  - name: eureka
    ref:
    apiVersion: services.apps.tanzu.vmware.com/v1alpha1
    kind: ResourceClaim
    name: eurekaserver-sample
  - name: db
    ref:
    apiVersion: services.apps.tanzu.vmware.com/v1alpha1
    kind: ClassClaim
    name: fortune-database
  - name: spring-properties
    ref:
    apiVersion: services.apps.tanzu.vmware.com/v1alpha1
    kind: ResourceClaim
    name: fortune-config
source:
  git:
  url: https://github.com/akrishna90/fortune-teller
  ref:
    branch: main
```

For more information about claiming an instance of MySQL using Bitnami services, see the tutorial Working with Bitnami Services.

Using Gradle for building Spring applications

By default, when the `bootJar` task is configured, the `jar` task is also configured and uses `plain` as the convention for its archive classifier. This ensures that `bootJar` and `jar` are executed and causes the executable archive and the plain archive to be built at the same time. For more information, see the Spring documentation.

Spring Boot buildpack behavior

The gradle buildpack produces both `<ARTIFACT_NAME>.jar` and `<ARTIFACT_NAME>-plain.jar`. The presence of a second JAR file in the build output means that the Gradle buildpack leaves both JAR files in compressed form instead of unpacking the `<ARTIFACT_NAME>.jar` file into the application directory. As a result, the spring-boot buildpack will fail to find a `<APPLICATION_ROOT>/META-INF/MANIFEST.MF` file and will fail to contribute the `spring-cloud-bindings` JAR file to the application. You will be able to run the application, but service bindings will not be processed.

To avoid this, deactivate the `jar` task in your `build.gradle` file to prevent the plain JAR from being created.

```
jar {
  enabled = false
}
```

Alternatively you can set `BP_GRADLE_ADDITIONAL_BUILD_ARGUMENTS="-x jar"` to prevent generating the plain JAR without modifying the `build.gradle` file.

Using Maven for building Spring applications

- No changes are required.
- When both Maven and Gradle builds are available, for example, in the case of the greeting repo, the Paketo Java buildpack checks Maven first and uses Maven to build the application.

Migrate to Cloud Native Buildpacks

This section tells you how to migrate from using Cloud Foundry buildpacks for Tanzu Application Service (commonly known as TAS for VMs) to using Cloud Native buildpacks for Tanzu Application Platform (commonly known as TAP).

In this section:

- Migrate to the .NET Core Cloud Native Buildpack
- Migrate from the Binary CF Buildpack to the Procfile Cloud Native Buildpack
- Migrate to the Go Cloud Native Buildpack
- Migrate to the Java Cloud Native Buildpack
- Migrate from the NGINX and Staticfile CF buildpack to the Web Server Cloud Native Buildpack
- Migrate to the Node.js Cloud Native Buildpack
- Migrate to the PHP Cloud Native Buildpack
- Migrate to the Python Cloud Native Buildpack
- Migrate to the Ruby Cloud Native Buildpack

Migrate to the .NET Core Cloud Native Buildpack

This topic tells you how to migrate your .NET Core app from using a Cloud Foundry buildpack for Tanzu Application Service (commonly known as TAS for VMs) to using a Cloud Native Buildpack for Tanzu Application Platform (commonly known as TAP).

Install specific .NET runtime and ASP.NET versions

The following table compares how Tanzu Application Service and Tanzu Application Platform handle installing specific versions.

Feature	Tanzu Application Service	Tanzu Application Platform
Detects version from `.csproj` `<RuntimeFrameworkVersion>` or `<TargetFramework>`		

Feature	Tanzu Application Service	Tanzu Application Platform
Detects version from `runtimeconfig.json` `runtimeOptions.framework.version`		
Detects versions from `.fsproj` and `.vbproj`.		
Override app-based version detection.	Use `builpack.yml`	Use `$BP_DOTNET_FRAMEWORK_VERSION`

Tanzu Application Service: Override version detection

Tanzu Application Service buildpacks allows you to specify a .NET Core SDK version using a `buildpack.yml`.

Example `buildpack.yml`:

```
dotnet-core:
  sdk: 7.0.x
```

Tanzu Application Platform: Override version detection

In Tanzu Application Platform, users set the `$BP_DOTNET_FRAMEWORK_VERSION` environment variable to specify which version of the .NET Core runtime to install. The buildpack automatically installs an SDK version that is compatible with the runtime version .NET Core runtime you selected.

The Tanzu Application Platform buildpack requires you to provide an exact version unlike in the Tanzu Application Service `buildpack.yml` where you can provide version patterns such as `7.x.x`.

Example `spec` section from a `workload.yaml`:

```
---
spec:
  build:
    env:
    - name: BP_DOTNET_FRAMEWORK_VERSION
       value: 7.0.10
  source:
    git:
      ref:
        branch: master
      url: https://github.com/cloudfoundry/dotnet-core-buildpack
    subPath: fixtures/source_apps/source-app-7
```

Configure multiple projects in an app

The following table compares how Tanzu Application Service and Tanzu Application Platform handle multiple projects in an app.

Feature	Tanzu Application Service	Tanzu Application Platform
Configure multiple projects in one app	Use a `.deployment` file	Use `$BP_DOTNET_PROJECT_PATH`
Projects referenced by the configured main project are also built		

Tanzu Application Service: Configure multiple projects

In Tanzu Application Service, users create a `.deployment` file in your app's root folder to designate the main project's path.

For example:

```
[config]
project = src/MyApp.Web/MyApp.Web.csproj
```

Tanzu Application Platform: Configure multiple projects

In Tanzu Application Platform, you configure the main project's path with `$BP_DOTNET_PROJECT_PATH` as the `spec` shown in this Tanzu Application Platform workload example. In Tanzu Application Platform, you point to the root of the project not the project file.

Example `spec` section from a `workload.yaml`:

```
---
spec:
  build:
    env:
    - name: BP_DOTNET_PROJECT_PATH
       value: ./src/asp_web_app
  source:
    git:
      ref:
        branch: master
      url: https://github.com/cloudfoundry/dotnet-core-buildpack
    subPath: fixtures/source_apps/multiple_projects_msbuild
```

Configure the publish command

The following table compares how you configure the publish command in Tanzu Application Service and Tanzu Application Platform.

Feature	Tanzu Application Service	Tanzu Application Platform
Configure the publish command	Not supported	Use `$BP_DOTNET_PUBLISH_FLAGS`

Configure the publish command in Tanzu Application Platform

Example `spec` section from a `workload.yaml`:

```
---
spec:
  build:
    env:
    - name: BP_DOTNET_PUBLISH_FLAGS
       value: "--verbosity=normal"
```

Provide a NuGet configuration

The following table compares how you provide a NuGet configuration in Tanzu Application Service and Tanzu Application Platform.

Feature	Tanzu Application Service	Tanzu Application Platform
Provide a NuGet.Config with the app		

Feature	Tanzu Application Service	Tanzu Application Platform
Provide a NuGet.Config through service bindings	Not supported	Use a binding of type `nugetconfig` that contains the `nuget.config`

Provide a NuGet configuration with sensitive data

In Tanzu Application Platform, if your NuGet config contains credentials or other sensitive data, you can provide it to the build without explicitly including the file in the application directory.

To provide your NuGet config without including it in the directory:

1. Create the service binding as a secret. For example:

```
---
apiVersion: v1
kind: Secret
metadata:
  name: nuget-config
  namespace: my-apps
type: service.binding/nugetconfig
stringData:
  type: nugetconfig
  nuget.config: |
    <?xml version="1.0" encoding="utf-8"?>
    <configuration>
      <packageSources>
        <clear />
        <add key="NuGet.org" value="https://api.nuget.org/v3/index.json" />
      </packageSources>
    </configuration>
```

2. Use the binding in the workload. For example:

```
---
kind: Workload
apiVersion: carto.run/v1alpha1
metadata:
name: dotnet-app
spec:
# ...
  params:
  - name: buildServiceBindings
    value:
      - name: nuget-config
        kind: Secret
        apiVersion: v1
```

For more information about service bindings, see Configure Tanzu Build Service properties on a workload.

Migrate from the Binary Cloud Foundry Buildpack to the Procfile Cloud Native Buildpack

This topic tells you how to migrate your binary app from using a Cloud Foundry buildpack for Tanzu Application Service (commonly known as TAS for VMs) to using a Cloud Native Buildpack for Tanzu Application Platform (commonly known as TAP).

The Tanzu Application Platform Procfile buildpack provides the capability corresponding to the Tanzu Application Service Binary buildpack.

In Tanzu Application Service, Procfile capability is built into the platform. For more information, see the Cloud Foundry documentation. However, due to the nature of the Cloud Foundry API, a no-op/null buildpack was required, and this null buildpack was named Binary buildpack.

A Procfile follows the same format in Tanzu Application Service and Tanzu Application Platform. If your app contains an executable file or a preceding buildpack generates an executable file in the build process, you can use the Tanzu Application Platform Procfile buildpack to run your app in the same way as the Tanzu Application Service Binary buildpack.

Example `spec` section from a `workload.yaml`:

```
---
spec:
  source:
    git:
      ref:
        branch: master
      url: https://github.com/cloudfoundry/binary-buildpack
    subPath: fixtures/default_app
```

Migrate to the Go Cloud Native Buildpack

This topic tells you how to migrate your Go app from using a Cloud Foundry buildpack for Tanzu Application Service (commonly known as TAS for VMs) to using a Cloud Native Buildpack for Tanzu Application Platform (commonly known as TAP).

Install a specific Go version

The following table compares how Tanzu Application Service and Tanzu Application Platform handle installing specific versions.

Feature	Tanzu Application Service	Tanzu Application Platform
Detects version from `go.mod`		
Override app-based version detection	Using `$GOVERSION`	Using `$BP_GO_VERSION`

Tanzu Application Service: Override version detection

To instruct the buildpack to select a specific version, set the environment variable `$GOVERSION`. For example, `$GOVERSION="go1.22"`.

Tanzu Application Platform: Override version detection

To configure the version, set the environment variable `$BP_GO_VERSION` in your `workload.yaml` file.

Example `spec` section from a `workload.yaml`:

```
---
spec:
  build:
    env:
    - name: BP_GO_VERSION
       value: "1.22.*"
```

Set LDFLAGS

The following table compares how you set LDFLAGS in Tanzu Application Service and Tanzu Application Platform.

Feature	Tanzu Application Service	Tanzu Application Platform
Set any LDFLAG		`$BP_GO_BUILD_LDFLAGS`
Set the -X LDFLAG	Using `buidpack.yml` OR `{$GO_LINKER_SYMBOL, $GO_LINKER_VALUE}`	`$BP_GO_BUILD_LDFLAGS`

Tanzu Application Service: Set LDFLAGS

In the Tanzu Application Service Go buildpack, you can only set the value of a variable when setting linker flags, that is, the `-X` flag.

Example Tanzu Application Service `buildpack.yml`:

```
---
go:
  ldflags:
    main.var1: val1
    main.var2: val2
```

Tanzu Application Platform: Set LDFLAGS

In Tanzu Application Platform Go buildpack, you can set any arbitrary LDFLAG.

Example `spec` section from a `workload.yaml`:

```
---
spec:
  build:
    env:
    - name: BP_GO_BUILD_LDFLAGS
       value: "-X 'main.var1=val1' -X 'main.var2=val2'"
```

Note

The Tanzu Application Service buildpack provided a special environment variable combination to set one variable value, `$GO_LINKER_SYMBOL` and `$GO_LINKER_VALUE`. If you used this combination, use the generic Tanzu Application Platform format shown in the preceding snippet for migration.

Custom build flags

The Tanzu Application Service buildpack did not allow users to set custom build flags other than the `-X ldflag`. A build flag of `"-tags cloudfoundry -buildmode pie"` is always added to the build in Tanzu Application Service.

In Tanzu Application Platform, you can set your own build flags with `$BP_GO_BUILD_FLAGS`, for example:

```
---
spec:
  build:
    env:
    - name: BP_GO_BUILD_FLAGS
       value: "-buildmode=default -tags=ilovetanzu"
```

For Static Stack in Tanzu Application Platform, unless you provide a `-buildmode` flag through `$BP_GO_BUILD_FLAGS`, the buildpack sets `CGO_ENABLED=0` and `-buildmode=default` build flag to enable support for the static stack by default.

Build non-default packages

The following table compares how Tanzu Application Service and Tanzu Application Platform handle building non-default packages.

Feature	Tanzu Application Service	Tanzu Application Platform
Set locations or targets of the Go programs to be compiled	`$GO_INSTALL_PACKAGE_SPEC`	`$BP_GO_TARGETS`

Tanzu Application Service: Build non-default packages

Example Tanzu Application Service `manifest.yml`:

```
---
applications:
- name: myapp
  env:
    GO_INSTALL_PACKAGE_SPEC: "./app1 ./app2"
```

Tanzu Application Platform: Build non-default packages

Example `spec` section from a `workload.yaml`:

```
---
spec:
  build:
    env:
    - name: BP_GO_TARGETS
       value: "./app1:./app2"
```

Pre-vendored apps based on Go mod

In Tanzu Application Service, you must set the `$GOPACKAGENAME` for go-mod based apps with dependencies vendored into a vendor directory. For more information, see the Cloud Foundry documentation.

In Tanzu Application Platform, this is not required because the Tanzu Application Platform Go buildpack can auto-detect your vendor directory, and use it if it exists.

Glide and Dep based apps

The Tanzu Application Platform Go buildpack does not support programs based on Glide and Go Dep dependency management tools. This is because the Go community has moved towards go modules. You must update your apps to use go modules.

Migrate to the Java Cloud Native Buildpack

This topic tells you how to migrate your Java app from using a Cloud Foundry buildpack for Tanzu Application Service (commonly known as TAS for VMs) to using a Cloud Native Buildpack for Tanzu Application Platform (commonly known as TAP).

Use a specific Java version

The following table compares how Tanzu Application Service and Tanzu Application Platform handle installing specific versions.

Feature (lowest to highest priority)	Tanzu Application Service	Tanzu Application Platform
Buildpack Default	8	11
Maven MANIFEST.MF property	Not supported	
Detects version from `.sdkmanrc` file	Not supported	
Override app-based version detection	Use `cf set-env`: `JBP_CONFIG_OPEN_JDK_JRE`	Use `$BP_JVM_VERSION`

Tanzu Application Service: Override version detection

In Tanzu Application Service, changing from the default JVM v8 requires you to configure the following app environment variable key and value:

```
JBP_CONFIG_OPEN_JDK_JRE '{ jre: { version: 17.+ }}'
```

This builds the app with the version of Java v17 that was bundled with the buildpack. Currently Java v8, v11, v17, and v21 are supported.

Tanzu Application Platform: Override version detection

In Tanzu Application Platform, set the `$BP_JVM_VERSION` build-time environment variable to specify which version of the JVM to install. The value can be the major version of one of the LTS Java releases included in the buildpack, currently 8, 11, 17, and 21. Patches for these majors are released quarterly along with OpenJDK release schedules. For the buildpack's exact versions, see the Java Buildpack Release Notes.

Example `spec` section from a `workload.yaml`:

```
---
spec:
  build:
    env:
    - name: BP_JVM_VERSION
       value: "17"
```

Configure Maven repository settings

The following table compares configuring Maven repository settings in Tanzu Application Service and Tanzu Application Platform.

Feature	Tanzu Application Service	Tanzu Application Platform
Connect to a private Maven repository	Not supported	Use a binding of type `maven` with key `settings.xml`

Configure Maven repository settings with sensitive data

In Tanzu Application Platform, if your Maven settings contain sensitive data, you can provide your own `settings.xml` to the build without explicitly including the file in the application directory.

To provide your `settings.xml` without including it in the directory:

1. Create the service binding as a secret. For example:

```
---
apiVersion: v1
kind: Secret
metadata:
  name: settings-xml
  namespace: my-apps
type: service.binding/maven
stringData:
  type: maven
  settings.xml: |
    <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 https://maven.
apache.org/xsd/settings-1.0.0.xsd">
      <servers>
        <server>
          <id>myrepo</id>
          <username>admin</username>
          <password>changeme</password>
        </server>
      </servers>

      <mirrors>
        <mirror>
          <id>myrepo</id>
          <name>myrepo</name>
          <url>http://myrepo/</url>
          <mirrorOf>*</mirrorOf>
        </mirror>
      </mirrors>
    </settings>
```

2. Apply the binding in the workload by running:

```
tanzu apps workload apply APP-NAME \
  --param-yaml buildServiceBindings='[{"name": "settings-xml", "kind": "Secre
t"}]'
```

   Where `APP-NAME` is the name of your app.

For more information about service bindings, see Configure Tanzu Build Service properties on a workload.

Service bindings

The following table compares service bindings in Tanzu Application Service and Tanzu Application Platform.

Feature	Tanzu Application Service	Tanzu Application Platform
Connect an app to a bound service		

Tanzu Application Service: Service bindings

In Tanzu Application Service, the Java Buildpack supplies Spring Boot v3 apps with the Java CF Env Library. This library parses the `VCAP_SERVICES` variable and allows the auto-configuration for Spring Boot to set properties and connect to the bound service.

Tanzu Application Platform: Service bindings

In Tanzu Application Platform, Tanzu Buildpacks supply a similar library, named Spring Cloud Bindings, which supports the Kubernetes-style service bindings. This does the following:

- Adds a PropertySource with a flattened representation, `k8s.bindings.{name}.*`, of the bindings.
- Adds a PropertySource with binding-specific Spring Boot configuration properties.

The auto-configuration for Spring Boot configures application properties appropriate for the type of binding encountered.

After you have applied a service binding, for example to a PostgreSQL database, Spring Cloud Bindings automatically uses the properties from the binding to set the relevant Spring Boot properties.

Example command to apply a service binding:

```
tanzu apps workload apply APP-NAME \
  --service-ref "psql=services.apps.tanzu.vmware.com/v1alpha1:ClassClaim:psql"
```

For more information about service bindings, see Configure Tanzu Build Service properties on a workload.

Deploy with Tomcat

The following table compares deploying with Tomcat for Tanzu Application Service and Tanzu Application Platform.

Feature	Tanzu Application Service	Tanzu Application Platform
Configure Tomcat Version	Not possible without creating a custom (unsupported) buildpack	Use `BP_TOMCAT_VERSION`
Use External Tomcat		

In Tanzu Application Platform, the Apache Tomcat buildpack includes versions 8, 9, and 10.1. The detection criteria to deploy with the buildpack-provided Tomcat remains the same as in Tanzu Application Service, which is the presence of a WEB-INF directory and no Main-Class entry in the manifest.

You can configure an external Tomcat instance by using environment variables. For more information about configuration options, see the Tanzu Java Buildpacks documentation.

Choose Java distribution

The following table compares deploying choosing a Java distribution for Tanzu Application Service and Tanzu Application Platform.

Feature	Tanzu Application Service	Tanzu Application Platform
Use an alternative JVM provider	Not possible without creating a custom (unsupported) buildpack	

Compare third-party integrations

The following table compares third-party integrations available for Tanzu Application Service and Tanzu Application Platform.

Partner	Tanzu Application Service	Tanzu Application Platform
Apache Skywalking		

Partner	Tanzu Application Service	Tanzu Application Platform
AppDynamics		
AppInternals (Aternity/Riverbed)		
AspectJ		
Azure App Insights		
Checkmarx		
Contrast Security		
Datadog		
Elastic APM		
Dynatrace		
Google Stackdriver		
Introscope (Broadcom)		?
JaCoCo		
Java Memory Assistant		
JProfiler		
JRebel		
Luna Sec Provider		
MariaDB Client PostgreSQL Client		See Notes
Metric Writer		See Notes
New Relic		
OpenTelemetry Agent		
ProtectApp (Thales)		
Sealights		
Seeker Security/Synopsys		
Spring Insight	See Notes	See Notes
Takipi		
YourKit		

Note

The following integrations do not have a Tanzu Application Platform equivalent:

- **MariaDB and PostgreSQL Client Libraries:** The recommended way to include these libraries is as an application dependency rather than supplied by the buildpack. Therefore in Tanzu Application Platform, included in the app, for example, using a `pom.xml` entry.
- **Metrics Writer:** This integration library provides Cloud Foundry-specific tags for micrometer enabled apps, and so does not have a Tanzu Application Platform equivalent.

- **Spring Insight:** This is no longer supported and is no longer available in Tanzu Application Service or Tanzu Application Platform.

Most of the third-party integrations that are supported in Tanzu Application Service are also supported in Tanzu Application Platform. The detection criteria to trigger enabling the integration, for example, adding a Java Agent to the JVM, is as follows:

- **Tanzu Application Service:** Bind a Tanzu Application Service service of the relevant type, or set an environment variable.
- **Tanzu Application Platform:** Provide a Tanzu Application Platform binding of the relevant type, or setting an environment variable.

Tanzu Application Platform only integrations

The following integrations are only available in Tanzu Application Platform through Cloud Native Buildpacks:

- **Checkmarx:** Contributes the Checkmarx CxIAST agent.
- **Snyk:** Contributes Snyk scanning and configures it to connect to the service.
- **Synopsys:** Contributes Synopsys scanning and configures it to connect to the service.

Configure debugging for your application

The following table compares configuring debugging for your app in Tanzu Application Service and Tanzu Application Platform.

Feature	Tanzu Application Service	Tanzu Application Platform
Enable Remote Debugging	`$JBP_CONFIG_DEBUG`	`$BPL_DEBUG_ENABLED`

Enabling remote debugging for Java apps in Tanzu Application Platform is similar to Tanzu Application Service. Specify a runtime environment variable in your `workload.yaml` as follows:

```
spec:
  env:
  - name: BPL_DEBUG_ENABLED
    value: "true"
```

This adds the JVM argument:

```
-agentlib:jdwp=transport=dt_socket,server=y,address=*:8000,suspend=n
```

(Optional) You can specify `BPL_DEBUG_PORT` and `BPL_DEBUG_SUSPEND` to change the defaults for these options.

Enable Application Performance Monitoring (APM) with Datadog

To enable Datadog, you must first install the Datadog Agent on your platform. After installing Datadog, the environment variable allows the buildpack to contribute the Datadaog Tracing library to the app classpath. To install the Datadog agent for Tanzu Application Platform, follow the instructions in Use Datadog as your observability tool.

Note

> Your Tanzu Application Platform environment might be restricted to the baseline pod security standard For more information about the baseline pod security standard, see the Kubernetes documentation. In that case, you must remove the restriction before you can install the Datadog Agent DaemonSet.

The following table compares enabling APM in Tanzu Application Service and Tanzu Application Platform.

Feature	Tanzu Application Service	Tanzu Application Platform
Enable Datadog	`$DD_API_KEY=XXX`	`$BP_DATADOG_ENABLED=true`

Example Datadog configuration

If you previously deployed the following `workload.yaml` to Tanzu Application Platform:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
 labels:
   app.kubernetes.io/part-of: petclinic
   apps.tanzu.vmware.com/has-tests: "true"
   apps.tanzu.vmware.com/workload-type: web
 name: petclinic
 namespace: apps
spec:
 build:
   env:
   - name: BP_JVM_VERSION
     value: "17"
 params:
 - name: annotations
   value:
     autoscaling.knative.dev/minScale: "1"
 source:
   git:
     ref:
       branch: main
     url: https://github.com/spring-projects/spring-petclinic
```

To enable Datadog, change the `workload.yaml` as follows:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
 labels:
   app.kubernetes.io/part-of: petclinic
   apps.tanzu.vmware.com/has-tests: "true"
   apps.tanzu.vmware.com/workload-type: web
   tags.datadoghq.com/env: "dev"
   tags.datadoghq.com/service: "petclinic"
   tags.datadoghq.com/version: "0.0.1"
 name: petclinic
 namespace: apps
spec:
 build:
   env:
   - name: BP_JVM_VERSION
     value: "17"
   - name: BP_DATADOG_ENABLED
     value: "true"
 env:
```

```
    - name: DD_AGENT_HOST
      value: "datadog-agent"
    - name: DD_ENV
      value: "dev"
    - name: DD_SERVICE
      value: "petclinic"
    - name: DD_VERSION
      value: "0.0.1"
  params:
  - name: annotations
    value:
      autoscaling.knative.dev/minScale: "1"
  source:
    git:
      ref:
        branch: main
      url: https://github.com/spring-projects/spring-petclinic
```

Configure the log level for buildpacks

The following table compares configuring the log level in Tanzu Application Service and Tanzu Application Platform.

Feature	Tanzu Application Service	Tanzu Application Platform
Enable Debug Logging	`$JBP_LOG_LEVEL=DEBUG`	`$BP_LOG_LEVEL=DEBUG`

In Tanzu Application Platform, changing the log level to DEBUG is similar to Tanzu Application Service. Set the environment variable in your `workload.yaml` as follows:

```
spec:
 build:
  env:
  - name: BP_LOG_LEVEL
    value: "DEBUG"
```

This shows debug level output for buildpacks during the build phase.

Migrate from the NGINX and Staticfile CF buildpack to the Web Server Cloud Native Buildpack

This topic tells you how to migrate your NGINX or Staticfile app from using a Cloud Foundry buildpack for Tanzu Application Service (commonly known as TAS for VMs) to using a Cloud Native Buildpack for Tanzu Application Platform (commonly known as TAP).

The Tanzu Application Platform Web Server buildpack provides the capability corresponding to the Tanzu Application Service NGINX buildpack and Tanzu Application Service Staticfile Buildpack.

Install a specific NGINX version

The following table compares how Tanzu Application Service and Tanzu Application Platform handle installing specific versions.

Feature	Tanzu Application Service	Tanzu Application Platform
Override app-based version detection.	Using `builpack.yml`	Using `$BP_NGINX_VERSION`

Tanzu Application Service: Override version detection

Tanzu Application Service buildpacks allows you to specify an NGINX version or version line, such as mainline or stable, using a `buildpack.yml` file.

Example `buildpack.yml`:

```
nginx:
  version: mainline
```

Tanzu Application Platform: Override version detection

In Tanzu Application Platform, set the `$BP_NGINX_VERSION` environment variable to specify which version or version line to install.

Example `spec` section from a `workload.yaml`:

```
spec:
  build:
    env:
    - name: BP_NGINX_VERSION
       value: mainline
```

Templating in the NGINX configuration file

The Tanzu Application Platform Web Servers buildpack supports templating in the `nginx.conf` file, such as `{{port}}`, `{{env "YOUR-VARIABLE"}}`, `{{module "module_name"}}`, similar to Tanzu Application Service.

Tanzu Application Platform does not support the template `{{nameservers}}` because it was intended only for the Cloud Foundry platform.

Static web apps

The Tanzu Application Platform Web Server buildpack can automatically generate an `nginx.conf` for your app when built with the environment `$BP_WEB_SERVER=nginx`. This is useful for apps that run in Tanzu Application Service using the Staticfile Buildpack.

You can configure the generated `nginx.conf` in combination with other environment variables, for example:

- `$BP_WEB_SERVER_ROOT`
- `$BP_WEB_SERVER_LOCATION_PATH`
- `$BP_WEB_SERVER_ENABLE_PUSH_STATE`
- `$BP_WEB_SERVER_FORCE_HTTPS`
- `$BP_NGINX_STUB_STATUS_PORT`

For more information about how to use these environment variables, see the Paketo documentation for NGINX.

Example `spec` section from a `workload.yaml`:

```
---
spec:
  build:
    env:
    - name: BP_WEB_SERVER
      value: nginx
    - name: BP_WEB_SERVER_ROOT
      value: build
```

```
  - name: BP_WEB_SERVER_ENABLE_PUSH_STATE
    value: "true"
```

Configure basic authentication

The following table compares how you set up basic authentication in Tanzu Application Service and Tanzu Application Platform.

Feature	Tanzu Application Service	Tanzu Application Platform
Provide HTTP basic authentication	Use a `Staticfile.auth`	Use a binding of type `htpasswd` containing `.htpasswd`

Tanzu Application Service: Set up basic authentication

In Tanzu Application Service Staticfile buildpack applications you use a file at the root of the app.

Tanzu Application Platform: Set up basic authentication

For the Tanzu Application Platform Web Server buildpack you provide basic authentication credentials by using a service binding as follows:

1. Create the service binding as a secret. For example:

```
---
apiVersion: v1
kind: Secret
metadata:
name: basic-auth-config
namespace: my-apps
type: service.binding/htpasswd
stringData:
type: htpasswd
.htpasswd: |
  user1:$foooo.oooo
  user2:$baaaaaaa.ar
```

2. Use the binding in the `workload.yaml`. For example:

```
---
kind: Workload
apiVersion: carto.run/v1alpha1
metadata:
name: nginx-app
spec:
# ...
  params:
  - name: buildServiceBindings
    value:
      - name: basic-auth-config
        kind: Secret
        apiVersion: v1
```

For more information about service bindings, see Configure Tanzu Build Service properties on a workload.

Migrate to the Node.js Cloud Native Buildpack

This topic tells you how to migrate your Node.js app from using a Cloud Foundry buildpack for Tanzu Application Service (commonly known as TAS for VMs) to using a Cloud Native Buildpack for

Tanzu Application Platform (commonly known as TAP).

Install a specific Node Engine version

The following table compares how Tanzu Application Service and Tanzu Application Platform handle installing specific versions.

Feature	Tanzu Application Service	Tanzu Application Platform
Detects version from `package.json` Detects version from `.nvmrc`		
Detects version from `.node-version`		
Override app-based version detection		Use `$BP_NODE_VERSION`

Tanzu Application Service: Override version detection

Specifying the version to install is not supported.

Tanzu Application Platform: Override version detection

In Tanzu Application Platform, users can set the `$BP_NODE_VERSION` environment variable to specify which version of the Node Engine to installed.

Example `spec` section from a `workload.yaml`:

```
---
spec:
  build:
    env:
    - name: BP_NODE_VERSION
       value: "20.*.*"
```

Heap memory optimization

The following table compares how to configure heap memory optimization in Tanzu Application Service and Tanzu Application Platform.

Feature	Tanzu Application Service	Tanzu Application Platform
Enable Heap Memory Optimization	Use `$OPTIMIZE_MEMORY=true`	Use `$BP_NODE_OPTIMIZE_MEMORY=true`

Provide npm configuration files

The following table compares how to provide npm configuration files in Tanzu Application Service and Tanzu Application Platform.

Feature	Tanzu Application Service	Tanzu Application Platform
Provide a `.npmrc` with the app		
Provide a `.npmrc` by using service bindings	Not supported	Use a binding of type npmrc containing `.npmrc`

Configure npm settings with sensitive data

In Tanzu Application Platform, if your npm configuration contains sensitive data, you can provide the npm configuration to the build without explicitly including the file in the application directory.

To provide your npm configuration file without including it in the directory:

1. Create the service binding as a secret. For example:

```
---
apiVersion: v1
kind: Secret
metadata:
  name: npmrc-binding
  namespace: my-apps
type: service.binding/npmrc
stringData:
  type: npmrc
  .npmrc: |
    registry=https://registry.npmjs.org/
    loglevel=warn
    save-exact=true
```

2. Use the binding in the `workload.yaml`. For example:

```
---
kind: Workload
apiVersion: carto.run/v1alpha1
metadata:
name: nodejs-app
spec:
# ...
  params:
  - name: buildServiceBindings
    value:
      - name: npmrc-binding
        kind: Secret
        apiVersion: v1
```

For more information about service bindings, see Configure Tanzu Build Service properties on a workload.

Provide yarn configuration files

The following table compares how to provide yarn configuration files in Tanzu Application Service and Tanzu Application Platform.

Feature	Tanzu Application Service	Tanzu Application Platform
Provide a `.yarnrc` with the app		
Provide a `.yarnrc` by using service bindings	Not supported	Use a binding of type yarnrc containing `.yarnrc`

Configure yarn settings with sensitive data

In Tanzu Application Platform, if your yarn configuration contains sensitive data, you can provide the yarn configuration to the build without explicitly including the file in the application directory.

To provide your yarn configuration file without including it in the directory:

1. Create the service binding as a secret. For example:

```
---
apiVersion: v1
kind: Secret
metadata:
```

```
  name: yarnrc-binding
  namespace: my-apps
type: service.binding/yarnrc
stringData:
  type: yarnrc
  .yarnrc: |
    registry "https://registry.yarnpkg.com"
    flat "true"
```

2. Use the binding in the workload. For example:

```
---
kind: Workload
apiVersion: carto.run/v1alpha1
metadata:
name: nodejs-app
spec:
# ...
  params:
  - name: buildServiceBindings
    value:
      - name: yarnc-binding
        kind: Secret
        apiVersion: v1
```

For more information about service bindings, see Configure Tanzu Build Service properties on a workload.

Build and serve a front end framework app

The Tanzu Application Platform Node.js buildpack is designed exclusively for building back end server node applications. If your app is using a front end framework that generates static content from JavaScript source code, such as React, Vue, Angular, you must use the Tanzu Application Platform Web Servers buildpack instead of the Tanzu Application Platform Node.js buildpack.

To build a front end app, set the environment variable `$BP_NODE_RUN_SCRIPTS` to instruct the Tanzu Application Platform Web Servers buildpack to run a specific npm or yarn script event during the build. For popular frameworks such as Angular and React, this is generally the build script.

Example `spec` section from a `workload.yaml`:

```
---
spec:
  build:
    env:
  - name: BP_NODE_RUN_SCRIPTS
    value: build
  - name: BP_WEB_SERVER
    value: nginx
  - name: BP_WEB_SERVER_ROOT
    value: build
```

For more information about using the Tanzu Application Platform Web Server buildpack, see Use the Tanzu Web Servers Buildpack in the Tanzu Buildpacks documentation.

Migrate to the PHP Cloud Native Buildpack

This topic tells you how to migrate your PHP app from using a Cloud Foundry buildpack for Tanzu Application Service (commonly known as TAS for VMs) to using a Cloud Native Buildpack for Tanzu Application Platform (commonly known as TAP).

PHP configuration

This section describes how you change PHP configurations when migrating from Tanzu Application Service to Tanzu Application Platform.

Install specific PHP versions

The following table compares how Tanzu Application Service and Tanzu Application Platform handle installing specific versions.

Feature	Tanzu Application Service	Tanzu Application Platform
Detects version from `composer.lock`		
Detects version from `composer.json`: `"require": { "php": ">=8.0" }`		
Override app-based version detection	Use `options.json`	Use `$BP_PHP_VERSION`

Tanzu Application Service: Override version detection

Tanzu Application Service buildpacks allows you to specify a PHP version using a `options.json` file.

Example `options.json`:

```
{
    "PHP_VERSION": "7.34.5"
}
```

The Tanzu Application Service `options.json` supports providing the exact version to use, or specifying a more general minor version line to use such as `PHP_80_LATEST` instead of `8.0.1`.

Tanzu Application Platform: Override version detection

In Tanzu Application Platform, set the `$BP_PHP_VERSION` environment variable to specify which version of the PHP distribution to install.

Example `spec` section from a `workload.yaml`:

```
---
spec:
  build:
    env:
    - name: BP_PHP_VERSION
      value: 8.0.0
```

Configure the PHP library directory

The following table compares how you configure the PHP library directory for Tanzu Application Service and Tanzu Application Platform.

Feature	Tanzu Application Service	Tanzu Application Platform
Set PHP library directory	Use `options.json`	Use `$BP_PHP_LIB_DIR`

Tanzu Application Service: Configure the PHP library directory

Tanzu Application Service buildpacks allow you to specify the PHP library directory using a `options.json` file.

Example `options.json`:

```
{
    "LIBDIR": "lib"
}
```

Tanzu Application Platform: Configure the PHP library directory

In Tanzu Application Platform, specify the PHP library directory by using the `$BP_PHP_LIB_DIR` environment variable.

Example `spec` section from a `workload.yaml`:

```
---
spec:
  build:
    env:
    - name: BP_PHP_LIB_DIR
      value: lib
```

Configure a custom `.ini` file

In Tanzu Application Service and Tanzu Application Platform, you can configure custom `.ini` files in addition to the default `php.ini` provided.

The following table compares how you configure a custom `.ini` file for Tanzu Application Service and Tanzu Application Platform.

Feature	Tanzu Application Service	Tanzu Application Platform
Custom `.ini` file in the application	Enabled by file in `.bp-config/php/php.ini.d/*.ini`	Enabled by file in `.php.ini.d/*.ini`

Create an `.ini` file under `.bp-config/php/php.ini.d/FILENAME.ini` in Tanzu Application Service, and under `.php.ini.d/FILENAME.ini` in Tanzu Application Platform.

Enable PHP extensions

In Tanzu Application Platform, the only extensions available to use at this time are the ones that come with the distribution of PHP.

The following table compares how you enable PHP extensions for Tanzu Application Service and Tanzu Application Platform.

Feature	Tanzu Application Service	Tanzu Application Platform
Enable PHP extensions by custom `.ini` snippet	Located in `.bp-config/php/php.ini.d/FILENAME.ini`	Located in `APP-ROOT/.php.ini.d/*.ini`
Enable PHP extensions by `composer.json`		
Enable additional custom extensions through `.extensions` directory. For more information, see the (php-buildpack README) in GitHub.		

Tanzu Application Service: Enable PHP extensions by custom `.ini` file

In Tanzu Application Service, you can specify extensions in a custom `.ini` file under `.bp-config/php/php.ini.d/FILENAME.ini` in the application. If an extension is already present and enabled in the compiled PHP, explicitly enabling the extension is not required in Tanzu Application Service.

For example:

```
extension=bz2.so
extension=curl.so
zend_extension=opcache.so
```

Tanzu Application Platform: Enable PHP extensions by custom `.ini` file

In Tanzu Application Platform, you can enable extensions by using a custom `.ini` file snippet. An `.ini` snippet is a valid PHP configuration file. The buildpacks look for any user-provided snippets under `APP-ROOT/.php.ini.d/*.ini`.

For example:

```
extension=bz2.so
extension=curl.so
```

Alternatively, in both Tanzu Application Service and Tanzu Application Platform, if you are using Composer as a package manager, you can specify extensions through the `composer.json` file.

For example:

```
{
    "require": {
        "php": ">=7.1",
        "ext-bz2": "*",
        "ext-curl": "*",
    },
}
```

Profile scripts

The following table compares enabling launch-time scripts for Tanzu Application Service and Tanzu Application Platform.

Feature	Tanzu Application Service	Tanzu Application Platform
Enable launch-time scripts (link for more info)	Located in `.profile.d` directory	

A `.profile.d` directory containing scripts to be run just before application launch is not currently supported in Tanzu Application Platform. These scripts are ignored during launch.

Server configuration

This section describes how you change server configurations when migrating from Tanzu Application Service to Tanzu Application Platform.

Select a web server

The following table compares how you select a web server in Tanzu Application Service and Tanzu Application Platform.

Feature	Tanzu Application Service	Tanzu Application Platform
Select a web server to use	Use `options.json`	Use `$BP_WEB_SERVER`

Tanzu Application Service: Select a web server

Tanzu Application Service buildpacks allow you to specify which web server to use by using an `options.json` file.

Example `options.json`:

```
{
    "WEB_SERVER": "httpd"
}
```

Tanzu Application Platform: Select a web server

In Tanzu Application Platform, you specify the web server by using the `$BP_WEB_SERVER` environment variable. For more information about the web servers you can, see Select a Web Server in the Tanzu Buildpacks documentation.

Example `spec` section from a `workload.yaml`:

```
---
spec:
  build:
    env:
    - name: BP_WEB_SERVER
      value: httpd
```

Set server configuration

The following table compares how you set server configuration in Tanzu Application Service and Tanzu Application Platform.

Feature	Tanzu Application Service	Tanzu Application Platform
HTTPD configuration	`.bp-config/httpd/*.conf`	`APP-DIRECTORY/.httpd.conf.d/*.conf`
NGINX configuration	`.bp-config/nginx/*.conf`	`APP-DIRECTORY/.nginx.conf.d/*-server.conf` OR `APP-DIRECTORY/.nginx.conf.d/*-http.conf`
FPM configuration	`.bp-config/php/fpm.d`	`APP-DIRECTORY/.php.fpm.d/*.conf.`

Tanzu Application Service: Set server configuration

For Tanzu Application Service buildpacks, you can add server configuration files under the `.bp-config` directory to override the buildpack-set configuration options. For more information, see the Cloud Foundry documentation.

Put these files under `.bp-config` in directories named for the relevant component, such as `httpd` for HTTPD, `nginx` for NGINX, and `fpm.d` for FPM configuration.

Tanzu Application Platform: Set server configuration

Tanzu Application Platform buildpacks also support providing custom settings for HTTPD, NGINX and FPM that are not supported through the buildpack environment variables using specially-

named directories:

- **HTTPD:** `APP-DIRECTORY/.httpd.conf.d/*.conf`
- **NGINX:** `APP-DIRECTORY/.nginx.conf.d/*-server.conf` OR `APP-DIRECTORY/.nginx.conf.d/*-http.conf`
- **FPM:** `APP-DIRECTORY/.php.fpm.d/*.conf.`

For more information, see Set Server Configuration.

Configure the web directory

The following table compares how you set the web directory in Tanzu Application Service and Tanzu Application Platform.

Feature	Tanzu Application Service	Tanzu Application Platform
Set the web directory	Use `options.json`	Use `$BP_WEB_SERVER`

Tanzu Application Service: Configure the web directory

Tanzu Application Service buildpacks allow you to specify the root directory of the files served by the web server, relative to the root of the app, through an `options.json`, for example:

```
{
    "WEBDIR": "htdocs"
}
```

Tanzu Application Platform: Configure the web directory

In Tanzu Application Platform you specify the web directory by using the `$BP_PHP_WEB_DIR` environment variable. For more information, see Configure Web Directory in the Tanzu Buildpacks documentation.

Example `spec` section from a `workload.yaml`:

```
---
spec:
  build:
    env:
    - name: BP_PHP_WEB_DIR
      value: htdocs
```

Set the app start command

The following table compares how you set the app start command in Tanzu Application Service and Tanzu Application Platform.

Feature	Tanzu Application Service	Tanzu Application Platform
Set a custom app start command	Use `options.json`	Use Procfile

Tanzu Application Service: Set the app start command

Tanzu Application Service buildpacks allow you to specify the start command for your app through an `options.json` file.

Example `options.json`:

```
{
    "APP_START_CMD": "app.php"
}
```

Tanzu Application Platform: Set the app start command

In Tanzu Application Platform buildpacks support setting a custom start command for your app by using a Procfile located at the root of the application. For more information, see Additional Configuration in the Tanzu Buildpacks documentation.

Example Procfile:

```
web: php -S 0.0.0.0:"${PORT:-80}" -t htdocs && echo hi
```

Activate or deactivate HTTPS redirect

The following table compares how you activate or deactivate the HTTPS redirect in Tanzu Application Service and Tanzu Application Platform.

Feature	Tanzu Application Service	Tanzu Application Platform
Activate or deactivate HTTPS redirect		Use Procfile

HTTPS redirect is enabled by default for NGINX and HTTPD. In Tanzu Application Platform, you can deactivate this by setting the `$BP_PHP_ENABLE_HTTPS_REDIRECT` environment variable to `false` at build time.

Example `spec` section from a `workload.yaml`:

```
---
spec:
  build:
    env:
    - name: BP_PHP_ENABLE_HTTPS_REDIRECT
      value: false
```

Composer configuration

This section describes how you change the Composer configurations when migrating from Tanzu Application Service to Tanzu Application Platform.

Install specific Composer versions

The following table compares how Tanzu Application Service and Tanzu Application Platform handle installing specific Composer versions.

Feature	Tanzu Application Service	Tanzu Application Platform
Override app-based version detection	Use `options.json`	Use `$BP_COMPOSER_VERSION`

Tanzu Application Service: Override Composer version detection

Tanzu Application Service buildpacks allow you to specify a Composer version using an `options.json` file.

Example `options.json`:

```
{
    "COMPOSER_VERSION": "2.6.6"
```

```
}
```

Tanzu Application Service: Override Composer version detection

Tanzu Application Platform buildpacks allow you to specify Composer version by using the `$BP_COMPOSER_VERSION` environment variable. For more information, see Set Composer Version in the Tanzu Buildpacks documentation.

Example `spec` section from a `workload.yaml`:

```
---
spec:
  build:
    env:
    - name: BP_COMPOSER_VERSION
      value: 2.6.6
```

Set Composer install options

The following table compares how you set install options for Composer in Tanzu Application Service and Tanzu Application Platform.

Feature	Tanzu Application Service	Tanzu Application Platform
Set a list of options to be passed to Composer install	Use `options.json`	Use `$BP_COMPOSER_INSTALL_OPTIONS`

Tanzu Application Service: Set Composer install options

Tanzu Application Service buildpacks allow you to specify Composer install options by using an `options.json` file.

Example `options.json`:

```
{
    "COMPOSER_INSTALL_OPTIONS": ["--no-interaction","--no-dev","--no-progress"]
}
```

Tanzu Application Platform: Set Composer install options

Tanzu Application Platform buildpacks allow you to specify composer install options by using the `$BP_COMPOSER_INSTALL_OPTIONS` environment variable. For more information, see Set Composer Install Options in the Tanzu Buildpacks documentation.

Example `spec` section from a `workload.yaml`:

```
---
spec:
  build:
    env:
    - name: BP_COMPOSER_INSTALL_OPTIONS
      value: "--no-dev --prefer-install=auto"
```

Set the `composer.json` path

The following table compares how you set the `composer.json` path in Tanzu Application Service and Tanzu Application Platform.

Feature	Tanzu Application Service	Tanzu Application Platform
Set a custom composer.json path	Use `$COMPOSER_PATH`	Use `$COMPOSER`

Tanzu Application Service: Set `composer.json` path

Tanzu Application Service buildpacks allow you to specify a `composer.json` path, relative to the project root, using the `$COMPOSER_PATH` environment variable either through the `manifest.yml` or the cf CLI, for example:

```
cf set-env YOUR_APP_NAME COMPOSER_PATH "PATH_TO_COMPOSER_JSON"
```

For more information, see the Cloud Foundry documentation.

Tanzu Application Platform: Set `composer.json` path

Tanzu Application Platform buildpacks allow you to specify the `composer.json` path, relative to the project root, by using the native `$COMPOSER` environment variable. For more information, see Set the composer.json Path. in the Tanzu Buildpacks documentation.

Example `spec` section from a `workload.yaml`:

```
---
spec:
  build:
    env:
    - name: COMPOSER
      value: some-other-composer-json
```

Set Composer-native environment variables

The following table compares how you set Composer-native environment variables Tanzu Application Service and Tanzu Application Platform.

Feature	Tanzu Application Service	Tanzu Application Platform
Override buildpack-set Composer environment variables	Use `options.json`	Use `$COMPOSER_ENV-VAR-NAME`. For example: `COMPOSER_VENDOR_DIR`

Tanzu Application Service: Set Composer-native environment variables

Tanzu Application Service buildpacks allow you to specify various Composer-native extensions using an `options.json`, which is passed through to Composer to override the buildpack-set defaults.

Example `options.json`:

```
{
    "COMPOSER_VENDOR_DIR": "vendor",
}
```

Tanzu Application Platform: Set Composer-native environment variables

Tanzu Application Platform buildpacks respect any Composer-native environment variables set at build-time.

Example `spec` section from a `workload.yaml`:

```
---
spec:
  build:
    env:
    - name: COMPOSER_VENDOR_DIR
      value: vendor
    - name: COMPOSER_BIN_DIR
      value: some-bin-dir
    - name: COMPOSER_CACHE_DIR
      value: some-cache-dir
```

Supply Composer authentication

The following table compares how you supply Composer authentication in Tanzu Application Service and Tanzu Application Platform.

Feature	Tanzu Application Service	Tanzu Application Platform
Supply Composer authentication (for example, GitHub token)	Use `$COMPOSER_GIHTUB_OAUTH_TOKEN`	Use `$COMPOSER_AUTH`

Tanzu Application Service: Supply Composer authentication

Tanzu Application Service buildpacks allow you to supply Composer authentication, mainly in the form of a Github token to bypass rate-limiting, by using an environment variable, for example:

```
cf set-env YOUR-APP-NAME COMPOSER_GITHUB_OAUTH_TOKEN "OAUTH-TOKEN-VALUE"
```

Where:

- `YOUR-APP-NAME` is the name of your app.
- `OAUTH-TOKEN-VALUE` is the token.

For more information, see the Cloud Foundry documentation.

Tanzu Application Platform: Supply Composer authentication

Tanzu Application Platform buildpacks respect any Composer-native environment variables set at build-time. You can supply a GitHub token, or any other authentication supported by Composer, through the native `COMPOSER_AUTH` environment variable. For more information, see Set Composer Authentication. in the Tanzu Buildpacks documentation.

Example `spec` section from a `workload.yaml`:

```
---
spec:
  build:
    env:
    - name: COMPOSER_AUTH
      value: '{"github-oauth": {"github.com": "<oauthtoken>"}}'
```

Session Handlers

This section describes how you change session handlers when migrating from Tanzu Application Service to Tanzu Application Platform.

Enable Redis or Memcached session handler

The following table compares how you enable a Redis or Memcached session handler in Tanzu Application Service and Tanzu Application Platform.

Feature	Tanzu Application Service	Tanzu Application Platform
Bind session handler service to application	Use `cf create-service`	Use Service Bindings

Tanzu Application Service: Enable session handler

In Tanzu Application Service, you can bind a Redis or Memcached instance to a PHP app using the cf CLI, for example:

```
$ cf create-service redis some-plan app-redis-sessions
$ cf bind-service app app-redis-sessions
$ cf restage app
```

For more information, see the Cloud Foundry documentation.

Tanzu Application Platform: Enable session handler

In Tanzu Application Platform, you can configure session handlers for Redis or Memcached by using Service Bindings:

1. Create the service binding as a secret. For example:

   ```
   ---
   apiVersion: v1
   kind: Secret
   metadata:
     name: php-redis-session
     namespace: my-apps
   type: service.binding/php-redis-session
   stringData:
     type: php-redis-session
     host: HOST # default: 127.0.0.1
   port: PORT # default: 6379
   password: PASSWORD # omitted if unset
   ```

2. Use the binding in the workload. For example:

   ```
   ---
   kind: Workload
   apiVersion: carto.run/v1alpha1
   metadata:
   name: php-app
   spec:
   # ...
     params:
     - name: buildServiceBindings
       value:
         - name: php-redis-session
           kind: Secret
           apiVersion: v1
   ```

For more information on using Service Bindings in Tanzu Application Platform, see Configure build-time service bindings.

New Relic

This section describes how you configure New Relic when migrating from Tanzu Application Service to Tanzu Application Platform.

Configure New Relic

Feature	Tanzu Application Service	Tanzu Application Platform
Bind session handler service to application	Use `$VCAP_SERVICES` or `$NEWRELIC_LICENSE`	Use Service Bindings

Tanzu Application Service: Configure New Relic

In Tanzu Application Service, you can configure New Relic for the PHP buildpack either by:

- Using a Cloud Foundry service: Your `VCAP_SERVICES` environment variable must contain a service named `newrelic`. The `newrelic` service must contain a key named `credentials` and the `credentials` key must contain a `licenseKey`.
- Obtaining a license key and setting the value of the environment variable `NEWRELIC_LICENSE` to your New Relic license key in the `manifest.yml` or through the cf CLI, for example:

  ```
  cf set-env NEWRELIC_LICENSE NEW-RELIC-LICENSE-KEY
  ```

  Where `NEW-RELIC-LICENSE-KEY` is you New Relic license key.

Tanzu Application Platform: Configure New Relic

In Tanzu Application Platform, the PHP language family buildpack includes the New Relic buildpack, which participates in a build when there is a service binding of type `NewRelic`.

To configure New Relic:

1. Create the service binding as a secret. For example:

   ```
   ---
   apiVersion: v1
   kind: Secret
   metadata:
     name: new-relic
     namespace: my-apps
   type: service.binding/NewRelic
   stringData:
     type: NewRelic
   ```

2. Use the binding in the workload. For example:

   ```
   ---
   kind: Workload
   apiVersion: carto.run/v1alpha1
   metadata:
   name: php-app
   spec:
   # ...
     params:
     - name: buildServiceBindings
       value:
         - name: new-relic
           kind: Secret
           apiVersion: v1
   ```

Migrate to the Python Cloud Native Buildpack

This topic tells you how to migrate your Python app from using a Cloud Foundry buildpack for Tanzu Application Service (commonly known as TAS for VMs) to using a Cloud Native Buildpack for Tanzu Application Platform (commonly known as TAP).

Install a specific Python version

The following table compares how Tanzu Application Service and Tanzu Application Platform handle installing specific versions.

Feature	Tanzu Application Service	Tanzu Application Platform
Detects version from `runtime.txt`		
Override app-based version detection.		Use `$BP_CPYTHON_VERSION`

Tanzu Application Platform: Specify version

In Tanzu Application Platform, set the `$BP_CPYTHON_VERSION` environment variable to specify which version of CPython 3 to install.

Example `spec` section from a `workload.yaml`:

```
---
spec:
  build:
    env:
    - name: BP_CPYTHON_VERSION
       value: "3.12.1"
```

Install a specific pip version

Both Tanzu Application Service and Tanzu Application Platform Python buildpacks provide an extra pip dependency. You can configure this dependency using the `$BP_PIP_VERSION` environment variable in the same way you configure `$BP_CPYTHON_VERSION` in Tanzu Application Platform: Specify version above.

Install a specific Pipenv version

Tanzu Application Service buildpack does not support selecting a version. However, for the Tanzu Application Platform Python buildpack, you con configure the Pipenv version using the `$BP_PIPENV_VERSION` environment variable.

Start command

The Tanzu Application Service buildpack does not generate a default start command for your apps. However, the Tanzu Application Platform Python buildpack sets the default start command to run the Python read-eval-print loop (REPL) at launch.

For production apps, use a Procfile to specify the start command for both Tanzu Application Service and Tanzu Application Platform. For more information about Procfile on Tanzu Application Platform, see the Tanzu Buildpacks documentation.

Migrate to the Ruby Cloud Native Buildpack

This topic tells you how to migrate your Ruby app from using a Cloud Foundry buildpack for Tanzu Application Service (commonly known as TAS for VMs) to using a Cloud Native Buildpack for Tanzu

Application Platform (commonly known as TAP).

Specifying the Ruby version to install

The following table compares how Tanzu Application Service and Tanzu Application Platform handle installing specific Ruby versions.

Feature	Tanzu Application Service	Tanzu Application Platform
Detects version from `Gemfile` `ruby '~> <version>'`		
Detects version from `$BP_MRI_VERSION` env var		

Tanzu Application Service: Override version detection

Specifying the version to install is not supported.

Tanzu Application Platform: Override version detection

In Tanzu Application Platform, set the `$BP_MRI_VERSION` environment variable to specify which version of Ruby MRI to install. You can set the version to any valid semver version or version constraint, for example, `2.7.4` or `2.7.*`. The buildpack automatically installs an Ruby MRI version that is compatible with the selected version.

Example `spec` section from a `workload.yaml`:

```
---
spec:
  build:
    env:
    - name: BP_MRI_VERSION
      value: 3.2.*
  source:
    git:
      ref:
        branch: master
      url: https://github.com/cloudfoundry/ruby-buildpack
    subPath: fixtures/bundler_2
```

Specifying the Bundler version to install

The following table compares how Tanzu Application Service and Tanzu Application Platform handle installing specific Bundler versions.

Feature	Tanzu Application Service	Tanzu Application Platform
Detects version from Gemfile.lock `BUNDLED_WITH <version>'`		
Detects version from `$BP_BUNDLER_VERSION` env var		

Specifying the Jruby version to install

Specifying the Jruby version is not supported in Tanzu Application Platform.

Feature	Tanzu Application Service	Tanzu Application Platform
Detects version from Gemfile		

Vendor in app dependencies before a build

The following table compares vendoring app dependencies before a build for Tanzu Application Service and Tanzu Application Platform.

Feature	Tanzu Application Service	Tanzu Application Platform
Packages in the default cache location `bundle package --all`		
Packages in a non-default cache location		

Migration to vendoring gems in a non-default cache location

In Tanzu Application Platform, you can vendor in your app dependencies before a build. To do this, put all `.gem` files into the custom path inside the app source code, for example, `custom_dir/custom_cache`. Then, create a `.bundle/config` file with the `BUNDLE_CACHE_PATH` setting configured:

```
---
BUNDLE_CACHE_PATH: "custom_dir/custom_cache"
```

Configure Rake tasks

The following table compares how to configure rake tasks with Tanzu Application Service and Tanzu Application Platform.

Feature	Tanzu Application Service	Tanzu Application Platform
Non-default Rake task support		
Default Rake task support		

Tanzu Application Service: Invoke Rake tasks

In Tanzu Application Service, you can automatically invoke a Rake task as follows:

1. Include a `.rake` file containing a Rake task in `lib/tasks`. For example:

   ```
   namespace :cf do
       desc "Only run on the first application instance"
       task :on_first_instance do
           instance_index = JSON.parse(ENV["VCAP_APPLICATION"])["instance_index"]
   rescue nil
       exit(0) unless instance_index == 0
       end
   end
   ```

2. Add the task to the `manifest.yml` with the command attribute to use this as the start command. For example:

   ```
   applications:
   - name: my-app
     command: bundle exec rake cf:on_first_instance db:migrate
   ```

Tanzu Application Platform: Invoke Rake tasks

In Tanzu Application Platform, you still specify a Rake task, but it goes in a Rakefile in the root of the application directory. Then, the buildpacks automatically set the default Rake task as the start

command.

- If you specify a default Rake task, this is the task executed by the buildpack.
- If you specify a non-default Rake task, use the Procfile Buildpack by including the start command for the non-default task in a Procfile file. Set as the web process, for example:

```
web: bundle exec rake non_default
```

Building a Rails application

The following table compares building a rails app on Tanzu Application Service and Tanzu Application Platform.

Feature	Tanzu Application Service	Tanzu Application Platform
Building a Rails application		

Building Rails applications is supported on both Tanzu Application Service and Tanzu Application Platform. In both cases, you must specify the rails gem in the Gemfile.

In Tanzu Application Platform, one of the following asset directories must also be present in the application source code:

- `app/assets`
- `lib/assets`
- `vendor/assets`
- `app/javascript`

Supported web servers

The following table compares supported web servers for Tanzu Application Service and Tanzu Application Platform.

Feature	Tanzu Application Service	Tanzu Application Platform
Rake		
Thin		
Rackup		
Rails Server		
Puma		
Unicorn	requires custom start command through the `manifest.yml`	
Passenger		

Supported dependencies

The following table compares supported dependencies for Tanzu Application Service and Tanzu Application Platform.

Feature	Tanzu Application Service	Tanzu Application Platform
Bundler		
Ruby		

Feature	Tanzu Application Service	Tanzu Application Platform
Node		
Yarn		
Jruby		
OpenJDK		
Ruby Gems		

Set up Tanzu Service Mesh

This topic tells you how to set up a Tanzu Application Platform application deployed on Kubernetes with Tanzu Service Mesh (commonly called TSM).

Sample applications are used to demonstrate how a global namespace can provide a network for Kubernetes workloads that are connected and secured within and across clusters, and across clouds.

Note

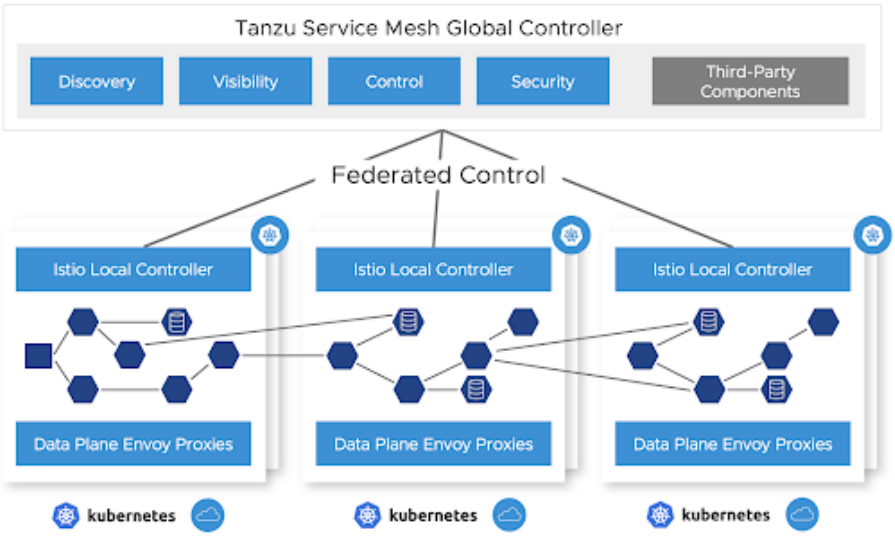
You can also use Tanzu Mission Control (commonly called TMC) to install Tanzu Application Platform (commonly called TAP) on a managed cluster. For more information, see the Tanzu Mission Control documentation.

Prerequisites

Meet the prerequisites, which includes having

- A supported Kubernetes platform
- The correct resource configuration (number of nodes, CPUs, RAM, and so on)
- The required connectivity requirements

Connectivity is only required from your local clusters out to Tanzu Service Mesh and not inwards. This can traverse a corporate proxy as well. In addition, connectivity in the data plane is required between the clusters that must communicate, specifically egress to ingress gateways. No data plane traffic needs to reach the Tanzu Service Mesh software as a service (SaaS) management plane.



Activate your Tanzu Service Mesh subscription

Activate your Tanzu Service Mesh subscription at cloud.vmware.com. After purchasing your Tanzu Service Mesh subscription, the VMware Cloud team sends you instructions. If you don't receive them, you can follow these instructions.

Onboard your clusters to Tanzu Service Mesh as described later in this topic. This deploys the Tanzu Service Mesh local control plane and OSS Istio on your Kubernetes cluster and connects the local control plane to your Tanzu Service Mesh tenant.

Set up Tanzu Application Platform

To enable Tanzu Service Mesh support in Tanzu Application Platform Build clusters:

1. Add the following key to `tap-values.yaml` under the `buildservice` top-level key:

   ```
   buildservice:
    injected_sidecar_support: true
   ```

2. Install Tanzu Application Platform on the run cluster.

End-to-end workload build and deployment scenario

The following sections describe how to build and deploy a workload.

Apply a workload resource to a build cluster

Workloads can be built by using a Tanzu Application Platform supply chain by applying a workload resource to a build cluster. At this time, Tanzu Service Mesh and Tanzu Application Platform cannot use the Knative resources that are the default runtime target when using the `web` resource type.

In Tanzu Application Platform v1.4 and later, two workload types support a Tanzu Service Mesh and Tanzu Application Platform integration: **server** and **worker**.

To work with Tanzu Service Mesh, web workloads must be converted to the `server` or `worker` workload type. Server workloads cause a Kubernetes `Deployment` resource to be created with a `Service` resource that uses port 8080 by default.

1. If the service port that you want is 80 or some other port, add port information to `workload.yaml`. The following example YAML snippets show the changes to make from the `web` to `server` workload type. This is an example before applying the changes:

   ```
   apiVersion: carto.run/v1alpha1
   kind: Workload
   metadata:
    name: where-for-dinner
    labels:
      apps.tanzu.vmware.com/workload-type: web
      app.kubernetes.io/part-of: where-for-dinner-api-gateway
   spec:
    params:
    - name: annotations
   value:
   autoscaling.knative.dev/minScale: "1"
    source:
      git:
        url: https://github.com/vmware-tanzu/application-accelerator-samples.git
      ref:
        branch: tap-1.6.x
      subPath: where-for-dinner/where-for-dinner-api-gateway
   ```

   This is an example modified for Tanzu Service Mesh, which includes the removal of the autoscaling annotation:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
 name: where-for-dinner
 labels:
 apps.tanzu.vmware.com/workload-type: server # modification
 app.kubernetes.io/part-of: where-for-dinner-api-gateway
spec:
 params:
 - name: ports # modification
 value:
 - port: 80 # modification
   containerPort: 8080 # modification
   name: http # modification
 source:
 git:
     url: https://github.com/vmware-tanzu/application-accelerator-samples.git
   ref:
       branch: tap-1.6.x
 subPath: where-for-dinner/where-for-dinner-api-gateway
```

This results in a deployment and a service that listens on port 80 and forwards traffic to port 8080 on the pod's workload container.

2. Submit the modified YAML to your build cluster by running:

```
tanzu apps NAMESPACE apply --file WORKLOAD-YAML-FILE
```

Where:

- `NAMESPACE` is the namespace that the build cluster uses for building.
- `WORKLOAD-YAML-FILE` is the name of your workload YAML file, such as `workload.yaml`.

After your workload is built a `Deliverable` resource is created.

Configure egress for Tanzu Build Service

For Tanzu Build Service to properly work, provide egress to access the registry where Tanzu Build Service writes application images, and define the registry in the `kp_default_repository` key and the Tanzu Application Platform install registry.

Additionally, configure egress for buildpack builds to download any required dependencies. This configuration varies with different buildpacks and language environments. For example, Java builds might need to download dependencies from Maven central.

Create a global namespace

Using the Tanzu Service Mesh portal or API, create a global namespace (GNS) that includes the namespaces where your application components are deployed. For more information, see Global Namespaces

Whether in a single cluster or multiple clusters, or within the same site or across clouds, after you add a namespace selection to the GNS, the services that Tanzu Application Platform deploys are connected based on the GNS configuration for service discovery and connectivity policies.

If a service must be accessible through the ingress from the outside, it can be configured through the public service option in Tanzu Service Mesh or directly through Istio on the clusters where that service resides. It's best practice to configure the service's accessibility through the GNS.

Run cluster deployment

Before deploying a workload to a run cluster, ensure that any prerequisite resources have already been created on the run cluster. This includes concepts such as data, messaging, routing, security services, RBAC, ResourceClaims, and so on.

After a successful build in a build cluster, workloads can be deployed to the run cluster by applying resulting deliverable resources to the run cluster as described in Getting Started with Multicluster Tanzu Application Platform.

Another option is to create a kapp application that references a GitOps repository to include all deliverable resources for a given cluster. See the following example of a kapp definition that points to a GitOps repository:

```
apiVersion: kappctrl.k14s.io/v1alpha1
kind: App
metadata:
  name: deliverable-gitops
  namespace: where-for-dinner
spec:
  serviceAccountName: default
  fetch:
  - git:
    url: https://github.com/gm2552/tap-play-gitops
    ref: origin/deliverables-tap-east01
    subPath: config
  template:
  - ytt: {}
  deploy:
  - kapp: {}
```

The advantage of this model is that applications can be deployed or uninstalled from a cluster by managing the contents of the deliverable resources from within the GitOps repository and enabling a GitOps workflow for application and service change control.

Deployment use case: Where For Dinner

The following instructions describe an end-to-end process for configuring, building, and deploying the Where For Dinner application into a Tanzu Service Mesh global namespace.

These instructions use the default configuration of Where For Dinner, which consists of only needing a single-node RabbitMQ cluster, an in-memory database, and no security. The application is deployed across two Tanzu Application Platform run clusters. It requires the `ytt` command to execute the build and deployment commands.

The configuration resources referenced in this scenario are located in the where-for-dinner-tap-tsm GitHub repository.

Create an initial set of configuration files from the accelerator

This use case deployment includes a pre-built set of configuration files in a Git repository. However, they were created from a set of configuration files by using a bootstrapped process that uses the Where For Dinner accelerator, and were later modified.

For reference, you can create an initial set of configuration files from the Where For Dinner accelerator, which is available in Tanzu Application Platform v1.3 and later.

This section does not include instructions for modifying the configuration files from the accelerator into configuration files used in a later section.

From the accelerator, accept all of the default options with the following exceptions:

- **Workload namespace:** Update this field with the name of the namespace you will use to build the application in your build cluster
- **Service namespace:** Update this field with the name of the namespace you will use to deploy a RabbitMQ cluster on your Tanzu Application Platform run cluster

Apply the workload resources to your build cluster

To build the application services, run the following command to apply the workload resources to your build cluster. You can also clone or fork the repository in this command to either use the YAML files locally or point to your own Git repository.

```
ytt -f workloads.yaml -v workloadNamespace=WORKLOAD-NAMESPACE | kubectl apply -f-
```

Where `WORKLOAD-NAMESPACE` is the name of your build namespace

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/where-for-dinner-tap-tsm/main/workload
s.yaml \
-v workloadNamespace=workloads | kubectl apply -f-
```

If you are using a GitOps workflow with your build cluster, after the workloads are built the deployment information is pushed to your GitOps repository.

If you follow these instructions without pull requests in the GitOps workflow, the config-writer pods that commit deployment information to the GtiOps repository might fail because of concurrency conflicts. A workaround for this is to delete the failed workloads from the build cluster and re-run the command provided in the instructions.

Install service claim resources on the cluster

Where For Dinner requires a RabbitMQ cluster installed on your run cluster. You must install RabbitMQ on the same run cluster that is named `RunCluster01` in the following deployment section. Additionally, you must install service claim resources on this cluster.

1. If you haven't already done so, install the RabbitMQ Cluster Operator on the run cluster by running:

   ```
   kubectl apply -f "https://github.com/rabbitmq/cluster-operator/releases/downloa
   d/v1.13.1/cluster-operator.yml"
   ```

2. Spin up an instance of a RabbitMQ cluster by running:

   ```
   kubectl create ns SERVICE-NAMESPACE

   ytt -f rmqCluster.yaml -v serviceNamespace=SERVICE-NAMESPACE | kubectl apply -f
   -
   ```

   Where `SERVICE-NAMESPACE` is the namespace of where you want to deploy your RabbitMQ cluster

   For example:

   ```
   kubectl create ns service-instances

   ytt -f https://raw.githubusercontent.com/gm2552/where-for-dinner-tap-tsm/main/r
   mqCluster.yaml -v \
   serviceNamespace=service-instances | kubectl apply -f-
   ```

3. Create service toolkit resources for the RabbitMQ class and resource claim by running:

```
ytt -f rmqResourceClaim.yaml -v serviceNamespace=SERVICE-NAMESPACE -v \
workloadNamespace=WORKLOAD-NAMESPACE | kubectl apply -f-
```

Where `SERVICE-NAMESPACE` and `WORKLOAD-NAMESPACE` are the namespaces where you deployed your RabbitMQ cluster and the namespace where the application service will run.

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/where-for-dinner-tap-tsm/main/r
mqResourceClaim.yaml \
-v serviceNamespace=service-instances -v workloadNamespace=where-for-dinner | k
ubectl apply -f-
```

Run cluster deployment

Workloads are deployed to the run cluster using deliverable resources. This section applies the deliverable resources directly to the run clusters instead of using a kapp application.

This deployment assumes that two clusters are part of the Tanzu Service Mesh GNS Where For Dinner. These example clusters are named `RunCluster01` and `RunCluster02`. The majority of the workload is deployed to `RunCluster01` while the crawler workload is deployed to `RunCluster02`.

The deliverable objects reference the GitOps repository, where the build cluster has written deployment information, and needs to reference this repository in the following commands.

Deploy the workloads to the run clusters by running these commands against their respective clusters:

```
ytt -f cluster01Deliverables.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
gitOpsSecret=GIT-OPS-SECRET -v gitOpsRepo=GIT-OPS-REPO | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads are deployed
- `GIT-OPS-SECRET` is the GitOps secret used to access the GitOps repository
- `GIT-OPS-REPO` is the URL of the GitOps repository where the build cluster wrote out deployment configuration information

```
ytt -f cluster02Deliverables.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
gitOpsSecret=GIT-OPS-SECRET -v gitOpsRepo=GIT-OPS-REPO | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads are deployed
- `GIT-OPS-SECRET` is the GitOps secret used to access the GitOps repository
- `GIT-OPS-REPO` is the URL of the GitOps repository where the build cluster wrote out deployment configuration information

To run this deployment on cluster `RunCluster01`, for example, you run:

```
ytt -f https://raw.githubusercontent.com/gm2552/where-for-dinner-tap-tsm/main/cluster0
1Deliverables.yaml -v \
workloadNamespace=where-for-dinner -v gitOpsSecret=tap-play-gitops-secret -v \
gitOpsRepo=https://github.com/gm2552/tap-play-gitops.git | kubectl apply -f-
```

To run this deployment on cluster `RunCluster02`, for example, you run:

```
ytt -f https://raw.githubusercontent.com/gm2552/where-for-dinner-tap-tsm/main/cluster0
2Deliverables.yaml -v \
```

```
workloadNamespace=where-for-dinner -v gitOpsSecret=tap-play-gitops-secret -v \
gitOpsRepo=https://github.com/gm2552/tap-play-gitops.git | kubectl apply -f-
```

You can create an Istio ingress resource on `RunCluster01` if you do not plan on using the GNS capabilities to expose the application to external networks.

You must create a domain name system address (DNS A) record in your DNS provider's configuration tool to point to the Istio load-balanced IP address of `RunCluster01`. The DNS configuration is out of the scope of this topic.

Create the ingress by running:

```
ytt -f ingress.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v domainName=DOMAIN-NAME
| kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workload is deployed
- `DOMAIN-NAME` is the public domain that will host your application

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/where-for-dinner-tap-tsm/main/ingress.
yaml -v \
workloadNamespace=where-for-dinner -v domainName=tsmdemo.perfect300rock.com | kubectl
apply -f-
```

Create a global namespace

The example clusters have the names `RunCluster01` and `RunCluster02`, and they assume the workload and service namespaces of where-for-dinner and service-instances, respectively.

1. Open the Tanzu Service Mesh console and create a new GNS.
2. Configure the following settings in each step:
    1. General details
        - **GNS Name:** where-for-dinner
        - **Domain:** where-for-dinner.lab
    2. Namespace mapping
        - Namespace mapping Rule 1
            - **Cluster name:** RunCluster01
            - **Namespace:** where-for-dinner
        - Namespace Mapping Rule 2
            - **Cluster name:** RunCluster02
            - **Namespace:** where-for-dinner
        - Namespace Mapping Rule 3
            - **Cluster name:** RunCluster01
            - **Namespace:** service-instances
    3. Autodiscovery. Use the default settings.
    4. Public services
        - **Service name:** where-for-dinner
        - **Service port:** 80

- **Public URL:** http where-for-dinner . Select a domain.

5. Global server load balancing and resiliency. Use the default settings.

You can now access the Where For Dinner application with the URL configured earlier.

Deployment use case: ACME Fitness Store

The following instructions describe an end-to-end process for configuring, building, and deploying the ACME Fitness Store application into a Tanzu Service Mesh GNS. In this use case, the application is deployed across two Tanzu Application Platform run clusters. ytt is used to run the build and deployment commands.

The configuration resources referenced in this scenario are in the acme-fitness-tap-tsm Git repository.

Deploy AppSSO

ACME requires the use of an AppSSO authorization server and client registration resource. Install these resources on the same run cluster that is named `RunCluster01` in the deployment section.

1. Deploy the authorization server instance by running:

   ```
   ytt -f appSSOInstance.yaml -v workloadNamespace=WORKLOAD-NAMESPACE \
   -v devDefaultAccountUsername=DEV-DEFAULT-ACCOUNT-USERNAME -v \
   devDefaultAccountPassword=DEV-DEFAULT-ACCOUNT-PASSWORD | kubectl apply -f-
   ```

   Where:

   - `WORKLOAD-NAMESPACE` is the namespace where the workloads will be deployed
   - `DEV-DEFAULT-ACCOUNT-USERNAME` is the user name for the ACME application authentication
   - `DEV-DEFAULT-ACCOUNT-PASSWORD` is the password for the ACME application authentication

   For example:

   ```
   ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/appSS
   OInstance.yaml -v \
   workloadNamespace=acme -v devDefaultAccountUsername=acme -v \
   devDefaultAccountPassword=fitness | kubectl apply -f-
   ```

2. Create a `ClientRegistration` resource by running:

   ```
   ytt -f appSSOInstance.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
   appSSORedirectURI=APP-SSO-REDIRECT-URI | kubectl apply -f-
   ```

   Where:

   - `WORKLOAD-NAMESPACE` is the namespace where the workloads will be deployed.
   - `APP-SSO-REDIRECT-URI` is the public URI that the authorization server redirects to after a login

   For example:

   ```
   ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/clien
   tRegistrationResourceClaim.yaml \
   -v workloadNamespace=acme -v \
   appSSORedirectURI=http://acme-fitness.tsmdemo.perfect300rock.com/login/oauth2/c
   ode/sso | kubectl apply -f-
   ```

3. Obtain the appSSO Issuer URI by running:

   ```
   kubectl get authserver -n WORKLOAD-NAMESPACE
   ```

   Where `WORKLOAD-NAMESPACE` is the name of the namespace where the workloads will be deployed.

4. Record the Issuer URI because you need it for the next section.

Apply the workload resources to your build cluster

To build the application services, run the following command to apply the workload resources to your build cluster. You can also clone or fork the repository in the following command to either use the YAML files locally or point to your own Git repository.

```
ytt -f workloads.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
appSSOIssuerURI=APP-SSO-ISSUER-URL | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the name of your build namespace
- `APP-SSO-ISSUER-URL` is the URL of the AppSSO authorization server that you deployed earlier

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/workloads.ya
ml -v \
workloadNamespace=workloads -v \
appSSOIssuerURI=http://appsso-acme-fitness.acme.tsmdemo.perfect300rock.com | kubectl a
pply -f-
```

If you are using a GitOps workflow with your build cluster then, after building the workloads, the deployment information is pushed to your GitOps repository.

If you follow these instructions without pull requests in the GitOps workflow, the `config-writer` pods that commit deployment information to the GitOps repository might fail because of concurrency conflicts. A workaround for this is to delete the failed workloads from the build cluster and re-run the command provided in these instructions.

Create the Istio ingress resources

The authorization server requires a publicly accessible URL and must be available before the Spring Cloud Gateway can deploy properly. The authorization server is deployed at the URI authserver app domain.

You must create a domain name system address (DNS A) record in your DNS provider's configuration tool to point to the Istio load-balanced IP address of `RunCluster01`. The DNS configuration is out of the scope of this topic.

Create the Istio ingress resources by running:

```
ytt -f istioGateway.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
appDomainName=APP-DOMAIN | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the name of your build namespace
- `APP-DOMAIN` is the application's DNS domain

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/istioGatewa
y.yaml -v \
workloadNamespace=acme -v appDomainName=tsmdemo.perfect300rock.com | kubectl apply -f-
```

Deploy Redis

A Redis instance is needed for caching the ACME fitness store cart service. Deploy the Redis instance by running:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/redis.yaml -
v \
workloadNamespace=WORKLOAD-NAMESPACE -v redisPassword=REDIS-PASSWORD | kb apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads will be deployed
- `REDIS-PASSWORD` is your password

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/redis.yaml -
v \
workloadNamespace=acme -v redisPassword=fitness | kubectl apply -f-
```

Run cluster deployment

Workloads are deployed to the run cluster by using deliverable resources. In this section you apply the deliverable resources directly to the run clusters, instead of using a kapp application. This deployment assumes that two clusters are part of the Tanzu Service Mesh GNS ACME. In this example these clusters are named `RunCluster01` and `RunCluster02`.

The deliverable objects reference the GitOps repository, where the build cluster has written deployment information, and need to reference this repository in the following commands.

To deploy the workloads to the run clusters, run these commands against their respective clusters:

```
ytt -f cluster01Deliverables.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
gitOpsSecret=GIT-OPS-SECRET -v gitOpsRepo=GIT-OPS-REPO | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads will be deployed
- `GIT-OPS-SECRET` is the GitOps secret used to access the GitOps repository
- `GIT-OPS-REPO` is the URL of the GitOps repository where the build cluster wrote out deployment configuration information

```
ytt -f cluster02Deliverables.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
gitOpsSecret=GIT-OPS-SECRET -v gitOpsRepo=GIT-OPS-REPO | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads will be deployed
- `GIT-OPS-SECRET` is the GitOps secret used to access the GitOps repository
- `GIT-OPS-REPO` is the URL of the GitOps repository where the build cluster wrote out deployment configuration information

For the `RunCluster01` example, run:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/cluster01Del
iverables.yaml \
-v workloadNamespace=acme -v gitOpsSecret=tap-play-gitops-secret -v \
gitOpsRepo=https://github.com/gm2552/tap-play-gitops.git | kubectl apply -f-
```

For the `RunCluster02` example, run:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/cluster02Del
iverables.yaml \
-v workloadNamespace=acme -v gitOpsSecret=tap-play-gitops-secret -v \
gitOpsRepo=https://github.com/gm2552/tap-play-gitops.git | kubectl apply -f-
```

Deploy Spring Cloud Gateway

The following sections describe how to deploy Spring Cloud Gateway.

Install the Spring Cloud Gateway package

The section requires the Spring Cloud Gateway for Kubernetes package to be installed on `RunCluster01`. If Spring Cloud Gateway is already installed on the run cluster, skip these install steps.

In Tanzu Application Platform v1.5 and later, Spring Cloud Gateway is included as an optional package in the Tanzu Application Platform Carvel bundle. Install the Spring Cloud Gateway package with the default settings by using this Tanzu CLI template:

```
tanzu package install scg --package spring-cloud-gateway.tanzu.vmware.com \
-version VERSION-NUMBER -n TAP-INSTALL-NAMESPACE
```

For example:

```
tanzu package install scg --package spring-cloud-gateway.tanzu.vmware.com \
--version 2.0.0-tap.3 -n tap-install
```

Configure the Spring Cloud Gateway instance and route

The Tanzu Application Platform fork of the ACME fitness store uses Spring Cloud Gateway for routing API classes from the web front end to the microservices.

Caution

The Spring Cloud Gateway `spec.service.name` configuration was not built with multicluster or cross-cluster support. The configuration for the gateway routes currently implements a workaround, which is brittle in terms of where certain services are deployed. Future releases of the gateway might have better support for this use case.

Deploy the gateway and applicable routes by running:

```
ytt -f scgInstance.yaml -v workloadNamespace=WORKLOAD-NAMESPACE
```

Where `WORKLOAD-NAMESPACE` is the namespace where the workload is deployed.

```
ytt -f scgRoutes.yaml -v workloadNamespace=WORKLOAD-NAMESPACE
```

Where `WORKLOAD-NAMESPACE` is the namespace where the workload is deployed.

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/scgInstance.
yaml -v \
workloadNamespace=acme | kubectl apply -f-
```

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/scgRoutes.ya
ml -v \
workloadNamespace=acme | kubectl apply -f-
```

Create a global namespace

The example clusters are named `RunCluster01` and `RunCluster02`, and they assume a workload namespace of ACME.

1. Open the Tanzu Service Mesh console and create a new global namespace.
2. Configure the following settings in each step:
    1. General details
        - **GNS name:** acme-tap
        - **Domain:** acme-tap.lab
    2. Namespace mapping
        - Namespace mapping Rule 1
            - **Cluster name:** RunCluster01
            - **Namespace:** acme
        - Namespace Mapping Rule 2
            - **Cluster name:** RunCluster02
            - **Namespace:** acme
    3. Autodiscovery. Use the default settings.
    4. Public Services
        - No Public service
    5. Global server load-balancing and resiliency. Use the default settings.

You can access the application by going to the URL `http://acme-fitness`.

Set up Tanzu Service Mesh

This topic tells you how to set up a Tanzu Application Platform application deployed on Kubernetes with Tanzu Service Mesh (commonly called TSM).

Sample applications are used to demonstrate how a global namespace can provide a network for Kubernetes workloads that are connected and secured within and across clusters, and across clouds.

Note

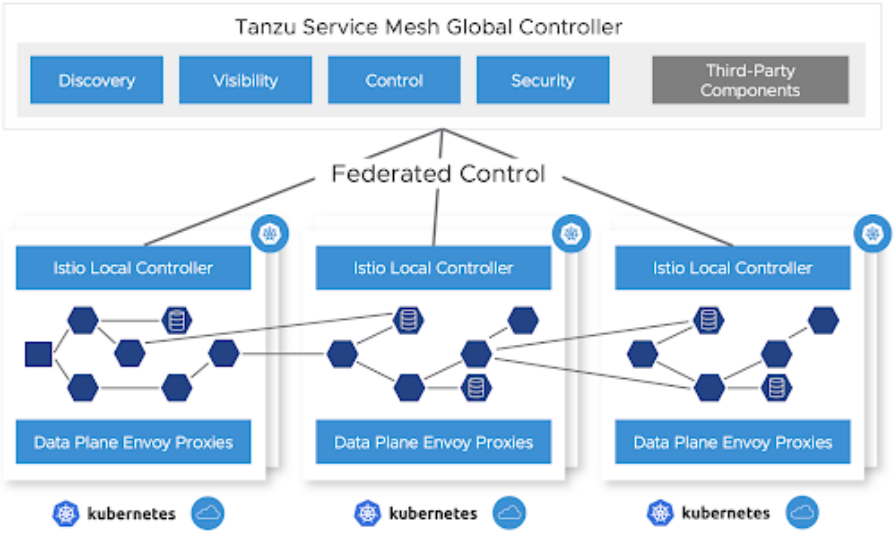
You can also use Tanzu Mission Control (commonly called TMC) to install Tanzu Application Platform (commonly called TAP) on a managed cluster. For more information, see the Tanzu Mission Control documentation.

Prerequisites

Meet the prerequisites, which includes having

- A supported Kubernetes platform
- The correct resource configuration (number of nodes, CPUs, RAM, and so on)
- The required connectivity requirements

Connectivity is only required from your local clusters out to Tanzu Service Mesh and not inwards. This can traverse a corporate proxy as well. In addition, connectivity in the data plane is required between the clusters that must communicate, specifically egress to ingress gateways. No data plane traffic needs to reach the Tanzu Service Mesh software as a service (SaaS) management plane.



Activate your Tanzu Service Mesh subscription

Activate your Tanzu Service Mesh subscription at cloud.vmware.com. After purchasing your Tanzu Service Mesh subscription, the VMware Cloud team sends you instructions. If you don't receive them, you can follow these instructions.

Onboard your clusters to Tanzu Service Mesh as described later in this topic. This deploys the Tanzu Service Mesh local control plane and OSS Istio on your Kubernetes cluster and connects the local control plane to your Tanzu Service Mesh tenant.

Set up Tanzu Application Platform

To enable Tanzu Service Mesh support in Tanzu Application Platform Build clusters:

1. Add the following key to `tap-values.yaml` under the `buildservice` top-level key:

   ```
   buildservice:
    injected_sidecar_support: true
   ```

2. Install Tanzu Application Platform on the run cluster.

End-to-end workload build and deployment scenario

The following sections describe how to build and deploy a workload.

Apply a workload resource to a build cluster

Workloads can be built by using a Tanzu Application Platform supply chain by applying a workload resource to a build cluster. At this time, Tanzu Service Mesh and Tanzu Application Platform cannot use the Knative resources that are the default runtime target when using the `web` resource type.

In Tanzu Application Platform v1.4 and later, two workload types support a Tanzu Service Mesh and Tanzu Application Platform integration: **server** and **worker**.

To work with Tanzu Service Mesh, web workloads must be converted to the `server` or `worker` workload type. Server workloads cause a Kubernetes `Deployment` resource to be created with a `Service` resource that uses port 8080 by default.

1. If the service port that you want is 80 or some other port, add port information to `workload.yaml`. The following example YAML snippets show the changes to make from the `web` to `server` workload type. This is an example before applying the changes:

   ```
   apiVersion: carto.run/v1alpha1
   kind: Workload
   metadata:
    name: where-for-dinner
    labels:
      apps.tanzu.vmware.com/workload-type: web
      app.kubernetes.io/part-of: where-for-dinner-api-gateway
   spec:
    params:
    - name: annotations
   value:
   autoscaling.knative.dev/minScale: "1"
    source:
      git:
        url: https://github.com/vmware-tanzu/application-accelerator-samples.git
      ref:
        branch: tap-1.6.x
      subPath: where-for-dinner/where-for-dinner-api-gateway
   ```

   This is an example modified for Tanzu Service Mesh, which includes the removal of the autoscaling annotation:

   ```
   apiVersion: carto.run/v1alpha1
   kind: Workload
   metadata:
    name: where-for-dinner
    labels:
    apps.tanzu.vmware.com/workload-type: server # modification
    app.kubernetes.io/part-of: where-for-dinner-api-gateway
   spec:
    params:
    - name: ports # modification
    value:
    - port: 80 # modification
      containerPort: 8080 # modification
      name: http # modification
    source:
    git:
        url: https://github.com/vmware-tanzu/application-accelerator-samples.git
      ref:
         branch: tap-1.6.x
    subPath: where-for-dinner/where-for-dinner-api-gateway
   ```

   This results in a deployment and a service that listens on port 80 and forwards traffic to port 8080 on the pod's workload container.

2. Submit the modified YAML to your build cluster by running:

```
tanzu apps NAMESPACE apply --file WORKLOAD-YAML-FILE
```

Where:

- `NAMESPACE` is the namespace that the build cluster uses for building.
- `WORKLOAD-YAML-FILE` is the name of your workload YAML file, such as `workload.yaml`.

After your workload is built a `Deliverable` resource is created.

Configure egress for Tanzu Build Service

For Tanzu Build Service to properly work, provide egress to access the registry where Tanzu Build Service writes application images, and define the registry in the `kp_default_repository` key and the Tanzu Application Platform install registry.

Additionally, configure egress for buildpack builds to download any required dependencies. This configuration varies with different buildpacks and language environments. For example, Java builds might need to download dependencies from Maven central.

Create a global namespace

Using the Tanzu Service Mesh portal or API, create a global namespace (GNS) that includes the namespaces where your application components are deployed. For more information, see Global Namespaces

Whether in a single cluster or multiple clusters, or within the same site or across clouds, after you add a namespace selection to the GNS, the services that Tanzu Application Platform deploys are connected based on the GNS configuration for service discovery and connectivity policies.

If a service must be accessible through the ingress from the outside, it can be configured through the public service option in Tanzu Service Mesh or directly through Istio on the clusters where that service resides. It's best practice to configure the service's accessibility through the GNS.

Run cluster deployment

Before deploying a workload to a run cluster, ensure that any prerequisite resources have already been created on the run cluster. This includes concepts such as data, messaging, routing, security services, RBAC, ResourceClaims, and so on.

After a successful build in a build cluster, workloads can be deployed to the run cluster by applying resulting deliverable resources to the run cluster as described in Getting Started with Multicluster Tanzu Application Platform.

Another option is to create a kapp application that references a GitOps repository to include all deliverable resources for a given cluster. See the following example of a kapp definition that points to a GitOps repository:

```
apiVersion: kappctrl.k14s.io/v1alpha1
kind: App
metadata:
  name: deliverable-gitops
  namespace: where-for-dinner
spec:
  serviceAccountName: default
  fetch:
  - git:
    url: https://github.com/gm2552/tap-play-gitops
    ref: origin/deliverables-tap-east01
    subPath: config
  template:
```

```
  - ytt: {}
  deploy:
  - kapp: {}
```

The advantage of this model is that applications can be deployed or uninstalled from a cluster by managing the contents of the deliverable resources from within the GitOps repository and enabling a GitOps workflow for application and service change control.

Deployment use case: Where For Dinner

The following instructions describe an end-to-end process for configuring, building, and deploying the Where For Dinner application into a Tanzu Service Mesh global namespace.

These instructions use the default configuration of Where For Dinner, which consists of only needing a single-node RabbitMQ cluster, an in-memory database, and no security. The application is deployed across two Tanzu Application Platform run clusters. It requires the `ytt` command to execute the build and deployment commands.

The configuration resources referenced in this scenario are located in the where-for-dinner-tap-tsm GitHub repository.

Create an initial set of configuration files from the accelerator

This use case deployment includes a pre-built set of configuration files in a Git repository. However, they were created from a set of configuration files by using a bootstrapped process that uses the Where For Dinner accelerator, and were later modified.

For reference, you can create an initial set of configuration files from the Where For Dinner accelerator, which is available in Tanzu Application Platform v1.3 and later.

This section does not include instructions for modifying the configuration files from the accelerator into configuration files used in a later section.

From the accelerator, accept all of the default options with the following exceptions:

- **Workload namespace:** Update this field with the name of the namespace you will use to build the application in your build cluster
- **Service namespace:** Update this field with the name of the namespace you will use to deploy a RabbitMQ cluster on your Tanzu Application Platform run cluster

Apply the workload resources to your build cluster

To build the application services, run the following command to apply the workload resources to your build cluster. You can also clone or fork the repository in this command to either use the YAML files locally or point to your own Git repository.

```
ytt -f workloads.yaml -v workloadNamespace=WORKLOAD-NAMESPACE | kubectl apply -f-
```

Where `WORKLOAD-NAMESPACE` is the name of your build namespace

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/where-for-dinner-tap-tsm/main/workload
s.yaml \
-v workloadNamespace=workloads | kubectl apply -f-
```

If you are using a GitOps workflow with your build cluster, after the workloads are built the deployment information is pushed to your GitOps repository.

If you follow these instructions without pull requests in the GitOps workflow, the config-writer pods that commit deployment information to the GtiOps repository might fail because of concurrency conflicts. A workaround for this is to delete the failed workloads from the build cluster and re-run the command provided in the instructions.

Install service claim resources on the cluster

Where For Dinner requires a RabbitMQ cluster installed on your run cluster. You must install RabbitMQ on the same run cluster that is named `RunCluster01` in the following deployment section. Additionally, you must install service claim resources on this cluster.

1. If you haven't already done so, install the RabbitMQ Cluster Operator on the run cluster by running:

   ```
   kubectl apply -f "https://github.com/rabbitmq/cluster-operator/releases/downloa
   d/v1.13.1/cluster-operator.yml"
   ```

2. Spin up an instance of a RabbitMQ cluster by running:

   ```
   kubectl create ns SERVICE-NAMESPACE

   ytt -f rmqCluster.yaml -v serviceNamespace=SERVICE-NAMESPACE | kubectl apply -f
   -
   ```

   Where `SERVICE-NAMESPACE` is the namespace of where you want to deploy your RabbitMQ cluster

   For example:

   ```
   kubectl create ns service-instances

   ytt -f https://raw.githubusercontent.com/gm2552/where-for-dinner-tap-tsm/main/r
   mqCluster.yaml -v \
   serviceNamespace=service-instances | kubectl apply -f-
   ```

3. Create service toolkit resources for the RabbitMQ class and resource claim by running:

   ```
   ytt -f rmqResourceClaim.yaml -v serviceNamespace=SERVICE-NAMESPACE -v \
   workloadNamespace=WORKLOAD-NAMESPACE | kubectl apply -f-
   ```

   Where `SERVICE-NAMESPACE` and `WORKLOAD-NAMESPACE` are the namespaces where you deployed your RabbitMQ cluster and the namespace where the application service will run.

   For example:

   ```
   ytt -f https://raw.githubusercontent.com/gm2552/where-for-dinner-tap-tsm/main/r
   mqResourceClaim.yaml \
   -v serviceNamespace=service-instances -v workloadNamespace=where-for-dinner | k
   ubectl apply -f-
   ```

Run cluster deployment

Workloads are deployed to the run cluster using deliverable resources. This section applies the deliverable resources directly to the run clusters instead of using a kapp application.

This deployment assumes that two clusters are part of the Tanzu Service Mesh GNS Where For Dinner. These example clusters are named `RunCluster01` and `RunCluster02`. The majority of the workload is deployed to `RunCluster01` while the crawler workload is deployed to `RunCluster02`.

The deliverable objects reference the GitOps repository, where the build cluster has written deployment information, and needs to reference this repository in the following commands.

Deploy the workloads to the run clusters by running these commands against their respective clusters:

```
ytt -f cluster01Deliverables.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
gitOpsSecret=GIT-OPS-SECRET -v gitOpsRepo=GIT-OPS-REPO | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads are deployed
- `GIT-OPS-SECRET` is the GitOps secret used to access the GitOps repository
- `GIT-OPS-REPO` is the URL of the GitOps repository where the build cluster wrote out deployment configuration information

```
ytt -f cluster02Deliverables.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
gitOpsSecret=GIT-OPS-SECRET -v gitOpsRepo=GIT-OPS-REPO | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads are deployed
- `GIT-OPS-SECRET` is the GitOps secret used to access the GitOps repository
- `GIT-OPS-REPO` is the URL of the GitOps repository where the build cluster wrote out deployment configuration information

To run this deployment on cluster `RunCluster01`, for example, you run:

```
ytt -f https://raw.githubusercontent.com/gm2552/where-for-dinner-tap-tsm/main/cluster0
1Deliverables.yaml -v \
workloadNamespace=where-for-dinner -v gitOpsSecret=tap-play-gitops-secret -v \
gitOpsRepo=https://github.com/gm2552/tap-play-gitops.git | kubectl apply -f-
```

To run this deployment on cluster `RunCluster02`, for example, you run:

```
ytt -f https://raw.githubusercontent.com/gm2552/where-for-dinner-tap-tsm/main/cluster0
2Deliverables.yaml -v \
workloadNamespace=where-for-dinner -v gitOpsSecret=tap-play-gitops-secret -v \
gitOpsRepo=https://github.com/gm2552/tap-play-gitops.git | kubectl apply -f-
```

You can create an Istio ingress resource on `RunCluster01` if you do not plan on using the GNS capabilities to expose the application to external networks.

You must create a domain name system address (DNS A) record in your DNS provider's configuration tool to point to the Istio load-balanced IP address of `RunCluster01`. The DNS configuration is out of the scope of this topic.

Create the ingress by running:

```
ytt -f ingress.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v domainName=DOMAIN-NAME
| kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workload is deployed
- `DOMAIN-NAME` is the public domain that will host your application

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/where-for-dinner-tap-tsm/main/ingress.
yaml -v \
workloadNamespace=where-for-dinner -v domainName=tsmdemo.perfect300rock.com | kubectl
apply -f-
```

Create a global namespace

The example clusters have the names `RunCluster01` and `RunCluster02`, and they assume the workload and service namespaces of where-for-dinner and service-instances, respectively.

1. Open the Tanzu Service Mesh console and create a new GNS.
2. Configure the following settings in each step:
    1. General details
        - **GNS Name:** where-for-dinner
        - **Domain:** where-for-dinner.lab
    2. Namespace mapping
        - Namespace mapping Rule 1
            - **Cluster name:** RunCluster01
            - **Namespace:** where-for-dinner
        - Namespace Mapping Rule 2
            - **Cluster name:** RunCluster02
            - **Namespace:** where-for-dinner
        - Namespace Mapping Rule 3
            - **Cluster name:** RunCluster01
            - **Namespace:** service-instances
    3. Autodiscovery. Use the default settings.
    4. Public services
        - **Service name:** where-for-dinner
        - **Service port:** 80
        - **Public URL:** http where-for-dinner . Select a domain.
    5. Global server load balancing and resiliency. Use the default settings.

You can now access the Where For Dinner application with the URL configured earlier.

Deployment use case: ACME Fitness Store

The following instructions describe an end-to-end process for configuring, building, and deploying the ACME Fitness Store application into a Tanzu Service Mesh GNS. In this use case, the application is deployed across two Tanzu Application Platform run clusters. ytt is used to run the build and deployment commands.

The configuration resources referenced in this scenario are in the acme-fitness-tap-tsm Git repository.

Deploy AppSSO

ACME requires the use of an AppSSO authorization server and client registration resource. Install these resources on the same run cluster that is named `RunCluster01` in the deployment section.

1. Deploy the authorization server instance by running:

```
ytt -f appSSOInstance.yaml -v workloadNamespace=WORKLOAD-NAMESPACE \
-v devDefaultAccountUsername=DEV-DEFAULT-ACCOUNT-USERNAME -v \
devDefaultAccountPassword=DEV-DEFAULT-ACCOUNT-PASSWORD | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads will be deployed
- `DEV-DEFAULT-ACCOUNT-USERNAME` is the user name for the ACME application authentication
- `DEV-DEFAULT-ACCOUNT-PASSWORD` is the password for the ACME application authentication

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/appSS
OInstance.yaml -v \
workloadNamespace=acme -v devDefaultAccountUsername=acme -v \
devDefaultAccountPassword=fitness | kubectl apply -f-
```

2. Create a `ClientRegistration` resource by running:

```
ytt -f appSSOInstance.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
appSSORedirectURI=APP-SSO-REDIRECT-URI | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads will be deployed.
- `APP-SSO-REDIRECT-URI` is the public URI that the authorization server redirects to after a login

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/clien
tRegistrationResourceClaim.yaml \
-v workloadNamespace=acme -v \
appSSORedirectURI=http://acme-fitness.tsmdemo.perfect300rock.com/login/oauth2/c
ode/sso | kubectl apply -f-
```

3. Obtain the appSSO Issuer URI by running:

```
kubectl get authserver -n WORKLOAD-NAMESPACE
```

Where `WORKLOAD-NAMESPACE` is the name of the namespace where the workloads will be deployed.

4. Record the Issuer URI because you need it for the next section.

Apply the workload resources to your build cluster

To build the application services, run the following command to apply the workload resources to your build cluster. You can also clone or fork the repository in the following command to either use the YAML files locally or point to your own Git repository.

```
ytt -f workloads.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
appSSOIssuerURI=APP-SSO-ISSUER-URL | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the name of your build namespace

- `APP-SSO-ISSUER-URL` is the URL of the AppSSO authorization server that you deployed earlier

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/workloads.ya
ml -v \
workloadNamespace=workloads -v \
appSSOIssuerURI=http://appsso-acme-fitness.acme.tsmdemo.perfect300rock.com | kubectl a
pply -f-
```

If you are using a GitOps workflow with your build cluster then, after building the workloads, the deployment information is pushed to your GitOps repository.

If you follow these instructions without pull requests in the GitOps workflow, the `config-writer` pods that commit deployment information to the GitOps repository might fail because of concurrency conflicts. A workaround for this is to delete the failed workloads from the build cluster and re-run the command provided in these instructions.

Create the Istio ingress resources

The authorization server requires a publicly accessible URL and must be available before the Spring Cloud Gateway can deploy properly. The authorization server is deployed at the URI authserver app domain.

You must create a domain name system address (DNS A) record in your DNS provider's configuration tool to point to the Istio load-balanced IP address of `RunCluster01`. The DNS configuration is out of the scope of this topic.

Create the Istio ingress resources by running:

```
ytt -f istioGateway.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
appDomainName=APP-DOMAIN | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the name of your build namespace
- `APP-DOMAIN` is the application's DNS domain

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/istioGatewa
y.yaml -v \
workloadNamespace=acme -v appDomainName=tsmdemo.perfect300rock.com | kubectl apply -f-
```

Deploy Redis

A Redis instance is needed for caching the ACME fitness store cart service. Deploy the Redis instance by running:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/redis.yaml -
v \
workloadNamespace=WORKLOAD-NAMESPACE -v redisPassword=REDIS-PASSWORD | kb apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads will be deployed
- `REDIS-PASSWORD` is your password

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/redis.yaml -
v \
workloadNamespace=acme -v redisPassword=fitness | kubectl apply -f-
```

Run cluster deployment

Workloads are deployed to the run cluster by using deliverable resources. In this section you apply the deliverable resources directly to the run clusters, instead of using a kapp application. This deployment assumes that two clusters are part of the Tanzu Service Mesh GNS ACME. In this example these clusters are named `RunCluster01` and `RunCluster02`.

The deliverable objects reference the GitOps repository, where the build cluster has written deployment information, and need to reference this repository in the following commands.

To deploy the workloads to the run clusters, run these commands against their respective clusters:

```
ytt -f cluster01Deliverables.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
gitOpsSecret=GIT-OPS-SECRET -v gitOpsRepo=GIT-OPS-REPO | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads will be deployed
- `GIT-OPS-SECRET` is the GitOps secret used to access the GitOps repository
- `GIT-OPS-REPO` is the URL of the GitOps repository where the build cluster wrote out deployment configuration information

```
ytt -f cluster02Deliverables.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
gitOpsSecret=GIT-OPS-SECRET -v gitOpsRepo=GIT-OPS-REPO | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads will be deployed
- `GIT-OPS-SECRET` is the GitOps secret used to access the GitOps repository
- `GIT-OPS-REPO` is the URL of the GitOps repository where the build cluster wrote out deployment configuration information

For the `RunCluster01` example, run:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/cluster01Del
iverables.yaml \
-v workloadNamespace=acme -v gitOpsSecret=tap-play-gitops-secret -v \
gitOpsRepo=https://github.com/gm2552/tap-play-gitops.git | kubectl apply -f-
```

For the `RunCluster02` example, run:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/cluster02Del
iverables.yaml \
-v workloadNamespace=acme -v gitOpsSecret=tap-play-gitops-secret -v \
gitOpsRepo=https://github.com/gm2552/tap-play-gitops.git | kubectl apply -f-
```

Deploy Spring Cloud Gateway

The following sections describe how to deploy Spring Cloud Gateway.

Install the Spring Cloud Gateway package

The section requires the Spring Cloud Gateway for Kubernetes package to be installed on `RunCluster01`. If Spring Cloud Gateway is already installed on the run cluster, skip these install

steps.

In Tanzu Application Platform v1.5 and later, Spring Cloud Gateway is included as an optional package in the Tanzu Application Platform Carvel bundle. Install the Spring Cloud Gateway package with the default settings by using this Tanzu CLI template:

```
tanzu package install scg --package spring-cloud-gateway.tanzu.vmware.com \
-version VERSION-NUMBER -n TAP-INSTALL-NAMESPACE
```

For example:

```
tanzu package install scg --package spring-cloud-gateway.tanzu.vmware.com \
--version 2.0.0-tap.3 -n tap-install
```

Configure the Spring Cloud Gateway instance and route

The Tanzu Application Platform fork of the ACME fitness store uses Spring Cloud Gateway for routing API classes from the web front end to the microservices.

> **Caution**
>
> The Spring Cloud Gateway `spec.service.name` configuration was not built with multicluster or cross-cluster support. The configuration for the gateway routes currently implements a workaround, which is brittle in terms of where certain services are deployed. Future releases of the gateway might have better support for this use case.

Deploy the gateway and applicable routes by running:

```
ytt -f scgInstance.yaml -v workloadNamespace=WORKLOAD-NAMESPACE
```

Where `WORKLOAD-NAMESPACE` is the namespace where the workload is deployed.

```
ytt -f scgRoutes.yaml -v workloadNamespace=WORKLOAD-NAMESPACE
```

Where `WORKLOAD-NAMESPACE` is the namespace where the workload is deployed.

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/scgInstance.
yaml -v \
workloadNamespace=acme | kubectl apply -f-
```

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/scgRoutes.ya
ml -v \
workloadNamespace=acme | kubectl apply -f-
```

Create a global namespace

The example clusters are named `RunCluster01` and `RunCluster02`, and they assume a workload namespace of ACME.

1. Open the Tanzu Service Mesh console and create a new global namespace.
2. Configure the following settings in each step:
    1. General details

        - **GNS name:** acme-tap
        - **Domain:** acme-tap.lab
    2. Namespace mapping
        - Namespace mapping Rule 1
            - **Cluster name:** RunCluster01
            - **Namespace:** acme
        - Namespace Mapping Rule 2
            - **Cluster name:** RunCluster02
            - **Namespace:** acme
    3. Autodiscovery. Use the default settings.
    4. Public Services
        - No Public service
    5. Global server load-balancing and resiliency. Use the default settings.

You can access the application by going to the URL `http://acme-fitness`.

Use external observability tools

This topic tells you how to generate metrics for your Tanzu Application Platform (commonly known as TAP) applications to enable observability with external tools. It explains how to set up Prometheus on your cluster or integrate an existing Datadog installation.

About Prometheus metrics

Prometheus is an open-source monitoring tool that defines a simple text-based metrics format with client libraries for a wide range of programming languages and frameworks.

By integrating these client libraries into your applications, you can effortlessly generate metrics in the Prometheus format. These metrics are accessible through an HTTP endpoint, enabling Prometheus and other observability tools to conveniently consume (scrape) them.

For Kubernetes, there is an established convention that facilitates the automatic discovery of pods exposing Prometheus metrics. This involves incorporating specific annotations on the pods exposing information for path and port of the metrics endpoint.

Prometheus and other observability tools like Datadog can discover annotated pods and collect the metrics from the endpoint.

Use Prometheus as your observability tool

There are multiple ways to install Prometheus on your cluster:

- Use the Prometheus Operator
- Use the `kube-prometheus-stack` Helm chart
- Use the Prometheus Helm chart

Prometheus Operator
The Prometheus Operator offers a simplified way to use custom resources to deploy and configure Prometheus, Alertmanager, and related monitoring components.

To install using the Prometheus Operator:

1. Install the Prometheus Operator bundle by running:

```
LATEST=$(curl -s https://api.github.com/repos/prometheus-operator/prometheus-operator/releases/latest | jq -cr .tag_name)
curl -sL https://github.com/prometheus-operator/prometheus-operator/releases/download/${LATEST}/bundle.yaml | kubectl create -f -
```

2. For RBAC-based environments, create the RBAC rules for the Prometheus service account by running:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ServiceAccount
metadata:
  name: prometheus
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: prometheus
rules:
- apiGroups: [""]
  resources:
  - nodes
  - nodes/metrics
  - services
  - endpoints
  - pods
  verbs: ["get", "list", "watch"]
- apiGroups: [""]
  resources:
  - configmaps
  verbs: ["get"]
- apiGroups:
  - networking.k8s.io
  resources:
  - ingresses
  verbs: ["get", "list", "watch"]
- nonResourceURLs: ["/metrics"]
  verbs: ["get"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: prometheus
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: prometheus
subjects:
- kind: ServiceAccount
  name: prometheus
  namespace: default
EOF
```

3. Configure a Prometheus scraping job. This job must monitor all pods that are marked with the designated Prometheus annotations.

Note

The Prometheus Operator does not support annotation-based discovery of services by default. To enable these annotations, you must set up a custom scrape job configuration.

1. Create a file named `prometheus-scrape-config.yaml` with the following content:

```
# Example scrape config for pods
#
# The relabeling allows the actual pod scrape endpoint to be configure
d through the
# following annotations:
#
# * `prometheus.io/scrape`: Only scrape pods that have a value of `tru
e`.
# * `prometheus.io/scheme`: If the metrics endpoint is secured then yo
u must
# set this to `https` and most likely set the `tls_config` of the scra
pe config.
# * `prometheus.io/path`: If the metrics path is not `/metrics` overri
de this.
# * `prometheus.io/port`: Scrape the pod on the indicated port instead
of the default of `9102`.
- job_name: 'kubernetes-pods'
  honor_labels: true

  kubernetes_sd_configs:
    - role: pod

  relabel_configs:
    - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_s
crape]
      action: keep
      regex: true
    - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_s
cheme]
      action: replace
      regex: (https?)
      target_label: __scheme__
    - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_p
ath]
      action: replace
      target_label: __metrics_path__
      regex: (.+)
    - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_p
ort, __meta_kubernetes_pod_ip]
      action: replace
      regex: (\d+);(([A-Fa-f0-9]{1,4}::?){1,7}[A-Fa-f0-9]{1,4})
      replacement: '[$2]:$1'
      target_label: __address__
    - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_p
ort, __meta_kubernetes_pod_ip]
      action: replace
      regex: (\d+);((([0-9]+?)(\.|$)){4})
      replacement: $2:$1
      target_label: __address__
    - action: labelmap
      regex: __meta_kubernetes_pod_annotation_prometheus_io_param_(.+)
      replacement: __param_$1
    - action: labelmap
      regex: __meta_kubernetes_pod_label_(.+)
    - source_labels: [__meta_kubernetes_namespace]
      action: replace
      target_label: namespace
    - source_labels: [__meta_kubernetes_pod_name]
```

```
      action: replace
      target_label: pod
    - source_labels: [__meta_kubernetes_pod_phase]
      regex: Pending|Succeeded|Failed|Completed
      action: drop
    - source_labels: [__meta_kubernetes_pod_node_name]
      action: replace
      target_label: node
```

2. Generate a secret using the file you just created by running:

```
kubectl create secret generic additional-scrape-configs --from-file=pr
ometheus-scrape-config.yaml
```

3. Create a Prometheus resource that uses this secret by running:

```
cat <<EOF | kubectl apply -f -
apiVersion: monitoring.coreos.com/v1
kind: Prometheus
metadata:
  name: prometheus
spec:
  serviceAccountName: prometheus
  podMonitorSelector: {}
  additionalScrapeConfigs:
    name: additional-scrape-configs
    key: prometheus-scrape-config.yaml
  resources:
    requests:
      memory: 400Mi
  enableAdminAPI: false
EOF
```

   The Prometheus Operator automatically detects the configuration and generates a scrape job from it. This job is executed regularly by the Prometheus instance that is set up.

4. To access the Prometheus web interface, you must make port `9090` of the Prometheus server pod accessible outside the cluster through a Kubernetes service or ingress. For development purposes, you can forward the port to your local machine using kubectl.

5. To see the scrape configuration that the Prometheus instance has picked up, run:

```
kubectl get secret prometheus-prometheus -ojson | jq -r '.data["prometheus.ya
ml.gz"]' | base64 -d | gunzip
```

For more information about the Prometheus Operator, including how to persist your metrics or activate alerting features, see the Prometheus Operator documentation.

kube-prometheus-stack Helm chart

The Prometheus community has developed the `kube-prometheus-stack` Helm chart, which establishes a comprehensive cluster monitoring stack. The Helm chart sets up various scraping jobs for metrics related to the Kubernetes cluster.

These are some of its features:

- Includes the Prometheus Operator
- Includes cluster monitoring
- Ensures high availability
- Integrates Node Exporter Grafana dashboards

For more information, see the kube-prometheus-stack Helm chart README in GitHub.

To install the `kube-prometheus-stack` Helm chart:

1. Add the Helm repository by running:

```
helm repo add prometheus-community https://prometheus-community.github.io/hel
m-charts
helm repo update
```

2. Because there isn't a predefined scrape job configuration to support annotation-based discovery of services, you must create the configuration secret `additional-scrape-configs`.

    1. Create a file named `prometheus-scrape-config.yaml` with the following content:

```
# Example scrape config for pods
#
# The relabeling allows the actual pod scrape endpoint to be configure
d through the
# following annotations:
#
# * `prometheus.io/scrape`: Only scrape pods that have a value of `tru
e`.
# * `prometheus.io/scheme`: If the metrics endpoint is secured then yo
u must
# set this to `https` and most likely set the `tls_config` of the scra
pe config.
# * `prometheus.io/path`: If the metrics path is not `/metrics` overri
de this.
# * `prometheus.io/port`: Scrape the pod on the indicated port instead
of the default of `9102`.
- job_name: 'kubernetes-pods'
  honor_labels: true

  kubernetes_sd_configs:
    - role: pod

  relabel_configs:
    - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_s
crape]
      action: keep
      regex: true
    - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_s
cheme]
      action: replace
      regex: (https?)
      target_label: __scheme__
    - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_p
ath]
      action: replace
      target_label: __metrics_path__
      regex: (.+)
    - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_p
ort, __meta_kubernetes_pod_ip]
      action: replace
      regex: (\d+);(([A-Fa-f0-9]{1,4}::?){1,7}[A-Fa-f0-9]{1,4})
      replacement: '[$2]:$1'
      target_label: __address__
    - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_p
ort, __meta_kubernetes_pod_ip]
      action: replace
      regex: (\d+);((([0-9]+?)(\.|$)){4})
      replacement: $2:$1
      target_label: __address__
```

```
    - action: labelmap
      regex: __meta_kubernetes_pod_annotation_prometheus_io_param_(.+)
      replacement: __param_$1
    - action: labelmap
      regex: __meta_kubernetes_pod_label_(.+)
    - source_labels: [__meta_kubernetes_namespace]
      action: replace
      target_label: namespace
    - source_labels: [__meta_kubernetes_pod_name]
      action: replace
      target_label: pod
    - source_labels: [__meta_kubernetes_pod_phase]
      regex: Pending|Succeeded|Failed|Completed
      action: drop
    - source_labels: [__meta_kubernetes_pod_node_name]
      action: replace
      target_label: node
```

2. Generate a secret using the file you just created by running:

```
kubectl create secret generic additional-scrape-configs --from-file=pr
ometheus-scrape-config.yaml
```

3. Install the `kube-prometheus-stack` Helm chart by running:

```
helm upgrade --install kube-prometheus-stack prometheus-community/kube-promet
heus-stack \
  --set prometheus.prometheusSpec.additionalScrapeConfigsSecret.name=addition
al-scrape-configs \
  --set prometheus.prometheusSpec.additionalScrapeConfigsSecret.key=prometheu
s-scrape-config.yaml \
  --set prometheus.prometheusSpec.additionalScrapeConfigsSecret.enabled=true
```

4. To see the scrape configuration that the Prometheus instance has picked up, run:

```
kubectl get secret prometheus-kube-prometheus-stack-prometheus -ojson | jq -r
'.data["prometheus.yaml.gz"]' | base64 -d | gunzip
```

Prometheus Helm chart

The Prometheus community provides the Prometheus Helm chart to install Prometheus on your Kubernetes cluster. The Helm chart installs scraping job configurations for pods and services tagged with Prometheus scraping annotations and sets up scraping for metrics related to the Kubernetes cluster. It includes key components such as Alert Manager, kube-state-metrics, Node Exporter, and Push Gateway.

For more information, see the Prometheus Helm chart README in GitHub.

To install the Prometheus Helm chart:

1. Add the Helm repository by running:

```
helm repo add prometheus-community https://prometheus-community.github.io/hel
m-charts
helm repo update
```

2. Install the Prometheus Helm chart by running:

```
helm upgrade --install prometheus prometheus-community/prometheus
```

Use Datadog as your observability tool

If you use Datadog, you can use it to scrape the Prometheus endpoints without having to install Prometheus itself. Datadog automatically gathers Prometheus metrics from pods that are annotated with the default Prometheus annotations. The Datadog Agent forwards the metrics to the Datadog servers.

To use Datadog as your observability tool:

1. Add the Helm repository by running:

   ```
   helm repo add datadog https://helm.datadoghq.com
   helm repo update
   ```

2. Install the Datadog Agent Helm chart by running:

   ```
   helm upgrade --install datadog-operator datadog/datadog-operator
   ```

   Note

   There is a known issue with Datadog Cluster Agent on Azure Kubernetes Service (AKS) clusters. For more information, see the troubleshooting item Datadog agent cannot reconcile webhook on AKS.

3. Generate a new API key in Datadog for the Agent that will push metrics to Datadog. You do this in the Datadog UI, under `Profile/Organization Settings/API Keys`.

4. Create a secret for the Datadog API key by running:

   ```
   kubectl create secret generic datadog-secret --from-literal api-key=API-KEY
   ```

   Where `API-KEY` is the API you generated in the previous step.

5. Install the Datadog Agent by running:

   ```
   cat <<EOF | kubectl apply -f -
   apiVersion: datadoghq.com/v2alpha1
   kind: DatadogAgent
   metadata:
     name: datadog
   spec:
     global:
       clusterName: YOUR-CLUSTER-NAME
       site: DATADOG-HOST-NAME
       credentials:
         apiSecret:
           secretName: datadog-secret
           keyName: api-key
     features:
       prometheusScrape:
         enabled: true
         enableServiceEndpoints: true
   EOF
   ```

   Where:

   - `YOUR-CLUSTER-NAME` is the name of your cluster as you want to see it in Datadog.
   - `DATADOG-HOST-NAME` is your Datadog host name, for example, `datadoghq.eu`.

Enable metric collection on Spring Boot workloads

To enable Spring Boot workloads to create Prometheus metrics, if using Maven, add the following dependencies to your `pom.xml`:

```
<dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <dependency>
      <groupId>io.micrometer</groupId>
      <artifactId>micrometer-registry-prometheus</artifactId>
      <scope>runtime</scope>
    </dependency>
</dependencies>
```

This creates default metrics for the JVM, HTTP traffic, and more. For a list of supported metrics, see the Spring Boot documentation.

When deploying Spring Boot workloads on Tanzu Application Platform, the Spring Boot conventions ensure that actuator endpoints are exposed. The Prometheus metrics endpoint is also made accessible.

For Prometheus to find this endpoint on a pod, you must include specific annotations in your `workload.yaml` file. These annotations must align with your configuration. After adding these annotations, deploy the changes:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: spring-petclinic
  annotations:
    prometheus.io/scrape: 'true'
    prometheus.io/path: '/actuator/prometheus'
    prometheus.io/port: '8081'
    # ...
```

For more information about the Spring Boot Prometheus integration, see the Spring Boot Reference Documentation.

Use a Grafana dashboard for Tanzu Application Platform observability

This topic tells you how to configure and set up a public Grafana-Prometheus Helm with Tanzu Application Platform (commonly known as TAP) `CustomResource` metrics collection.

Prerequisites

Before proceeding you must have:

- kubectl
- Helm
- A Kubernetes cluster with Tanzu Application Platform and associated workloads

Download and unzip the Tanzu CLI binary

To download the Tanzu CLI binary and set up the cluster:

1. Go to VMware Tanzu Network.

2. Select 1.9.1 from the release drop-down menu for Tanzu Application Platform.
3. Click **Grafana Dashboard for Tanzu Application Platform (Beta)**.
4. Download `prometheus-grafana-dashboard-for-tap-1.0.0-beta.1.zip`.
5. If on macOS or Linux, unzip `prometheus-grafana-dashboard-for-tap-1.0.0-beta.1.zip`. If you're using Windows, use the Windows extractor tool to unzip it.

Install a public Helm chart

To install a public Helm chart:

1. Apply `tap-metrics.yaml` on the Tanzu Application Platform cluster, which enables collection of the Tanzu Application Platform `CustomResource` metrics.
2. Install a public Helm chart for Prometheus with a public HTTP endpoint in all Tanzu Application Platform clusters where necessary for observability.

   For example, if you have a View cluster, a Build cluster, and a Run cluster:

   1. Set the context to View cluster and install the Prometheus public Helm chart by running:

      ```
      # Add Prometheus helm repo
      helm repo add prometheus-community https://prometheus-community.github.i
      o/helm-charts
      helm repo update

      # Install prometheus chart
      helm install prometheus -f DOWNLOADED-TAP-METRICS-YAML-FILE \
      prometheus-community/kube-prometheus-stack --namespace tap-monitoring --c
      reate-namespace
      ```

   2. Repeat the previous step for the Build cluster and the Run cluster.
   3. You now have a Prometheus endpoint, for each cluster, which acts as a data source for Grafana in later steps. See the public LoadBalancer IP `<PUBLIC_ENDPOINT>` in each cluster where Prometheus is installed by running:

      ```
      kubectl get svc -n tap-monitoring | grep prometheus
      ```

      Example output:

      ```
      $ kubectl get svc -n tap-monitoring | grep prometheus
      prometheus-operated                         ClusterIP      None
      <none>            9090/TCP                       20h
      tap-observability-kube-pro-prometheus       LoadBalancer   10.0.19.222
      <PUBLIC_ENDPOINT> 9090:32357/TCP,8080:31323/TCP   20h
      tap-observability-prometheus-node-exporter  ClusterIP      10.0.34.213
      <none>            9100/TCP                       20h
      ```

      Note

      This example is given only for HTTP endpoints exposed publicly. Your own security policy might require private endpoints, a TLS mechanism, or other restrictions.

Create a Grafana dashboard

To create a Grafana dashboard:

1. Create `grafana-values.yaml` with the following content to establish the Prometheus endpoints with the port (9090 by default for Prometheus):

```
datasources:
    datasources.yaml:
      apiVersion: 1
      datasources:
      - name: prometheus-view-cluster
        type: prometheus
        url: VIEW-CLUSTER-PROMETHEUS-ENDPOINT:9090
        access: proxy
        isDefault: true
      - name: prometheus-build-cluster
        type: prometheus
        url: BUILD-CLUSTER-PROMETHEUS-ENDPOINT:9090
        access: proxy
        isDefault: false
      - name: prometheus-run-cluster
        type: prometheus
        url: RUN-CLUSTER-PROMETHEUS-ENDPOINT:9090
        access: proxy
        isDefault: false
```

2. Install Grafana in a cluster dedicated to the Grafana dashboard by running:

```
kubectl create ns grafana

# Add Grafana Helm repository
helm repo add grafana https://grafana.github.io/helm-charts
helm repo update

# Install the chart by setting your Grafana password
helm install grafana grafana/grafana \
  --namespace grafana \
  --set persistence.storageClassName=DEFAULT-PVC-STORAGE-CLASS \
  --set persistence.enabled=true \
  --set adminPassword='GRAFANA-ENDPOINT-PASSWORD' \
  --values  grafana-values.yaml \
  --set service.type=LoadBalancer
```

   Where:

   - `DEFAULT-PVC-STORAGE-CLASS` is the default PVC storage class. For some examples there is `default` for Azure, `gp2` for Amazon Web Services, and so on.
   - `GRAFANA-ENDPOINT-PASSWORD` is the password for logging in to the Grafana endpoint.

   If any auto-generate mechanism is used, you can get your Grafana user ID and password by running:

```
kubectl get secret --namespace grafana grafana -o jsonpath="{.data.admin-passwo
rd}" | base64 \
--decode ; echo
```

3. Access the Grafana UI by using the `<PUBLIC_ENDPOINT>` created by the default LoadBalancer. This can be configured behind any ingress service.

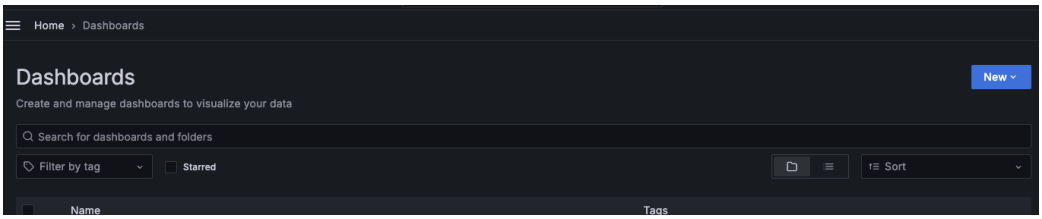
```
kubectl get svc -n grafana
NAME      TYPE           CLUSTER-IP     EXTERNAL-IP        PORT(S)        AGE
grafana   LoadBalancer   10.0.133.110   <PUBLIC_ENDPOINT>  80:31147/TCP   20h
```

4. Log in to the Grafana UI through the endpoint and configured user name and password. The default user name is `admin`.

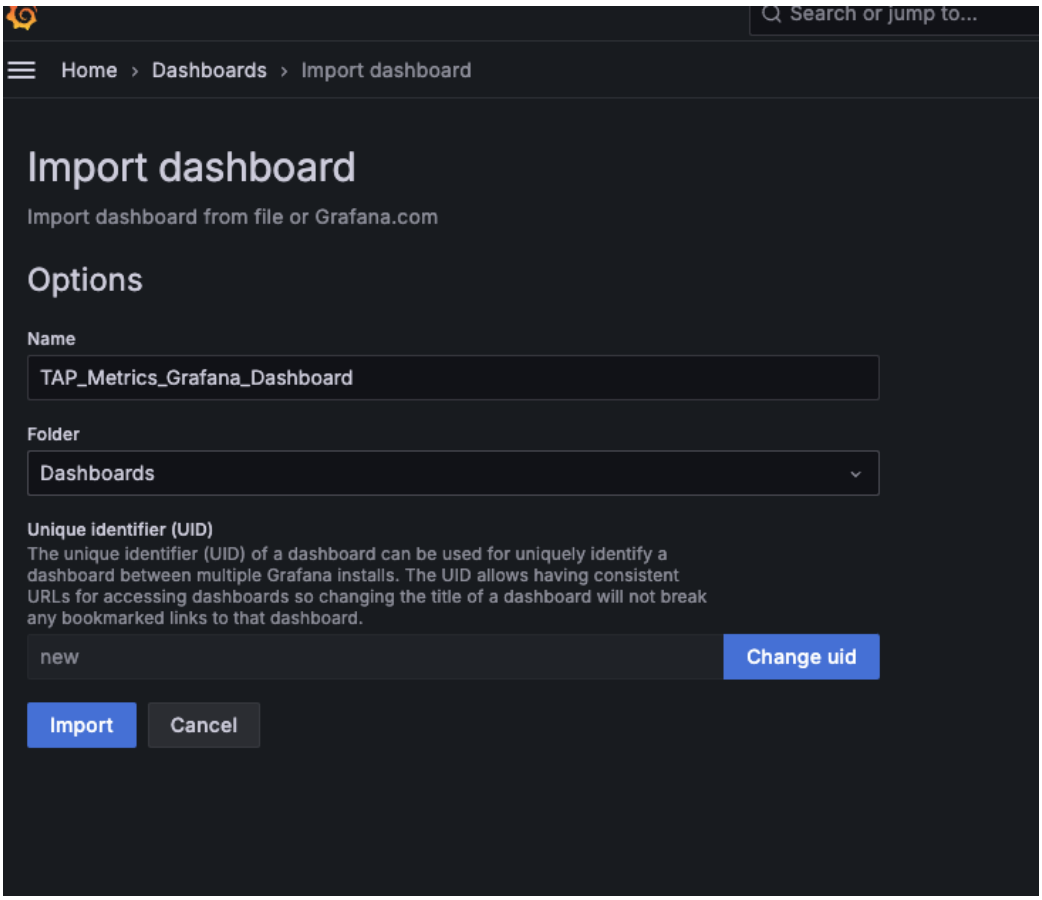
Import the Grafana dashboard

To import `TAP_Metrics_Grafana_Dashboard.json` from the Tanzu Application Platform artifact you downloaded earlier:

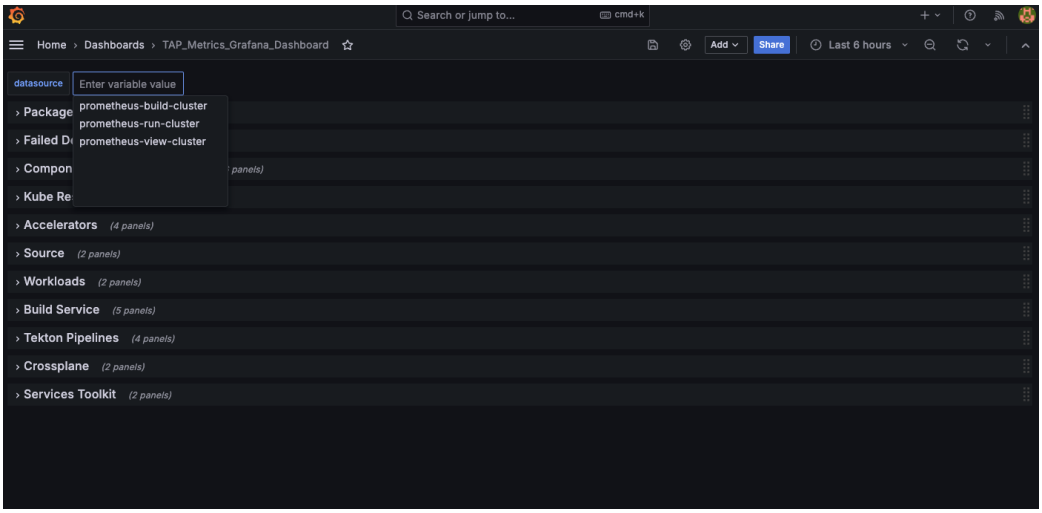
1. Go to the **Dashboards** page.

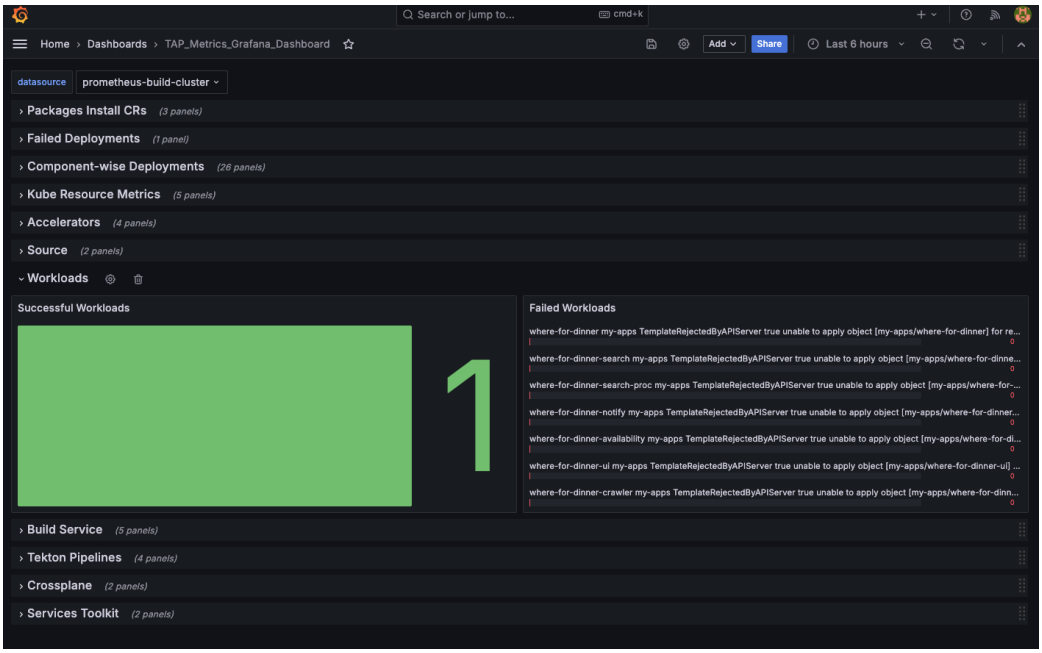


2. Click **New** > **Import**.
3. Select and import the dashboard downloaded earlier named `TAP_Metrics_Grafana_Dashboard.json`.



4. View the Tanzu Application Platform health dashboard for different Tanzu Application Platform clusters by selecting the appropriate datasource.

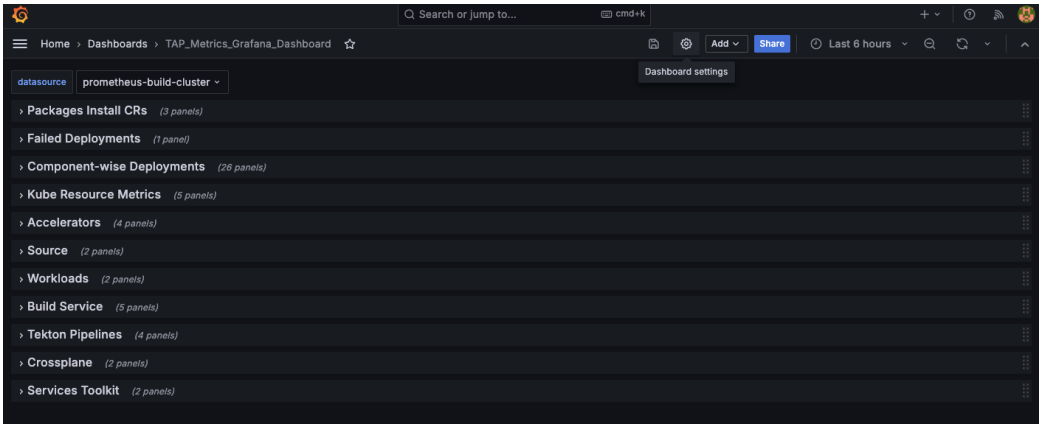




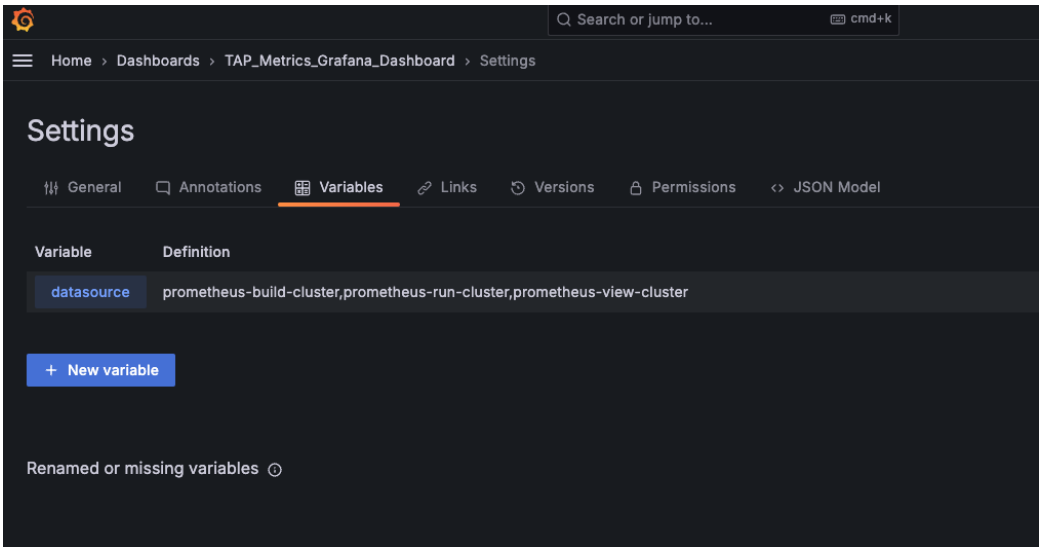
(Optional) Use your own datasources

To use your own set of datasources:

1. Go to **Dashboard Settings**.



2. Go to the **Variables** tab, click the datasource variable, and then update the proper datasource names in the datasource variable.


Search or jump to...
cmd+k
Home › Dashboards › TAP_Metrics_Grafana_Dashboard › Settings
Settings
General
Annotations
Variables
Links
Versions
Permissions
JSON Model
Variable
Definition
datasource
prometheus-build-cluster,prometheus-run-cluster,prometheus-view-cluster
+ New variable
Renamed or missing variables

Overview of workloads

This topic provides you with an overview of workload types in Tanzu Application Platform (commonly known as TAP).

Workload features

Tanzu Application Platform allows you to quickly build and test applications regardless of your familiarity with Kubernetes.

You can turn source code into a workload that runs in a container with a URL. You can also use supply chains to build applications that process work from a message queue, or provide arbitrary network services.

A workload allows you to choose application specifications, such as repository location, environment variables, service binding, and so on. For more information about workload creation and management, see Create or update a workload.

The Out of the Box supply chains support a range of workload types, including

- Scalable web applications (`web`)
- Traditional application servers (`server`)
- Background applications (`worker`)
- Serverless functions

You can use a collection of workloads of different types to deploy microservices that function as a logical application. Alternatively, you can deploy your entire application as a single monolith.

If you build your own supply chains, you can define additional deployment methods beyond those in the Out of the Box supply-chain templates.

Available workload types

When using the Out of the Box supply chain, the `apps.tanzu.vmware.com/workload-type` annotation selects which style of deployment is suitable for your application. The valid values are:

Type	Description	Indicators
`web`	Scalable web applications	• Scales based on request load • Automatically exposed by HTTP Ingress • Does not perform background work • Works with Service Bindings • Stateless • Quick startup time

Type	Description	Indicators
`server`	Traditional applications	• Provides HTTP or TCP services on the network • Exposed by external Ingress or LoadBalancer settings • Might perform background work from a queue • Works with Service Bindings • Fixed scaling, no disk persistence • Startup time not an issue
`worker`	Background applications	• Does not provide network services • Not exposed externally as a network service • Performs background work from a queue • Works with Service Bindings • Fixed scaling, no disk persistence • Startup time not an issue

Overview of workloads

This topic provides you with an overview of workload types in Tanzu Application Platform (commonly known as TAP).

Workload features

Tanzu Application Platform allows you to quickly build and test applications regardless of your familiarity with Kubernetes.

You can turn source code into a workload that runs in a container with a URL. You can also use supply chains to build applications that process work from a message queue, or provide arbitrary network services.

A workload allows you to choose application specifications, such as repository location, environment variables, service binding, and so on. For more information about workload creation and management, see Create or update a workload.

The Out of the Box supply chains support a range of workload types, including

- Scalable web applications (`web`)
- Traditional application servers (`server`)
- Background applications (`worker`)
- Serverless functions

You can use a collection of workloads of different types to deploy microservices that function as a logical application. Alternatively, you can deploy your entire application as a single monolith.

If you build your own supply chains, you can define additional deployment methods beyond those in the Out of the Box supply-chain templates.

Available workload types

When using the Out of the Box supply chain, the `apps.tanzu.vmware.com/workload-type` annotation selects which style of deployment is suitable for your application. The valid values are:

Type	Description	Indicators
`web`	Scalable web applications	• Scales based on request load • Automatically exposed by HTTP Ingress • Does not perform background work • Works with Service Bindings • Stateless • Quick startup time
`server`	Traditional applications	• Provides HTTP or TCP services on the network • Exposed by external Ingress or LoadBalancer settings • Might perform background work from a queue • Works with Service Bindings • Fixed scaling, no disk persistence • Startup time not an issue
`worker`	Background applications	• Does not provide network services • Not exposed externally as a network service • Performs background work from a queue • Works with Service Bindings • Fixed scaling, no disk persistence • Startup time not an issue

Use web workloads

This topic tells you how to use the `web` workload type in Tanzu Application Platform (commonly known as TAP).

Overview

The `web` workload type allows you to deploy web applications on Tanzu Application Platform. Using an application workload specification, you can turn source code into a scalable, stateless application that runs in a container with an automatically-assigned URL. This type of application is often called serverless, and is deployed using Knative.

The `web` workload type is suitable for modern stateless web applications that follow the twelve-factor app methodology and have the following characteristics:

- Perform all work through HTTP requests, including gRPC and WebSocket
- Do not perform work except when processing a request
- Start up quickly
- Store state in external databases instead of storing state locally

Applications using the `web` workload type have the following features:

- Automatic request-based scaling, including scale-to-zero
- Automatic URL provisioning and optional certificate provisioning
- Automatic health-check definitions, if not provided by a convention
- Blue-green application rollouts

When creating a workload with the `tanzu apps workload create` command, you can use the `--type=web` argument to select the `web` workload type. For more information, see Use the web Workload Type later in this topic.

You can also use the `apps.tanzu.vmware.com/workload-type:web` label in the YAML workload description to support this deployment type.

Use the `web` workload type

The `tanzu-java-web-app` workload mentioned in Deploy an app on Tanzu Application Platform is a good match for the `web` workload type. It is a good match because it serves HTTP requests and does not perform any background processing.

You can experiment with the differences between the `web` and `server` workload types by changing the workload type. To change the workload type run:

```
tanzu apps workload apply tanzu-java-web-app --type=server
```

After changing the workload type to `server`, the application does not auto-scale or expose an external URL. For more information about the server workload type, see Use Server workloads.

Switch back to the `web` workload by running:

```
tanzu apps workload apply tanzu-java-web-app --type=web
```

Use this to test which applications can function well as serverless web applications, and which are more suited to the `server` application style.

Use server workloads

This topic tells you how to use the `server` workload type in Tanzu Application Platform (commonly known as TAP).

Overview

The `server` workload type allows you to deploy traditional network applications on Tanzu Application Platform.

Using an application workload specification, you can build and deploy application source code to a manually-scaled Kubernetes deployment which exposes an in-cluster Service endpoint. If required, you can use environment-specific LoadBalancer Services or Ingress resources to expose these applications outside the cluster.

The `server` workload is suitable for traditional applications, including HTTP applications, which have the following characteristics:

- Store state locally
- Run background tasks outside of requests
- Provide multiple network ports or non-HTTP protocols
- Are not a good match for the `web` workload type

An application using the `server` workload type has the following features:

- Does not natively autoscale, but you can use these applications with the Kubernetes Horizontal Pod Autoscaler.
- By default, is exposed only within the cluster using a `ClusterIP` service.

- Uses health checks if defined by a convention.
- Uses a rolling update pattern by default.

When creating a workload with the `tanzu apps workload create` command, you can use the `--type=server` argument to select the `server` workload type. For more information, see Use the server Workload Type later in this topic. You can also use the `apps.tanzu.vmware.com/workload-type:server` annotation in the YAML workload description to support this deployment type.

Use the `server` workload type

The `spring-sensors-consumer-web` workload in Bind an application workload to the service instance in the Get started guide is a good match for the `server` workload type.

This is because it runs continuously to extract information from a RabbitMQ queue, and stores the resulting data locally in memory and presents it through a web UI.

In the Services Toolkit example in Bind an application workload to the service instance, you can update the `spring-sensors-consumer-web` workload to use the `server` supply chain by changing the workload:

```
tanzu apps workload apply spring-sensors-consumer-web --type=server
```

This shows the change in the workload label and prompts you to accept the change. After the workload finishes the new deployment, there are a few differences:

- The workload no longer exposes a URL. It's available within the cluster as `spring-sensors-consumer-web` within the namespace, but you must use `kubectl port-forward service/spring-sensors-consumer-web 8080` to access the web service on port 8080.

  You can set up a Kubernetes Ingress rule to direct traffic from outside the cluster to the workload. Use an Ingress rule to specify that specific host names or paths must be routed to the application. For more information about Ingress rules, see the Kubernetes documentation
- The workload no longer autoscales based on request traffic. For the `spring-sensors-consumer-web` workload, this means that it never spawns a second instance that consumes part of the request queue. Also, it does not scale down to zero instances.

`server`-specific workload parameters

In addition to the common supply chain parameters, `server` workloads can expose one or more network ports from the application to the Kubernetes cluster by using the `ports` parameter. This parameter is a list of port objects, similar to a Kubernetes service specification.

If you do not configure the `ports` parameter, the applied container conventions in the cluster establishes the set of exposed ports.

The following configuration exposes two ports on the Kubernetes cluster under the `my-app` host name:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: my-app
  labels:
    apps.tanzu.vmware.com/workload-type: server
spec:
  params:
  - name: ports
```

```
    value:
    - containerPort: 2025
      name: smtp
      port: 25
    - port: 8080
  ...
```

This snippet configures:

- One service on port 25, which is redirected to port 2025 on the application
- One service on port 8080, which is routed to port 8080 on the application

You can set the `ports` parameter from the `tanzu apps workload create` command as `--param-yaml 'ports=[{"port": 8080}]'`.

The following values are valid within the `ports` argument:

Field	Value
`port`	The port on which the application is exposed to the rest of the cluster
`containerPort`	The port on which the application listens for requests. Defaults to `port` if not set.
`name`	A human-readable name for the port. Defaults to `port` if not set.

Expose `server` workloads outside the cluster

You have several options for exposing `server` workloads outside the cluster:

- Expose server workloads outside the cluster automatically
- Expose HTTP server workloads outside the cluster manually
- Define a workload type that exposes server workloads outside the cluster
- Expose workloads outside the cluster using AVI L4/L7

Use server workloads

This topic tells you how to use the `server` workload type in Tanzu Application Platform (commonly known as TAP).

Overview

The `server` workload type allows you to deploy traditional network applications on Tanzu Application Platform.

Using an application workload specification, you can build and deploy application source code to a manually-scaled Kubernetes deployment which exposes an in-cluster Service endpoint. If required, you can use environment-specific LoadBalancer Services or Ingress resources to expose these applications outside the cluster.

The `server` workload is suitable for traditional applications, including HTTP applications, which have the following characteristics:

- Store state locally
- Run background tasks outside of requests
- Provide multiple network ports or non-HTTP protocols
- Are not a good match for the `web` workload type

An application using the `server` workload type has the following features:

- Does not natively autoscale, but you can use these applications with the Kubernetes Horizontal Pod Autoscaler.
- By default, is exposed only within the cluster using a `ClusterIP` service.
- Uses health checks if defined by a convention.
- Uses a rolling update pattern by default.

When creating a workload with the `tanzu apps workload create` command, you can use the `--type=server` argument to select the `server` workload type. For more information, see Use the server Workload Type later in this topic. You can also use the `apps.tanzu.vmware.com/workload-type:server` annotation in the YAML workload description to support this deployment type.

Use the `server` workload type

The `spring-sensors-consumer-web` workload in Bind an application workload to the service instance in the Get started guide is a good match for the `server` workload type.

This is because it runs continuously to extract information from a RabbitMQ queue, and stores the resulting data locally in memory and presents it through a web UI.

In the Services Toolkit example in Bind an application workload to the service instance, you can update the `spring-sensors-consumer-web` workload to use the `server` supply chain by changing the workload:

```
tanzu apps workload apply spring-sensors-consumer-web --type=server
```

This shows the change in the workload label and prompts you to accept the change. After the workload finishes the new deployment, there are a few differences:

- The workload no longer exposes a URL. It's available within the cluster as `spring-sensors-consumer-web` within the namespace, but you must use `kubectl port-forward service/spring-sensors-consumer-web 8080` to access the web service on port 8080.

  You can set up a Kubernetes Ingress rule to direct traffic from outside the cluster to the workload. Use an Ingress rule to specify that specific host names or paths must be routed to the application. For more information about Ingress rules, see the Kubernetes documentation
- The workload no longer autoscales based on request traffic. For the `spring-sensors-consumer-web` workload, this means that it never spawns a second instance that consumes part of the request queue. Also, it does not scale down to zero instances.

`server`-specific workload parameters

In addition to the common supply chain parameters, `server` workloads can expose one or more network ports from the application to the Kubernetes cluster by using the `ports` parameter. This parameter is a list of port objects, similar to a Kubernetes service specification.

If you do not configure the `ports` parameter, the applied container conventions in the cluster establishes the set of exposed ports.

The following configuration exposes two ports on the Kubernetes cluster under the `my-app` host name:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: my-app
  labels:
```

```
    apps.tanzu.vmware.com/workload-type: server
spec:
  params:
  - name: ports
    value:
    - containerPort: 2025
      name: smtp
      port: 25
    - port: 8080
  ...
```

This snippet configures:

- One service on port 25, which is redirected to port 2025 on the application
- One service on port 8080, which is routed to port 8080 on the application

You can set the `ports` parameter from the `tanzu apps workload create` command as `--param-yaml 'ports=[{"port": 8080}]'`.

The following values are valid within the `ports` argument:

Field	Value
`port`	The port on which the application is exposed to the rest of the cluster
`containerPort`	The port on which the application listens for requests. Defaults to `port` if not set.
`name`	A human-readable name for the port. Defaults to `port` if not set.

Expose `server` workloads outside the cluster

You have several options for exposing `server` workloads outside the cluster:

- Expose server workloads outside the cluster automatically
- Expose HTTP server workloads outside the cluster manually
- Define a workload type that exposes server workloads outside the cluster
- Expose workloads outside the cluster using AVI L4/L7

Expose server workloads outside the cluster automatically

In Tanzu Application Platform (commonly known as TAP) v1.6 and later, Carvel Package Supply Chains produce server workloads that have Ingress by default. For more information, see Overview of the Carvel Package Supply Chains.

Expose HTTP server workloads outside the cluster manually

You can expose HTTP `server` workloads outside the cluster by creating an Ingress resource and using cert-manager to provision TLS-signed certificates. To do so:

1. Using the `spring-sensors-consumer-web` workload from Bind an application workload to the service instance as an example, create the following `Ingress`:

   ```
   apiVersion: networking.k8s.io/v1
   kind: Ingress
   metadata:
    name: spring-sensors-consumer-web
    namespace: DEVELOPER-NAMESPACE
    annotations:
   ```

```
    cert-manager.io/cluster-issuer: tap-ingress-selfsigned
    ingress.kubernetes.io/force-ssl-redirect: "true"
    kubernetes.io/ingress.class: contour
    kubernetes.io/tls-acme: "true"
spec:
 tls:
   - secretName: spring-sensors-consumer-web
     hosts:
       - "spring-sensors-consumer-web.INGRESS-DOMAIN"
 rules:
   - host: "spring-sensors-consumer-web.INGRESS-DOMAIN"
     http:
       paths:
         - pathType: Prefix
           path: /
           backend:
             service:
               name: spring-sensors-consumer-web
               port:
                 number: 8080
```

- Replace `DEVELOPER-NAMESPACE` with your developer namespace.
- Replace `INGRESS-DOMAIN` with the domain name defined in `tap-values.yaml` during the installation.
- Set the annotation `cert-manager.io/cluster-issuer` to the `shared.ingress_issuer` value configured during installation or leave it as `tap-ingress-selfsigned` to use the default value.
- Update the port exposed by your `Service` resource, which is set as `8080` in the example.

2. Access the `server` workload with HTTPS:

```
curl -k https://spring-sensors-consumer-web.INGRESS-DOMAIN
```

Define a workload type that exposes server workloads outside the cluster

Tanzu Application Platform (commonly known as TAP) allows you to create new workload types. You start by adding an `Ingress` resource to the `server-template ClusterConfigTemplate` when this new type of workload is created.

1. Delete the `Ingress` resource previously created.
2. Install the `yq` CLI on your local machine.
3. Save the existing `server-template` in a local file by running:

```
kubectl get ClusterConfigTemplate server-template -o yaml > secure-server-templ
ate.yaml
```

4. Extract the `.spec.ytt` field from this file and create another file by running:

```
yq eval '.spec.ytt' secure-server-template.yaml > spec-ytt.yaml
```

5. In the next step, you add the `Ingress` resource snippet to `spec-ytt.yaml`. This step provides a sample `Ingress` resource snippet. Make the following edits before adding the `Ingress` resource snippet to `spec-ytt.yaml`:
   - Replace `INGRESS-DOMAIN` with the Ingress domain you set during the installation.

- Set the annotation `cert-manager.io/cluster-issuer` to the `shared.ingress_issuer` value configured during installation or leave it as `tap-ingress-selfsigned` to use the default one.
- This configuration is based on your workload service running on port `8080`.

The `Ingress` resource snippet looks like this:

```
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: #@ data.values.workload.metadata.name
  annotations:
    cert-manager.io/cluster-issuer: tap-ingress-selfsigned
    ingress.kubernetes.io/force-ssl-redirect: "true"
    kubernetes.io/ingress.class: contour
    kubernetes.io/tls-acme: "true"
    kapp.k14s.io/change-rule: "upsert after upserting Services"
  labels: #@ merge_labels({ "app.kubernetes.io/component": "run", "carto.run/wo
rkload-name": data.values.workload.metadata.name })
spec:
  tls:
    - secretName: #@ data.values.workload.metadata.name
      hosts:
        - #@ data.values.workload.metadata.name + ".INGRESS-DOMAIN"
  rules:
    - host: #@ data.values.workload.metadata.name + ".INGRESS-DOMAIN"
      http:
        paths:
          - pathType: Prefix
            path: /
            backend:
              service:
                name: #@ data.values.workload.metadata.name
                port:
                  number: 8080
```

6. Add the `Ingress` resource snippet to the `spec-ytt.yaml` file and save. Look for the `Service` resource, and insert the snippet before the last `#@ end`. For example:

```

# THE TOP OF THE FILE IS NOT SHOWN

---
apiVersion: v1
kind: Service
metadata:
  name: #@ data.values.workload.metadata.name
  labels: #@ merge_labels({ "app.kubernetes.io/component": "run", "carto.run/wo
rkload-name": data.values.workload.metadata.name })
spec:
  selector: #@ data.values.config.metadata.labels
  ports:
  #@ hasattr(data.values.params, "ports") and len(data.values.params.ports) or
assert.fail("one or more ports param must be provided.")
  #@ declared_ports = {}
  #@ if "ports" in data.values.params:
  #@   declared_ports = data.values.params.ports
  #@ else:
  #@   declared_ports = struct.encode([{ "containerPort": 8080, "port": 8080,
"name":   "http"}])
  #@ end
  #@ for p in merge_ports(declared_ports, data.values.config.spec.containers):
  - #@ p
```

```
  #@ end

# NEW INGRESS RESOURCE
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: #@ data.values.workload.metadata.name
  annotations:
    cert-manager.io/cluster-issuer: tap-ingress-selfsigned
    ingress.kubernetes.io/force-ssl-redirect: "true"
    kubernetes.io/ingress.class: contour
    kubernetes.io/tls-acme: "true"
    kapp.k14s.io/change-rule: "upsert after upserting Services"
  labels: #@ merge_labels({ "app.kubernetes.io/component": "run", "carto.run/wo
rkload-name": data.values.workload.metadata.name })
spec:
  tls:
    - secretName: #@ data.values.workload.metadata.name
      hosts:
        - #@ data.values.workload.metadata.name + ".INGRESS-DOMAIN"
  rules:
    - host: #@ data.values.workload.metadata.name + ".INGRESS-DOMAIN"
      http:
        paths:
          - pathType: Prefix
            path: /
            backend:
              service:
                name: #@ data.values.workload.metadata.name
                port:
                  number: 8080
# END NEW INGRESS RESOURCE

#@ end

---
apiVersion: v1
kind: ConfigMap
metadata:
  name: #@ data.values.workload.metadata.name + "-server"
  labels: #@ merge_labels({ "app.kubernetes.io/component": "config" })
data:
  delivery.yml: #@ yaml.encode(delivery())
```

7. Add the snippet to the `.spec.ytt` property in `secure-server-template.yaml`:

```
SPEC_YTT=$(cat spec-ytt.yaml) yq eval -i '.spec.ytt |= strenv(SPEC_YTT)' secure
-server-template.yaml
```

8. Change the name of the `ClusterConfigTemplate` to `secure-server-template` by running:

```
yq eval -i '.metadata.name = "secure-server-template"' secure-server-template.y
aml
```

9. Create the new `ClusterConfigTemplate` by running:

```
kubectl apply -f secure-server-template.yaml
```

10. Verify the new `ClusterConfigTemplate` is in the cluster by running:

```
kubectl get ClusterConfigTemplate
```

Expected output:

```
kubectl get ClusterConfigTemplate
NAME                      AGE
api-descriptors           82m
config-template           82m
convention-template       82m
secure-server-template    22s
server-template           82m
service-bindings          82m
worker-template           82m
```

11. Add the new workload type to the `tap-values.yaml`. The new workload type is named `secure-server` and the `cluster_config_template_name` is `secure-server-template`.

```
ootb_supply_chain_basic:
  supported_workloads:
    - type: web
      cluster_config_template_name: config-template
    - type: server
      cluster_config_template_name: server-template
    - type: worker
      cluster_config_template_name: worker-template
    - type: secure-server
      cluster_config_template_name: secure-server-template
```

12. Update your Tanzu Application Platform installation as follows:

```
tanzu package installed update tap -p tap.tanzu.vmware.com --values-file \
"/path/to/your/config/tap-values.yaml"  -n tap-install
```

13. Give privileges to the `deliverable` role to manage `Ingress` resources:

```
cat <<EOF | kubectl apply -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
 name: deliverable-with-ingress
 labels:
   apps.tanzu.vmware.com/aggregate-to-deliverable: "true"
rules:
- apiGroups:
 - networking.k8s.io
 resources:
 - ingresses
 verbs:
 - get
 - list
 - watch
 - create
 - patch
 - update
 - delete
 - deletecollection
EOF
```

14. Update the workload type to `secure-server`:

Note

If you created the `Ingress` resource manually in the previous section, delete it before this.

```
tanzu apps workload apply spring-sensors-consumer-web --type=secure-server
```

15. After the process finishes, verify that the resources Deployment, Service, and Ingress appear by running:

    ```
    kubectl get ingress,svc,deploy -l carto.run/workload-name=spring-sensors-consum
    er-web
    ```

    Expected output:

    ```
    kubectl get ingress,svc,deploy -l carto.run/workload-name=tanzu-java-web-app-js
    NAME                                                    CLASS    HOSTS
    ADDRESS          PORTS     AGE
    ingress.networking.k8s.io/spring-sensors-consumer-web   <none>   spring-sensors
    -consumer-web.INGRESS-DOMAIN   34.111.111.111   80, 443   37s

    NAME                                  TYPE        CLUSTER-IP     EXTERNAL-IP
    PORT(S)    AGE
    service/spring-sensors-consumer-web   ClusterIP   10.32.15.194   <none>
    8080/TCP   36m

    NAME                                          READY   UP-TO-DATE   AVAILABLE
    AGE
    deployment.apps/spring-sensors-consumer-web   1/1     1            1
    37s
    ```

16. Access your `secure-server` workload with HTTPS by running:

    ```
    curl -k https://spring-sensors-consumer-web.INGRESS-DOMAIN
    ```

Expose workloads outside the cluster using AVI L4/L7

To expose workloads outside the cluster by using AVI L4/L7, see the Tanzu Reference Architecture documentation.

Use worker workloads

This topic tells you how to create and install a supply chain for the `worker` workload type in Tanzu Application Platform (commonly known as TAP).

Overview

The `worker` workload type allows you to deploy applications that run continuously without network input on Tanzu Application Platform. Using an application workload specification, you can build and deploy application source code to a manually scaled Kubernetes deployment with no network exposure.

The `worker` workload is a good match for applications that manage their own work by reading from a worker or a background scheduled time source, and don't expose any network interfaces.

An application using the `worker` workload type has the following features:

- Does not natively auto-scale but you can use it with the Kubernetes Horizontal Pod Autoscaler
- Does not expose any network services
- Uses health checks if defined by a convention

- Uses a rolling update pattern by default

When creating a workload with `tanzu apps workload create`, you can use the `--type=worker` argument to select the `worker` workload type. For more information, see the Use the worker Workload Type section. You can also use the `apps.tanzu.vmware.com/workload-type:worker` annotation in the YAML workload description to support this deployment type.

Use the `worker` workload type

The `spring-sensors-producer` workload in the example in Consume services on Tanzu Application Platform is a good match for the `worker` workload type. This is because it runs continuously without a UI to report sensor information to a RabbitMQ topic.

If you followed the Services Toolkit example, you can update the `spring-sensors-producer` to use the `worker` supply chain by changing the workload type. To do so, run:

```
tanzu apps workload apply spring-sensors-producer --type=worker
```

This shows a difference in the workload label, and prompts you to accept the change. After the workload finishes the new deployment, there are a few differences:

- The workload no longer has a URL. Because the workload does not present a web UI, this more closely matches the original application intent.
- The workload no longer auto-scales based on request traffic. For the `spring-sensors-producer` workload, this means that it does not scale down to zero instances when there is no request traffic.

Service-to-Service communication

This topic tells you about service-to-service communication for `web` and `server` workloads.

Calling `web` workloads within a cluster

When a `web` workload type is created, a Knative service is deployed to the cluster. To access your application, you need the URL for the route created by the Knative Service.

Obtain the URL by running one of these commands:

- To use the `tanzu apps` command, run:

  ```
  tanzu apps workload get WORKLOAD-NAME --namespace DEVELOPER-NAMESPACE
  ```

- To use the `kubectl get` command, run:

  ```
  kubectl get ksvc WORKLOAD-NAME -n YOUR-DEVELOPER-NAMESPACE -ojsonpath="{status.
  address.url}"
  ```

Note

When calling a Knative service, both the Service name and namespace are required. This behavior is distinct from `server` type workloads, which do not rely on the namespace name to establish service-to-service communication between applications within the same namespace.

Example of service-to-service communication for `web` and `server` workloads

You have three applications deployed to the namespace called `dev-namespace`:

- A `server` type workload named `server-workload`
- A `web` type workload named `web-workload`
- A pod running the busybox image with `curl` that is named `busybox`

Open a shell to the running container of the `busybox` pod and send requests to the `server` and `web` workloads using curl. Specify the namespace for both, as follows:

```
kubectl exec busybox -n dev-namespace -- curl server-workload.dev-namespace.svc.cluste
r.local -v
kubectl exec busybox -n dev-namespace -- curl web-workload.dev-namespace.svc.cluster.l
ocal -v
```

You can alternatively reference a web service as `workload-name.namespace` or `web-workload.dev-namespace` in this example. You can also reference a `server` workload that exists in the same namespace as `workload-name`, or `server-workload`.

Parameter reference

This topic tells you about the default supply chains and templates provided by Tanzu Application Platform (commonly known as TAP). It describes the `workload.spec.params` parameters that are configured in workload objects, and the `deliverable.spec.params` parameters that are configured in the deliverable object.

Workload Parameter Reference

The supply chains and templates provided by the Out of the Box packages contain a series of parameters that customize supply chain behavior. This section describes the `workload.spec.params` parameters that can be configured in workload objects.

The following table provides a list of supply chain resources organized by the resource in the supply chain where they are used. Some of these resources might not be applicable depending on the supply chain in use.

List of Supply Chain Resources for Workload Object

Supply Chain Resource	Output Type	Purpose	Basic	Testing	Scanning
source-provider	Source	Fetches source code	Yes	Yes	Yes
source-tester	Source	Tests source code	No	Yes	Yes
source-scanner	Source	Scans source code	No	No	Yes
image-provider	Image	Builds application container image	Yes	Yes	Yes
image-scanner	Image	Scans application container image	No	No	Yes
config-provider	Podtemplate spec	Tailors a pod spec based on the application image and conventions set up in the cluster	Yes	Yes	Yes
app-config	Kubernetes configuration	Creates Kubernetes config files (knative service/deployment - depending on workload type)	Yes	Yes	Yes

Supply Chain Resource	Output Type	Purpose	Basic	Testing	Scanning
service-bindings	Kubernetes configuration	Adds service bindings to the set of config files	Yes	Yes	Yes
api-descriptors	Kubernetes configuration	Adds api descriptors to the set of config files	Yes	Yes	Yes
config-writer	Kubernetes configuration	Writes configuration to a destination (git or registry) for further deployment to a run cluster	Yes	Yes	Yes
deliverable	Kubernetes configuration	Writes deliverable content to be extracted for use in a run cluster	Yes	Yes	Yes

For information about supply chains, see:

- Out of the Box Supply Chain Basic
- Out of the Box Supply Chain Testing
- Out of the Box Supply Chain Testing Scanning

source-provider

The `source-provider` resource in the supply chain creates objects that fetch either source code or pre-compiled Java applications depending on how the workload is configured. For more information, see Building from Source.

GitRepository

Use `gitrepository` when fetching source code from Git repositories. This resource makes further resources available in the supply chain, such as the contents of the Git repository as a tarball available in the cluster.

Parameters:

Parameter name	Meaning	Example
`gitImplementation`	VMware recommends that you use the underlying library for fetching the source code.	- name: gitImplementation value: go-git
`source_credentials_secret`	The name of the secret in the same namespace as the `Workload` used for providing credentials for fetching source code from the Git repository. For more information, see Git authentication.	- name: source_credentials_secret value: git-credentials
`gitops_ssh_secret`	Deprecated. Use `source_credentials_secret` instead. The name of the secret in the same namespace as the `Workload` used for providing credentials for fetching source code from the Git repository. For more information, see Git authentication.	- name: gitops_ssh_secret value: git-credentials

For information about the features supported by each implementation, see Git implementation in the Flux documentation.

For information about how to create a workload that uses a GitHub repository as the provider of source code, see Create a workload from GitHub repository.

For more information about GitRepository objects, see Git Repository in the Flux documentation.

ImageRepository

Use the ImageRepository when fetching source code from container images. It makes the contents of the container image available as a tarball to further resources in the supply chain. The contents of the container image are fetched by using Git or Maven. For more information, see Create a workload from local source code.

Parameters:

Parameter Name	Meaning	Example
`serviceAccount`	Name of the service account (in the same namespace as the workload) to use to provide the credentials to `ImageRepository` for fetching the container images.	- name: serviceAccount value: default

The `--service-account` flag sets the `spec.serviceAccountName` key in the workload object. To configure the `serviceAccount` parameter, use `--param serviceAccount=SERVICE-ACCOUNT`.

For information about custom resource details, see the ImageRepository reference topic.

For information about how to use ImageRepository with the Tanzu CLI, see Create a workload.

MavenArtifact

When carrying pre-built Java artifacts, `MavenArtifact` makes the artifact available to further resources in the supply chain as a tarball. You can wrap the tarball as a container image for further deployment. Differently from `git` and `image`, its configuration is solely driven by parameters in the workload.

Parameters:

Parameter Name	Meaning	Example
`maven`	Points to the maven artifact to fetch and the polling interval.	- name: maven value: artifactId: springboot-initial groupId: com.example version: RELEASE classifier: sources # optional type: # optional artifactRetryTimeout: 1m0s # optional

For information about the custom resource, see the MavenArtifact reference documentation.

For information about how to use the custom resource with the Tanzu apps CLI plug-in, see the Tanzu CLI Command Reference documentation.

source-tester

The `source-tester` resource is in `ootb-supply-chain-testing` and `ootb-supply-chain-testing-scanning`. This resource is responsible for instantiating a Tekton PipelineRun object that calls the execution of a Tekton Pipeline, in the same namespace as the workload, whenever its inputs change. For example, the source code revision that you want to test changes.

A Runnable object is instantiated to ensure that there's always a run for a particular set of inputs. The parameters are passed from the workload down to Runnable's Pipeline selection mechanism through `testing_pipeline_matching_labels` and the execution of the PipelineRuns through `testing_pipeline_params`.

Parameters:

Parameter name	Meaning	Example
`testing_pipeline_matching_labels`	The set of labels to use when searching for Tekton Pipeline objects in the same namespace as the workload. By default, a Pipeline labeled as `apps.tanzu.vmware.com/pipeline: test` is selected, but when using this parameter, it's possible to override the behavior.	- name: testing_pipeline_matching_labels value: apps.tanzu.com/pipeline: test my.company/language: golang
`testing_pipeline_params`	The set of extra parameters, aside from `source-url` and `source-revision`, to pass to the Tekton Pipeline. The Tekton Pipeline **must** declare both the required parameters `source-url` and `source-revision` and the extra ones declared in this table.	- name: testing_pipeline_params value: verbose: true

For information about how to set up the Workload namespace for testing with Tekton, see Out of the Box Supply Chain with Testing.

For information about how to use the parameters to customize this resource to test using a Jenkins cluster, see Out of the Box Supply Chain with Testing on Jenkins.

source-scanner

The `source-scanner` resource is available in `ootb-supply-chain-testing-scanning`. It scans the source code that is tested by pointing a SourceScan object at the same source code as the tests.

You can customize behavior for both CVEs evaluation with parameters.

Parameters:

Parameter name	Meaning	Example
`scanning_source_template`	The name of the ScanTemplate object (in the same namespace as the workload) to use for running the scans against the source code.	- name: scanning_source_template value: private-source-scan-template
`scanning_source_policy`	The name of the ScanPolicy object (in the same namespace as the workload) to use when evaluating the scan results of a source scan.	- name: scanning_source_policy value: allowlist-policy

For more information, see Out of the Box Supply Chain with Testing and Scanning for details about how to set up the workload namespace with the ScanPolicy and ScanTemplate required for this resource, and SourceScan reference for details about the SourceScan custom resource.

For information about how the artifacts found during scanning are catalogued, see Supply Chain Security Tools for Tanzu – Store.

image-provider

The `image-provider` in the supply chains provides a container image carrying the application already built to further resources.

Different semantics apply, depending on how the workload is configured, for example, if using pre-built images or building from source:

- pre-built: an `ImageRepository` object is created aiming at providing a reference to the latest image found matching the name as specified in `workload.spec.image`

- building from source: an image builder object is created (either Kpack's `Image` or a `Runnable` for creating Tekton TaskRuns for building images from Dockerfiles)

Kpack Image

Use the Kpack Image object to build a container image out of source code or pre-built Java artifact.

This makes the container image available to further resources in the supply chain through a content addressable image reference that's carried to the final deployment objects unchanged. For more information, see Tanzu Build Service.

Parameters:

Parameter name	Meaning	Example
`serviceAccount`	The name of the serviceaccount (in the same namespace as the workload) to use for providing credentials to `Image` for pushing the container images it builds to the configured registry.	- name: serviceAccount value: default
`clusterBuilder`	The name of the Kpack cluster builder to use in the Kpack Image object created.	- name: clusterBuilder value: nodejs-cluster-builder
`buildServiceBindings`	The definition of a list of service bindings to use at build time. For example, providing credentials for fetching dependencies from repositories that require credentials.	- name: buildServiceBindings value: - name: settings-xml kind: Secret apiVersion: v1
`additionalTags`	The additionalTags is a list of locations the built OCI image will be written to in addition to the default tag. Additional tags must be in the same registry as the default tag.	- name: additionalTags value: - my-registry.com/my-app - my-other-registry.com/other-app:other-tag
`live-update`	Enables the use of Tilt's live-update function.	- name: live-update value: "true"

The `--service-account` flag sets the `spec.serviceAccountName` key in the workload object. To configure the `serviceAccount` parameter, use `--param serviceAccount=SERVICE-ACCOUNT`.

For information about the integration with Tanzu Build Service, see Tanzu Build Service Integration.

For information about `live-update`, see Developer Conventions and Overview of Tanzu Developer Tools for IntelliJ.

For information about using Kpack builders with `clusterBuilder`, see Builders.

For information about `buildServiceBindings`, see Service Bindings.

Runnable (TaskRuns for Dockerfile-based builds)

To perform Dockerfile-based builds, all the supply chains instantiate a Runnable object that instantiates Tekton TaskRun objects to call the execution of kaniko builds.

Parameters:

Parameter name	Meaning	Example
`dockerfile`	The relative path to the Dockerfile file in the build context.	./Dockerfile
`docker_build_context`	The relative path to the directory where the build context is.	.
`docker_build_extra_args`	List of flags to pass directly to Kaniko, such as providing arguments to a build.	--build-arg=FOO=BAR

For information about how to use Dockerfile-based builds and limitations associated with the function, see Dockerfile-based builds.

Pre-built image (ImageRepository)

For applications that already have their container images built outside the supply chains, such as providing an image reference under `workload.spec.image`, an `ImageRepository` object is created to keep track of any images pushed under that name.

This makes the content-addressable name, such as the image name containing the digest, available for further resources in the supply chain.

Parameters:

Parameter name	Meaning	Example
`serviceAccount`	The name of the serviceaccount (in the same namespace as the workload) to use for providing the credentials to `ImageRepository` for fetching the container images.	- name: serviceAccount value: default

The `--service-account` flag sets the `spec.serviceAccountName` key in the workload object. To configure the `serviceAccount` parameter, use `--param serviceAccount=...`.

For information about the ImageRepository resource, see the ImageRepository reference documentation. For information about the prebuild image function, see Using a prebuilt image.

image-scanner

The `image-scanner` resource is included only in `ootb-supply-chain-testing-scanning`.

This resource scans a container image (either built by using the supply chain or prebuilt), persisting the results in the store, and gating the image from moving forward in case the CVEs found are not compliant with the ScanPolicy referenced by the ImageScan object create for doing so.

Parameters:

Parameter name	Meaning	Example
`scanning_image_template`	The name of the ScanTemplate object (in the same namespace as the workload) to use for running the scans against a container image.	- name: scanning_image_template value: private-image-scan-template
`scanning_image_policy`	The name of the ScanPolicy object (in the same namespace as the workload) to use when evaluating the scan results of an image scan.	- name: scanning_image_policy value: allowlist-policy

For information about the ImageScan custom resource, see ImageScan reference.

For information about how the artifacts found during scanning are catalogued, see Supply Chain Security Tools for Tanzu – Store.

config-provider

The `config-provider` resource in the supply chains generates a PodTemplateSpec to use in application configs, such as Knative services and deployments, to represent the desired pod configuration to instantiate to run the application in containers. For more information, see PodTemplateSpec in the Kubernetes documentation.

The `config-provider` resource manages a PodIntent object that represents the intention of having PodTemplateSpec enhanced with conventions installed in the cluster whose final representation is then passed forward to other resources to form the final deployment configuration.

Parameters:

Parameter name	Meaning	Example
`serviceAccount`	The name of the serviceaccount (in the same namespace as the workload) to use for providing the necessary credentials to `PodIntent` for fetching the container image to inspect the metadata to pass to convention servers and the serviceAccountName set in the podtemplatespec.	- name: serviceAccount value: default
`annotations`	An extra set of annotations to pass down to the PodTemplateSpec.	- name: annotations value: name: my-application version: v1.2.3 team: store
`debug`	Put the workload in debug mode.	- name: debug value: "true"
`live-update`	Enable live-updating of the code (for innerloop development).	- name: live-update value: "true"

The `--service-account` flag sets the `spec.serviceAccountName` key in the workload object. To configure the `serviceAccount` parameter, use `--param serviceAccount=SERVICE-ACCOUNT`.

For more information about the controller behind `PodIntent`, see Cartographer Conventions.

For more details about the two convention servers enabled by default in Tanzu Application Platform installations, see Developer Conventions and Spring Boot conventions.

app-config

The `app-config` resource prepares a ConfigMap with the Kubernetes configuration that is used for instantiating an application in the form of a particular workload type in a cluster.

The resource is configured in the supply chain to allow, by default, three types of workloads with the selection of which workload type to apply based on the labels set in the workload object created by the developer:

- `apps.tanzu.vmware.com/workload-type: web`
- `apps.tanzu.vmware.com/workload-type: worker`
- `apps.tanzu.vmware.com/workload-type: server`

Only the `server` workload type has the following configurable parameters:

Parameter name	Meaning	example
`ports`	The set of network ports to expose from the application to the Kubernetes cluster.	- name: ports value: - containerPort: 2025 name: smtp port: 25

For more information about the three different types of workloads, see Overview of workloads.

For a more detailed overview of the ports parameter, see server-specific Workload parameters.

service-bindings

The `service-bindings` resource adds ServiceBindings to the set of Kubernetes configuration files to promote for deployment.

Parameters:

Parameter name	Meaning	Example
`annotations`	The extra set of annotations to pass down to the ServiceBinding and ResourceClaim objects.	- name: annotations value: name: my-application version: v1.2.3 team: store

For an example, see the Tanzu CLI Command Reference documentation.

For an overview of the function, see Consume services on Tanzu Application Platform.

api-descriptors

The `api-descriptor` resource adds an APIDescriptor to the set of Kubernetes objects to deploy. This enables API auto registration.

Parameters:

Parameter name	Meaning	Example
`annotations`	An extra set of annotations to pass down to the APIDescriptor object.	- name: annotations value: name: my-application version: v1.2.3 team: store
`api_descriptor`	Information used to fill the state that you want of the APIDescriptor object (its spec).	- name: api_descriptor value: type: openapi location: baseURL: http://petclinic-hard-coded.my-apps.tapdemo.vmware.com/ path: "/v3/api" owner: team-petclinic system: pet-clinics description: "example"

The workload must include the `apis.apps.tanzu.vmware.com/register-api: "true"` label to activate this function.

For more details about API auto registration, see Use API Auto Registration.

config-writer (git or registry)

The `config-writer` resource is responsible for performing the last mile of the supply chain: persisting in an external system (registry or Git repository) the Kubernetes configuration generated throughout the supply chain.

There are three methods:

- Publishing the configuration to a container image registry
- Publishing the configuration to a Git repository by using the push of a commit
- Publishing the configuration to a Git repository by pushing a commit and opening a pull request

For more information about the different modes of operation, see Use GitOps or RegistryOps with Supply Chain Choreographer.

deliverable

The `deliverable` resource creates a `deliverable` object that represents the intention of delivering to the cluster the configurations that are produced by the supply chain.

Parameters:

Parameter name	Meaning	Example
`serviceAccount`	The name of the serviceaccount (in the same namespace as the deliverable) to use for providing the necessary permissions to create the children objects for deploying the objects created by the supply chain to the cluster.	- name: serviceAccount value: default

The `--service-account` flag sets the `spec.serviceAccountName` key in the workload object. To configure the `serviceAccount` parameter, use `--param serviceAccount=SERVICE-ACCOUNT`.

On build clusters where a corresponding `ClusterDelivery` doesn't exist, the deliverable takes no effect (similarly to a workload without a SupplyChain, no action is taken).

Deliverable Parameters Reference

The deliverable object applies the configuration produced by the resources defined by a ClusterSupplyChain to a Kubernetes cluster.

This section describes the `deliverable.spec.params` parameters that can be configured in the deliverable object. The following section describes the two resources defined in the ClusterDelivery resources section. These are part of the `ootb-delivery-basic` package:

List of Cluster Delivery Resources for Deliverable Object

Cluster Delivery Resource	Output Type	Purpose
source provider	Source	Fetches the Kubernetes configuration file from Git repository or image registry
app deployer	Source	Applies configuration produced by a supply chain to the cluster

For information about the ClusterDelivery shipped with `ootb-delivery-basic`, and the templates used by it, see:

- Out of the Box Delivery Basic
- Out of the Templates

For information about the use of the deliverable object in a multicluster environment, see Getting started with multicluster Tanzu Application Platform.

For reference information about deliverable, see Deliverable and Delivery custom resources in the Cartographer documentation.

source-provider

The `source-provider` resource in the basic ClusterDelivery creates objects that continuously fetch Kubernetes configuration files from a Git repository or container image registry so that it can apply those to the cluster.

Regardless of where it fetches that Kubernetes configuration from (Git repository or image registry), it exposes those files to further resources along the ClusterDelivery as a tarball.

GitRepository

A GitRepository object is instantiated when `deliverable.spec.source.git` is configured to continuously look for a Kubernetes configuration pushed to a Git repository, making it available for resources in the ClusterDelivery.

Parameters:

Parameter name	Meaning	Example
`gitImplementation`	VMware recommends that you use the underlying library for fetching the source code. `go-git`.	- name: gitImplementation value: go-git
`source_credentials_secret`	The name of the secret in the same namespace as the `Deliverable` used for providing credentials for fetching build configuration from the Git repository. For more information, see Git authentication.	- name: source_credentials_secret value: git-credentials
`gitops_ssh_secret`	Deprecated. Use `source_credentials_secret` instead. The name of the secret in the same namespace as the `Deliverable` used for providing credentials for fetching build configuration from the Git repository. For more information, see Git authentication.	- name: gitops_ssh_secret value: git-credentials

For information about the features supported by each implementation, see git implementation in the Flux documentation.

For information about how to create a workload that uses a GitHub repository as the provider of source code, see Create a workload from GitHub repository.

For information about GitRepository objects, see GitRepository.

ImageRepository

An ImageRepository object is instantiated when `deliverable.spec.source.image` is configured to continuously look for Kubernetes configuration files pushed to a container image registry as opposed to a Git repository.

Parameters:

Parameter name	Meaning	Example
`serviceAccount`	The name of the service account, in the same namespace as the deliverable, you want to use to provide the necessary permissions for kapp-controller to deploy the objects to the cluster.	- name: serviceAccount value: default

The `--service-account` flag sets the `spec.serviceAccountName` key in the deliverable object. To configure the `serviceAccount` parameter, use `--param serviceAccount=SERVICE-ACCOUNT`.

For information about custom resource details, see the ImageRepository reference documentation.

app deployer

The `app-deploy` resource in the ClusterDelivery applies the Kubernetes configuration that is built by the supply chain, pushed to either a Git repository or image repository, and applied to the cluster.

App

Regardless of where the configuration comes from, an App object is instantiated to deploy the set of Kubernetes configuration files to the cluster.

Parameters:

Parameter name	Meaning	Example
`serviceAccount`	The name of the service account, in the same namespace as the deliverable, you want to use to provide the necessary privileges for `App` to apply the Kubernetes objects to the cluster.	- name: serviceAccount value: default
`gitops_sub_path` (deprecated)	The subdirectory within the configuration bundle used for looking up the files to apply to the Kubernetes cluster.	- name: gitops_sub_path value: ./config

The `gitops_sub_path` parameter is deprecated. Use `deliverable.spec.source.subPath` instead.

The `--service-account` flag sets the `spec.serviceAccountName` key in the deliverable object.

To configure the `serviceAccount` parameter, use `--param serviceAccount=SERVICE-ACCOUNT`.

For details about RBAC and how `kapp-controller` uses the ServiceAccount provided to it using the `serviceAccount` parameter in the `deliverable` object, see kapp-controller's Security Model in the Carvel documentation.

Troubleshoot Tanzu Application Platform

These topics provide you with troubleshooting information to help resolve issues with your Tanzu Application Platform (commonly known as TAP):

- Troubleshoot installing Tanzu Application Platform
- Troubleshoot using Tanzu Application Platform
- Troubleshoot Tanzu Application Platform components
- Troubleshoot Tanzu GitOps Reference Implementation (RI)

Troubleshoot Tanzu Application Platform

These topics provide you with troubleshooting information to help resolve issues with your Tanzu Application Platform (commonly known as TAP):

- Troubleshoot installing Tanzu Application Platform
- Troubleshoot using Tanzu Application Platform
- Troubleshoot Tanzu Application Platform components
- Troubleshoot Tanzu GitOps Reference Implementation (RI)

Troubleshoot installing Tanzu Application Platform

This topic tells you how to troubleshoot installing Tanzu Application Platform (commonly known as TAP).

Developer cannot be verified when installing Tanzu CLI on macOS

You see the following error when you run Tanzu CLI commands, for example `tanzu version`, on macOS:

```
"tanzu" cannot be opened because the developer cannot be verified
```

Explanation

Security settings are preventing installation.

Solution

To resolve this issue:

1. Click **Cancel** in the macOS prompt window.
2. Open **System Preferences** > **Security & Privacy**.
3. Click **General**.
4. Next to the warning message for the Tanzu binary, click **Allow Anyway**.

5. Enter your system username and password in the macOS prompt window to confirm the changes.
6. In the terminal window, run:

   ```
   tanzu version
   ```

7. In the macOS prompt window, click **Open**.

Access `.status.usefulErrorMessage` details

When installing Tanzu Application Platform, you receive an error message that includes the following:

```
(message: Error (see .status.usefulErrorMessage for details))
```

Explanation

A package fails to reconcile and you must access the details in `.status.usefulErrorMessage`.

Solution

Access the details in `.status.usefulErrorMessage` by running:

```
kubectl get packageinstall PACKAGE-NAME -n tap-install -o yaml
```

Where `PACKAGE-NAME` is the name of the package to target.

“Unauthorized to access” error

When running the `tanzu package install` command, you receive an error message that includes the error:

```
UNAUTHORIZED: unauthorized to access repository
```

For example:

```
$ tanzu package install app-live-view -p appliveview.tanzu.vmware.com -v 0.1.0 -n tap-
install --values-file ./app-live-view.yaml

Error: package reconciliation failed: vendir: Error: Syncing directory '0':
  Syncing directory '.' with imgpkgBundle contents:
    Imgpkg: exit status 1 (stderr: Error: Checking if image is bundle: Collecting imag
es: Working with registry.tanzu.vmware.com/app-live-view/application-live-view-install
-bundle@sha256:b13b9ba81bcc985d76607cfc04bcbb8829b4cc2820e64a99e0af840681da12aa: GET h
ttps://registry.tanzu.vmware.com/v2/app-live-view/application-live-view-install-bundl
e/manifests/sha256:b13b9ba81bcc985d76607cfc04bcbb8829b4cc2820e64a99e0af840681da12aa: U
NAUTHORIZED: unauthorized to access repository: app-live-view/application-live-view-in
stall-bundle, action: pull: unauthorized to access repository: app-live-view/applicati
on-live-view-install-bundle, action: pull
```

Note

This example shows an error received when with Application Live View as the package. This error can also occur with other packages.

Explanation

The Tanzu Network credentials needed to access the package may be missing or incorrect.

Solution

To resolve this issue:

1. Repeat the step to create a secret for the namespace. For instructions, see Add the Tanzu Application Platform Package Repository in *Installing the Tanzu Application Platform Package and Profiles*. Ensure that you provide the correct credentials.

   When the secret has the correct credentials, the authentication error should resolve itself and the reconciliation succeed. Do not reinstall the package.

2. List the status of the installed packages to confirm that the reconcile has succeeded. For instructions, see Verify the Installed Packages in *Installing Individual Packages*.

"Serviceaccounts already exists" error

When running the `tanzu package install` command, you receive the following error:

```
failed to create ServiceAccount resource: serviceaccounts already exists
```

For example:

```
$ tanzu package install app-accelerator -p accelerator.apps.tanzu.vmware.com -v 0.2.0
-n tap-install --values-file app-accelerator-values.yaml

Error: failed to create ServiceAccount resource: serviceaccounts "app-accelerator-tap-
install-sa" already exists
```

Note

This example shows an error received with App Accelerator as the package. This error can also occur with other packages.

Explanation

The `tanzu package install` command may be executed again after failing.

Solution

To update the package, run the following command after the first use of the `tanzu package install` command

```
tanzu package installed update
```

After package installation, one or more packages fails to reconcile

You run the `tanzu package install` command and one or more packages fails to install.

For example:

```
tanzu package install tap -p tap.tanzu.vmware.com -v 0.4.0 -n tap-install --values-fil
e tap-values.yaml
- Installing package 'tap.tanzu.vmware.com'
\ Getting package metadata for 'tap.tanzu.vmware.com'
| Creating service account 'tap-tap-install-sa'
/ Creating cluster admin role 'tap-tap-install-cluster-role'
| Creating cluster role binding 'tap-tap-install-cluster-rolebinding'
| Creating secret 'tap-tap-install-values'
```

```
| Creating package resource
- Waiting for 'PackageInstall' reconciliation for 'tap'
/ 'PackageInstall' resource install status: Reconciling
| 'PackageInstall' resource install status: ReconcileFailed

Please consider using 'tanzu package installed update' to update the installed package
with correct settings


Error: resource reconciliation failed: kapp: Error: waiting on reconcile packageinstal
l/tap-gui (packaging.carvel.dev/v1alpha1) namespace: tap-install:
  Finished unsuccessfully (Reconcile failed:  (message: Error (see .status.usefulError
Message for details))). Reconcile failed: Error (see .status.usefulErrorMessage for de
tails)
Error: exit status 1
```

Explanation

Often, the cause is one of the following:

- Your infrastructure provider takes longer to perform tasks than the timeout value allows.
- A race-condition between components exists. For example, a package that uses `Ingress` completes before the shared Tanzu ingress controller becomes available.

The VMware Carvel tools kapp-controller continues to try in a reconciliation loop in these cases. However, if the reconciliation status is `failed` then there might be a configuration issue in the provided `tap-config.yaml` file.

Solution

1. Verify if the installation is still in progress by running:

   ```
   tanzu package installed list -A
   ```

   If the installation is still in progress, the command produces output similar to the following example, and the installation is likely to finish successfully.

   ```
   \ Retrieving installed packages...
     NAME                       PACKAGE-NAME
   PACKAGE-VERSION  STATUS               NAMESPACE
     accelerator                accelerator.apps.tanzu.vmware.com
   1.0.0            Reconcile succeeded  tap-install
     api-portal                 api-portal.tanzu.vmware.com
   1.0.6            Reconcile succeeded  tap-install
     appliveview                run.appliveview.tanzu.vmware.com
   1.0.0-build.3    Reconciling          tap-install
     appliveview-conventions    build.appliveview.tanzu.vmware.com
   1.0.0-build.3    Reconcile succeeded  tap-install
     buildservice               buildservice.tanzu.vmware.com
   1.4.0-build.1    Reconciling          tap-install
     cartographer               cartographer.tanzu.vmware.com
   0.1.0            Reconcile succeeded  tap-install
     cert-manager               cert-manager.tanzu.vmware.com
   1.5.3+tap.1      Reconcile succeeded  tap-install
     cnrs                       cnrs.tanzu.vmware.com
   1.1.0            Reconcile succeeded  tap-install
     contour                    contour.tanzu.vmware.com
   1.18.2+tap.1     Reconcile succeeded  tap-install
     conventions-controller     controller.conventions.apps.tanzu.vmware.com
   0.4.2            Reconcile succeeded  tap-install
     developer-conventions      developer-conventions.tanzu.vmware.com
   0.4.0-build1     Reconcile succeeded  tap-install
     fluxcd-source-controller   fluxcd.source.controller.tanzu.vmware.com
   0.16.0           Reconcile succeeded  tap-install
     grype                      grype.scanning.apps.tanzu.vmware.com
   ```

```
1.0.0            Reconcile succeeded  tap-install
  image-policy-webhook      image-policy-webhook.signing.apps.tanzu.vmware.com
1.0.0-beta.3     Reconcile succeeded  tap-install
  ootb-delivery-basic       ootb-delivery-basic.tanzu.vmware.com
0.5.1            Reconcile succeeded  tap-install
  ootb-supply-chain-basic   ootb-supply-chain-basic.tanzu.vmware.com
0.5.1            Reconcile succeeded  tap-install
  ootb-templates            ootb-templates.tanzu.vmware.com
0.5.1            Reconcile succeeded  tap-install
  scanning                  scanning.apps.tanzu.vmware.com
1.0.0            Reconcile succeeded  tap-install
  metadata-store            metadata-store.apps.tanzu.vmware.com
1.0.2            Reconcile succeeded  tap-install
  service-bindings          service-bindings.labs.vmware.com
0.6.0            Reconcile succeeded  tap-install
  services-toolkit          services-toolkit.tanzu.vmware.com
0.7.1            Reconcile succeeded  tap-install
  source-controller         controller.source.apps.tanzu.vmware.com
0.2.0            Reconcile succeeded  tap-install
  spring-boot-conventions   spring-boot-conventions.tanzu.vmware.com
0.2.0            Reconcile succeeded  tap-install
  tap                       tap.tanzu.vmware.com
0.4.0-build.12   Reconciling          tap-install
  tap-gui                   tap-gui.tanzu.vmware.com
1.0.0-rc.72      Reconcile succeeded  tap-install
  tap-telemetry             tap-telemetry.tanzu.vmware.com
0.1.0            Reconcile succeeded  tap-install
  tekton-pipelines          tekton.tanzu.vmware.com
0.30.0           Reconcile succeeded  tap-install
```

If the installation has stopped running, one or more reconciliations have likely failed, as seen in the following example:

```
NAME                       PACKAGE NAME
PACKAGE VERSION   DESCRIPTION
AGE
accelerator                accelerator.apps.tanzu.vmware.com
1.0.1             Reconcile succeeded
109m
api-portal                 api-portal.tanzu.vmware.com
1.0.9             Reconcile succeeded
119m
appliveview                run.appliveview.tanzu.vmware.com
1.0.2-build.2     Reconcile succeeded
109m
appliveview-conventions    build.appliveview.tanzu.vmware.com
1.0.2-build.2     Reconcile succeeded
109m
buildservice               buildservice.tanzu.vmware.com
1.5.0             Reconcile succeeded
119m
cartographer               cartographer.tanzu.vmware.com
0.2.1             Reconcile succeeded
117m
cert-manager               cert-manager.tanzu.vmware.com
1.5.3+tap.1       Reconcile succeeded
119m
cnrs                       cnrs.tanzu.vmware.com
1.1.0             Reconcile succeeded
109m
contour                    contour.tanzu.vmware.com
1.18.2+tap.1      Reconcile succeeded
117m
conventions-controller     controller.conventions.apps.tanzu.vmware.com
0.5.0             Reconcile succeeded
```

```
117m
developer-conventions      developer-conventions.tanzu.vmware.com
0.5.0             Reconcile succeeded
109m
fluxcd-source-controller   fluxcd.source.controller.tanzu.vmware.com
0.16.1            Reconcile succeeded
119m
grype                      grype.scanning.apps.tanzu.vmware.com
1.0.0             Reconcile failed: Error (see .status.usefulErrorMessage for d
etails)   109m
image-policy-webhook       image-policy-webhook.signing.apps.tanzu.vmware.com
1.0.1             Reconcile succeeded
117m
metadata-store             metadata-store.apps.tanzu.vmware.com
1.0.2             Reconcile succeeded
117m
ootb-delivery-basic        ootb-delivery-basic.tanzu.vmware.com
0.6.1             Reconcile succeeded
103m
ootb-supply-chain-basic    ootb-supply-chain-basic.tanzu.vmware.com
0.6.1             Reconcile succeeded
103m
ootb-templates             ootb-templates.tanzu.vmware.com
0.6.1             Reconcile succeeded
109m
scanning                   scanning.apps.tanzu.vmware.com
1.0.0             Reconcile succeeded
119m
service-bindings           service-bindings.labs.vmware.com
0.6.0             Reconcile succeeded
119m
services-toolkit           services-toolkit.tanzu.vmware.com
0.7.1             Reconcile succeeded
119m
source-controller          controller.source.apps.tanzu.vmware.com
0.2.0             Reconcile succeeded
119m
spring-boot-conventions    spring-boot-conventions.tanzu.vmware.com
0.3.0             Reconcile succeeded
109m
tap                        tap.tanzu.vmware.com
1.0.1             Reconcile failed: Error (see .status.usefulErrorMessage for d
etails)   119m
tap-gui                    tap-gui.tanzu.vmware.com
1.0.2             Reconcile succeeded
109m
tap-telemetry              tap-telemetry.tanzu.vmware.com
0.1.3             Reconcile succeeded
119m
tekton-pipelines           tekton.tanzu.vmware.com
0.30.0            Reconcile succeeded
119m
```

In this example, `packageinstall/grype` and `packageinstall/tap` have reconciliation errors.

2. To get more details on the possible cause of a reconciliation failure, run:

   ```
   kubectl describe packageinstall/NAME -n tap-install
   ```

   Where `NAME` is the name of the failing package. For this example it would be `grype`.

3. Use the displayed information to search for a relevant troubleshooting issue in this topic. If none exists, and you are unable to fix the described issue yourself, please contact support.

4. Repeat these diagnosis steps for any other packages that failed to reconcile.

Failure to accept an End User License Agreement error

You cannot access Tanzu Application Platform or one of its components from VMware Tanzu Network.

Explanation

You cannot access Tanzu Application Platform or one of its components from VMware Tanzu Network before accepting the relevant EULA in VMware Tanzu Network.

Solution

Follow the steps in Accept the End User License Agreements in *Installing the Tanzu CLI*.

Ingress is broken on Kind cluster

Your Contour installation cannot provide ingress to workloads when installed on a Kind cluster without a LoadBalancer solution. Your Kind cluster was created with port mappings.

Explanation

In Tanzu Application Platform v1.9.1, the default configuration for `contour.envoy.service.type` is `LoadBalancer`. However, for the Envoy pods to be accessed by using the port mappings on your Kind cluster, the service must be of type `NodePort`.

Solution

Configure `contour.evnoy.service.type` to be `NodePort`. Then, configure `envoy.service.nodePorts.http` and `envoy.service.nodePorts.https` to the corresponding port mappings on your Kind node. Otherwise, the NodePort service is assigned random ports, which are not accessible through your Kind cluster.

Service binding package fails to reconcile

You receive the following error message when deploying or upgrading to Tanzu Application Platform v1.9:

```
ValidationError(Package.spec.template.spec.template[0].ytt.valuesFrom[0]): unknown fie
ld "downwardAPI"
```

Explanation

The version of Cluster Essentials is not supported. To deploy successfully, the `servicebinding.tanzu.vmware.com` package requires Cluster Essentials `v1.3.0` or later.

Solution

Upgrade to Cluster Essentials v1.9. For more information about the upgrade procedures, see the Cluster Essentials documentation.

Reconciliation fails with Supply Chain Security Tools - Scan 2.0 upgrade

When deploying Supply Chain Security Tools - Scan 2.0 `app-scanning.apps.tanzu.vmware.com` to a cluster and upgrading Tanzu Application Platform from v1.6 to v1.7 or later, you might encounter a reconciliation error as follows:

```
Expected to find at least one version, but did not (details: all=1 ->
      after-prereleases-filter=1 -> after-kapp-controller-version-check=1 -> after-con
straints-filter=0)
```

Explanation

The upgrade instructions overwrite the Tanzu Application Platform v1.6 repository with the 1.7 version or later, which leads to a reconciliation error.

Solution

You can resolve this issue by using either of the following two solutions:

1. Upgrade `app-scanning.apps.tanzu.vmware.com` to v0.2.1.
2. Add a new package repository with Tanzu Application Platform v1.6.

Troubleshoot using Tanzu Application Platform

This topic tells you how to troubleshoot using Tanzu Application Platform (commonly known as TAP).

Use events to find possible causes

Events can highlight issues with components in a supply chain. For example, high occurrences of `StampedObjectApplied` or `ResourceOutputChanged` can indicate problems with trashing on a component.

To view the recent events for a workload, run:

```
kubectl describe workload.carto.run <workload-name> -n <workload-ns>
```

Missing build logs after creating a workload

Symptom:

You create a workload, but no logs appear when you run:

```
tanzu apps workload tail workload-name --since 10m --timestamp
```

Explanation:

Common causes include:

- Misconfigured repository
- Misconfigured service account
- Misconfigured registry credentials

Solution:

To resolve this issue, run:

```
kubectl get clusterbuilder.kpack.io -o yaml
```

```
kubectl get image.kpack.io <workload-name> -o yaml
```

```
kubectl get build.kpack.io -o yaml
```

Workload creation stops responding with “Builder default is not ready” message

Symptom:

You can see the “Builder default is not ready” message in either of two places:

1. The “Messages” section of the `tanzu apps workload get my-app` command.
2. The Supply Chain section of Tanzu Developer Portal.

This message indicates there is something wrong with the Builder (the component that builds the container image for your workload).

Explanation:

This message is typically encountered when the core component of the Builder (`kpack`) transitions into a bad state.

Although this isn’t the only scenario where this can happen, `kpack` can transition into a bad state when Tanzu Application Platform is deployed to a local `kind` cluster, and especially when that `kind` cluster is restarted.

Solution:

1. Restart `kpack` by deleting the `kpack-controller` and `kpack-webhook` pods in the `kpack` namespace. Deleting these resources triggers their recreation:
    - `kubectl delete pods --all --namespace kpack`
2. Verify status of the replacement pods:
    - `kubectl get pods --namespace kpack`
3. Verify the workload status after the new kpack pods `STATUS` are `Running`:
    - `tanzu apps workload get YOUR-WORKLOAD-NAME`

“Workload already exists” error after updating the workload

Symptom:

When you update the workload, you receive the following error:

```
Error: workload "default/APP-NAME" already exists
Error: exit status 1
```

Where `APP-NAME` is the name of the app.

For example, when you run:

```
tanzu apps workload create tanzu-java-web-app \
--git-repo https://github.com/dbuchko/tanzu-java-web-app \
--git-branch main \
--type web \
--label apps.tanzu.vmware.com/has-tests=true \
--yes
```

You receive the following error

```
Error: workload "default/tanzu-java-web-app" already exists
Error: exit status 1
```

Explanation:

The app is running before performing a Live Update using the same app name.

Solution:

To resolve this issue, either delete the app or use a different name for the app.

Workload creation fails due to authentication failure in Docker Registry

Symptom:

You might encounter an error message similar to the following when creating or updating a workload by using IDE or `apps` CLI plug-in:

```
Error: Writing 'index.docker.io/shaileshp2922/build-service/tanzu-java-web-app:lates
t': Error while preparing a transport to talk with the registry: Unable to create roun
d tripper: GET https://auth.ipv6.docker.com/token?scope=repository%3Ashaileshp2922%2Fb
uild-service%2Ftanzu-java-web-app%3Apush%2Cpull&service=registry.docker.io: unexpected
status code 401 Unauthorized: {"details":"incorrect username or password"}
```

Explanation:

This type of error frequently occurs when the URL set for `source image` (IDE) or `--source-image` flag (`apps` CLI plug-in) is not Docker registry compliant.

Solution:

1. Verify that you can authenticate directly against the Docker registry and resolve any failures by running:

   ```
   docker login -u USER-NAME
   ```

2. Verify your `--source-image` URL is compliant with Docker.

   The URL in this example `index.docker.io/shaileshp2922/build-service/tanzu-java-web-app` includes nesting. Docker registry, unlike many other registry solutions, does not support nesting.

3. To resolve this issue, you must provide an unnested URL. For example, `index.docker.io/shaileshp2922/tanzu-java-web-app`

Telemetry component logs show errors fetching the “reg-creds” secret

Symptom:

When you view the logs of the `tap-telemetry` controller by running `kubectl logs -n tap-telemetry <tap-telemetry-controller-<hash> -f`, you see the following error:

```
"Error retrieving secret reg-creds on namespace tap-telemetry","error":"secrets \"reg-
creds\" is forbidden: User \"system:serviceaccount:tap-telemetry:controller\" cannot g
et resource \"secrets\" in API group \"\" in the namespace \"tap-telemetry\""
```

Explanation:

The `tap-telemetry` namespace misses a role that allows the controller to list secrets in the `tap-telemetry` namespace. For more information about roles, see Role and ClusterRole Kubernetes documentation.

Solution:

To resolve this issue, run:

```
kubectl patch roles -n tap-telemetry tap-telemetry-controller --type='json' -p='[{"o
p": "add", "path": "/rules/-", "value": {"apiGroups": [""],"resources": ["secrets"],"v
erbs": ["get", "list", "watch"]} }]'
```

Debug convention might not apply

Symptom:

If you upgrade from Tanzu Application Platform v0.4, the debug convention can not apply to the app run image.

Explanation:

The Tanzu Application Platform v0.4 lacks SBOM data.

Solution:

Delete existing app images that were built using Tanzu Application Platform v0.4.

Execute bit not set for App Accelerator build scripts

Symptom:

You cannot execute a build script provided as part of an accelerator.

Explanation:

Build scripts provided as part of an accelerator do not have the execute bit set when a new project is generated from the accelerator.

Solution:

Explicitly set the execute bit by running the `chmod` command:

```
chmod +x BUILD-SCRIPT-NAME
```

Where `BUILD-SCRIPT-NAME` is the name of the build script.

For example, for a project generated from the “Spring PetClinic” accelerator, run:

```
chmod +x ./mvnw
```

“No live information for pod with ID” error

Symptom:

After deploying Tanzu Application Platform workloads, Tanzu Developer Portal shows a “No live information for pod with ID” error.

Explanation:

The connector must discover the application instances and render the details in Tanzu Application Platform GUI.

Solution:

Recreate the Application Live View connector pod by running:

```
kubectl -n app-live-view delete pods -l=name=application-live-view-connector
```

This allows the connector to discover the application instances and render the details in Tanzu Application Platform GUI.

“image-policy-webhook-service not found” error

Symptom:

When installing a Tanzu Application Platform profile, you receive the following error:

```
Internal error occurred: failed calling webhook "image-policy-webhook.signing.apps.tanzu.vmware.com": failed to call webhook: Post "https://image-policy-webhook-service.image-policy-system.svc:443/signing-policy-check?timeout=10s": service "image-policy-webhook-service" not found
```

Explanation:

The “image-policy-webhook-service” service cannot be found.

Solution:

Redeploy the `trainingPortal` resource.

“Increase your cluster resources” error

Symptom:

You receive an “Increase your cluster’s resources” error.

Explanation:

Node pressure can be caused by an insufficient number of nodes or a lack of resources on nodes necessary to deploy the workloads.

Solution:

Follow instructions from your cloud provider to scale out or scale up your cluster.

CrashLoopBackOff from password authentication fails

Symptom:

SCST - Store does not start. You see the following error in the `metadata-store-app` Pod logs:

```
$ kubectl logs pod/metadata-store-app-* -n metadata-store -c metadata-store-app
...
[error] failed to initialize database, got error failed to connect to `host=metadata-store-db user=metadata-store-user database=metadata-store`: server error (FATAL: password authentication failed for user "metadata-store-user" (SQLSTATE 28P01))
```

Explanation:

The database password has changed between deployments. This is not supported.

Solution:

Redeploy the app either with the original database password or follow the latter steps to erase the data on the volume:

1. Deploy `metadata-store app` with kapp.
2. Verify that the `metadata-store-db-*` pod fails.
3. Run:

```
kubectl exec -it metadata-store-db-KUBERNETES-ID -n metadata-store /bin/bash
```

Where `KUBERNETES-ID` is the ID generated by Kubernetes and appended to the pod name.

4. To delete all database data, run:

   ```
   rm -rf /var/lib/postgresql/data/*
   ```

   This is the path found in `postgres-db-deployment.yaml`.

5. Delete the `metadata-store` app with kapp.
6. Deploy the `metadata-store` app with kapp.

Password authentication fails

Symptom:

SCST - Store does not start. You see the following error in the `metadata-store-app` pod logs:

```
$ kubectl logs pod/metadata-store-app-* -n metadata-store -c metadata-store-app
...
[error] failed to initialize database, got error failed to connect to `host=metadata-s
tore-db user=metadata-store-user database=metadata-store`: server error (FATAL: passwo
rd authentication failed for user "metadata-store-user" (SQLSTATE 28P01))
```

Explanation

The database password has changed between deployments. This is not supported.

Solution:

Redeploy the app either with the original database password or follow the latter steps to erase the data on the volume:

1. Deploy `metadata-store app` with kapp.
2. Verify that the `metadata-store-db-*` pod fails.
3. Run:

   ```
   kubectl exec -it metadata-store-db-KUBERNETES-ID -n metadata-store /bin/bash
   ```

   Where `KUBERNETES-ID` is the ID generated by Kubernetes and appended to the pod name.

4. To delete all database data, run:

   ```
   rm -rf /var/lib/postgresql/data/*
   ```

   This is the path found in `postgres-db-deployment.yaml`.

5. Delete the `metadata-store` app with kapp.
6. Deploy the `metadata-store` app with kapp.

`metadata-store-db` pod fails to start

Symptom:

When SCST - Store is deployed, deleted, and then redeployed, the `metadata-store-db` pod fails to start if the database password changed during redeployment.

Explanation:

The persistent volume used by `PostgreSQL` retains old data, even though the retention policy is set to `DELETE`.

Solution:

Redeploy the app either with the original database password or follow the later steps to erase the data on the volume:

1. Deploy `metadata-store app` with kapp.
2. Verify that the `metadata-store-db-*` pod fails.
3. Run:

   ```
   kubectl exec -it metadata-store-db-KUBERNETES-ID -n metadata-store /bin/bash
   ```

   Where `KUBERNETES-ID` is the ID generated by Kubernetes and appended to the pod name.
4. To delete all database data, run:

   ```
   rm -rf /var/lib/postgresql/data/*
   ```

   This is the path found in `postgres-db-deployment.yaml`.
5. Delete the `metadata-store` app with kapp.
6. Deploy the `metadata-store` app with kapp.

Missing persistent volume

Symptom:

After SCST - Store is deployed, `metadata-store-db` pod fails for missing volume while `postgres-db-pv-claim` pvc is in the `PENDING` state.

Explanation:

The cluster where SCST - Store is deployed does not have `storageclass` defined. The provisioner of `storageclass` is responsible for creating the persistent volume after `metadata-store-db` attaches `postgres-db-pv-claim`.

Solution:

1. Verify that your cluster has `storageclass` by running:

   ```
   kubectl get storageclass
   ```
2. Create a `storageclass` in your cluster before deploying SCST - Store. For example:

   ```
   # This is the storageclass that Kind uses
   kubectl apply -f https://raw.githubusercontent.com/rancher/local-path-provision
   er/master/deploy/local-path-storage.yaml

   # set the storage class as default
   kubectl patch storageclass local-path -p '{"metadata": {"annotations":{"storage
   class.kubernetes.io/is-default-class":"true"}}}'
   ```

Failure to connect Tanzu CLI to AWS EKS clusters

Symptom:

When using the Tanzu CLI to connect to AWS EKS clusters, you might see one of the following errors:

- `Error: Unable to connect: connection refused. Confirm kubeconfig details and try again`
- `invalid apiVersion "client.authentication.k8s.io/v1alpha1"`

Explanation:

The cause is Kubernetes v1.24 dropping support for `client.authentication.k8s.io/v1alpha1`. For more information, see aws/aws-cli/issues/6920 in GitHub.

Solution:

Follow these steps to update your `aws-cli` to a supported v2.7.35 or later, and update the `kubeconfig` entry for your EKS clusters:

1. Update `aws-cli` to the latest version. For more information see AWS documentation.
2. Update the `kubeconfig` entry for your EKS clusters:

   ```
   aws eks update-kubeconfig --name ${EKS_CLUSTER_NAME} --region ${REGION}
   ```

3. In a new terminal window, run a Tanzu CLI command to verify the connection issue is resolved. For example:

   ```
   tanzu apps workload list
   ```

   Expect the command to execute without error.

Invalid repository paths are propagated

Symptom:

When inputting `shared.image_registry.project_path`, invalid repository paths are propagated.

Explanation:

The key `shared.image_registry.project_path`, which takes input as `SERVER-NAME/REPO-NAME`, cannot take “/” at the end of the string.

Solution:

Do not append “/” to the end of the string.

---

x509: certificate signed by unknown authority

Explanation:

Tanzu Application Platform v1.4 introduces Shared Ingress Issuer to secure ingress communication by default. The Certificate Authority for Shared Ingress Issuer is generated as self-signed. As a result, you might see one of the following errors:

- `connection refused`
- `x509: certificate signed by unknown authority`

Solution:

You can choose one of the following options to mitigate the issue:

Option 1: Configure the Shared Ingress Issuer's Certificate Authority as a trusted Certificate Authority

Important

This is the recommended option for a secure instance.

Follow these steps to trust the Shared Ingress Issuer's Certificate Authority in Tanzu Application Platform:

1. Extract the ClusterIssuer's Certificate Authority.

   For default installations where `ingress_issuer` is not set in `tap_values.yml`, you can extract the ClusterIssuer's Certificate Authority from cert-manager:

   ```
   kubectl get secret tap-ingress-selfsigned-root-ca -n cert-manager -o yaml | yq
   .data | cut -d' ' -f2 | head -1 | base64 -d
   ```

   If you overrode the default `ingress_issuer` while installing Tanzu Application Platform, you must refer to your issuer's documentation to extract your ClusterIssuer's Certificate Authority instead of using the command above.

2. Add the certificate to the list of trusted certificate authorities by appending the certificate authority to the `shared.ca_cert_data` field in your `tap-values.yml`.

3. Reapply your configuration:

   ```
   tanzu package install tap -p tap.tanzu.vmware.com -v ${TAP_VERSION} --values-fi
   le tap-values.yml -n tap-install
   ```

Option 2: Deactivate the shared ingress issuer

Important

This option is recommended for testing purposes only.

Follow these steps to deactivate TLS for Cloud Native Runtimes, AppSSO and Tanzu Developer Portal:

1. Set `shared.ingress_issuer` to `""` in your `tap-values.yml`:

   ```
   shared:
     ingress_issuer: ""
   ```

2. Reapply your configuration:

   ```
   tanzu package install tap -p tap.tanzu.vmware.com -v ${TAP_VERSION} --values-fi
   le tap-values.yml -n tap-install
   ```

Datadog agent cannot reconcile webhook on AKS

Symptom:

On Azure Kubernetes Service (AKS), you receive an error because the Datadog Cluster Agent cannot reconcile the webhook.

Solution:

> **Note**
>
> This workaround deactivates the admission controller, which might have implications for certain features. See the Datadog documentation or contact support for guidance based on your specific use case.

To work around this issue:

1. Create a custom values file for Datadog named `values.yaml`.
2. Enter the following YAML into your `values.yaml` file, which sets `clusterAgent.admissionController` to `false` and sets the environment variable `DD_ADMISSION_CONTROLLER_ADD_AKS_SELECTORS` to `true`:

   ```
   clusterAgent:
     admissionController:
       enabled: false
     env:
       - name: "DD_ADMISSION_CONTROLLER_ADD_AKS_SELECTORS"
         value: "true"
   ```

3. Install the Datadog Agent Helm chart with your custom `values.yaml` file:

   ```
   helm upgrade --install datadog-operator datadog/datadog-operator -f values.yaml
   ```

Troubleshoot Tanzu Application Platform components

For component-level troubleshooting, see these topics:

- Troubleshoot Application Live View
- Troubleshoot AWS Services
- Troubleshoot Bitnami Services
- Troubleshoot Cloud Native Runtimes for Tanzu
- Troubleshoot Contour
- Troubleshoot Crossplane
- Troubleshoot Service Bindings
- Troubleshoot Services Toolkit
- Troubleshoot Source Controller
- Troubleshoot Spring Boot conventions
- Troubleshoot Supply Chain Security Tools - Scan
- Troubleshoot Supply Chain Security Tools - Store
- Troubleshoot Tanzu Build Service
- Tanzu Build Service FAQ
- Troubleshoot Tanzu Developer Portal

Troubleshoot Tanzu GitOps Reference Implementation (RI)

This topic tells you how to troubleshoot Tanzu GitOps Reference Implementation (commonly known as RI).

Tanzu Sync application error

After the Tanzu Sync application is installed in the cluster, the main resource to check is the sync app in the `tanzu-sync` namespace:

```
kubectl -n tanzu-sync get app/sync --template='{{.status.usefulErrorMessage}}'
```

Example error:

```
kapp: Error: waiting on reconcile packageinstall/tap (packaging.carvel.dev/v1alpha1) n
amespace: tap-install:
  Finished unsuccessfully (Reconcile failed:  (message: Error (see .status.usefulError
Message for details)))
```

This indicates that the resource `packageinstall/tap` in the namespace `tap-install` failed. See the following section for the solution details.

Tanzu Application Platform install error

After the Tanzu Sync application is installed in the cluster, the Tanzu Application Platform starts to install. The resource to check is the Tanzu Application Platform package install in the `tap-install` namespace:

```
kubectl -n tap-install get packageinstall/tap --template='{{.status.usefulErrorMessag
e}}'
```

Common errors

You might encounter one of the following errors:

Given data value is not declared in schema

Error: Reconciliation fails with `Given data value is not declared in schema`

```
^ Reconcile failed:  (message: ytt: Error: Overlaying data values (in following order:
tap-install/.tanzu-managed/version.yaml, additional data values):
One or more data values were invalid
====================================

Given data value is not declared in schema
tap-values.yaml:
    |
  1 | shared:
    |

    = found: shared
    = expected: a map item with the key named "tap_install" (from tap-install/.tanzu-m
anaged/schema--tap-sensitive-values.yaml:3)
```

Problem: The values files were not generated according to the expected schema.

Solution: Ensure both non-sensitive and sensitive Tanzu Application Platform values files to adhere to the schema described in configure values.

Incorrect values example:

```
shared:
  ingress_domain: example.vmware.com
```

Correct values example:

```
tap_install:
  values:
    ingress_domain: example.vmware.com
```

`ExternalSecret` not found

Error: Reconciliation of `ExternalSecret` fails with `secret not found`

Example:

```
kubectl describe ExternalSecret install-registry-dockerconfig -n tanzu-sync

Status:
  Conditions:
    Last Transition Time:  2023-07-20T08:14:04Z
    Message:               could not get secret data from provider
    Reason:                SecretSyncedError
    Status:                False
    Type:                  Ready
Events:
  Type     Reason         Age                   From              Message
  ----     ------         ----                  ----              -------
  Warning  UpdateFailed  12s (x15 over 6m14s)  external-secrets  secret not found
```

- **Problem 1:** Incorrect secret reference

  Solution 1: Ensure the references in `tanzu-sync/app/values/tanzu-sync-vault-values.yaml` and `cluster-config/values/tap-install-vault-values.yaml` are correct.

- **Problem 2:** Incorrect configuration of the secret engine for Vault

  Solution 2: Validate and update the version of the Vault secrets engine configuration to `v1` or `v2`. By default, `v2` is chosen.

  Update `tanzu-sync/app/values/tanzu-sync-vault-values.yaml`:

  ```
  ---
  secrets:
    eso:
      vault:
        ...
        version: "v1" # v1 or v2
  ```

  Update `cluster-config/values/tap-install-vault-values.yaml`:

  ```
  ---
  tap_install:
    secrets:
      eso:
        vault:
          ...
          version: "v1" # v1 or v2
  ```

Error occurs when deleting Kubernetes authentication

Problem: The following error occurs when deleting Kubernetes authentication:

```
./tanzu-sync/vault/scripts/setup/delete-kubernetes-auth.sh
flag provided but not defined: -path
```

Cause: The command to disable vault authentication is incorrect.

Solution: Update the command `vault auth disable -path=$CLUSTER_NAME kubernetes` to `vault auth disable $CLUSTER_NAME` in the file `./tanzu-sync/vault/scripts/setup/delete-kubernetes-auth.sh`

Uninstall your Tanzu Application Platform by using Tanzu CLI

This document tells you how to uninstall Tanzu Application Platform (commonly known as TAP) packages from your Tanzu Application Platform package repository by using Tanzu CLI.

To uninstall Tanzu Application Platform:

- Delete the Packages
- Delete the Tanzu Application Platform Package Repository
- Remove Tanzu CLI, plug-ins, and associated files
- Remove Cluster Essentials

Delete the packages

- If you installed Tanzu Application Platform through predefined profiles, delete the `tap` metadata package by running:

  ```
  tanzu package installed delete tap --namespace tap-install
  ```

- If you installed any additional packages that were not in the predefined profiles, delete the individual packages by running:

  1. List the installed packages by running:

     ```
     tanzu package installed list --namespace tap-install
     ```

  2. Remove a package by running:

     ```
     tanzu package installed delete PACKAGE-NAME --namespace tap-install
     ```

     For example:

     ```
     $ tanzu package installed delete cloud-native-runtimes --namespace tap-in
     stall
     | Uninstalling package 'cloud-native-runtimes' from namespace 'tap-instal
     l'
     / Getting package install for 'cloud-native-runtimes'
     \ Deleting package install 'cloud-native-runtimes' from namespace 'tap-in
     stall'
     \ Package uninstall status: Reconciling
     / Package uninstall status: Deleting
     | Deleting admin role 'cloud-native-runtimes-tap-install-cluster-role'
     | Deleting role binding 'cloud-native-runtimes-tap-install-cluster-rolebi
     nding'
     | Deleting secret 'cloud-native-runtimes-tap-install-values'
     / Deleting service account 'cloud-native-runtimes-tap-install-sa'

      Uninstalled package 'cloud-native-runtimes' from namespace 'tap-install'
     ```

     Where `PACKAGE-NAME` is the name of a package listed in step 1.

3. Repeat step 2 for each individual package installed.

Delete the Tanzu Application Platform package repository

To delete the Tanzu Application Platform package repository:

1. Retrieve the name of the Tanzu Application Platform package repository by running:

   ```
   tanzu package repository list --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package repository list --namespace tap-install
   - Retrieving repositories...
     NAME                  REPOSITORY
   STATUS               DETAILS
     tanzu-tap-repository  registry.tanzu.vmware.com/tanzu-application-platform/ta
   p-packages:0.2.0  Reconcile succeeded
   ```

2. Remove the Tanzu Application Platform package repository by running:

   ```
   tanzu package repository delete PACKAGE-REPO-NAME --namespace tap-install
   ```

   Where `PACKAGE-REPO-NAME` is the name of the packageRepository from the earlier step.

   For example:

   ```
   $ tanzu package repository delete tanzu-tap-repository --namespace tap-install
   - Deleting package repository 'tanzu-tap-repository'...
    Deleted package repository 'tanzu-tap-repository' in namespace 'tap-install'
   ```

Remove Tanzu CLI, plug-ins, and associated files

To completely remove the Tanzu CLI, plug-ins, and associated files, run the script for your OS:

- For Linux or MacOS, run:

  ```
  #!/bin/zsh
  rm -rf $HOME/tanzu/cli        # Remove previously downloaded cli files
  sudo rm /usr/local/bin/tanzu  # Remove CLI binary (executable)
  rm -rf ~/.config/tanzu/       # current location # Remove config directory
  rm -rf ~/.tanzu/              # old location # Remove config directory
  rm -rf ~/.cache/tanzu         # remove cached catalog.yaml
  rm -rf ~/Library/Application\ Support/tanzu-cli/* # Remove plug-ins
  ```

- Uninstall tanzu cli installed using package manager:
  - On mac:

    ```
    brew uninstall vmware-tanzu/tanzu/tanzu-cli
    ```

  - On Linux:

    ```
    sudo apt remove tanzu-cli
    ```

    ```
    sudo yum remove tanzu-cli
    ```

  - On Windows:

```
choco uninstall tanzu-cli
```

Remove Cluster Essentials

To completely remove Cluster Essentials, see Cluster Essentials documentation.

Remove Crossplane resources

To remove Crossplane resources, see Delete Crossplane resources when you uninstall Tanzu Application Platform.

Uninstall Tanzu Application Platform by using GitOps

This document tells you how to uninstall Tanzu Application Platform (commonly known as TAP) when installed by using GitOps.

> **Caution**
>
> Tanzu Application Platform (GitOps) is currently in beta and is intended for evaluation and test purposes only. Do not use in a production environment.

To uninstall your Tanzu Application Platform:

- Delete Tanzu Sync Application
- Delete external resources (ESO installation only)
- Remove the Tanzu CLI, plug-ins, and associated files
- Remove Cluster Essentials

Delete Tanzu Sync Application

> **Caution**
>
> - Deleting Tanzu Sync application removes all associated resources of Tanzu Application Platform on the cluster.
> - You must delete any applications that were installed manually into the `tap-install` namespace, because they might interfere with the deletion of Tanzu Application Platform.

To delete Tanzu Sync Application, run:

```
kapp delete -a tanzu-sync
```

Delete external resources from AWS Secrets Manager

To delete external resources from AWS, run:

```
cd $HOME/REPO-NAME/clusters/CLUSTER-NAME
./tanzu-sync/aws/scripts/setup/delete-irsa.sh
./tanzu-sync/aws/scripts/setup/delete-policies.sh
```

Delete external resources from Hashicorp Vault

To delete external resources from AWS, run:

```
cd $HOME/REPO-NAME/clusters/CLUSTER-NAME
./tanzu-sync/vault/scripts/setup/delete-roles.sh
./tanzu-sync/vault/scripts/setup/delete-policies.sh
./tanzu-sync/vault/scripts/setup/delete-kubernetes-auth.sh
```

Remove the Tanzu CLI, plug-ins, and associated files

To completely remove the Tanzu CLI, plug-ins, and associated files, run the following scrips for Linux or MacOS:

```
#!/bin/zsh
rm -rf $HOME/tanzu/cli         # Remove previously downloaded cli files
sudo rm /usr/local/bin/tanzu   # Remove CLI binary (executable)
rm -rf ~/.config/tanzu/        # current location # Remove config directory
rm -rf ~/.tanzu/               # old location # Remove config directory
rm -rf ~/.cache/tanzu          # remove cached catalog.yaml
rm -rf ~/Library/Application\ Support/tanzu-cli/* # Remove plug-ins
```

Remove Cluster Essentials

To completely remove Cluster Essentials, see Cluster Essentials documentation.

Glossary

This topic gives you definitions of terms used in the Tanzu Application Platform documentation.

A

Terms beginning with “A”.

Term	Definition
API Auto Registration	This Tanzu Application Platform component automates registering API specifications defined in a workload's configuration.
API portal	This Tanzu Application Platform component enables API consumers to find APIs they can use in their own applications.
API Scoring and Validation	This Tanzu Application Platform component focuses on scanning and validating OpenAPI specifications. It ensures your APIs are secure and robust by providing feedback and recommendations early on in the software development life cycle.
Application Accelerator	This Tanzu Application Platform component provides a method to create and use application project templates that codify best practices and ensure that important configurations and structures are in place. Application developers use Application Accelerator to quickly bootstrap applications.
Application Accelerator engine	The component that performs the file transformations specified in an accelerator's `accelerator.yaml` file.
Application Configuration Service	This Tanzu Application Platform component provides a Kubernetes-native experience to enable the runtime configuration of existing Spring applications that were previously leveraged by using Spring Cloud Config Server.
Application Live View	A set of Tanzu Application Platform components that provide insight and troubleshooting capability that helps application developers and application operators look inside running applications.
Application Live View APIServer	This Tanzu Application Platform component is the part of Application Live View that generates a unique token when a user receives access validation to a pod. The Application Live View connector component verifies the token against the Application Live View APIServer before proxying the actuator data from the application. This ensures that the actuator data is secured and only the user who has valid access to view the live information for the pod can retrieve the data.
Application Live View back end	This Tanzu Application Platform component is the central server for Application Live View that contains a list of registered apps. It is responsible for proxying the request to fetch the actuator information related to the app.
Application Live View connector	This Tanzu Application Platform component is the part of Application Live View that is responsible for discovering the app pods running on the Kubernetes cluster and registering the instances to the Application Live View back end for it to be observed. The Application Live View connector is also responsible for proxying the actuator queries to the app pods running in the Kubernetes cluster.
Application Live View convention server	This Tanzu Application Platform component is the part of Application Live View that provides a webhook handler for the Tanzu convention controller. The webhook handler is registered with Tanzu convention controller. The webhook handler detects supply-chain workloads running a Spring Boot. Such workloads are annotated automatically to enable Application Live View to monitor them.
Application Single Sign-On (AppSSO)	This Tanzu Application Platform component provides APIs for curating and consuming a “Single Sign-On as a service” offering on Tanzu Application Platform.

Term	Definition
Aria Operations for Applications dashboard	This Tanzu Application Platform component, powered by Aria Operations for Applications (formerly Tanzu Observability), helps you monitor the health of a cluster by showing whether the deployed Tanzu Application Platform components are behaving as expected.
Artifact Metadata Repository (AMR)	AMR Observer is a set of managed controllers that watches for relevant updates on resources of interest. When relevant events are observed, a CloudEvent is generated and sent to AMR CloudEvent Handler.
Artifact Metadata Repository (AMR) CloudEvent Handler	AMR CloudEvent Handler receives CloudEvents and stores relevant information in the Artifact Metadata Repository or Metadata Store.
Artifact Metadata Repository (AMR) Observer	A set of managed controllers that watches for relevant updates on resources of interest. When relevant events are observed, a CloudEvent is generated and sent to AMR CloudEvent Handler and relayed for storage in the metadata store.
AWS Services	This Tanzu Application Platform component provides integration with AWS. Through integration with Crossplane and Services Toolkit, you can offer services from AWS to apps teams to consume with only minimal setup and configuration required from ops teams.

B

Terms beginning with “B”.

Term	Definition
Bitnami Services	This Tanzu Application Platform component provides a set of services backed by corresponding Bitnami Helm Charts. Through integration with Crossplane and Services Toolkit, these Bitnami Services are immediately ready for apps teams to consume, with no additional setup or configuration required from ops teams.
Build Cluster	A cluster with network access to your run clusters that controls the deployment on all the run clusters.
Build profile	This Tanzu Application Platform profile is intended for the transformation of source revisions to workload revisions. Specifically, hosting workloads and Supply Chains.

C

Terms beginning with “C”.

Term	Definition
Cartographer Conventions	This Tanzu Application Platform component supports defining and applying conventions to pods. It applies these conventions to developer workloads as they are deployed to the platform.
Claim	A mechanism in which requests for service instances can be declared and fulfilled without requiring detailed knowledge of the service instances themselves.
Claimable service instance	Any service instance that you can claim using a resource claim from a namespace.
Class	The common name for a service instance class.

Term	Definition
Class claim	A type of claim that references a class from which a service instance is either selected (pool-based) or provisioned (provisioner-based).
Cloud Native Runtimes	This Tanzu Application Platform component that is a serverless application runtime for Kubernetes. It is based on Knative and runs on a single Kubernetes cluster.
Component	A distinct unit or module within Tanzu Application Platform that performs a specific function or set of functions. Components can be software-based, such as services, APIs, or libraries, or they can be configuration-based, such as profiles or settings. Each component is interoperable and plays a role in the overall functionality and extensibility of Tanzu Application Platform.
Convention controller	The convention controller provides the metadata to the convention server and executes the updates to a PodTemplateSpec in accordance with convention server's requests.
Convention server	The convention server receives and evaluates metadata associated with a workload and requests updates to the PodTemplateSpec associated with that workload. You can have one or more convention servers for a single controller instance.

D

Terms beginning with “D”.

Term	Definition
Developer Conventions	This Tanzu Application Platform component enables you to configure your workloads to support live updates and debug operations.
Dynamic provisioning	A capability of Services Toolkit in which class claims that reference provisioner-based classes are fulfilled automatically through the provisioning of new service instances.

F

Terms beginning with “F”.

Term	Definition
Full profile	This profile contains most Tanzu Application Platform packages. This includes the necessary defaults for the meta-package, or parent Tanzu Application Platform package, subordinate packages, or individual child packages.

I

Terms beginning with “I”.

Term	Definition
Iterate cluster	A cluster for “inner loop” development iteration. Developers connect to the Iterate cluster by using their IDE to rapidly iterate on new software features.
Iterate profile	This Tanzu Application Platform profile is intended for iterative application development.

L

Terms beginning with “L”.

Term	Definition
Local Application Accelerator engine server	Also known as the local engine server. The local engine server allows you to serve your accelerators with their fragments on `localhost`, including any changes you have locally. You can use the VS Code Tanzu App Accelerator extension or the Tanzu CLI Accelerator plug-in to generate new projects based on these local files.
Local Source Proxy (LSP)	This Tanzu Application Platform component serves as a proxy registry server with Open Container Initiative (OCI) compatibility. Its main purpose is to handle image push requests by forwarding them to an external registry server, which is configured through `tap-values.yaml`.

M

Terms beginning with “M”.

Term	Definition
Multicluster	A setup or architecture where multiple Kubernetes clusters are used either for geographical distribution, high availability, scalability, or separation of workloads. In Tanzu Application Platform, multicluster capabilities ensure that you can deploy, manage, and operate applications and platform services across these multiple clusters seamlessly.

N

Terms beginning with “N”.

Term	Definition
Namespace Provisioner	This Tanzu Application Platform component provides an automated way for you to provision namespaces with the resources and namespace-level privileges required for your workloads to function as intended in Tanzu Application Platform.

P

Terms beginning with “P”.

Term	Definition
Package	A package contains the resources needed to install the components needed by the cluster.
Package Repository	A centralized store or registry containing packages that users can access, retrieve, and deploy. A package repository ensures that you have a consistent and curated set of software components, which are versioned and that you can fetch to deploy on your Tanzu Application Platform installation.
Pool-based class	A type of service instance class for which claims are fulfilled by selecting a service instance from a pool.
Profile	A predefined group of Tanzu Application Platform packages that you can deploy. You can deploy the full profile, which includes most Tanzu Application Platform packages, or you can deploy a profile that includes a subset of packages intended to suit a certain use case by deploying the iterate, build, run, or view profiles. The profiles allow Tanzu Application Platform to scale across an organization's multicluster, multi-cloud, or hybrid cloud infrastructure.
Provisioned service	Any service resource that defines a `.status.binding.name` that points to a secret in the same namespace that contains credentials and connectivity information for the resource.
Provisioner-based class	A type of service instance class for which claims are fulfilled by provisioning new service instances.

R

Terms beginning with “R”.

Term	Definition
Resource claim	A type of claim that references a specific service instance.
Run Cluster	The clusters that serve as your deployment environments. They can either be Tanzu Application Platform clusters, or regular Kubernetes clusters, but they must have kapp-controller and Contour installed.
Run profile	This Tanzu Application Platform profile is intended for the transformation of workload revisions to running pods. Specifically, hosting deliveries and deliverables.

S

Terms beginning with “S”.

Term	Definition
Service	A broad, high-level term that describes something used in either the development of, or running of application workloads. It is often, but not exclusively, synonymous with the concept of a backing service as defined by the Twelve Factor App.
Service binding	A mechanism in which service instance credentials and other related connectivity information are automatically communicated to application workloads.
Service Bindings	This Tanzu Application Platform component specifies a Kubernetes-wide specification for communicating service secrets to workloads in an automated way. Service Bindings provides a proprietary package of the Service Binding for Kubernetes open source project.
Service cluster	A service cluster is applicable within the context of Service API Projection and Service Resource Replication. It is a Kubernetes cluster that has Service Resource Lifecycle APIs installed and a corresponding controller managing their life cycle.
Service instance	A service instance is an abstraction over one, or a group, of interrelated service resources that together provide the functions for a particular service. One of the service resources that make up an instance must either adhere to the definition of provisioned service, or be a secret conforming to the service binding specification for Kubernetes. This guarantees that you can claim a service and subsequently bind service instances to application workloads. You make service instances discoverable through service instance classes.
Service instance class	A service instance class is more commonly called a class. Service instance classes provide a way to describe categories of service instances. They enable service instances belonging to the class to be discovered. They come in one of two varieties: pool-based or provisioner-based.
Service Registry	This Tanzu Application Platform component provides on-demand Eureka servers for your Tanzu Application Platform (commonly known as TAP) clusters. With Service Registry, you can create Eureka servers in your namespaces and bind Spring Boot workloads to them.
Service resource	A Kubernetes resource that provides some of the functions related to a Service.
Service resource life cycle API	Any Kubernetes API that you can use to manage the life cycle—create, read, update, and delete (CRUD)—of a service resource.
Services Toolkit	This Tanzu Application Platform component provides backing for service capabilities. This includes the integration of an extensive list of cloud-based and on-prem services, through to the offering and discovery of those services, and finally to the claiming and binding of service instances to application workloads.

Term	Definition
Supply Chain Choreographer	This Tanzu Application Platform component is based on open source Cartographer. It allows App Operators to create pre-approved paths to production by integrating Kubernetes resources with the elements of their existing toolchains, for example, Jenkins.
Supply Chain Security Tools - Policy	This Tanzu Application Platform component helps you ensure that the container images in your registry are not tampered with.
Supply Chain Security Tools - Scan	This Tanzu Application Platform component lets you build and deploy secure, trusted software that complies with your corporate security requirements by using scanning and gatekeeping capabilities.
Supply Chain Security Tools - Store	This Tanzu Application Platform component saves software bills of materials (SBoMs) to a database and allows you to query for image, source code, package, and vulnerability relationships.

T

Terms beginning with “T”.

Term	Definition
Tanzu Application Platform Telemetry	This is a set of objects that collect data about the use of Tanzu Application Platform and send it back to VMware for product improvements.
Tanzu Build Service	This Tanzu Application Platform component enables you to automate container creation, management, and governance at an enterprise scale. Tanzu Build Service uses the open-source Cloud Native Buildpacks project to turn application source code into container images.
Tanzu Buildpacks	Buildpacks provide framework and runtime support for applications. Use Buildpacks to determine what dependencies are required for your applications to communicate with bound services. Tanzu Buildpacks provides a proprietary package of the Paketo Buildpacks open-source project.
Tanzu CLI	A command-line tool that connects you to Tanzu.
Tanzu CLI plug-in	Tanzu CLI has a pluggable architecture. Plug-ins extend the Tanzu CLI core with additional CLI commands. Each plug-in is an executable binary that packages a group of CLI commands.
Tanzu CLI plug-in group	Tanzu CLI commands are organized into command groups. For each Tanzu Application Platform version, there is a Tanzu CLI plug-in group.
Tanzu Developer Portal	This Tanzu Application Platform component enables developers to view apps and services running for an organization, including dependencies, relationships, technical documentation, and the service status.
Tanzu Developer Portal Configurator	This tool is used for customizing Tanzu Developer Portal with Backstage plug-ins.
Tanzu Developer Tools for IntelliJ	This extension for IntelliJ IDEA helps you develop with Tanzu Application Platform and enables you to rapidly iterate on your workloads on supported Kubernetes clusters that have Tanzu Application Platform installed.
Tanzu Developer Tools for Visual Studio	This extension for Visual Studio helps you develop with Tanzu Application Platform and enables you to rapidly iterate on your workloads on supported Kubernetes clusters that have Tanzu Application Platform installed.
Tanzu Developer Tools for VS Code	This extension for Visual Studio Code helps you develop with Tanzu Application Platform and enables you to rapidly iterate on your workloads on supported Kubernetes clusters that have Tanzu Application Platform installed.

Term	Definition
Tanzu GitOps Reference Implementation (RI)	This is built upon Carvel, which shares the same packaging APIs as Tanzu Application Platform. Carvel packaging APIs support all the GitOps features and enables a native GitOps flow.
Tanzu Service CLI plug-in	A plug-in for the Tanzu CLI used by application operators and application developers to create claims for service instances.

U

Terms beginning with “U”.

Term	Definition
Unmanaged service	The services available in the Bitnami Services package where the resulting service instances run on the cluster, that is, they are not a managed service running in the cloud.

V

Terms beginning with “V”.

Term	Definition
VMware Tanzu Application Catalog	A customizable selection of trusted, pre-packaged application components that are continuously maintained and verifiably tested for use in production environments.
View Cluster	The cluster that runs the web applications for Tanzu Application Platform.
View profile	This Tanzu Application Platform profile is intended for instances of applications related to centralized developer experiences. Specifically, Tanzu Developer Portal and Metadata Store.

W

Terms beginning with “W”.

Term	Definition
Workload cluster	A cluster that has developer-created applications running on it. A workload cluster is applicable within the context of Service API Projection and Service Resource Replication.

Component documentation for Tanzu Application Platform

Tanzu Application Platform (commonly known as TAP) is a modular platform that you can enhance by installing components. Most of the Tanzu Application Platform components are documented in this section. In some cases, a component's documentation is hosted on a separate site, and you'll find a link to it in this section.

Component documentation for Tanzu Application Platform

Tanzu Application Platform (commonly known as TAP) is a modular platform that you can enhance by installing components. Most of the Tanzu Application Platform components are documented in this section. In some cases, a component's documentation is hosted on a separate site, and you'll find a link to it in this section.

Overview of Tanzu CLI

This topic tells you about the Tanzu command-line interface (commonly known as Tanzu CLI).

Tanzu CLI

The Tanzu CLI is a command-line interface that connects you to Tanzu. For example, some functions of Tanzu CLI include:

- Configure the Tanzu CLI itself
- Install and manage packages
- Run Tanzu services and components
- Create and manage application workloads

For more information about Tanzu CLI, see the VMware Tanzu CLI documentation.

Tanzu CLI architecture

The Tanzu CLI has a pluggable architecture. Plug-ins extend the Tanzu CLI core with additional CLI commands. Use the `vmware-tap/default:v1.9` plug-in group with Tanzu Application Platform. For more information, see Install Tanzu CLI plug-ins.

The `vmware-tap/default:v1.9` plug-in group consists of the following plug-ins:

```
NAME              DESCRIPTION                                                TAR
GET      VERSION
  telemetry         configure cluster-wide settings for vmware tanzu telemetry       g
lobal      v1.1.0
  accelerator       Manage accelerators in a Kubernetes cluster                      k
ubernetes  v1.7.0
  apps              Applications on Kubernetes                                       k
ubernetes  v0.12.1
  build-service     plugin to interact with tanzu build service (tbs) crds           k
```

```
ubernetes  v1.0.0
  external-secrets  interacts with external-secrets.io resources                    k
ubernetes  v0.1.0
  insight           post & query image, package, source, and vulnerability data      k
ubernetes  v1.7.0
  package           Tanzu package management                                         k
ubernetes  v0.29.0
  secret            Tanzu secret management                                          k
ubernetes  v0.29.0
  services          Commands for working with service instances, classes and claims  k
ubernetes  v0.8.0
```

Note

The Telemetry, Package, and Secret plug-ins are not Tanzu Application Platform specific, they are common to a number of other plug-in groups, for more information, see Command Groups in the Tanzu CLI Product Documentation.

Command Reference Documentation	Release Notes
Accelerator command reference	Accelerator release notes
Apps command reference	Apps release notes
Build Service command reference	Build Service release notes
External Secrets command reference	External Secrets release notes
Insight command reference	Insight release notes
Services command reference	Services release notes

Tanzu CLI installation

To install the Tanzu CLI, see Install Tanzu CLI.

Tanzu CLI command groups

Tanzu CLI commands are organized into command groups. View a list of available command groups by running:

```
tanzu
```

The list of command groups that you see depends on which CLI plug-ins are installed on your local machine.

Install new plug-ins

Install the CLI plug-ins required for Tanzu Application Platform by running:

```
tanzu plugin install --group vmware-tap/default:v1.9
```

Plug-ins for the Tanzu CLI are distributed by using a centralized plug-in repository.

The centralized plug-in repository contains:

- CLI plug-ins, which you can install individually

- Plug-in groups, which are a collection of plug-ins that you can install by using a single command

To discover and install plug-ins and plug-in groups:

1. Discover which plug-ins or plug-in groups are available for installation by running:

```
tanzu plugin search
```

```
tanzu plugin group search
```

> **Note**
>
> Include the `--show-details` flag to see all plug-in or group versions available for installation.

2. Install the latest version of a plug-in or plug-in group by running:

```
tanzu plugin install PLUG-IN-NAME
```

```
tanzu plugin install --group GROUP-NAME
```

3. Install a specific version of a plug-in or plug-in group by running:

```
tanzu plugin install PLUG-IN-NAME --version VERSION
```

```
tanzu plugin install --group GROUP-NAME:VERSION
```

Overview of Tanzu CLI

This topic tells you about the Tanzu command-line interface (commonly known as Tanzu CLI).

Tanzu CLI

The Tanzu CLI is a command-line interface that connects you to Tanzu. For example, some functions of Tanzu CLI include:

- Configure the Tanzu CLI itself
- Install and manage packages
- Run Tanzu services and components
- Create and manage application workloads

For more information about Tanzu CLI, see the VMware Tanzu CLI documentation.

Tanzu CLI architecture

The Tanzu CLI has a pluggable architecture. Plug-ins extend the Tanzu CLI core with additional CLI commands. Use the `vmware-tap/default:v1.9` plug-in group with Tanzu Application Platform. For more information, see Install Tanzu CLI plug-ins.

The `vmware-tap/default:v1.9` plug-in group consists of the following plug-ins:

```
NAME              DESCRIPTION                                                 TAR
GET      VERSION
```

```
  telemetry          configure cluster-wide settings for vmware tanzu telemetry       g
lobal      v1.1.0
  accelerator        Manage accelerators in a Kubernetes cluster                      k
ubernetes  v1.7.0
  apps               Applications on Kubernetes                                       k
ubernetes  v0.12.1
  build-service      plugin to interact with tanzu build service (tbs) crds           k
ubernetes  v1.0.0
  external-secrets   interacts with external-secrets.io resources                     k
ubernetes  v0.1.0
  insight            post & query image, package, source, and vulnerability data      k
ubernetes  v1.7.0
  package            Tanzu package management                                         k
ubernetes  v0.29.0
  secret             Tanzu secret management                                          k
ubernetes  v0.29.0
  services           Commands for working with service instances, classes and claims  k
ubernetes  v0.8.0
```

Note

The Telemetry, Package, and Secret plug-ins are not Tanzu Application Platform specific, they are common to a number of other plug-in groups, for more information, see Command Groups in the Tanzu CLI Product Documentation.

Command Reference Documentation	Release Notes
Accelerator command reference	Accelerator release notes
Apps command reference	Apps release notes
Build Service command reference	Build Service release notes
External Secrets command reference	External Secrets release notes
Insight command reference	Insight release notes
Services command reference	Services release notes

Tanzu CLI installation

To install the Tanzu CLI, see Install Tanzu CLI.

Tanzu CLI command groups

Tanzu CLI commands are organized into command groups. View a list of available command groups by running:

```
tanzu
```

The list of command groups that you see depends on which CLI plug-ins are installed on your local machine.

Install new plug-ins

Install the CLI plug-ins required for Tanzu Application Platform by running:

```
tanzu plugin install --group vmware-tap/default:v1.9
```

Plug-ins for the Tanzu CLI are distributed by using a centralized plug-in repository.

The centralized plug-in repository contains:

- CLI plug-ins, which you can install individually
- Plug-in groups, which are a collection of plug-ins that you can install by using a single command

To discover and install plug-ins and plug-in groups:

1. Discover which plug-ins or plug-in groups are available for installation by running:

   ```
   tanzu plugin search
   ```

   ```
   tanzu plugin group search
   ```

   Note

   Include the `--show-details` flag to see all plug-in or group versions available for installation.

2. Install the latest version of a plug-in or plug-in group by running:

   ```
   tanzu plugin install PLUG-IN-NAME
   ```

   ```
   tanzu plugin install --group GROUP-NAME
   ```

3. Install a specific version of a plug-in or plug-in group by running:

   ```
   tanzu plugin install PLUG-IN-NAME --version VERSION
   ```

   ```
   tanzu plugin install --group GROUP-NAME:VERSION
   ```

Overview of Tanzu CLI plug-ins

The topics in this section tell you about the Tanzu CLI plug-ins in Tanzu Application Platform (commonly known as TAP).

- accelerator - The Application Accelerator Tanzu CLI plug-in includes commands for developers and operators to create and use accelerators.
- apps - This Tanzu Apps CLI plug-in provides the ability to create, view, update, and delete application workloads on any Kubernetes cluster that has the Tanzu Application Platform components installed.
- build-service - The Tanzu Build Service CLI plug-in provides the ability to view all the Tanzu Build Service resources on any Kubernetes cluster.
- insight - The Tanzu Insight CLI plug-in enables querying vulnerability, image, and package data.

Overview of Tanzu Apps CLI plug-in

This topic gives you an overview of the Tanzu Apps CLI plug-in. Use the Tanzu Apps CLI plug-in to create, view, update, and delete application workloads on any Kubernetes cluster that has the Tanzu Application Platform (commonly known as TAP) components installed.

About workloads

Tanzu Application Platform enables you to quickly build and test applications regardless of your familiarity with Kubernetes. You can turn source code into a workload that runs in a container with a URL. A workload enables you to choose application specifications, such as repository location, environment variables, and service binding.

Tanzu Application Platform can support a range of workloads, including a serverless process that starts on demand, a constellation of microservices that functions as a logical application, or a small hello-world test application.

For information about installing the Tanzu Apps CLI plug-in, see Install Tanzu Apps CLI.

Overview of Tanzu Apps CLI plug-in

This topic gives you an overview of the Tanzu Apps CLI plug-in. Use the Tanzu Apps CLI plug-in to create, view, update, and delete application workloads on any Kubernetes cluster that has the Tanzu Application Platform (commonly known as TAP) components installed.

About workloads

Tanzu Application Platform enables you to quickly build and test applications regardless of your familiarity with Kubernetes. You can turn source code into a workload that runs in a container with a URL. A workload enables you to choose application specifications, such as repository location, environment variables, and service binding.

Tanzu Application Platform can support a range of workloads, including a serverless process that starts on demand, a constellation of microservices that functions as a logical application, or a small hello-world test application.

For information about installing the Tanzu Apps CLI plug-in, see Install Tanzu Apps CLI.

Get started with Apps CLI plug-in

The Getting Started section contains the following topics:

- Install the Apps CLI plug-in
- Configure the Apps CLI plug-in

Install Tanzu Apps CLI plug-in

This topic tells you how to install the Tanzu Apps CLI plug-in on Tanzu Application Platform (commonly known as TAP).

Prerequisites

Ensure that you installed or updated the core Tanzu CLI. For more information, see Install Tanzu CLI.

Install Tanzu Apps CLI plug-in

Run:

```
tanzu plugin install apps --group vmware-tap/default:v1.9
```

Verify that the plug-in is installed correctly:

```
tanzu apps version
# sample output
v0.12.1
```

Uninstall Tanzu Apps CLI

Run:

```
tanzu plugin delete apps
```

Configure the Tanzu Apps CLI plug-in

This topic tells you how to configure the Tanzu Apps CLI plug-in on Tanzu Application Platform (commonly known as TAP).

Changing clusters with –context

The Tanzu Apps CLI plug-in references the default kubeconfig file to access a Kubernetes cluster. When you run a `tanzu apps` command, the Tanzu Apps CLI plug-in uses the default context. The default context is defined in the kubeconfig file located by default in: `HOME/.kube/config`.

There are two ways to change the target cluster:

1. Use `kubectl config use-context CONTEXT-NAME` to change the default context. All subsequent `tanzu apps` commands target the cluster defined in the new default kubeconfig context.
2. Include the `--context CONTEXT-NAME` flag when running any `tanzu apps` command.

Note

Any subsequent `tanzu apps` commands that do not include the `--context CONTENT-NAME` flag continue to use the default context set in the kubeconfig.

Overriding the default kubeconfig

There are two approaches to overriding the default kubeconfig:

1. To change the kubeconfig the Tanzu Apps CLI will reference, set the environment variable `KUBECONFIG=PATH` . All subsequent `tanzu apps` commands reference the non-default kubeconfig assigned to the environment variable.
2. Include the `--kubeconfig path` flag when running any `tanzu apps` command.

Note

Any subsequent `tanzu apps` commands that do not include the `--context CONTEXT-NAME` flag continue to use the default context set in the kubeconfig.

For more information about kubeconfig, see the Kubernetes documentation.

Suppressing color with –no-color flag

Most Apps Plug-in commands have emojis and colored output. In some cases, color, emojis, and other characters are not needed, such as for automated scripting, or where the misinterpretation of these features by a terminal could result in a poor user experience. Use the `--no-color` flag to suppress color, emojis, and animation.

The following example creates a workload with code from `--local-path`. The `--no-color` flag suppresses the emojis and animated upload progress bar:

```
tanzu apps workload apply my-workload --local-path path/to/my/source --type web --no-c
olor
The files and/or directories listed in the .tanzuignore file are being excluded from t
he uploaded source code.
Publishing source in "path/to/my/source" to "local-source-proxy.tap-local-source-syste
m.svc.cluster.local/source:default-my-workload""...
Published source

Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  annotations:
      6 + |    local-source-proxy.apps.tanzu.vmware.com: registry.io/project/source:de
fault-my-workload@sha256:447db92e289dbe3a6969521917496ff2b6b0a1d6fbff1beec3af726430ce8
493
      7 + |  labels:
      8 + |    apps.tanzu.vmware.com/workload-type: web
      9 + |  name: my-workload
     10 + |  namespace: default
     11 + |spec:
     12 + |  source:
     13 + |    image: registry.io/project/source:default-my-workload@sha256:447db92e28
9dbe3a6969521917496ff2b6b0a1d6fbff1beec3af726430ce8493
? Do you want to create this workload? [yN]:
```

Persist the suppression of color, emojis, and animation across commands by setting the `NO_COLOR` environment variable.

```
export NO_COLOR=true
```

`tanzuignore` file

Use the optional `.tanzuignore` file at the root of your project directory to indicate which files or directories in your project are not required to build or run your application such as `README.md`, .git, or docs. Including these files in your `.tanzuignore` provides the following benefits:

1. Items included in the `.tanzuignore` file are not uploaded when you upload your source code. This helps to avoid unnecessary consumption of resources.
2. When iterating on code with the `--live-update` flag enabled, changes to directories or files listed in the `.tanzuignore` file do not trigger the automatic re-deployment of source code.

The following are some guidelines for the `.tanzuignore` file:

- The `.tanzuignore` file should include a reference to itself, as it provides no value when deployed.
- Directories must not end with the system separator `/`, or `\`.
- Add comments using hashtag `#`.
- If the `.tanzuignore` file contains files or directories that are not found in the source code, they are ignored.

Example of a .tanzuignore file

```
.tanzuignore # must contain itself in order to be ignored
# This is a comment
this/is/a/folder/to/exclude

this-is-a-file.ext
```

Registry flags and environment variables

You can either trust a custom certificate on a system or pass the path to the certificate via flags. Environment variables can also be set to avoid having to pass the flags and values for every command incantation.

Below is a list of each of the flags with the corresponding environment variable equivalent in parenthesis:

- `--registry-ca-cert`: This is the path of the self-signed certificate needed for the custom or private registry (`TANZU_APPS_REGISTRY_CA_CERT`).
- `--registry-password`: Use this when the registry requires credentials to push (`TANZU_APPS_REGISTRY_PASSWORD`).
- `--registry-username`: Use with `--registry-password` to set the registry credentials (`TANZU_APPS_REGISTRY_USERNAME`).
- `--registry-token`: Set when the registry authentication is done through a token (`TANZU_APPS_REGISTRY_TOKEN`).

For example:

```
tanzu apps workload apply WORKLOAD \
--local-path PATH-TO-REPO --source-image registry.url.nip.io/PACKAGE/IMAGE \
--type web --registry-ca-cert path/to/ca/cert.nip.io.crt \
--registry-username USERNAME \
--registry-password PASSWORD
```

Alternatively, run as:

```
export TANZU_APPS_REGISTRY_CA_CERT=path/to/ca/cert.nip.io.crt
export TANZU_APPS_REGISTRY_PASSWORD=USERNAME
export TANZU_APPS_REGISTRY_USERNAME=PASSWORD

tanzu apps workload apply WORKLOAD \
--local-path PATH-TO-REPO \
--source-image registry.url.nip.io/PACKAGE/IMAGE
```

Use the `--type` flag to specify the type of workload. Persist the workload type value across commands by setting the `TANZU_APPS_TYPE` environment variable. The default value of `web` is set automatically if no `--type` flag or `TANZU_APPS_TYPE` value is provided.

```
export TANZU_APPS_TYPE=server
```

Autocompletion

The Tanzu Apps CLI plug-in provides auto-completion support for commands, positional arguments, flags, and flag values.

Add one of the following commands to the shell config file according to your setup:

Bash

```
tanzu completion bash >  HOME/.tanzu/completion.bash.inc
```

Zsh

```
echo "autoload -U compinit; compinit" >> ~/.zshrc
tanzu completion zsh > "${fpath[1]}/_tanzu"
```

Tanzu Apps CLI plug-in dependency matrix

This topic tells you which versions of Cartographer are supported in Tanzu Application Platform (commonly known as TAP).

Tanzu Application Platform Version	Apps CLI Version	Required Cartographer Version
v1.3.x	v0.9.x	v0.5.x or later
v1.4.x	v0.10.x	v0.6.x or later
v1.5.x	v0.11.x	v0.7.x or later
v1.6.x	v0.12.x	v0.7.x or later

Check Cartographer version

To see the Cartographer version installed in the cluster, run:

```
kubectl get -n tap-install packageinstalls.packaging.carvel.dev cartographer
```

Tanzu Apps CLI plug-in tutorials

This section contains the following topics:

- Create/Update a Workload
- Get Workload Status
- Delete a workload

Create or update a workload

This topic tells you how to create a workload in Tanzu Application Platform (commonly known as TAP) from a `workload.yaml` file, a URL, a Git source, a local source, and a pre-built image.

For more information about the different types of workload creation, see Supply Chain How-to-guides.

Create a workload from a `workload.yaml` file or from a URL

You can create a workload from a `workload.yaml` file or from a URL.

Create a workload from a YAML file

In many cases, workload life cycles are managed through CLI commands. However, there might be cases where managing the workload through direct interactions and edits of a `yaml` file is preferred. The Apps CLI plug-in supports using `yaml` files to meet the requirements.

When a workload is managed using a `yaml` file, that file must contain a single workload definition.

For example, a valid file looks similar to the following example:

```
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: tanzu-java-web-app
  labels:
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/workload-type: web
spec:
  source:
    git:
      url: https://github.com/vmware-tanzu/application-accelerator-samples
      ref:
        tag: tap-1.6.0
    subPath: tanzu-java-web-app
```

To create a workload from a file like the earlier example:

```
tanzu apps workload create --file my-workload-file.yaml
```

Note

When flags are passed in combination with `--file my-workload-file.yaml` the flag values take precedence over the associated property or values in the YAML.

Create a workload from stdin

The workload `yaml` definition can also be passed in through stdin as follows:

```
tanzu apps workload create --file - --yes
```

The console waits for input, and the content with valid `yaml` definitions for a workload can either be written or pasted. Then click `Ctrl-D` three times to start the workload creation. This can also be done with the `workload apply` command.

To pass a workload through `stdin`, the `--yes` flag is required. If not provided, the command fails.

```
tanzu apps workload create -f -y
  WARNING: Configuration file update strategy is changing. By default, provided config
uration files will replace rather than merge existing configuration. The change will t
ake place in the January 2024 TAP release (use "--update-strategy" to control strategy
explicitly).

---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: tanzu-java-web-app
  labels:
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/workload-type: web
spec:
  source:
    git:
      url: https://github.com/vmware-tanzu/application-accelerator-samples
      ref:
        tag: tap-1.6.0
```

```
    subPath: tanzu-java-web-app
  Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    app.kubernetes.io/part-of: tanzu-java-web-app
      7 + |    apps.tanzu.vmware.com/workload-type: server
      8 + |  name: tanzu-java-web-app
      9 + |  namespace: default
     10 + |spec:
     11 + |  source:
     12 + |    git:
     13 + |      ref:
     14 + |        tag: tap-1.6.0
     15 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
     16 + |    subPath: tanzu-java-web-app
  Created workload "tanzu-java-web-app"

To see logs:   "tanzu apps workload tail tanzu-java-web-app --timestamp --since 1h"
To get status: "tanzu apps workload get tanzu-java-web-app"
```

Create a workload from URL

Another way to pass a workload with the `--file` flag is using a URL, which must contain a raw file with the workload definition.

Create workload from Git source

Use the `--git-repo`, `--git-branch`, `--git-tag`, and `--git-commit` flags to create a workload from an existing Git repository. This allows the supply chain to get the source from the given repository to deploy the application.

To create a named workload and specify a Git source code location, run:

```
tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanz
u/application-accelerator-samples --sub-path tanzu-java-web-app --git-tag tap-1.6.0 --
type web
```

Respond `Y` to prompts to complete process.

Where:

- `tanzu-java-web-app` is the name of the workload.
- `--git-repo` is the location of the code to build the workload from.
- `--sub-path` (optional) is the relative path inside the repository to treat as application root.
- `--git-tag` (optional) specifies which tag in the repository to pull the code from.
- `--git-branch` (optional) specifies which branch in the repository to pull the code from.
- `--type` distinguishes the workload type.

This process can also be done with non-publicly accessible repositories. These require authentication using credentials stored in a Kubernetes secret. The supply chain is in charge of managing these credentials.

View the full list of supported workload configuration options by running `tanzu apps workload apply --help`.

Unset Git fields

There are various ways to update a workload. Use flags to change workload fields. Use a YAML file with the required changes, and run the `tanzu apps workload apply` with the `--update-strategy` set as `replace`. For more information, see Control Workload Merge Behavior.

To delete fields, set the `--git-*` flags as empty strings within the command. This removes the `workload.spec.source.git` fields.

For example, for a workload that includes specifications such as `--git-tag`, `--git-commit` or `--git-branch`, remove these by setting them as empty strings in the command as shown in the following example:

```
# existing workload definition
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  labels:
    apps.tanzu.vmware.com/workload-type: web
  name: tanzu-java-web-app
  namespace: default
spec:
  source:
    git:
      ref:
        branch: main
        tag: tap-1.6.0
      url: https://github.com/vmware-tanzu/application-accelerator-samples
    subPath: tanzu-java-web-app
```

```
# Update the workload to remove one of its git fields
tanzu apps workload apply tanzu-java-web-app --git-branch ""
  Update workload:
...
  9,  9   |spec:
 10, 10   |  source:
 11, 11   |    git:
 12, 12   |      ref:
 13     - |        branch: main
 14, 13   |        tag: tap-1.6.0
 15, 14   |      url: https://github.com/vmware-tanzu/application-accelerator-samples
 16, 15   |    subPath: tanzu-java-web-app
  Really update the workload "tanzu-java-web-app"? [yN]: y
  Updated workload "tanzu-java-web-app"

To see logs:   "tanzu apps workload tail tanzu-java-web-app --timestamp --since 1h"
To get status: "tanzu apps workload get tanzu-java-web-app"

# Export the workload to see that `spec.source.git.ref.tag` is not part of the definit
ion
tanzu apps workload get tanzu-java-web-app --export
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  labels:
    apps.tanzu.vmware.com/workload-type: web
  name: tanzu-java-web-app
  namespace: default
spec:
  source:
    git:
      ref:
        tag: tap-1.6.0
```

```
      url: https://github.com/vmware-tanzu/application-accelerator-samples
    subPath: tanzu-java-web-app
```

Note

If `--git-repo` is set to empty, then the whole Git section is removed from the workload definition.

```
tanzu apps workload apply tanzu-java-web-app --git-repo ""
  Update workload:
...
  5,  5   |  labels:
  6,  6   |    apps.tanzu.vmware.com/workload-type: web
  7,  7   |  name: tanzu-java-web-app
  8,  8   |  namespace: default
  9      - |spec:
 10      - |  source:
 11      - |    git:
 12      - |      ref:
 13      - |        tag: tap-1.6.0
 14      - |      url: https://github.com/vmware-tanzu/application-accelerator-samples
 15      - |    subPath: tanzu-java-web-app
       9 + |spec: {}
  NOTICE: no source code or image has been specified for this workload.
  Really update the workload "tanzu-java-web-app"? [yN]: y
  Updated workload "tanzu-java-web-app"

To see logs:   "tanzu apps workload tail tanzu-java-web-app --timestamp --since 1h"
To get status: "tanzu apps workload get tanzu-java-web-app"

# Export the workload and check that the git source section does not exist
tanzu apps workload get tanzu-java-web-app --export
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  labels:
    apps.tanzu.vmware.com/workload-type: web
  name: tanzu-java-web-app
  namespace: default
spec: {}
```

Subpath

Use the `--subpath` flag to create workloads within a repository, where the repository, such as a monorepo, consists of multiple folders or projects.

```
# The Git repository for this sample contains several applications, each on its own fo
lder
tanzu apps workload apply tanzu-where-for-dinner --git-repo https://github.com/vmware-
tanzu/application-accelerator-samples --git-branch main --sub-path where-for-dinner
  Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    apps.tanzu.vmware.com/workload-type: web
      7 + |  name: tanzu-where-for-dinner
      8 + |  namespace: default
      9 + |spec:
```

```
     10 + |  source:
     11 + |    git:
     12 + |      ref:
     13 + |        branch: main
     14 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
     15 + |    subPath: where-for-dinner
  Do you want to create this workload? [yN]: y
  Created workload "tanzu-where-for-dinner"

To see logs:   "tanzu apps workload tail tanzu-where-for-dinner --timestamp --since 1
h"
To get status: "tanzu apps workload get tanzu-where-for-dinner"
```

Create a workload from Local Source

There are multiple ways to upload local source code to a Tanzu Application Platform cluster.

Using Local Source Proxy

Use Local Source Proxy to push local source code to the registry configured during Tanzu Application Platform installation.

For more information, see Install Local Source Proxy. To create a workload that pushes to an already configured registry through Local Source Proxy, use `--local-path` flag without `--source-image`, like the following example:

```
# Point the local path flag to the folder containing the source code
tanzu apps workload create tanzu-java-web-app --local-path /path/to/java/app

The files and/ directories listed in the .tanzuignore file are being excluded from t
he uploaded source code.
Publishing source in "/path/to/java/app" to "local-source-proxy.tap-local-source-sys
tem.svc.cluster.local/source:default-tanzu-java-web-app"...
  Published source

  Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  annotations:
      6 + |    local-source-proxy.apps.tanzu.vmware.com: registry.io/project/source:
default-tanzu-java-web-app@sha256:447db92e289dbe3a6969521917496ff2b6b0a1d6fbff1beec3
af726430ce8493
      7 + |  labels:
      8 + |    apps.tanzu.vmware.com/workload-type: web
      9 + |  name: tanzu-java-web-app
     10 + |  namespace: default
     11 + |spec:
     12 + |  source:
     13 + |    image: registry.io/project/source:default-tanzu-java-web-app@sha256:4
47db92e289dbe3a6969521917496ff2b6b0a1d6fbff1beec3af726430ce8493
  Do you want to create this workload? [yN]:
```

Note

A workload created using Local Source Proxy is easily recognizable because it has the `local-source-proxy.apps.tanzu.vmware.com` annotation with a value the same as the `spec.source.image` field.

Using Source Image

If the Local Source Proxy component is not installed, upload your local source code to a registry of your choice by passing in the `--source-image` flag. Use this flag to specify the registry path where the local source code is uploaded as an image. Both the cluster and the developer's machine must be configured to properly provide credentials for accessing the container image registry where the local source code is published to. For more information about authentication requirements, see Building from Local Source. To create a workload using a source image, use `--local-path` flag with `--source-image`, like the following example:

```
tanzu apps workload create tanzu-java-web-app --local-path /path/to/java/app --sourc
e-image registry.io/path/to/project/image-name

The files and directories listed in the .tanzuignore file are being excluded from th
e uploaded source code.
Publishing source in "." to "registry.io/path/to/project/image-name"...
  Published source

  Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    apps.tanzu.vmware.com/workload-type: web
      7 + |  name: tanzu-java-web-app
      8 + |  namespace: default
      9 + |spec:
     10 + |  source:
     11 + |    image: registry.io/path/to/project/image-name:latest@sha256:447db92e28
9dbe3a6969521917496ff2b6b0a1d6fbff1beec3af726430ce8493
  Do you want to create this workload? [yN]:
```

`--live-update`

Use the `--live-update` flag to ensure that local source code changes are reflected quickly on the running workload. This is particularly valuable when iterating on features that require the workload to be deployed and running to validate.

Live update is ideally situated for running from within one of our supported IDE extensions, but it can also be utilized independently as shown in the following Spring Boot application example:

Spring Boot application example

Prerequisites: Tilt must be installed on the client.

1. Clone the repository by running:

   ```
   git clone https://github.com/vmware-tanzu/application-accelerator-samples
   ```

2. Change into the `tanzu-java-web-app` directory.
3. In `Tiltfile`, first, change the `SOURCE_IMAGE` variable to use your registry and project.
4. At the very end of the file add:

   ```
   allow_k8s_contexts('your-cluster-name')
   ```

5. Inside the directory, run:

   ```
   tanzu apps workload apply tanzu-java-web-app --live-update --local-path . -s
   gcr.io/PROJECT/tanzu-java-web-app-live-update -y
   ```

Expected output:

```
The files and directories listed in the .tanzuignore file are being excluded fr
om the uploaded source code.
Publishing source in "." to "gcr.io/PROJECT/tanzu-java-web-app-live-update"...
  Published source

  Create workload:
   1 + |---
   2 + |apiVersion: carto.run/v1alpha1
   3 + |kind: Workload
   4 + |metadata:
   5 + |  name: tanzu-java-web-app
   6 + |  namespace: default
   7 + |spec:
   8 + |  params:
   9 + |  - name: live-update
  10 + |    value: "true"
  11 + |  source:
  12 + |    image: gcr.io/PROJECT/tanzu-java-web-app-live-update:latest@sha256:
3c9fd738492a23ac532a709301fcf0c9aa2a8761b2b9347bdbab52ce9404264b
  Created workload "tanzu-java-web-app"

To see logs:   "tanzu apps workload tail tanzu-java-web-app --timestamp --since
1h"
To get status: "tanzu apps workload get tanzu-java-web-app"
```

6. Run Tilt to deploy the workload.

```
tilt up

Tilt started on http://localhost:10350/
v0.23.6, built 2022-01-14

(space) to open the browser
(s) to stream logs (--stream=true)
(t) to open legacy terminal mode (--legacy=true)
(ctrl-c) to exit
Tilt started on http://localhost:10350/
v0.23.6, built 2022-01-14

Initial Build • (Tiltfile)
Loading Tiltfile at: /path/to/repo/tanzu-java-web-app/Tiltfile
Successfully loaded Tiltfile (1.500809ms)
tanzu-java-w… │
tanzu-java-w… │ Initial Build • tanzu-java-web-app
tanzu-java-w… │ WARNING: Live Update failed with unexpected error:
tanzu-java-w… │   Cannot extract live updates on this build graph structure
tanzu-java-w… │ Falling back to a full image build + deploy
tanzu-java-w… │ STEP 1/1 — Deploying
tanzu-java-w… │      Objects applied to cluster:
tanzu-java-w… │        → tanzu-java-web-app:workload
tanzu-java-w… │
tanzu-java-w… │      Step 1 - 8.87s (Deploying)
tanzu-java-w… │      DONE IN: 8.87s
tanzu-java-w… │
tanzu-java-w… │
tanzu-java-w… │ Tracking new pod rollout (tanzu-java-web-app-build-1-build-po
d):
tanzu-java-w… │      ┊ Scheduled       - (…) Pending
tanzu-java-w… │      ┊ Initialized     - (…) Pending
tanzu-java-w… │      ┊ Ready           - (…) Pending
...
```

Subpath

For local source workloads, specify a subpath. A subpath points to a specific subfolder within the root folder.

```
# After cloning repo in https://github.com/vmware-tanzu/application-accelerator-sample
s and install Local Source Proxy

cd application-accelerator-samples
tanzu apps workload apply tanzu-java-web-app --local-path . --sub-path tanzu-java-web-
app
Publishing source in "." to "local-source-proxy.tap-local-source-system.svc.cluster.lo
cal/source:default-tanzu-java-web-app"...
  Published source

  Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  annotations:
      6 + |    local-source-proxy.apps.tanzu.vmware.com: gcr.io/tanzu-framework-playgr
ound/source:default-tanzu-java-web-app@sha256:e6ee774bc427273afb6dcf6388aca8edd83b83c7
2e4de00bf0cf8dfce72f8446
      7 + |  labels:
      8 + |    apps.tanzu.vmware.com/workload-type: web
      9 + |  name: tanzu-java-web-app
     10 + |  namespace: default
     11 + |spec:
     12 + |  source:
     13 + |    image: gcr.io/tanzu-framework-playground/source:default-tanzu-java-web-
app@sha256:e6ee774bc427273afb6dcf6388aca8edd83b83c72e4de00bf0cf8dfce72f8446
     14 + |    subPath: tanzu-java-web-app
  Do you want to create this workload? [yN]: y
  Created workload "tanzu-java-web-app"

To see logs:   "tanzu apps workload tail tanzu-java-web-app --timestamp --since 1h"
To get status: "tanzu apps workload get tanzu-java-web-app"
```

Create a workload from a pre-built image

Create a workload from an existing registry image by providing the reference to that image through the `--image` flag. The supply chain references the provided registry image when the workload is deployed.

For example:

```
tanzu apps workload create petclinic-image --image springcommunity/spring-framework-pe
tclinic
  Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    apps.tanzu.vmware.com/workload-type: web
      7 + |  name: petclinic-image
      8 + |  namespace: default
      9 + |spec:
     10 + |  image: springcommunity/spring-framework-petclinic
  Do you want to create this workload? [yN]:
```

For information about requirements for prebuilt images and how to configure prebuilt images in a supply chains, see Use an existing image with Supply Chain Choreographer.

Create a workload from a Maven repository artifact

Create a workload from a Maven repository artifact by setting some specific properties as YAML parameters or using `--maven-*` flags in the workload when using the supply chain. For more information about Maven repository artifact, see Source-Controller.

The Maven repository URL is set when the supply chain is created.

Maven workload created with `--maven-*` flags

To create a Maven workload using the CLI provided flags, run:

```
tanzu apps workload apply my-workload \
      --maven-artifact hello-world \
      --maven-type jar
      --maven-version 0.0.1 \
      --maven-group carto.run \
      --type web -y
```

For information about available commands, see the Tanzu CLI Command Reference documentation.

Maven workload created with YAML or JSON parameters

- Param name: maven
- Param value:
    - YAML:

    ```
    artifactId: ...
    type: ... # default jar if not provided
    version: ...
    groupId: ...
    ```

    - JSON:

    ```
    {
        "artifactId": ...,
        "type": ..., // default jar if not provided
        "version": ...,
        "groupId": ...
    }
    ```

For example, to create a workload from a Maven artifact using parameters, run:

```
# YAML
tanzu apps workload create my-workload --type web --param-yaml maven=$"artifactId:hell
o-world\ntype:jar\nversion:0.0.1\ngroupId:carto.run"

# JSON
tanzu apps workload create my-workload --type web --param-yaml maven="{"artifactId":"h
ello-world", "type": "jar", "version": "0.0.1", "groupId": "carto.run"}"
```

For information about how to configure the Maven artifact authentication credentials, see Maven Repository Secret.

Create a workload from a Dockerfile

For any source-based supply chains, when you specify the new dockerfile parameter in a workload, the builds switch from using Kpack to using kaniko. Source-based supply chains are supply chains that don't take a pre-built image. kaniko is an open-source tool for building container images from a Dockerfile without running Docker inside a container. For more information, see Dockerfile-based builds.

Get workload status

This topic tells you about the Tanzu Apps CLI plug-in commands you can use to get information about the status of a workload.

tanzu apps workload list

The `tanzu apps workload list` command gets a list of the workloads present in the cluster, either in the current namespace, in another namespace, or all namespaces.

Filter the list result by the `--namespace`, `--type` or `--app` flags. The `--output` flag supports the `yaml` and `json` formats.

–all-namespaces, -A flag

Show workloads in all namespaces in the cluster.

```
tanzu apps workload list -A

NAMESPACE   TYPE   NAME                  APP                  READY
AGE
default     web    nginx4                <empty>              Ready
7d9h
default     web    rmq-sample-app        <empty>              Ready
179m
default     web    spring-pet-clinic     <empty>              Unknown
3h1m
default     web    tanzu-java-web-app    tanzu-java-web-app   Ready
40m
nginx-ns    web    nginx2                <empty>              TemplateRejectedByAPISer
ver   8d
```

–app flag

Filter workloads by application. Shows workloads for the application specified in the command.

```
tanzu apps workload list --app spring-petclinic

NAME                TYPE   READY     AGE
spring-petclinic2   web    Unknown   29d
spring-petclinic3   web    Ready     29d
```

–namespace, -n flag

Filter workloads by namespace. Lists all the workloads present in the specified namespace.

```
tanzu apps workload list --namespace my-namespace

NAME   TYPE   APP       READY                          AGE
app1   web    <empty>   TemplateRejectedByAPIServer   8d
app2   web    <empty>   Ready                          8d
app3   web    <empty>   Unknown                        8d
```

–output, -o flag

List all workloads in the specified namespace in `yaml`, `yml` or `json` format.

- yaml/yaml

```
# shorthand for --output is -o
# alternatives are
# tanzu apps workload list -o yaml / -o json
# tanzu apps workload list -oyaml / -ojson
tanzu apps workload list --output yaml
```

```
---
- apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
    creationTimestamp: "2022-05-17T22:06:49Z"
    generation: 1
    labels:
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/workload-type: web
    managedFields:
    ...
    ...
    manager: cartographer
    operation: Update
    time: "2022-05-17T22:06:52Z"
name: tanzu-java-web-app2
namespace: default
resourceVersion: "6071972"
uid: 7fbcd40d-4eb3-41dc-a1db-657b64148708
spec:
    source:
        git:
            ref:
              tag: tap-1.3
            url: https://github.com/vmware-tanzu/application-accelerator-sample
s
        subPath: tanzu-java-web-app
...
...
---
- apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
    creationTimestamp: "2022-05-17T22:06:49Z"
    generation: 1
    labels:
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/workload-type: web
    managedFields:
    ...
    ...
    manager: cartographer
    operation: Update
    time: "2022-05-17T22:06:52Z"
name: tanzu-java-web-app
namespace: default
resourceVersion: "6071972"
uid: 7fbcd40d-4eb3-41dc-a1db-657b64148708
spec:
    source:
        git:
            ref:
              tag: tap-1.3
```

```
            url: https://github.com/vmware-tanzu/application-accelerator-sample
s
        subPath: tanzu-java-web-app
...
...
```

- json

```
tanzu apps workload list --output json
```

```
[
    {
        "kind": "Workload",
        "apiVersion": "carto.run/v1alpha1",
        "metadata": {
            "name": "tanzu-java-web-app2",
            "namespace": "default",
            "uid": "7fbcd40d-4eb3-41dc-a1db-657b64148708",
            "resourceVersion": "6071972",
            "generation": 1,
            "creationTimestamp": "2022-05-17T22:06:49Z",
            "labels": {
                "app.kubernetes.io/part-of": "tanzu-java-web-app",
                "apps.tanzu.vmware.com/workload-type": "web"
            },
        ...
        }
    ...
    },
    {
        "kind": "Workload",
        "apiVersion": "carto.run/v1alpha1",
        "metadata": {
            "name": "tanzu-java-web-app",
            "namespace": "default",
            "uid": "7fbcd40d-4eb3-41dc-a1db-657b64148708",
            "resourceVersion": "6071972",
            "generation": 1,
            "creationTimestamp": "2022-05-17T22:06:49Z",
            "labels": {
                "app.kubernetes.io/part-of": "tanzu-java-web-app",
                "apps.tanzu.vmware.com/workload-type": "web"
            },
        ...
        }
    ...
    },
...
...
]
```

tanzu apps workload get

The `tanzu apps workload get` command provides detailed information and status about a workload. Filter workloads with the `--namespace` flag, which specifies the namespace where the workload is deployed.

There are multiple sections in the workload get command output. The following data is displayed:

- Name of the workload and its status.
- Displays source information of workload.
- If the workload was matched with a supply chain, the name of the supply chain is provided.

- A table providing the name, status, health, and resulting resource for each step defined in the supply chain or delivery for the workload.
- If there are any issues, such as errors or status messages associated with any of the steps defined by the supply chain or delivery for the workload, the name and corresponding message are included in the `Messages` section.
- Workload related resource information and status like services claims, related pods, and Knative services.

At the very end of the command output, a hint to follow up commands is also displayed.

Note

The `Supply Chain` and `Delivery` sections are included in the command output depending on whether those resources are present on the target cluster, for example, if the target includes only build components, there is no `Delivery` resources available and therefore the `Delivery` section is not included in the command output.

```
# --namespace flag shorthand is -n
# An alternative way to use this command is
# tanzu apps workload get tanzu-java-web-app -n development
tanzu apps workload get tanzu-java-web-app --namespace development

  Overview
   name:         tanzu-java-web-app
   type:         web
   namespace:    development

  Source
   type:       git
   url:        https://github.com/vmware-tanzu/application-accelerator-samples
   tag:        tap-1.6.0
   sub-path:   tanzu-java-web-app
   revision:   tap-1.6.0/bb9e655b33e39b242b6e5d9e218578e75c1d833c

  Supply Chain
   name:   source-to-url

   NAME                READY   HEALTHY   UPDATED   RESOURCE
   source-provider     True    True      31m       gitrepositories.source.toolkit.fluxc
d.io/tanzu-java-web-app
   image-provider      True    True      30m       images.kpack.io/tanzu-java-web-app
   config-provider     True    True      30m       podintents.conventions.carto.run/tan
zu-java-web-app
   app-config          True    True      30m       configmaps/tanzu-java-web-app
   service-bindings    True    True      30m       configmaps/tanzu-java-web-app-with-c
laims
   api-descriptors     True    True      30m       configmaps/tanzu-java-web-app-with-a
pi-descriptors
   config-writer       True    True      30m       runnables.carto.run/tanzu-java-web-a
pp-config-writer

  Delivery
   name:   delivery-basic

   NAME              READY   HEALTHY   UPDATED   RESOURCE
   source-provider   True    True      30m       imagerepositories.source.apps.tanzu.v
mware.com/tanzu-java-web-app-delivery
   deployer          True    True      30m       apps.kappctrl.k14s.io/tanzu-java-web-
app
```

```
  Messages
   No messages found.

  Pods
   NAME                                          READY   STATUS      RESTARTS   AGE
   tanzu-java-web-app-build-11-build-pod         0/1     Completed   0          6d12h
   tanzu-java-web-app-build-12-build-pod         0/1     Completed   0          22h
   tanzu-java-web-app-build-3-build-pod          0/1     Completed   0          60d
   tanzu-java-web-app-config-writer-655rb-pod    0/1     Completed   0          21d
   tanzu-java-web-app-config-writer-7h8bn-pod    0/1     Completed   0          6d12h
   tanzu-java-web-app-config-writer-7xr6m-pod    0/1     Completed   0          60d
   tanzu-java-web-app-config-writer-g9gp8-pod    0/1     Completed   0          45d

  Knative Services
   NAME                READY   URL
   tanzu-java-web-app  Ready   http://tanzu-java-web-app.default.127.0.0.1.nip.io

To see logs: "tanzu apps workload tail tanzu-java-web-app --namespace development --ti
mestamp --since 1h"
```

tanzu apps workload tail

`tanzu apps workload tail` checks the runtime logs of a workload. The workload can be filtered by the namespace it was created on by using the `--namespace` flag, and the workload logs can also be filtered by a specific `--component`.

Use the `--since` flag to retrieve logs within a specific time-frame and display them with their `--timestamp`.

```
tanzu apps workload tail tanzu-java-web-app -n development --since 1h --component buil
d --timestamp

tanzu-java-web-app-build-1-build-pod[prepare] 2023-06-15T10:45:13.236584161-05:00 Buil
d reason(s): CONFIG
tanzu-java-web-app-build-1-build-pod[prepare] 2023-06-15T10:45:13.237805020-05:00 CONF
IG:
tanzu-java-web-app-build-1-build-pod[prepare] 2023-06-15T10:45:13.237847016-05:00
+ env:
tanzu-java-web-app-build-1-build-pod[prepare] 2023-06-15T10:45:13.237853995-05:00
+ - name: BP_OCI_SOURCE
tanzu-java-web-app-build-1-build-pod[prepare] 2023-06-15T10:45:13.237857982-05:00
+   value: tap-1.6.0/bb9e655b33e39b242b6e5d9e218578e75c1d833c
tanzu-java-web-app-build-1-build-pod[prepare] 2023-06-15T10:45:13.237861706-05:00
resources: {}
tanzu-java-web-app-build-1-build-pod[prepare] 2023-06-15T10:45:13.237864954-05:00
- source: {}
tanzu-java-web-app-build-1-build-pod[prepare] 2023-06-15T10:45:13.237868502-05:00
+ source:
tanzu-java-web-app-build-1-build-pod[prepare] 2023-06-15T10:45:13.237871941-05:00
+   blob:
tanzu-java-web-app-build-1-build-pod[prepare] 2023-06-15T10:45:13.237875932-05:00
+     url: http://fluxcd-source-controller.flux-system.svc.cluster.local./gitrepositor
y/default/tanzu-java-web-app/bb9e655b33e39b242b6e5d9e218578e75c1d833c.tar.gz
tanzu-java-web-app-build-1-build-pod[prepare] 2023-06-15T10:45:13.237880878-05:00
+   subPath: tanzu-java-web-app
tanzu-java-web-app-build-1-build-pod[prepare] 2023-06-15T10:45:13.237884317-05:00 Load
ing registry credentials from service account secrets
tanzu-java-web-app-build-1-build-pod[prepare] 2023-06-15T10:45:13.238417749-05:00 Load
ing secret for "gcr.io" from secret "registry-credentials" at location "/var/build-sec
rets/registry-credentials"
tanzu-java-web-app-build-1-build-pod[prepare] 2023-06-15T10:45:13.238437378-05:00 Load
ing secret for "https://gcr.io/" from secret "registry-credentials" at location "/var/
```

```
build-secrets/registry-credentials"
...
```

tail a workload while creating or applying a workload

You can tail a workload directly as part of the `tanzu apps workload create` or `tanzu apps workload apply` commands by providing the `--tail` flag and you can display timestamps for each log entry by including the `--tail-timestamp` flag.

```
tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanz
u/application-accelerator-samples --git-tag tap-1.6.0 --sub-path tanzu-java-web-app --
tail --tail-timestamp
   Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    apps.tanzu.vmware.com/workload-type: web
      7 + |  name: tanzu-java-web-app
      8 + |  namespace: default
      9 + |spec:
     10 + |  source:
     11 + |    git:
     12 + |      ref:
     13 + |        tag: tap-1.6.0
     14 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
     15 + |    subPath: tanzu-java-web-app
  Do you want to create this workload? [yN]: y
  Created workload "tanzu-java-web-app"

To see logs:   "tanzu apps workload tail tanzu-java-web-app --timestamp --since 1h"
To get status: "tanzu apps workload get tanzu-java-web-app"

Waiting for workload "tanzu-java-web-app" to become ready...
+ tanzu-java-web-app-build-1-build-pod › prepare
tanzu-java-web-app-build-1-build-pod[prepare] 2023-06-15T11:28:12.746812119-05:00 Buil
d reason(s): CONFIG
tanzu-java-web-app-build-1-build-pod[prepare] 2023-06-15T11:28:12.747149910-05:00 CONF
IG:
tanzu-java-web-app-build-1-build-pod[prepare] 2023-06-15T11:28:12.747186215-05:00
+ env:
tanzu-java-web-app-build-1-build-pod[prepare] 2023-06-15T11:28:12.747194864-05:00
+ - name: BP_OCI_SOURCE
tanzu-java-web-app-build-1-build-pod[prepare] 2023-06-15T11:28:12.747201214-05:00
+   value: tap-1.6.0/bb9e655b33e39b242b6e5d9e218578e75c1d833c
tanzu-java-web-app-build-1-build-pod[prepare] 2023-06-15T11:28:12.747206421-05:00
resources: {}
tanzu-java-web-app-build-1-build-pod[prepare] 2023-06-15T11:28:12.747211761-05:00
- source: {}
tanzu-java-web-app-build-1-build-pod[prepare] 2023-06-15T11:28:12.747217114-05:00
+ source:
tanzu-java-web-app-build-1-build-pod[prepare] 2023-06-15T11:28:12.747222613-05:00
+   blob:
tanzu-java-web-app-build-1-build-pod[prepare] 2023-06-15T11:28:12.747228733-05:00
+     url: http://fluxcd-source-controller.flux-system.svc.cluster.local./gitrepositor
y/default/tanzu-java-web-app/bb9e655b33e39b242b6e5d9e218578e75c1d833c.tar.gz
tanzu-java-web-app-build-1-build-pod[prepare] 2023-06-15T11:28:12.747235497-05:00
+   subPath: tanzu-java-web-app
tanzu-java-web-app-build-1-build-pod[prepare] 2023-06-15T11:28:12.747240321-05:00 Load
ing registry credentials from service account secrets
tanzu-java-web-app-build-1-build-pod[prepare] 2023-06-15T11:28:12.747402066-05:00 Load
ing secret for "gcr.io" from secret "registry-credentials" at location "/var/build-sec
rets/registry-credentials"
```

```
tanzu-java-web-app-build-1-build-pod[prepare] 2023-06-15T11:28:12.747417620-05:00 Load
ing secret for "https://gcr.io/" from secret "registry-credentials" at location "/var/
build-secrets/registry-credentials"
...
```

–export flag

A clean workload definition export can be committed to GitHub or applied to a different environment without having to make significant edits because the export only includes the fields specified by the developer who created it.

By default, the `--export` flag is configured to display the workload in `yaml` format. However, the behavior can be modified by combining it with the `--output` flag, which accepts `yaml`, `yml`, and `json` as values.

For example, if a workload is created with:

```
tanzu apps workload apply tanzu-where-for-dinner --git-repo https://github.com/vmware-
tanzu/application-accelerator-samples --git-branch main --sub-path where-for-dinner
```

When querying the workload with `--export`, the default export format in `yaml` is as follows:

```
# with yaml format
tanzu apps workload get tanzu-where-for-dinner --export
```

```
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  labels:
    apps.tanzu.vmware.com/workload-type: web
  name: tanzu-where-for-dinner
  namespace: default
spec:
  source:
    git:
      ref:
        branch: main
      url: https://github.com/vmware-tanzu/application-accelerator-samples
    subPath: where-for-dinner
```

```
# with json format
# shorthand for --output flag is -o
# an alternative for this command would be
# tanzu apps workload get rmq-sample-app --export -ojson
tanzu apps workload get rmq-sample-app --export --output json
```

```
{
    "apiVersion": "carto.run/v1alpha1",
    "kind": "Workload",
    "metadata": {
        "labels": {
            "apps.tanzu.vmware.com/workload-type": "web"
        },
        "name": "rmq-sample-app",
        "namespace": "default"
    },
    "spec": {
        "serviceClaims": [
            {
                "name": "rmq",
```

```
                "ref": {
                    "apiVersion": "rabbitmq.com/v1beta1",
                    "kind": "RabbitmqCluster",
                    "name": "example-rabbitmq-cluster-1"
                }
            }
        ],
        "source": {
            "git": {
                "ref": {
                    "branch": "main"
                },
                "url": "https://github.com/jhvhs/rabbitmq-sample"
            }
        }
    }
}
```

–output flag with `tanzu apps workload get` command

Use the `--output` flag with `tanzu apps workload get` to retrieve a workload with all the cluster-specifics.

```
# with json format
tanzu apps workload get rmq-sample-app --output json # can also be used as tanzu apps
workload get rmq-sample-app -ojson
```

```
{
    "kind": "Workload",
    "apiVersion": "carto.run/v1alpha1",
    "metadata": {
        "name": "rmq-sample-app",
        "namespace": "default",
        "uid": "3619ff6d-9e73-473a-9112-891a6d8aee9e",
        "resourceVersion": "11657434",
        "generation": 2,
        "creationTimestamp": "2022-11-28T05:10:32Z",
        "labels": {
            "apps.tanzu.vmware.com/workload-type": "web"
        },
     ...
    },
    ...
        "status": {
        "observedGeneration": 2,
        "conditions": [
            {
                "type": "SupplyChainReady",
                "status": "True",
                "lastTransitionTime": "2022-11-28T05:10:32Z",
                "reason": "Ready",
                "message": ""
            },
            ...
        ],
        "supplyChainRef": {
            "kind": "ClusterSupplyChain",
            "name": "source-to-url"
        },
        "resources": [
            {
                "name": "source-provider",
                "stampedRef": {
```

```
                    "kind": "GitRepository",
                    "namespace": "default",
                    "name": "rmq-sample-app",
                    "apiVersion": "source.toolkit.fluxcd.io/v1beta1",
                    "resource": "gitrepositories.source.toolkit.fluxcd.io"
                },
                "templateRef": {
                    "kind": "ClusterSourceTemplate",
                    "name": "source-template",
                    "apiVersion": "carto.run/v1alpha1"
                },
            ...
            }
        ...
        ]
        ...
    }
    ...
}
```

```
## with yaml format
tanzu apps workload get rmq-sample-app --output yaml # can also be used as tanzu apps
workload get rmq-sample-app -oyaml
```

```
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  creationTimestamp: "2022-11-28T05:10:32Z"
  generation: 2
  labels:
    apps.tanzu.vmware.com/workload-type: web
  ...
  name: rmq-sample-app
  namespace: default
  resourceVersion: "11657434"
  uid: 3619ff6d-9e73-473a-9112-891a6d8aee9e
spec:
  serviceClaims:
  - name: rmq
    ref:
      apiVersion: rabbitmq.com/v1beta1
      kind: RabbitmqCluster
      name: example-rabbitmq-cluster-1
  source:
    git:
      ref:
        branch: main
      url: https://github.com/jhvhs/rabbitmq-sample
status:
  conditions:
  - lastTransitionTime: "2022-11-28T05:10:32Z"
    message: ""
    reason: Ready
    status: "True"
    type: SupplyChainReady
  ...
  observedGeneration: 2
  resources:
  ...
    name: source-provider
    outputs:
    - digest: sha256:97b2cb779b4ea31339595cd204a3fec0053805eeacbbd6d6dd23af7d3000a6ae
      lastTransitionTime: "2022-11-28T05:16:01Z"
      name: url
```

```
      preview: |
        http://fluxcd-source-controller.flux-system.svc.cluster.local./gitrepository/d
efault/rmq-sample-app/73c6311eefbf724fee9ad6f4524fa24ec842ff34.tar.gz
    - digest: sha256:e7884b071fe1bbb2551d42a171043d061a7591e744705572136e689c2a154b7a
      lastTransitionTime: "2022-11-28T05:16:01Z"
      name: revision
      preview: |
        HEAD/73c6311eefbf724fee9ad6f4524fa24ec842ff34
      ...
```

–output flag with `tanzu apps workload apply` command

Use this flag to retrieve the workload in the `yaml`, `yml`, or `json` format after it is applied. When combined with the `--yes` flag, all prompts are bypassed, and only the workload definition is returned. Additionally, use the `--wait` or `--tail` flags, to retrieve the workload with its current status.

```
tanzu apps workload create tanzu-where-for-dinner --git-repo https://github.com/vmware
-tanzu/application-accelerator-samples --git-branch main --sub-path where-for-dinner -
oyaml
  Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    apps.tanzu.vmware.com/workload-type: web
      7 + |  name: tanzu-where-for-dinner
      8 + |  namespace: default
      9 + |spec:
     10 + |  source:
     11 + |    git:
     12 + |      ref:
     13 + |        branch: main
     14 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
     15 + |    subPath: where-for-dinner
  Do you want to create this workload? [yN]: y
  Created workload "tanzu-where-for-dinner"

To see logs:   "tanzu apps workload tail tanzu-where-for-dinner --timestamp --since 1
h"
To get status: "tanzu apps workload get tanzu-where-for-dinner"

---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  creationTimestamp: "2023-07-06T17:22:34Z"
  generation: 1
  labels:
    apps.tanzu.vmware.com/workload-type: web
  name: tanzu-where-for-dinner
  namespace: default
  resourceVersion: "249375192"
  uid: 8a8132e5-9ee4-41f3-af92-9578ea1efb01
spec:
  source:
    git:
      ref:
        branch: main
      url: https://github.com/vmware-tanzu/application-accelerator-samples
    subPath: where-for-dinner
```

```
status:
  supplyChainRef: {}
```

Using `--yes` flag with the command

```
tanzu apps workload create tanzu-where-for-dinner --git-repo https://github.com/vmware
-tanzu/application-accelerator-samples --git-branch main --sub-path where-for-dinner -
ojson -y
{
    "apiVersion": "carto.run/v1alpha1",
    "kind": "Workload",
    "metadata": {
        "creationTimestamp": "2023-07-06T18:03:07Z",
        "generation": 1,
        "labels": {
            "apps.tanzu.vmware.com/workload-type": "web"
        },
        "name": "tanzu-where-for-dinner",
        "namespace": "default",
        "resourceVersion": "249455362",
        "uid": "06a9793b-6747-4f9e-8455-6f5b342c3631"
    }
    "spec": {
        "source": {
            "git": {
                "ref": {
                    "branch": "main"
                },
                "url": "https://github.com/vmware-tanzu/application-accelerator-sample
s"
            },
            "subPath": "where-for-dinner"
        }
    },
    "status": {
        "supplyChainRef": {}
    }
}
```

Delete a workload

This topic tells you about the Tanzu Apps CLI plug-in `tanzu apps workload delete` command.

Use the `tanzu apps workload delete` command to remove one or more workloads from a specific namespace on the target cluster.

Note

This command does not automatically remove a workload's corresponding images from the registry.

To specify the workload for deletion, either provide the workload name and namespace or use a `yaml` file containing the workload definition.

To control the deletion process, use the `--wait` flag, which waits until the workload is deleted, or the `--wait-timeout` flag, which sets a specific timeout duration for the deletion process.

Delete all workloads in a namespace

Delete workloads all at once with the `--all` flag. If there is a particular namespace where all workloads should be deleted, use the `--namespace` flag to specify it.

```
# --namespace shorthand is -n
# an alternative for this same command is
# tanzu apps workload delete --all -n my-namespace
tanzu apps workload delete --all --namespace my-namespace
  Really delete all workloads in the namespace "my-namespace"? Yes
  Deleted workloads in namespace "my-namespace"
```

Apps CLI plug-in how-to-guides

The how-to-guides section contains the following topics:

- Integrate with Local Source Proxy
- Manage workload merge behavior

Integrate with Local Source Proxy

This topic tells you how to integrate the Tanzu Apps CLI with Local Source Proxy.

You can configure workloads to push local source code to a registry that is predefined using the Local Source Proxy component.

For more information about Local Source Proxy, see Overview of Local Source Proxy and Create a workload from Local Source.

Check Local Source Proxy health

To check the installation and health of Local Source Proxy and determine if failures originate from the Local Source Proxy or the upstream registry, run:

```
tanzu apps local-source-proxy health
```

This returns an overview of the health of the Local Source Proxy. This information is obtained directly from the Local Source Proxy and provides a status code and message.

Use the `--output` flag to specify one of the following formats: `yaml`, `yml`, and `json`. The default format is `yaml`.

The API is called at the `/health` endpoint, and there are five possible outputs based on the response received:

- All checks pass. Format is `yaml`.

  ```
  user_has_permission: true
  reachable: true
  upstream_authenticated: true
  overall_health: true
  message: "All health checks passed"
  ```

- You do not have permission to list the service. The possible reason is a 403 error from the Kubernetes API Server. Format is `yaml`.

  ```
  user_has_permission: false
  reachable: false
  upstream_authenticated: false
  overall_health: false
  ```

```
message: "The current user does not have permission to access the local source
proxy"
```

- You can list the services, but it's not there. The possible reason is a 404 error from the Kubernetes API Server. Format is Source-Controller

```
user_has_permission: true
reachable: false
upstream_authenticated: false
overall_health: false
message: "Local source proxy is not installed on the cluster"
```

- `/health` from Local Source Proxy returns a 5xx. Format is `yaml`.

```
user_has_permission: true
reachable: true
upstream_authenticated: false
overall_health: false
message: "Local source proxy is not healthy. Error: <error>"
```

- `/health` from upstream returns a non-2xx, 4xx, 5xx. `/health` from Local Source Proxy will still be 2xx. Format is `yaml`.

```
user_has_permission: true
reachable: true
upstream_authenticated: false
overall_health: false
message: "Local source proxy was unable to authenticate against the target regi
stry. Error: <error>"
```

- All checks pass. Format is `json`.

```
{
"user_has_permission": true
"reachable": true
"upstream_authenticated": true
"overall_health": true
"message": ""
}
```

Update the local source code for workloads

To create a workload from local source code, there are two options available:

1. Use Local Source Proxy, which automatically defaults to a predefined registry.
2. Use the `--source-image` flag to specify a registry. You must be authenticated to access the registry.

To distinguish whether a workload was created with the Local Source Proxy or the `--source-image` flag, check if the workload contains the `local-source-proxy.apps.tanzu.vmware.com` annotation. This annotation indicates the method used to create the workload.

For more information, see Create a workload from Local Source.

Use the `--source-image` flag

After a workload is created using the `--source-image` flag, it is not possible to update it to use Local Source Proxy. However, you do not need to repeatedly specify the `--source-image` flag during subsequent updates, as it is retrieved from the existing workload specification within the cluster. If

you do want to change the registry or storage location for the source code, specify the `--source-image` flag again with a new value.

```
# create a workload using source image
# inside the local source code folder
tanzu apps workload apply java-web-app --local-path . -s my-registry.io/my-project/jav
a-app
The files and/or directories listed in the .tanzuignore file are being excluded from t
he uploaded source code.
Publishing source in "." to "my-registry.io/my-project/java-app"...
  Published source

  Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    apps.tanzu.vmware.com/workload-type: web
      7 + |  name: java-web-app
      8 + |  namespace: default
      9 + |spec:
     10 + |  source:
     11 + |    image: my-registry.io/my-project/java-app:latest@sha256:447db92e289dbe3
a6969521917496ff2b6b0a1d6fbff1beec3af726430ce8493
  Do you want to create this workload? [yN]: y
  Created workload "java-web-app"

To see logs:   "tanzu apps workload tail java-web-app --timestamp --since 1h"
To get status: "tanzu apps workload get java-web-app"

# update the workload and check the source image is not changed to use Local Source Pr
oxy
tanzu apps workload apply java-web-app --label hello=world
  Update workload:
...
  3,  3   |kind: Workload
  4,  4   |metadata:
  5,  5   |  labels:
  6,  6   |    apps.tanzu.vmware.com/workload-type: web
      7 + |    hello: world
  7,  8   |  name: java-web-app
  8,  9   |  namespace: default
  9, 10   |spec:
 10, 11   |  source:
...
  Really update the workload "java-web-app"? [yN]:
```

Switch from Local Source Proxy to `--source-image` flag

If a workload was initially created using the Local Source Proxy, it can be updated to use the `--source-image` flag. The workload transitions to use the specified source image. As a result, the annotation that establishes the connection between the workload and the Local Source Proxy is removed.

```
# inside the folder that contains the local source code
tanzu apps workload apply java-web-app --local-path .
The files and/or directories listed in the .tanzuignore file are being excluded from t
he uploaded source code.
Publishing source in "." to "local-source-proxy.tap-local-source-system.svc.cluster.lo
cal/source:default-java-web-app"...
  Published source
```

```
  Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  annotations:
      6 + |    local-source-proxy.apps.tanzu.vmware.com: my-registry.io/my-project/sou
rce:default-java-web-app@sha256:447db92e289dbe3a6969521917496ff2b6b0a1d6fbff1beec3af72
6430ce8493
      7 + |  labels:
      8 + |    apps.tanzu.vmware.com/workload-type: web
      9 + |  name: java-web-app
     10 + |  namespace: default
     11 + |spec:
     12 + |  source:
     13 + |    image: my-registry.io/my-project/source:default-java-web-app@sha256:447
db92e289dbe3a6969521917496ff2b6b0a1d6fbff1beec3af726430ce8493
  Do you want to create this workload? [yN]: y
  Created workload "java-web-app"

To see logs:   "tanzu apps workload tail java-web-app --timestamp --since 1h"
To get status: "tanzu apps workload get java-web-app"

# update to use a source image and see how the Local Source Proxy annotation is remove
d
# and the `spec.source.image` field also changes
tanzu apps workload apply java-web-app --local-path . -s my-registry.io/my-project/jav
a-web-app
The files and/or directories listed in the .tanzuignore file are being excluded from t
he uploaded source code.
Publishing source in "." to "gcr.io/tanzu-framework-playground/java-web-app"...
37.58 kB / 37.16 kB [-------------------------------------------------------------------
-------------------------------------------------------------------------------] 101.14%
31.94 kB p/s
  Published source

  Update workload:
  1,  1   |---
  2,  2   |apiVersion: carto.run/v1alpha1
  3,  3   |kind: Workload
  4,  4   |metadata:
  5     - |  annotations:
  6     - |    local-source-proxy.apps.tanzu.vmware.com: my-registry.io/my-project/sou
rce:default-java-web-app@sha256:447db92e289dbe3a6969521917496ff2b6b0a1d6fbff1beec3af72
6430ce8493
  7,  5   |  labels:
  8,  6   |    apps.tanzu.vmware.com/workload-type: web
  9,  7   |  name: java-web-app
 10,  8   |  namespace: default
 11,  9   |spec:
 12, 10   |  source:
 13     - |    image: my-registry.io/my-project/source:default-java-web-app@sha256:447
db92e289dbe3a6969521917496ff2b6b0a1d6fbff1beec3af726430ce8493
     11 + |    image: my-registry.io/my-project/java-web-app:latest@sha256:447db92e289
dbe3a6969521917496ff2b6b0a1d6fbff1beec3af726430ce8493
  Really update the workload "java-web-app"? [yN]:
```

Manage workload merge behavior

This topic tells you how to manage the workload update behavior with the Tanzu Apps CLI `--update-strategy` flag.

When updating a workload from a file, manage the workload update behavior with the `--update-strategy` flag. There are two possible values: `merge` or `replace`. The default value is `merge`.

merge

If the `--file workload.yaml` deletes an existing on-cluster property or value, it is not removed from the on-cluster definition. If the `--file workload.yaml` includes a new property or value, it is added to the on-cluster workload property value. If the `--file workload.yaml` updates an existing value for a property, it is updated on the on-cluster definition.

replace

The on-cluster workload is updated to exactly what is specified in the `--file workload.yaml` definition.

The current default merge strategy intents to prevent unintentional deletions of critical properties from existing workloads.

Note

The default value for the `--update-strategy flag` will change from merge to replace in Tanzu Application Platform v1.7.0 and later.

Examples of the outcomes of both the `merge` and `replace` values are provided in the following examples:

- ```
  # Export workload if there is no previous yaml definition
  tanzu apps workload get spring-petclinic --export > spring-petclinic.yaml

  # modify the workload definition
  vi rmq-sample-app.yaml
  ---
  apiVersion: carto.run/v1alpha1
  kind: Workload
  metadata:
  name: spring-petclinic
  labels:
    app.kubernetes.io/part-of: spring-petclinic
    apps.tanzu.vmware.com/workload-type: web
  spec:
  resources:
    requests:
      memory: 1Gi
    limits:           # delete this line
      memory: 1Gi     # delete this line
      cpu: 500m       # delete this line
  source:
    git:
      url: https://github.com/sample-accelerators/spring-petclinic
      ref:
        tag: tap-1.1
  ```

After saving the file, to verify how both of the update strategy options behave, run:

```
tanzu apps workload apply -f ./spring-petclinic.yaml --update-strategy merge # if flag
is not specified, merge is taken as default
```

This produces the following output:

```
  WARNING: Configuration file update strategy is changing. By default, provided config
uration files
will replace rather than merge existing configuration. The change will take place in t
```
```

```
he January 2024
Tanzu Application Platform release (use "--update-strategy" to control strategy explic
itly).

Workload is unchanged, skipping update
```

By contrast, use `replace` as follows:

```
tanzu apps workload apply -f ./spring-petclinic.yaml --update-strategy replace
```

This produces the following output:

```
  WARNING: Configuration file update strategy is changing. By default, provided config
uration files
will replace rather than merge existing configuration. The change will take place in t
he January 2024
Tanzu Application Platform release (use "--update-strategy" to control strategy explic
itly).

  Update workload:
...
  8,  8   |  name: spring-petclinic
  9,  9   |  namespace: default
 10, 10   |spec:
 11, 11   |  resources:
 12     - |    limits:
 13     - |      cpu: 500m
 14     - |      memory: 1Gi
 15, 12   |    requests:
 16, 13   |      memory: 1Gi
 17, 14   |  source:
 18, 15   |    git:
...
  Really update the workload "spring-petclinic"? [yN]:
```

The lines that were deleted in the `yaml` file are deleted as well in the workload running in the cluster. The only text boxes that remain exactly as they were created are the system populated metadata text boxes; `resourceVersion`, `uuid`, `generation`, `creationTimestamp`, and `deletionTimestamp`.

Troubleshoot Apps CLI

The troubleshooting section contains the following topics:

- Troubleshooting workloads
- Troubleshooting local source proxy integration

Troubleshoot workloads

This topic tells you how to use the Tanzu Apps CLI plug-in to troubleshoot workloads in Tanzu Application Platform (commonly known as TAP).

Check build logs

After a workload is created, tail the workload to view the build and runtime logs.

Run:

```
tanzu apps workload tail WORKLOAD --since 10m --timestamp
```

Where:

- `WORKLOAD` is the name of the workload.
- `--since` is optional.The amount of time to go back to begin streaming logs. The default is 1 second.
- `--timestamp` is optional. Prints the timestamp with each log entry.

Get the workload status and details

After the workload build process is complete, a Knative service can be created to run the workload. Workload details can be viewed at any time during the process. Some details, such as the workload URL, are only available after the workload is running.

Run:

```
tanzu apps workload get WORKLOAD
```

Where:

`WORKLOAD` is the name of the workload to be checked.

Now the workload should be in a running state. When the workload is created, `tanzu apps workload get` includes the URL for the running workload. In some terminals, you can **Ctrl+click** the URL to view it. You can also copy the URL into a web browser to see the application.

Common workload errors

A workload can either be ready, be in an error state, or have an unknown status.

There are known errors that cause the workload to enter an error or unknown status. Look at the supply chain or delivery steps for status and review the messages section for clues when the workload appears to be having issues.

Local Path Development Error Cases

The section describes the cause and resolution for some of the most common issues.

Message: Writing `registry/project/repo/workload:latest`: Writing image: Unexpected status code *401 Unauthorized* (HEAD responses have no body, use GET for details)

Cause: Apps plug-in cannot talk to the registry because the registry credentials are missing or invalid.

Resolution: Run `docker logout registry` and `docker login registry` commands and specify the valid credentials for the registry.

Message: Writing `registry/project/workload:latest`: Writing image: HEAD Unexpected status code *400 Bad Request* (HEAD responses have no body, use GET for details)

Cause: Certain registries like Harbor or GCR have a concept of `Project`. A 400 Bad request is sent when either the project does not exist, the user does not have access to it, or the path in the `--source-image` flag is missing either project or repository.

Resolution: Fix the path in the `--source-image` flag value to point to a valid repository path.

WorkloadLabelsMissing/SupplyChainNotFound

Message: No supply chain found where full selector is satisfied by `labels: map[app.kubernetes.io/part-of:spring-petclinic]`.

Cause: The labels and attributes in the workload object did not fully satisfy any installed supply chain on the cluster.

Resolution: Use the `tanzu apps cluster-supply-chain list` (alias `csc`) and `tanzu apps csc get <supply-chain-name>` commands to see the workload selection criteria for the supply chain available on the cluster. Apply any missing labels to a workload by using `tanzu apps workload apply --label required-label-name=required-label-value`. For example:

```
tanzu apps workload apply workload-name —-type web
# or
tanzu apps workload apply workload-name --label apps.tanzu.vmware.com/workload-type=we
b
```

MissingValueAtPath

Message: Waiting to read value `[.status.artifact.url]` from resource gitrepository.source.toolkit.fluxcd.io in namespace `[ns]`

Possible Cause 1: The Git `url/tag/branch/commit` parameters passed in the workload are not valid.

Resolution 1: Fix the invalid Git parameters by using `tanzu apps workload apply`.

Possible Cause 2: The Git repository is not accessible from the cluster.

Resolution 2: Configure the cluster networking or the Git repository networking so that they can communicate with each other.

Possible Cause 3: The namespace is missing the Git secret for communicating with the private repository.

Resolution 3: For more information, see Git authentication.

TemplateRejectedByAPIServer

Message: Unable to apply object `[ns/workload-name]` for resource `[source-provider]` in supply chain `[source-to-url]`: failed to get unstructured `[ns/workload-name]` from API server: imagerepositories.source.apps.tanzu.vmware.com “workload-name” is forbidden: User “system:serviceaccount:ns:default” cannot get resource “imagerepositories” in API group “source.apps.tanzu.vmware.com” in the namespace “ns”

Cause: This error happens when the service account in the workload object does not have permission to create objects that are stamped out by the supply chain.

Resolution: Set up the Set up developer namespaces to use your installed packages with the required service account and permissions. For more information, see Developer namespace setup for Supply Chain Security Tools - Store.

Review supply chain steps

After a workload is created with the `tanzu apps workload create` or `tanzu apps workload apply` command, run the `tanzu apps workload get` command to display the current condition of each supply chain.

For example:

```
...
  Supply Chain
   name:   source-to-url

   NAME                READY   HEALTHY   UPDATED   RESOURCE
   source-provider     True    True      71m       gitrepositories.source.toolkit.fluxc
d.io/spring-petclinic
```

```
   image-provider     True    True      70m       images.kpack.io/spring-petclinic
   config-provider    True    True      69m       podintents.conventions.carto.run/spr
ing-petclinic
   app-config         True    True      69m       configmaps/spring-petclinic
   service-bindings   True    True      69m       configmaps/spring-petclinic-with-cla
ims
   api-descriptors    True    True      69m       configmaps/spring-petclinic-with-api
-descriptors
   config-writer      True    True      69m       runnables.carto.run/spring-petclinic
-config-writer

  Delivery
   name:   delivery-basic

   NAME               READY   HEALTHY   UPDATED   RESOURCE
   source-provider    True    True      69m       imagerepositories.source.apps.tanzu.v
mware.com/spring-petclinic-delivery
   deployer           True    True      69m       apps.kappctrl.k14s.io/spring-petclini
c

  Messages
   No messages found.
...
```

The `Supply Chain` section displays the supply chain steps associated with the workload. If a step fails, the `READY` column value is `Unknown` or `False`, and the `HEALTHY` column value is `False`. If a resource is in the `Unknown` or `False` status, inspect it with:

```
kubectl describe RESOURCE-NAME
```

Where `RESOURCE-NAME` is the name of the stamped out resource, displayed in the `RESOURCE` column.

For example, if `tanzu apps workload get` command returns this resource:

```
NAME               READY   HEALTHY   UPDATED   RESOURCE
source-provider    False   False     3h12m     gitrepositories.source.toolkit.fluxcd.i
o/spring-petclinic
```

The resource can be checked with:

```
kubectl describe gitrepositories.source.toolkit.fluxcd.io/spring-petclinic
```

The `Messages` section might give a hint about what went wrong in the process. For example, a message similar to the following is shown:

```
  Messages
   Workload [HealthyConditionRule]:   failed to checkout and determine revision: faile
d to resolve commit object for '425ae9a2a2f84d195a9f3862668e8b2abf81418a': object not
found
```

This might mean that the commit does not belong to the specified branch or does not exist in the repository.

Additional Troubleshooting References

For more workload troubleshooting tips, see Troubleshoot using Tanzu Application Platform page.

Troubleshooting Local Source Proxy integration

This topic tells you how to troubleshoot the Tanzu Apps CLI plug-in integration with Local Source Proxy.

Use the following topics to troubleshoot the Local Source Proxy integration:

Integrate with Local Source Proxy

Troubleshoot Local Source Proxy.

Tanzu Apps CLI plug-in command reference

The Tanzu Apps CLI plug-in command reference has moved to the Tanzu CLI Command Reference documentation.

Overview of Tanzu Accelerator CLI

The Tanzu Accelerator Tanzu CLI includes commands for developers and operators to create and use accelerators.

Server API connections for operators and developers

The Tanzu Accelerator CLI must connect to a server for all provided commands except for the `help` and `version` commands.

Operators typically use **create**, **update**, and **delete** commands for managing accelerators in a Kubernetes context. They also use the **fragment** commands to manage accelerator fragments. These commands require a Kubernetes context where the operator is already authenticated and is authorized to create and edit the accelerator resources. Operators can also use the **get** and **list** commands by using the same authentication. For any of these commands, the operator can specify the `--context` flag to access accelerators in a specific Kubernetes context.

Developers use the **list**, **get**, and **generate** commands for using accelerators available in an Application Accelerator server. Developers use the `--server-url` to point to the Application Accelerator server they want to use.

You can either use the proxy that is part of TAP-GUI or you can use the URL for the Application Accelerator server, if that is configured to be exposed. VMware recommends using the TAP-GUI address.

Using TAP-GUI URL

1. Specify `--server-url` as:

   ```
   https://tap-gui.DOMAIN
   ```

   Where `DOMAIN` defaults to the `shared.ingress_domain` value provided in the Tanzu Application Platform values file.

2. Add the following flags to the `tap-values.yaml` file when `shared.ingress_domain` is set.

   ```
   accelerator:
     ingress:
       include: true
   ```

Using Application Accelerator Server URL

If you cannot use the TAP-GUI URL, the fallback is to use Application Accelerator server directly. In this case the URL depends on the configuration settings for Application Accelerator:

- For installations configured with a **shared ingress** and where the Application Accelerator server is configured to use the ingress, use `https://accelerator.<domain>` where `domain`

defaults to the `shared.ingress_domain` value provided in the values file of Tanzu Application Platform.

- For installations using a **LoadBalancer**, look up the External IP address by using:

  ```
  kubectl get -n accelerator-system service/acc-server
  ```

  Use `http://<External-IP>` as the URL.

- For any other configuration, you can use port forwarding by using:

  ```
  kubectl port-forward service/acc-server -n accelerator-system 8877:80
  ```

  Use `http://localhost:8877` as the URL.

Using “ACC_SERVER_URL” environment variable

The developer can set an `ACC_SERVER_URL` environment variable to avoid having to provide the same `--server-url` flag for every command. Run `export ACC_SERVER_URL=<URL>` for the terminal session in use. If the developer explicitly specifies the `--server-url` flag, it overrides the `ACC_SERVER_URL` environment variable if it is set.

Installation

To install the Tanzu Accelerator CLI plug-in, see Install new CLI plug-ins.

Command reference

For information about available commands, see the Tanzu CLI Command Reference documentation.

Overview of Tanzu Accelerator CLI

The Tanzu Accelerator Tanzu CLI includes commands for developers and operators to create and use accelerators.

Server API connections for operators and developers

The Tanzu Accelerator CLI must connect to a server for all provided commands except for the `help` and `version` commands.

Operators typically use **create**, **update**, and **delete** commands for managing accelerators in a Kubernetes context. They also use the **fragment** commands to manage accelerator fragments. These commands require a Kubernetes context where the operator is already authenticated and is authorized to create and edit the accelerator resources. Operators can also use the **get** and **list** commands by using the same authentication. For any of these commands, the operator can specify the `--context` flag to access accelerators in a specific Kubernetes context.

Developers use the **list**, **get**, and **generate** commands for using accelerators available in an Application Accelerator server. Developers use the `--server-url` to point to the Application Accelerator server they want to use.

You can either use the proxy that is part of TAP-GUI or you can use the URL for the Application Accelerator server, if that is configured to be exposed. VMware recommends using the TAP-GUI address.

Using TAP-GUI URL

1. Specify `--server-url` as:

   ```
   https://tap-gui.DOMAIN
   ```

   Where `DOMAIN` defaults to the `shared.ingress_domain` value provided in the Tanzu Application Platform values file.

2. Add the following flags to the `tap-values.yaml` file when `shared.ingress_domain` is set.

   ```
   accelerator:
     ingress:
       include: true
   ```

Using Application Accelerator Server URL

If you cannot use the TAP-GUI URL, the fallback is to use Application Accelerator server directly. In this case the URL depends on the configuration settings for Application Accelerator:

- For installations configured with a **shared ingress** and where the Application Accelerator server is configured to use the ingress, use `https://accelerator.<domain>` where `domain` defaults to the `shared.ingress_domain` value provided in the values file of Tanzu Application Platform.
- For installations using a **LoadBalancer**, look up the External IP address by using:

  ```
  kubectl get -n accelerator-system service/acc-server
  ```

  Use `http://<External-IP>` as the URL.

- For any other configuration, you can use port forwarding by using:

  ```
  kubectl port-forward service/acc-server -n accelerator-system 8877:80
  ```

  Use `http://localhost:8877` as the URL.

Using “ACC_SERVER_URL” environment variable

The developer can set an `ACC_SERVER_URL` environment variable to avoid having to provide the same `--server-url` flag for every command. Run `export ACC_SERVER_URL=<URL>` for the terminal session in use. If the developer explicitly specifies the `--server-url` flag, it overrides the `ACC_SERVER_URL` environment variable if it is set.

Installation

To install the Tanzu Accelerator CLI plug-in, see Install new CLI plug-ins.

Command reference

For information about available commands, see the Tanzu CLI Command Reference documentation.

Tanzu Accelerator CLI plug-in command reference

The Tanzu Accelerator CLI plug-in command reference has moved to the Tanzu CLI Command Reference documentation.

Overview of Build Service CLI plug-in

The Tanzu Build Service CLI plug-in command reference has moved to the Tanzu CLI Command Reference documentation.

Overview of Tanzu Insight plug-in

The Tanzu Insight CLI plug-in helps you query vulnerability, image, and package data.

Follow these steps to install, configure, and use your Tanzu Insight CLI plug-in:

Note: Prior to using the CLI plug-in, you must install the Supply Chain Security Tools - Store, either as its own package, or as part of Tanzu Application Platform View profile.

1. Install the Tanzu Insight plug-in. The Tanzu Insight plug-in is in the Tanzu Application Platform plug-ins group, see Install Tanzu CLI plug-ins.
2. Configure your Tanzu Insight CLI plug-in.

When the Tanzu Insight CLI plug-in is set up, complete the following steps:

1. Add data to your Supply Chain Security Tools - Store
2. Query vulnerabilities, images, and packages
3. Query Software Bill of Material reports
4. Triage vulnerabilities

Overview of Tanzu Insight plug-in

The Tanzu Insight CLI plug-in helps you query vulnerability, image, and package data.

Follow these steps to install, configure, and use your Tanzu Insight CLI plug-in:

Note: Prior to using the CLI plug-in, you must install the Supply Chain Security Tools - Store, either as its own package, or as part of Tanzu Application Platform View profile.

1. Install the Tanzu Insight plug-in. The Tanzu Insight plug-in is in the Tanzu Application Platform plug-ins group, see Install Tanzu CLI plug-ins.
2. Configure your Tanzu Insight CLI plug-in.

When the Tanzu Insight CLI plug-in is set up, complete the following steps:

1. Add data to your Supply Chain Security Tools - Store
2. Query vulnerabilities, images, and packages
3. Query Software Bill of Material reports
4. Triage vulnerabilities

Configure your Tanzu Insight CLI plug-in

This topic tells you how to configure your Tanzu Insight CLI plug-in.

Set the target and certificate authority (CA) certificate

These instructions are for the recommended configuration where Ingress is enabled. For instructions on non Ingress setups, see Configure target endpoint and certificate.

Set the endpoint host to `metadata-store.INGRESS-DOMAIN`, such as `metadata-store.example.domain.com`. Where `INGRESS-DOMAIN` isthe value of the `ingress_domain` property in your deployment yaml.

Note In a multi-cluster setup, a DNS record is **required** for the domain. The below instructions for single cluster setup do not apply, skip to Set Target section.

Single Cluster setup

In a single-cluster setup, a DNS record is still recommended. However, if no accessible DNS record exists for the domain, edit the `/etc/hosts` file to add a local record:

```
ENVOY_IP=$(kubectl get svc envoy -n tanzu-system-ingress -o jsonpath="{.status.loadBal
ancer.ingress[0].ip}")

# Replace with your domain
METADATA_STORE_DOMAIN="metadata-store.example.domain.com"

# Delete any previously added entry
sudo sed -i '' "/$METADATA_STORE_DOMAIN/d" /etc/hosts

echo "$ENVOY_IP $METADATA_STORE_DOMAIN" | sudo tee -a /etc/hosts > /dev/null
```

Set Target

To get the certificate, run:

```
kubectl get secret tap-ingress-selfsigned-root-ca -n cert-manager -o json | jq -r '.da
ta."ca.crt"' | base64 -d > insight-ca.crt
```

Set the target by running:

```
tanzu insight config set-target https://$METADATA_STORE_DOMAIN --ca-cert insight-ca.cr
t
```

Important

The `tanzu insight config set-target` does not initiate a test connection. Use `tanzu insight health` to test connecting using the configured endpoint and CA certificate. Neither commands test whether the access token is correct. For that you must use the plug-in to add data and query data.

Set the access token

When using the `insight` plug-in, you must set the `METADATA_STORE_ACCESS_TOKEN` environment variable, or use the `--access-token` flag. VMware discourages using the `--access-token` flag as the token appears in your shell history.

The following command retrieves the access token from the default `metadata-store-read-write-client` service account and stores it in `METADATA_STORE_ACCESS_TOKEN`:

```
export METADATA_STORE_ACCESS_TOKEN=$(kubectl get secrets metadata-store-read-write-cli
ent -n metadata-store -o jsonpath="{.data.token}" | base64 -d)
```

Verify the connection

Verify that your configuration is correct and you can make a connection using `tanzu insight health`.

Important

The `tanzu insight health` command tests the configured endpoint and CA certificate. However, it does not test whether the access token is correct. For that, you must use the plug-in to add and query data.

For example:

```
$ tanzu insight health
Success: Reached Metadata Store!
```

Query vulnerabilities, images, and packages

This topic tells you how to query the database to understand vulnerability, image, and dependency relationships. The Tanzu Insight CLI plug-in queries the database for vulnerability scan reports or Software Bill of Materials (commonly known as SBoM) files.

Supported use cases

The following are use cases supported by the CLI:

- What packages and CVEs exist in a particular image? (`image`)
- What dependencies are affected by a specific CVE? (`vulnerabilities`)

Query using the Tanzu Insight CLI plug-in

Install the Tanzu Insight plug-in. The Tanzu Insight plug-in is in the Tanzu Application Platform plug-ins group, see Install Tanzu CLI plug-ins.

There are four commands for querying and adding data.

- `image` - Post an image SBOM or query images for packages and vulnerabilities.
- `package` - Query packages for vulnerabilities or by image or source code.
- `source` - Post a source code SBOM or query source code for packages and vulnerabilities.
- `vulnerabilities` - Query vulnerabilities by image, package, or source code.

For more information about Tanzu Insight CLI plug-in commands, see the VMware Tanzu CLI documentation.

Example 1: What packages and CVEs does a specific image contain?

To query an image scan for vulnerabilities, you need the image digest value, which you can get from the image scan resource.

Find an image digest value

Find an image digest value using Supply Chain Tools - Scan 2.0 or Supply Chain Tools - Scan Pre-2.0.

Find an image digest using Supply Chain Tools - Scan 2.0

When using Supply Chain Tools - Scan 2.0, find the image digest value by looking inside the corresponding image vulnerability scan custom resource.

To get a list of image vulnerability scans, run:

```
kubectl get imagevulnerabilityscan -n WORKLOAD-NAMESPACE
```

For example:

```
$ kubectl get imagevulnerabilityscan -n my-apps
NAME                                 SUCCEEDED   REASON
tanzu-java-web-app-grype-scan-jb76m   True        Succeeded
```

The name of the image vulnerability scan starts with the name of the workload.

To describe the image vulnerability scan, run:

```
kubectl describe imagevulnerabilityscan IMAGE-VULNERABILITY-SCAN-NAME -n WORKLOAD-NAME
SPACE
```

For example:

```
kubectl describe imagevulnerabilityscan tanzu-java-web-app-grype-scan-jb76m -n my-apps
```

In the resource, look for the `Spec.Image` field. The value points to the image that was scanned, including its digest.

For example:

```
Spec:
  Image: fake.oci-registry.io/dev-cluster/supply-chain-apps/tanzu-java-web-app-my-apps
@sha256:a24a8d8eb724b6816f244925cc6625a84c15f6ced6a19335121343424be693cd
```

In this example, the image digest value is:
`sha256:a24a8d8eb724b6816f244925cc6625a84c15f6ced6a19335121343424be693cd`

Find an image digest value using Supply Chain Tools - Scan Pre-2.0

When using Supply Chain Tools - Scan Pre-2.0, find the image digest value by looking inside the corresponding image scan custom resource.

Run:

```
kubectl get imagescan WORKLOAD-NAME -n WORKLOAD-NAMESPACE
```

For example:

```
kubectl get imagescan tanzu-java-web-app -n my-apps
```

In the resource, look for the `Spec.Registry.Image` field. The value points to the image that was scanned, including its digest.

For example:

```
Spec:
  Registry:
    Image: fake.oci-registry.io/dev-cluster/supply-chain-apps/tanzu-java-web-app-my-ap
ps@sha256:e8c648533c4c7440ee9a93142ac7480205e0f7669e4f86771cede8bfaacdc2cf
```

In this example, the image digest is:
`sha256:e8c648533c4c7440ee9a93142ac7480205e0f7669e4f86771cede8bfaacdc2cf`

Query an image with the image digest value

Run:

```
tanzu insight image get --digest DIGEST
```

Where:

- `DIGEST` is the component version or image digest.

For example:

```
$ tanzu insight image get --digest sha256:sha256:e8c648533c4c7440ee9a93142ac7480205e0f
7669e4f86771cede8bfaacdc2cf
Registry:       fake.oci-registry.com
Image Name:     dev-cluster/supply-chain-apps/tanzu-java-web-app-my-apps
Digest:         sha256:sha256:e8c648533c4c7440ee9a93142ac7480205e0f7669e4f86771cede8bf
aacdc2cf
Packages:
        1. alpine-baselayout@3.1.2-r0
        2. alpine-keys@2.1-r2
        3. apk-tools@2.10.4-r2
        CVEs:
                1. CVE-2021-30139 (High)
                2. CVE-2021-36159 (Critical)
        4. busybox@1.30.1-r3
        CVEs:
                1. CVE-2021-28831 (High)
...
```

Example #: What dependencies are affected by a specific CVE?

Run:

```
tanzu insight vulnerabilities get --cveid CVE-IDENTIFIER
```

Where:

- `CVE-IDENTIFIER` is the CVE identifier, for example, CVE-2021-30139.

For example:

```
$ tanzu insight vulnerabilities get --cveid CVE-2010-4051
1. CVE-2010-4051 (Low)
Packages:
        1. libc-bin@2.28-10
        2. libc-l10n@2.28-10
        3. libc6@2.28-10
        4. locales@2.28-10
```

Add data

For more information about manually adding data, see [Add Data](Add data to your Supply Chain Security Tools - Store.hbs.md).

Add data to your Supply Chain Security Tools - Store

This topic tells you how to add vulnerability scan reports or Software Bill of Materials (commonly known as SBoM) files to your Supply Chain Security Tools (commonly known as SCST) - Store.

Supported formats and file types

Currently, only CycloneDX XML and JSON files are accepted.

Source commits and image files have been tested. Additional file types might work, but are not fully supported (for example, JAR files).

If you are not using a source commit or image file, you must ensure the `component.version` field in the CycloneDX file is non-null.

Generate a CycloneDX file

A CycloneDX file is needed to post data. Supply Chain Security Tools - Scan outputs CycloneDX files automatically. For more information, see Supply Chain Security Tools - Scan.

To generate a file to post manually, use Grype or another tool in the CycloneDX Tool Center.

To use Grype to scan an image and generate an image report in CycloneDX format:

1. Install Grype.
2. Scan the image and generate a report by running:

   ```
   grype REPO:TAG -o cyclonedx > IMAGE-CVE-REPORT
   ```

   Where:

   - `REPO` is the name of your repository
   - `TAG` is the name of a tag
   - `IMAGE-CVE-REPORT` is the resulting file name of the Grype image scan report

   For example:

   ```
   $ grype docker.io/checkr/flagr:1.1.12 -o cyclonedx > image-cve-report
    ✔ Vulnerability DB        [updated]
    ✔ Parsed image
    ✔ Cataloged packages      [21 packages]
    ✔ Scanned image           [8 vulnerabilities]
   ```

Add data with the Tanzu Insight plug-in

Use the following commands to add data:

- `image add`
- `source add`

If you are not using a source commit or image file, you can select either option.

Example #1: Add an image report

To use a CycloneDX-formatted image report:

1. Run:

   ```
   tanzu insight image add --cyclonedxtype TYPE --path IMAGE-CVE-REPORT
   ```

   Where:

   - `TYPE` specifies XML or JSON, the two supported file types
   - `IMAGE-CVE-REPORT` is the location of a Cyclone DX formatted file

For example:

```
$ tanzu insight image add --cyclonedxtype xml --path downloads/image-cve-report
Image report created.
```

Note

The Metadata Store only stores a subset of CycloneDX file data. Support for more data might be added in the future.

Example #2: Add a source report

To use a CycloneDX-formatted source report:

1. Run:

   ```
   tanzu insight source add --cyclonedxtype TYPE --path SOURCE-CVE-REPORT
   ```

   Where:

   - `TYPE` specifies XML or JSON, the two supported file types
   - `SOURCE-CVE-REPORT` is the location of a Cyclone DX formatted file

   For example:

   ```
   $ tanzu insight source add --cyclonedxtype json --path source-cve-report
   Source report created.
   ```

Note

Supply Chain Security Tools - Store only stores a subset of a CycloneDX file's data. Support for more data might be added in the future.

Query Software Bill of Materials Reports

This topic tells you how to query the database to retrieve a Software Bill of Materials (SBoM) report for a specific image.

Use the `tanzu insight report get --uid` command to query the database to retrieve an SBoM report.

Firstly, query for a list of existing SBoM reports associated with an image, and when you have determined the unique identifier (UID) for an SBoM report, query for a specific SBoM report.

Querying for a specific SBoM report returns all common vulnerabilities and exposures (CVEs), and packages for an image for a specific point in time with a specific vulnerability scan tool.

The information returned is similar to the SBoM returned by the `tanzu insight image get --digest` command. However, the output returned is not for a specific point in time with a specific vulnerability scan tool. For more information about querying for images, see the What packages and CVEs does a specific image contain example.

Query the database to retrieve an SBoM report

1. Query for the list of all reports associated with an image. To fetch a specific SBoM report, you must retrieve the unique identifier (UID) for the SBoM report. Use the unique identifier returned by this command in the next step. Run:

   ```
   tanzu insight report list --digest DIGEST
   ```

   Where:

   - `DIGEST` is the component version or image digest.

   For example:

   ```
   $ tanzu insight report list --digest sha256:20521f76ff3d27f436e03dc666cc97a511b
   be71e8e8495f851d0f4bf57b0bab6
   1.      UID:            b84bd3e8-8d71-4012-a7fa-310d4406658c
     Entity Type:          image
     Entity UID:   sha256:20521f76ff3d27f436e03dc666cc97a511bbe71e8e8495f851d0f4bf
   57b0bab6
     Generated At:         2022-08-01T15:55:56Z
     Tool:
       Name:    trivy
       Version: 0.45.0
       Vendor:  anchore
   2.      UID:            b6bed111-24eb-4814-9cbd-bbc26dd76d7f
     Entity Type:          image
     Entity UID:   sha256:20521f76ff3d27f436e03dc666cc97a511bbe71e8e8495f851d0f4bf
   57b0bab6
     Generated At:         2022-07-15T09:01:56Z
     Tool:
       Name:    grype
       Version: 0.38.0
       Vendor:  anchore
   3.      UID:            510de185-3000-405e-a734-c420f64b1b94
     Entity Type:          image
     Entity UID:   sha256:20521f76ff3d27f436e03dc666cc97a511bbe71e8e8495f851d0f4bf
   57b0bab6
     Generated At:         2022-07-01T13:18:56Z
     Tool:
       Name:    trivy
       Version: 0.45.0
       Vendor:  anchore
   4.      UID:            a007bb9b-877c-485b-9ffc-d359586c5bfc
     Entity Type:          image
     Entity UID:   sha256:20521f76ff3d27f436e03dc666cc97a511bbe71e8e8495f851d0f4bf
   57b0bab6
     Generated At:         2022-06-01T19:28:56Z
     Tool:
       Name:    grype
       Version: 0.38.0
       Vendor:  anchore
   ```

2. Fetch a specific SBoM report based on the UID returned in output in the previous step. In the output in the previous step, the report UID for the image on `2022-07-15`, is `b6bed111-24eb-4814-9cbd-bbc26dd76d7f`. Use this UID to fetch the SBoM report. Run:

   ```
   tanzu insight report get --uid UID
   ```

   Where:

   - `UID` specifies the SBoM report unique identifier

   For example:

```
$ tanzu insight report get --uid b6bed111-24eb-4814-9cbd-bbc26dd76d7f
UID:               b6bed111-24eb-4814-9cbd-bbc26dd76d7f
Generated At:      2022-07-15T09:01:56Z
Tool:
  Name:    grype
  Version: 0.38.0
  Vendor:  anchore
Entity:
  Type:     image
  Name:     checkr/flagr:1.1.12
  Digest:   sha256:20521f76ff3d27f436e03dc666cc97a511bbe71e8e8495f851d0f4bf57b0
bab6
  Registry:
  Packages:
    1. alpine-baselayout@3.1.2-r0
    2. alpine-keys@2.1-r2
    3. apk-tools@2.10.4-r2
    CVEs:
      1. CVE-2021-30139 (High)
      2. CVE-2021-36159 (Critical)
    4. busybox@1.30.1-r3
    CVEs:
      1. CVE-2021-28831 (High)
      2. CVE-2021-42374 (Medium)
      3. CVE-2021-42376 (Medium)
      4. CVE-2021-42378 (High)
      5. CVE-2021-42379 (High)
      6. CVE-2021-42380 (High)
      7. CVE-2021-42381 (High)
      8. CVE-2021-42382 (High)
      9. CVE-2021-42384 (High)
      10. CVE-2021-42385 (High)
      11. CVE-2021-42386 (High)
      12. CVE-2022-28391 (Critical)
    5. ca-certificates@20191127-r2
    6. ca-certificates-cacert@20190108-r0
    7. cloud.google.com/go@v0.37.4
    8. curl@7.66.0-r1
...
```

This SBoM report only includes packages and vulnerabilities for this image on this specific date and time and the vulnerability scan tool version.

Triage vulnerabilities (alpha)

This topic tells you how to add vulnerability analysis associated with a workload in the Supply Chain Security Tools (SCST) - Store. This is an experimental feature, and the API is prone to changes in subsequent releases.

Important

The capability to triage scan results in SCST - Store is in the alpha stage, which means that it is still in early development and is subject to change at any point. You might encounter unexpected behavior from it.

Triage

Vulnerability analysis, or triage is the process of evaluating a reported vulnerability to decide on an effective remediation plan. Triage helps application teams generate useful insights about the

vulnerabilities in their software so that they can make the right decisions about when and how to mitigate them. The current implementation of triage follows CycloneDX's Vulnerability Exploitability eXchange (VEX) specification, and is designed specifically to work with Tanzu workloads.

Prerequisites

Before you begin vulnerability analysis, you must:

- Install the Tanzu Insight plug-in. The Tanzu Insight plug-in is in the Tanzu Application Platform plug-ins group, see Install Tanzu CLI plug-ins.
- Add vulnerability scan reports to the SCST - Store. You can do this either by using the `tanzu insight image add` command or by installing the SCST - Scan. For more information, see Add data and Supply Chain Security Tools - Scan.

Create vulnerability analyses

A vulnerability analysis contains the following data:

1. state: Declares the current state of an occurrence of a vulnerability, after automated or manual analysis.
2. justification: The rationale of why the impact analysis state was asserted.
3. response: A response to the vulnerability by the manufacturer, supplier, or project responsible for the affected component or service.
4. comment: Free form comments to provide additional details.

For more information about the supported values for each of these fields, see the Tanzu CLI Command Reference documentation.

For example, if you are interested in a vulnerability affecting a specific image in your workload, and are investigating its impact, you can add this information to the SCST - Store:

```
tanzu insight triage update \
  --cveid $CVEID \
  --pkg-name $PKG-NAME \
  --pkg-version $PKG-VERSION \
  --img-digest $IMG-DIGEST \
  --artifact-group-uid $ARTIFACT-GROUP-UID \
  --state in_triage
```

Where:

- `CVEID` is the unique identifier of the vulnerability
- `PKG-NAME` and `PKG-VERSION` are the name and version of the Application and OS package affected by the vulnerability
- `IMG-DIGEST` is the digest of the image that contains the affected Application and OS package
- `ARTIFACT-GROUP-UID` is the unique identifier for the workload that contains the image. If your workload was deployed with Tanzu CLI, you can find its unique identifier with the following command:

  ```
  kubectl get workload $MY_WORKLOAD_NAME --namespace $MY_WORKLOAD_NAMESPACE --out
  put jsonpath='{.metadata.uid}'
  ```

Note

> If your affected package is linked to a source instead of an image, you can use `--src-commit` instead of `--img-digest`

As you continue to investigate the vulnerability, you can update your analysis with the latest findings by using the `tanzu insight triage update` command as many times as needed.

View existing analysis

To view all the existing analysis in SCST - Store, run:

```
tanzu insight triage list
```

The results are paginated by default. You can switch the current page or the number of results returned by providing the `--page` or `--limit` flags respectively. You can also filter the results by image or source. For more information, use the `--help` flag or see the Tanzu CLI Command Reference documentation.

Copy an analysis

Sometimes, you might run into scenarios where an existing analysis might be shared between multiple images, for example, when a new version of an existing image is deployed by your workload and it contains the same vulnerability as the previous version, or when you create an analysis for an image that is shared between multiple workloads.

To speed up triage in those cases, you can use the `copy` subcommand:

```
tanzu insight triage copy \
  --triage-uid-to-copy $TRIAGE-UID \
  --img-digest $TARGET-IMAGE
```

Where:

- `TRIAGE-UID` is the uid of an existing analysis
- `TARGET-IMAGE` is the digest of an image you want to copy the analysis to

The following conditions are required for this action:

1. If specified, the targeted image or source must contain the package affected by the vulnerability in the existing analysis.
2. If only an image or source is specified, they must belong to the same workload as the one in the existing analysis.
3. If only an `artifact-group-uid` is specified, it must contain the image or source associated with the existing analysis.

> **Note**
>
> The responsibility of assessing a vulnerability's impact is up to the person in charge of triage. Images and sources with the same package and version might use the package differently and might not have the same analysis values.

Rebase multiple analyses

When you carry out vulnerability analysis on a workload image, you might want to carry this forward after the workload source code is updated and a new image is built and deployed. This process is

called rebase, and you can run it with the following command:

```
tanzu insight triage rebase \
  --img-digest $TARGET-IMAGE
  --artifact-group-uid $ARTIFACT-GROUP-UID
```

Where:

- `TARGET-IMAGE` is the digest of the image you want to rebase the analysis into
- `ARTIFACT-GROUP-UID` is the unique identifier for the workload that contains the image, and where existing analysis will be searched for

This command returns a list of existing analyses that can be automatically rebased into your target image. Each analysis on the list meets all the following criteria:

- The analysis exists for a vulnerability that the target image is affected by.
- The analysis is linked to a previous version of an image.
- There is no existing analysis for the same vulnerability and the target image, or their state is ‘in_triage’

In this context, image A is considered to be a previous version of image B when they have the same name, different digests and image A was created before image B. This will be bound on the workload’s context, using the provided `--artifact-group-uid`.

Known limitations

1. You can only rebase analyses for images, sources are not currently supported.
2. If you are deploying Tanzu Application Platform workloads from pre-built images, or have a custom Supply Chain that changes the name of the deployed image in between builds, you can’t use this feature and must manually copy the existing analyses, see Copy an analysis above.

Tanzu Insight CLI plug-in command reference

The Tanzu Insight CLI plug-in command reference has moved to the Tanzu CLI Command Reference documentation.

Overview of API Auto Registration

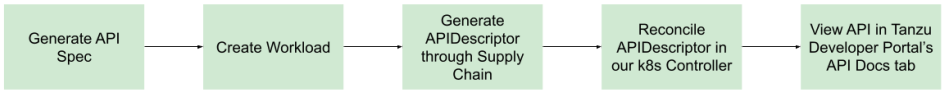
This topic gives you an overview of API Auto Registration for Tanzu Application Platform.

Overview

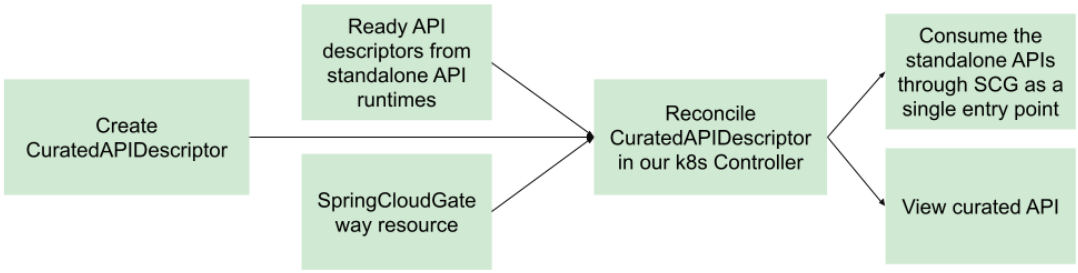
API Auto Registration automatically generates API specifications from a workload’s configuration and registration of the API entity in the Tanzu Developer Portal’s catalog. You can access the registered API specification in Tanzu Developer Portal with no additional steps. As an experimental alpha feature, API Auto Registration generates a curated API by combining APIs exposed from multiple workloads.

You can use an automated workflow with a supply chain to create and manage a Kubernetes Custom Resource (CR) of kind `APIDescriptor` from your workload. API Auto Registration’s Kubernetes controller periodically reconciles the CR and updates the API entity in Tanzu Developer Portal to achieve automated API specification registration from origin workloads.

You can also use API Auto Registration without supply chain automation, with other GitOps processes, or by directly applying an `APIDescriptor` CR to the cluster.



For combining and curating multiple standalone APIDescriptors in your run clusters, you can configure and apply `CuratedAPIDescriptor` to your clusters to provide a single curated entry point for users consuming your APIs. With Spring Cloud Gateway for Kubernetes available as the route provider, API Auto Registration's Kubernetes controller automatically generates and manages necessary routing resources for the curated API. If configured, each curated API is exposed and navigable on your preferred Spring Cloud Gateway instance.



Getting started

For information about API Auto Registration architecture, the `APIDescriptor` CR, the `CuratedAPIDescriptor` CR, and API entities in Tanzu Developer Portal, see Key Concepts.

For information about configuring iterate, run, and full Tanzu Application Platform cluster profiles, see Configure API Auto Registration.

For information about generating API specifications and registering them with Tanzu Developer Portal catalog, see Use API Auto Registration.

For information about curating workloads into Curated APIs and generating Spring Cloud Gateway resources, see API Curation (alpha).

For information about other profiles, install the `api-auto-registration` package. See Install API Auto Registration.

For information about troubleshooting and debugging API Auto Registration, see Troubleshooting.

Overview of API Auto Registration

This topic gives you an overview of API Auto Registration for Tanzu Application Platform.

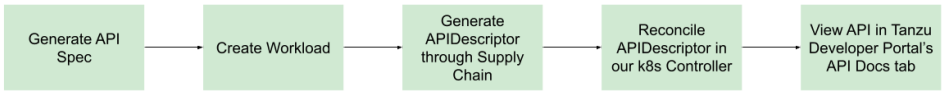
Overview

API Auto Registration automatically generates API specifications from a workload's configuration and registration of the API entity in the Tanzu Developer Portal's catalog. You can access the registered API specification in Tanzu Developer Portal with no additional steps. As an experimental alpha feature, API Auto Registration generates a curated API by combining APIs exposed from multiple workloads.

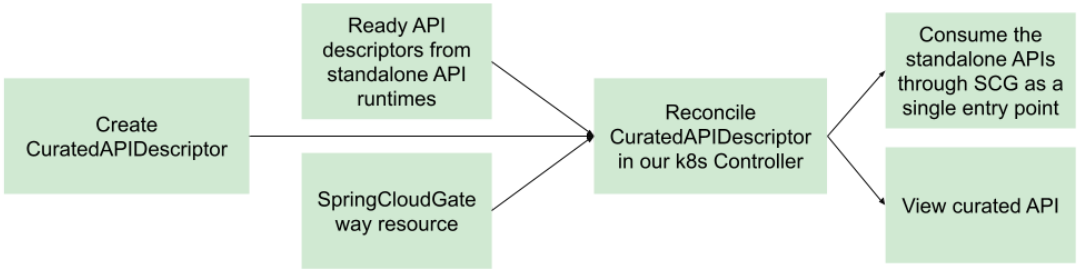
You can use an automated workflow with a supply chain to create and manage a Kubernetes Custom Resource (CR) of kind `APIDescriptor` from your workload. API Auto Registration's

Kubernetes controller periodically reconciles the CR and updates the API entity in Tanzu Developer Portal to achieve automated API specification registration from origin workloads.

You can also use API Auto Registration without supply chain automation, with other GitOps processes, or by directly applying an `APIDescriptor` CR to the cluster.



For combining and curating multiple standalone APIDescriptors in your run clusters, you can configure and apply `CuratedAPIDescriptor` to your clusters to provide a single curated entry point for users consuming your APIs. With Spring Cloud Gateway for Kubernetes available as the route provider, API Auto Registration's Kubernetes controller automatically generates and manages necessary routing resources for the curated API. If configured, each curated API is exposed and navigable on your preferred Spring Cloud Gateway instance.



Getting started

For information about API Auto Registration architecture, the `APIDescriptor` CR, the `CuratedAPIDescriptor` CR, and API entities in Tanzu Developer Portal, see Key Concepts.

For information about configuring iterate, run, and full Tanzu Application Platform cluster profiles, see Configure API Auto Registration.

For information about generating API specifications and registering them with Tanzu Developer Portal catalog, see Use API Auto Registration.

For information about curating workloads into Curated APIs and generating Spring Cloud Gateway resources, see API Curation (alpha).

For information about other profiles, install the `api-auto-registration` package. See Install API Auto Registration.

For information about troubleshooting and debugging API Auto Registration, see Troubleshooting.

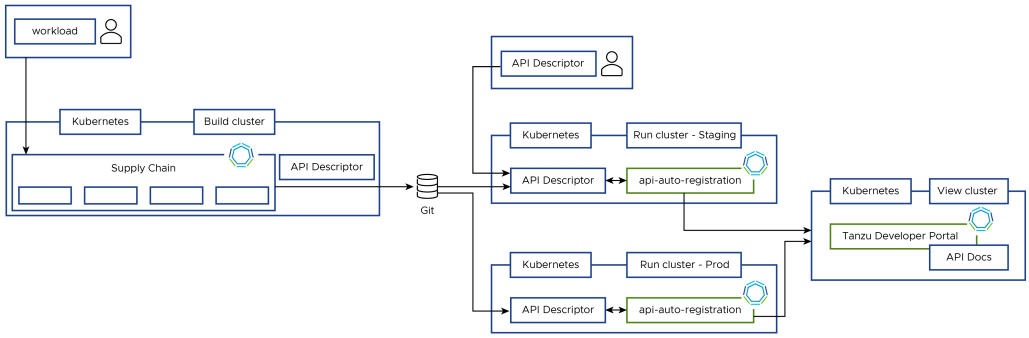
Key Concepts for API Auto Registration

This topic explains key concepts you use with API Auto Registration.

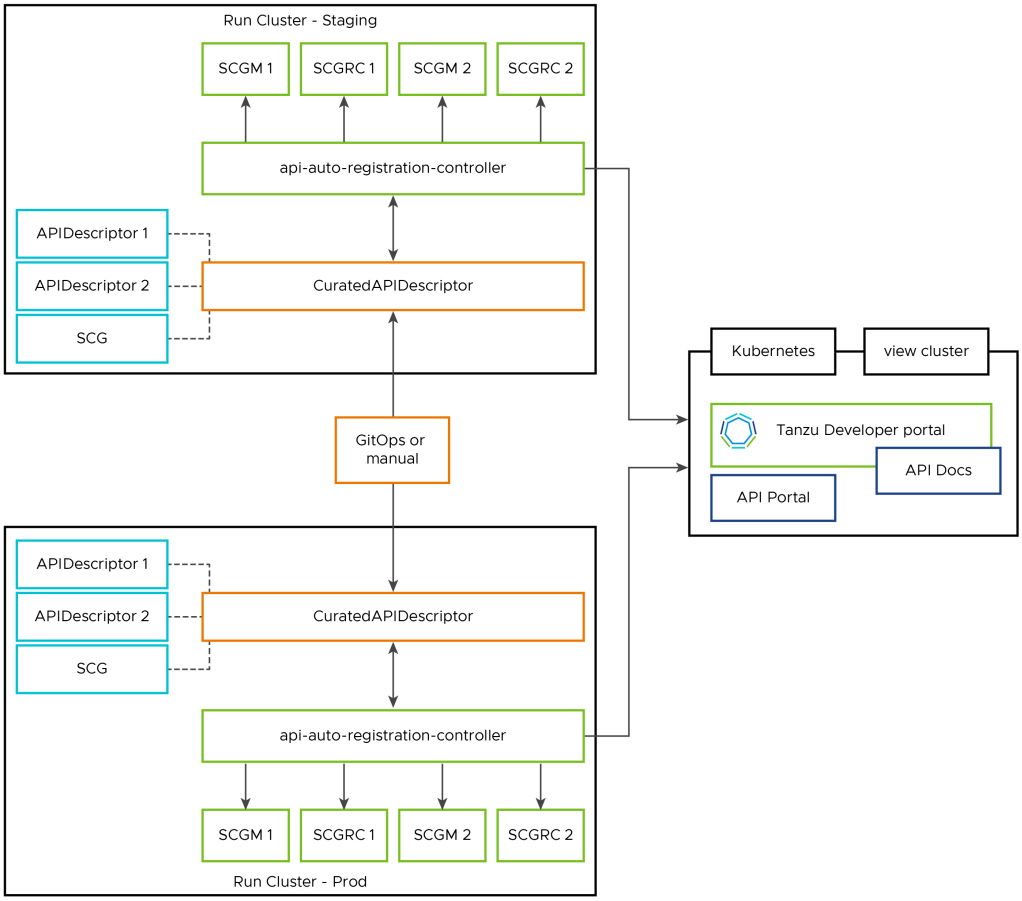
API Auto Registration architecture

You can use the full potential of API Auto Registration by using a distributed environment, as shown in the following diagrams:

- The workloads that expose APIs through the supply chains cause generated `APIDescriptors`. This triggers API Auto Registration's Kubernetes controller to generate and register API entities in Tanzu Developer Portal.



- Aggregate one or more `CuratedAPIDescriptor` into a curated API by using `APIDescriptors`. Optionally, this can trigger Spring Cloud Gateway routing resource generation for the referenced APIs.



APIDescriptor custom resource explained

To initiate API registration, the supply chain must create the custom resource of type `APIDescriptor` automatically or through other processes. The information from this custom resource constructs an API entity in Tanzu Developer Portal.

This custom resource exposes the following text boxes:

```
apiVersion: apis.apps.tanzu.vmware.com/v1alpha1
kind: APIDescriptor
metadata:
```

```
  name:                    # name of your APIDescriptor
  namespace:               # optional: namespace of your APIDescriptor
spec:
  type:                    # type of the API spec. oneOf(openapi, grpc, asyncapi, graph
ql)
  description:             # description for the API exposed
  system:                  # system that the API is part of
  owner:                   # person/team that owns the API
  location:
    apiSpec:
      path:                # sub-path where the API spec is available (previously `loca
tion.path`)
      url:                 # optional: static absolute base URL for the API spec
    server:                # base URL object where the API spec is available. oneOf(ur
l, ref) (previously `location.baseURL`)
      url:                 # optional: static absolute base URL for the API server
      ref:                 # optional: object ref to oneOf(HTTPProxy, Knative Service,
Ingress)
        apiVersion:
        kind:
        name:
        namespace:
```

The text boxes cause specific behavior in Tanzu Developer Portal:

- The system and owner are copied to the API entity. You might have to separately create and add the System and Group kind to the catalog.
- Tanzu Developer Portal uses the namespace for the API entity where the APIDescriptor CR is applied. This causes the API entity's name, system, and owner to all be in that namespace.
- To explicitly use a system or owner in a different namespace, you can specify that in the `system: my-namespace/my-other-system` or `owner: my-namespace/my-other-team` text boxes.
- If the system or owner you are trying to link doesn't have a namespace specified, you can qualify them with the `default` namespace. For example, `system: default/my-default-system`

Important

`spec.location.path` is deprecated in favor of `spec.location.apiSpec.path`, and `spec.location.baseURL` is deprecated in favor of `spec.location.server`. This change supports having a different API server location from the specification's location. These deprecated fields will be removed in Tanzu Application Platform 1.8.

With an absolute URL

To create an APIDescriptor with a static `server.url`, you must apply the following YAML to your cluster.

```
apiVersion: apis.apps.tanzu.vmware.com/v1alpha1
kind: APIDescriptor
metadata:
  name: sample-absolute-url
spec:
  type: openapi
  description: A set of API endpoints to manage the resources within the petclinic ap
p.
  system: spring-petclinic
```

```
  owner: team-petclinic
  location:
    apiSpec:
      path: "/v3/api-docs.yaml"
    server:
      url: https://myservice.mynamespace.svc.cluster.local:6789
```

With an object ref

You can use an object reference, instead of hard coding the URL, to point to a HTTPProxy, Knative Service, or Ingress. VMware does not support referencing Kubernetes `Service` with Object Ref. To point to your Kubernetes `Service` directly, you can use the static URL with cluster DNS address. For example, `https://myservice.mynamespace.svc.cluster.local:6789`.

With an HTTPPRoxy object ref

This section includes an example YAML that points to an HTTPProxy from which the controller extracts the `.spec.virtualhost.fqdn` as the baseURL.

```
apiVersion: apis.apps.tanzu.vmware.com/v1alpha1
kind: APIDescriptor
metadata:
  name: sample-contour-ref
spec:
  type: openapi
  description: A set of API endpoints to manage the resources within the petclinic ap
p.
  system: spring-petclinic
  owner: team-petclinic
  location:
    apiSpec:
      path: "/test/openapi"
    server:
      ref:
        apiVersion: projectcontour.io/v1
        kind: HTTPProxy
        name: my-httpproxy
        namespace: my-namespace # optional
```

With a Knative service object ref

To use a Knative Service, your controller reads the `status.url` as the baseURL. For example:

```
# all other fields similar to the above example
    server:
      ref:
        apiVersion: serving.knative.dev/v1
        kind: Service
        name: my-knative-service
        namespace: my-namespace # optional
```

With an ingress object ref

To use an Ingress instead, your controller reads the URL from the jsonPath specified. When jsonPath is left empty, your controller reads the `"{.spec.rules[0].host}"` as the URL. For example:

```
# all other fields similar to the above example
    server:
      ref:
```

```
        apiVersion: networking.k8s.io/v1
        kind: Ingress
        name: my-ingress
        jsonPath: "{.spec.rules[1].host}"
        namespace: my-namespace # optional
```

APIDescriptor status fields

When API Auto Registration processes an APIDescriptor, it adds several fields describing the progress to the `status`. Including `conditions`, which provides information useful for troubleshooting. For information about the conditions, see Troubleshoot API Auto Registration.

In addition to `conditions`, the `status` contains other useful fields showing the resolved API's details. The following is a list of these fields with a brief explanation of what they contain.

```
status:
  registeredEntityURL:    # Url of the corresponding API Entity in Tanzu Developer Por
tal
  registeredTapUID:       # Unique identifier for the corresponding API Entity in Tanz
u Developer Portal
  resolvedAPIServerURL:   # Url to the API runtime server
  resolvedAPISpecURL:     # Url used to retrieve the full API Spec by Api Auto Registr
ation
  resolvedAPISpec:        # Full API Spec as retrieved by Api Auto Registration
  resolvedAPISpecHash:    # Hash value of the `resolvedAPISpec`. This field can be use
d to see whether the spec has been updated or not.
  apiSpecLastUpdateTime:  # Timestamp representing the server time when API Spec was l
ast updated. It is not guaranteed to be set in happens-before order across separate op
erations. It is represented in RFC3339 form and is in UTC.
```

CuratedAPIDescriptor custom resource explained

To curate one or more Workload OpenAPI specifications into a single aggregated API, create a custom resource of type `CuratedAPIDescriptor`. The information from this custom resource references a list of APIDescriptors and how path-based routing aggregates them.

If you specify a valid route provider, for example, `spring-cloud-gateway` for Spring Cloud Gateway for Kubernetes (SCG), the API Auto Registration controller finds the SpringCloudGateway resource and automatically creates the following routing resources for you to expose your curated APIs as:

- SpringCloudGatewayRouteConfig (SCGRC): a custom resource that describes all the API endpoints and optional routing modifiers to access the endpoints. This is generated from the resolved OpenAPI Specification of the APIDescriptor through SCG OpenAPI conversion service.
- SpringCloudGatewayMapping (SCGM): a custom resource that binds a SCGRC resource to a SCG resource.

This custom resource exposes the following text boxes:

```
apiVersion: apis.apps.tanzu.vmware.com/v1alpha1
kind: CuratedAPIDescriptor
metadata:
  name:                 # name of your CuratedAPIDescriptor
  namespace:            # optional: namespace of your CuratedAPIDescriptor
  annotations:
    "apis.apps.tanzu.vmware.com/route-provider": "spring-cloud-gateway"   # specify ro
ute provider
spec:
  type: openapi         # type of the API spec. oneOf(openapi, grpc, asyncapi, graphq
l)
  title:                # title of the curated API
```

```
  description:          # description of the curated API spec
  documentation:        # documentation for the curated API spec
  groupId:              # groupID of the curated API.
  version:              # version of the curated API
  apiDescriptors:
    - name:             # name of a APIDescriptor to include in this curated API
      namespace:        # namespace of the APIDescriptor
      pathPrefix:       # Path prefix that the API endpoints from the linked APIDescri
ptor should have
      routeConfig:
        ssoEnabled:     # whether to enable SSO on the gateway to perform authenticati
on for users
        tokenRelay:     # whether to enable TokenRelay on the gateway to pass along th
e SSO ID token to the upstream service. This setting depends on `ssoEnabled`.
        filters:        # additional service lever filters to set on all the endpoints
in this API
    - ...
```

There are some key behaviors generated from the text boxes:

- The `apis.apps.tanzu.vmware.com/route-provider` annotation specified how you want to provide routing to the curated API. VMware only supports `spring-cloud-gateway`.
- `groupId` is a concept that's aligned with API portal to group APIs from different high-availability zones/locations or with different `version`.
- `groupId` and `version` identify a matching gateway that route traffic for the curated API
- `routeConfig` section specifies service level configuration you add when generating the routing resource for the API. For information about spring-cloud-gateway fields, see OpenAPI route conversion.
    - Prior to Spring Cloud Gateway (SCG) for Kubernetes v2.1.3, there is a known issue in the SCG OpenAPI Conversion Service to support adding token relay at the service level. The `tokenRelay: true` setting only works with SCG v2.1.3 and above.
    - `routeConfig.filters` section specifies service level filters for all the routes exposed in each API. You can add modifications to your endpoints, such as `RateLimit=5,10s` or `RemoveRequestHeader=X-Request-Foo`. For information about available filters, see SCG commercial filters.
    - Your controller automatically prepends each endpoint path with the `pathPrefix` you specified for each APIDescriptor, and adds the `StripPrefix` filter to the end of the filter list to facilitate a successful path-based redirect. Additionally, you can add more `StripPrefix` filters to the service level filters to skip common paths in your specifications that are not in your service.

CuratedAPIDescriptor status fields

When processing an CuratedAPIDescriptor, several fields are added to the `status`. One of these is `conditions`, which provide information useful for troubleshooting. The conditions are explained in the Troubleshooting Guide.

In addition to `conditions` the `status` contains a couple of other useful fields. The following is a list of these fields with a brief explanation of what they contain.

```
status:
  ResolvedBaseURL:        # Base url to access the curated API. Empty if no route prov
ider is configured.
  ResolvedAPISpecURL:     # Url to the generated aggregated API spec
  AggregatedAPISpec:      # Generated full API Spec as aggregated by Api Auto Registra
tion
  AggregatedAPISpecHash:  # Hash of the aggregated API Spec. This field can be used to
see whether the spec has been updated or not.
```

```
  LastUpdateTime:         # Timestamp representing the server time when API Spec was l
ast updated. It is not guaranteed to be set in happens-before order across separate op
erations. It is represented in RFC3339 form and is in UTC.
  MatchedGateway:         # The Gateway resource the curated API is matched on. We cur
rently only support SCG.
```

Install API Auto Registration

This topic describes how you can install API Auto Registration from the Tanzu Application Platform package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install API Auto Registration. For information about profiles, see Components and installation profiles.

Tanzu Application Platform prerequisites

Before installing API Auto Registration, complete all prerequisites to install Tanzu Application Platform. See Tanzu Application Platform Prerequisites.

Using with TLS

Starting in Tanzu Application Platform v1.4, TLS is turned on by default for several components. API Auto Registration automatically trusts the CA for the shared `ingress_issuer` when using the default ClusterIssuer `tap-ingress-selfsigned`. This change means that a `Certificate` is automatically generated using this issuer.

If you do not want a `Certificate` to generate automatically, you can set the `auto_generate_cert` flag to `false` in the values file. To replace the default with a custom ingress issuer, see Security and compliance. Whenever you do not use the default ClusterIssuer `tap-ingress-selfsigned`, do not automatically generate certificates, or use other custom CAs, you must manually set the certificate. See Troubleshooting.

Install

To install the API Auto Registration package:

1. List version information for the package by running:

   ```
   tanzu package available list apis.apps.tanzu.vmware.com --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package available list apis.apps.tanzu.vmware.com --namespace tap-insta
   ll
   - Retrieving package versions for apis.apps.tanzu.vmware.com...
     NAME                                VERSION  RELEASED-AT
     apis.apps.tanzu.vmware.com  0.1.0        2022-08-30 19:00:00 -0500 -05
     apis.apps.tanzu.vmware.com  0.2.0        2022-11-24 12:20:00 -0500 -05
   ```

2. (Optional) Gather values schema.

   Display values schema of the package:

```
tanzu package available get apis.apps.tanzu.vmware.com/VERSION-NUMBER --values-
schema --namespace tap-install
```

Where `VERSION-NUMBER` is the version of the package you retrieved.

For example:

```
tanzu package available get apis.apps.tanzu.vmware.com/0.4.0 --values-schema --
namespace tap-install

Retrieving package details for apis.apps.tanzu.vmware.com/0.4.0...
KEY                                                         DEFAULT
TYPE     DESCRIPTION
ca_cert_data
string   Optional: PEM-encoded certificate data for the controller to trust TL
S.
ingress_issuer
string   Optional: Name of the default cluster issuer used to generate certific
ates
auto_generate_cert                                          true
boolean  Flag that indicates if a cert-manager certificate should be generated
using the ingress_issuer. Only applies if the ingress_issuer is specified conne
ctions with a custom CA
cluster_name                                                dev
string   Name of the cluster used for setting the API entity lifecycle in TAP G
UI. The value should be unique for each run cluster.
route_provider.spring_cloud_gateway.enabled                 false
boolean  Flag that indicates if VMware Spring Cloud Gateway for Kubernetes is s
upported as route provider
route_provider.spring_cloud_gateway.scg_openapi_service_url  http://scg-openapi
-service.spring-cloud-gateway.svc.cluster.local  string   FQDN URL for SCG oper
ator openapi conversion service
sync_period                                                 5m
string   Time period used for reconciling an APIDescriptor
cluster_name                                                dev
string   Name of the cluster that will be used for setting the API entity lifec
ycle in TAP GUI. The value should be unique for each run cluster.
curated_api_server.port                                     8082
integer
curated_api_server.service_type                             ClusterIP
string
sync_period                                                 5m
string   Time period used for reconciling an APIDescriptor.
tap_gui_url                                                 http://server.tap-
gui.svc.cluster.local:7000                        string   FQDN URL for TAP GUI.
replicas                                                    1
integer  Number of controller replicas to deploy.
resources.limits.cpu                                        500m
string   CPU limit of the controller.
resources.limits.memory                                     500Mi
string   Memory limit of the controller.
resources.requests.cpu                                      20m
string   CPU request of the controller.
resources.requests.memory                                   100Mi
string   Memory request of the controller.
logging_profile                                             production
string   Logging profile for controller. If set to development, use console log
ging with full stack traces, else use JSON logging.
```

3. Locate the Tanzu Developer Portal (formerly Tanzu Application Platform GUI) URL.

   When running a full profile on a Tanzu Application Platform cluster, the default value of Tanzu Developer Portal URL is sufficient. You can edit this to match the externally available

FQDN of Tanzu Developer Portal to display the entity URL in the externally accessible APIDescriptor status.

When installed in a run cluster or with a profile where Tanzu Developer Portal is not installed in the same cluster, you must set the `tap_gui_url` parameters correctly for successful entity registration with Tanzu Developer Portal.

You can locate the `tap_gui_url` by going to the view cluster with the Tanzu Developer Portal you want to register the entity with:

```
kubectl get secret tap-values -n tap-install -o jsonpath="{.data['tap-values\.y
aml']}" | base64 -d | yq '.tap_gui.app_config.app.baseUrl'
```

4. (Optional) VMware recommends creating `api-auto-registration-values.yaml`.

   To overwrite the default values when installing the package, create a `api-auto-registration-values.yaml` file:

   ```
   tap_gui_url: https://tap-gui.view-cluster.com
   cluster_name: staging-us-east
   ca_cert_data:  |
       -----BEGIN CERTIFICATE-----
       MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...
       -----END CERTIFICATE-----
   sync_period: 2m
   ```

5. Install the package using Tanzu CLI:

   ```
   tanzu package install api-auto-registration
   --package apis.apps.tanzu.vmware.com
   --namespace tap-install
   --version VERSION-NUMBER
   --values-file api-auto-registration-values.yaml
   ```

   Where `VERSION-NUMBER` is the version of the package you retrieved in the earlier step.

6. Verify the package installation by running:

   ```
   tanzu package installed get api-auto-registration -n tap-install
   ```

   Verify that `STATUS` is `Reconcile succeeded`:

   ```
   kubectl get pods -n api-auto-registration
   ```

7. Verify that applying an APIDescriptor resource to your cluster causes the `STATUS` showing `Ready`:

   ```
   kubectl apply -f - <<EOF
   apiVersion: apis.apps.tanzu.vmware.com/v1alpha1
   kind: APIDescriptor
   metadata:
     name: sample-api-descriptor-with-absolute-url
   spec:
     type: openapi
     description: A sample APIDescriptor to validate package installation successf
   ul
     system: test-installation
     owner: test-installation
     location:
       apiSpec:
         path: "/api/v3/openapi.json"
       server:
   ```

```
      url: https://petstore3.swagger.io
EOF
```

Verify that the APIDescriptor status shows `Ready`:

```
kubectl get apidescriptors
NAME                                        STATUS
sample-api-descriptor-with-absolute-url     Ready

kubectl get apidescriptors -owide
NAME                                        STATUS    TAP GUI ENTITY URL     API
SPEC URL
sample-api-descriptor-with-absolute-url     Ready     <url-to-the-entity>    <ur
l-to-the-api-spec>
```

If the status does not show `Ready`, you can inspect and show the reason by running:

```
kubectl get apidescriptor sample-api-descriptor-with-absolute-url -o jsonpath
='{.status.conditions[?(@.type=="Ready")].message}'
```

Verify that the entity is created in your Tanzu Developer Portal: `TAP-GUI-URL/catalog/default/api/sample-api-descriptor-with-absolute-url`

Configure API Auto Registration

This topic describes configuration options for API Auto Registration.

Update install values for api-auto-registration package

You can view the values available for the package by finding the available package version and finding the corresponding schema. After you select the values that you want to update, update them either by updating the `tap` package or by updating the `api-auto-registration` package if you installed it on its own.

1. Find the version of the available package:

```
tanzu package available list apis.apps.tanzu.vmware.com -n tap-install

/ Retrieving package versions for apis.apps.tanzu.vmware.com...
NAME                        VERSION  RELEASED-AT
apis.apps.tanzu.vmware.com  0.4.0    2023-10-31 13:47:14 -0400 EDT
```

2. Explore the values available:

```
tanzu package available get apis.apps.tanzu.vmware.com/VERSION-NUMBER --values-
schema --namespace tap-install
```

Where `VERSION-NUMBER` is the version of the package listed in the earlier step.

For example:

```
tanzu package available get apis.apps.tanzu.vmware.com/0.4.0 --values-schema --
namespace tap-install

Retrieving package details for apis.apps.tanzu.vmware.com/0.4.0...
KEY                       DEFAULT                                          TYPE
DESCRIPTION
ca_cert_data                                                               string
Optional: PEM-encoded certificate data for the controller to trust TLS.
ingress_issuer                                                             string
Optional: Name of the default cluster issuer used to generate certificates
```

```
auto_generate_cert         true                                          boolea
n  Flag that indicates if a cert-manager certificate should be generated using
the ingress_issuer. Only applies if the ingress_issuer is specified
connections with a custom CA
cluster_name               dev                                           string
Name of the cluster used for setting the API entity lifecycle in TAP GUI. The v
alue should be unique for each run cluster.
sync_period                5m                                            string
Time period used for reconciling an APIDescriptor.
tap_gui_url                http://server.tap-gui.svc.cluster.local:7000  string
FQDN URL for TAP GUI.
replicas                   1                                             intege
r  Number of controller replicas to deploy.
resources.limits.cpu       500m                                          string
CPU limit of the controller.
resources.limits.memory    500Mi                                         string
Memory limit of the controller.
resources.requests.cpu     20m                                           string
CPU request of the controller.
resources.requests.memory  100Mi                                         string
Memory request of the controller.
logging_profile            production                                    string
Logging profile for controller. If set to development, use console logging with
full stack traces, else use JSON logging.
```

3. Create `api-auto-registration-values.yaml` if you installed the API Auto Registration package on its own.

4. Update the tap-values.yaml if you used a package to install.

   A `api-auto-registration-values.yaml` file might contain the following:

   ```
   tap_gui_url: https://tap-gui.view-cluster.com
   cluster_name: staging-us-east
   ca_cert_data:  |
       -----BEGIN CERTIFICATE-----
       MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...
       -----END CERTIFICATE-----
   ```

   A `tap-values.yaml` file might contain the following values, in addition to other values.

   ```
   shared:
       ingress_domain: "INGRESS-DOMAIN"
       ingress_issuer: # Optional: can denote a cert-manager.io/v1/ClusterIssuer o
   f your choice. Defaults to "tap-ingress-selfsigned".
       ca_cert_data: | # To be passed if using custom certificates.
           -----BEGIN CERTIFICATE-----
           MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...
           -----END CERTIFICATE-----
   api_auto_registration:
       tap_gui_url: https://tap-gui.view-cluster.com
       cluster_name: staging-us-east
   ```

   See the Full Profile sample.

5. If you installed the API Auto Registration package on its own, not as part of Tanzu Application Platform, update the package using the Tanzu CLI:

   ```
   tanzu package installed update api-auto-registration
   --package apis.apps.tanzu.vmware.com
   --namespace tap-install
   --version $VERSION
   --values-file api-auto-registration-values.yaml
   ```

6. If you installed API Auto Registration as part of Tanzu Application Platform, see Upgrading Tanzu Application Platform.

Use API Auto Registration

This topic describes how you can use API Auto Registration.

Overview

The run profile requires you to update the install values before proceeding. For iterate and full profiles, the default values work but you might prefer to update them. For information about profiles, see About Tanzu Application Platform profiles.

Prerequisites

API Auto Registration requires the following:

1. A location exposing a dynamic or static API specification.
2. An APIDescriptor Custom Resource (CR) with that location created in the cluster.
3. (Optional) Configurations:
    1. Cross-Origin Resource Sharing (CORS) for OpenAPI specifications
    2. Connect the component to the auto-registered API in Tanzu Developer Portal

Configurations

To generate OpenAPI Spec:

- By creating a simple Spring Boot app
- By scaffolding a new project using App Accelerator Template
- In an existing Spring Boot project

To create APIDescriptor Custom Resource:

- Using Out Of The Box Supply Chains
- Using Custom Supply Chains
- Using other GitOps processes or Manually

Additional configurations:

- CORS for viewing OpenAPI Spec in Tanzu Developer Portal
- Connection between the Component and API in Tanzu Developer Portal

Generate OpenAPI specifications

This section tells you how to generate OpenAPI specifications.

Using a Spring Boot app with a REST service

You can use a Spring Boot example app built using Building a RESTful Web Service guide. and has the Springdoc dependency.

Example of a workload using the Spring Boot app:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: simple-rest-app
  labels:
    ...
    apis.apps.tanzu.vmware.com/register-api: "true"
spec:
  source:
    ...
  params:
    - name: api_descriptor
      value:
        type: openapi
        location:
          path: "/v3/api-docs"
        system: dev
        owner: team-a
        description: "A set of API endpoints."
```

Using App Accelerator template

If you are creating a new application exposing an API, you might use the java-rest-service App Accelerator template to get a pre-built app that includes a `workload.yaml` with a basic REST API. From your Tanzu Developer Portal Accelerators tab, search for the accelerator and scaffold it according to your needs.

Using an existing Spring Boot project using springdoc

If you have an existing Spring Boot app that exposes an API, you can generate OpenAPI specifications using springdoc. See the springdoc documentation

After you have springdoc configured and an OpenAPI automatically generated, you can choose one of the following methods to create the APIDescriptor custom resource:

- VMware recommends having your Spring Boot app to be managed using Workloads and the Out-Of-The-Box (OOTB) supply chain. See the Use Out-Of-The-Box (OOTB) supply chains for further instructions.
- To use custom supply chains, see Using Custom Supply Chains. -To use a different Gitops process or manage the APIDescriptor CR manually, see the Using other GitOps processes or Manually section.

Create APIDescriptor custom resource

This section tells you how to create an APIDescriptor custom resource.

Use Out-Of-The-Box (OOTB) supply chains

All the Out-Of-The-Box (OOTB) supply chains are modified so that they can use API Auto Registration. If you want your workload to be auto registered, you must make modifications to your workload YAML:

1. Add the label `apis.apps.tanzu.vmware.com/register-api: "true"`.
2. Add a parameter of `type api_descriptor`:

```
params:
  - name: api_descriptor
    value:
      type: openapi   # We currently support any of openapi, aysncapi, graphq
```

```
l, grpc
        location:
          path: "/v3/api-docs"  # The path to the api documentation
          baseURL: "http://my-spec.url" # Optional: The base URL to the api doc
umentation if not served from the API runtime
        owner: team-petclinic   # The team that owns this
        system: petclinic       # The Backstage system entity this API belongs
to
        description: "A set of API endpoints to manage the resources within the
petclinic app."
```

The default supply chains use Knative to deploy your applications and are referenced as the server location for the generated APIDescriptor.

For API specifications, there are 2 options for the location:

- If the API specifications are generated and served from your application endpoint, the only location information you must set is the path to the API documentation. The controller can figure out the base URL by using the Knative server reference.
- If your API specifications are served from a static location, you can hardcode the URL using the `location.baseURL` property.

Example workload that exposes the API specifications from a Knative service:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: petclinic-knative
  labels:
    ...
    apis.apps.tanzu.vmware.com/register-api: "true"
spec:
  source:
    ...
  params:
    - name: api_descriptor
      value:
        type: openapi
        location:
          path: "/v3/api-docs"
        system: pet-clinics
        owner: team-petclinic
        description: "A set of API endpoints to manage the resources within the petcli
nic app."
```

Example of a workload with a hardcoded URL to the API documentation:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: petclinic-hard-coded
  labels:
    ...
    apis.apps.tanzu.vmware.com/register-api: "true"
spec:
  source:
    ...
  params:
    - name: api_descriptor
      value:
        type: openapi
        location:
          baseURL: http://petclinic-hard-coded.my-apps.tapdemo.vmware.com/
```

```
          path: "/v3/api-docs"
        owner: team-petclinic
        system: pet-clinics
        description: "A set of API endpoints to manage the resources within the petcli
nic app."
```

After the supply chain runs, it creates an `APIDescriptor` custom resource. Tanzu Application Platform uses this resource to auto register your API. See APIDescriptor explained.

Using custom supply chains

If you are creating custom supply chains, you can still use API Auto Registration. To write a supply chain pipeline, use `ClusterConfigTemplate` by the name of `config-template` in your pipeline. To write a custom task, verify how the template is written to read parameters, interpret baseURL from Knative Services, and construct APIDescriptor CRs.

In the Delivery pipeline, you must directly create an APIDescriptor custom resource. You must grant permissions to create the CR from the delivery pipeline.

For information about APIDescriptors, see APIDescriptor explained.

Using other GitOps processes or manually

Using your GitOps process, or manually, you must stamp out an APIDescriptor CR and apply it in the cluster you choose. Specify all the required fields for an APIDescriptor CR to reconcile.

For information about APIDescriptors, see APIDescriptor explained.

Additional configuration

This section tells you how to perform additional configuration.

Setting up CORS for OpenAPI specifications

The agent, usually a browser, uses the CORS protocol to verify whether the current origin uses an API. To use the `Try it out`` feature for OpenAPI specifications from the API Documentation plug-in, you must configure CORS to allow successful requests.

Your API must be configured to allow CORS Requests from Tanzu Developer Portal. How you accomplish this varies based on your programming language and framework. If you are using Spring, see CORS support in spring framework.

At a high level, your API must accept the Tanzu Developer Portal domain must be accepted as a valid cross-origin.

To do this:

- **Origins allowed** header: `Access-Control-Allow-Origin`: A list of comma-separated values. This list must include your Tanzu Developer Portal host.
- **Methods allowed** header: `Access-Control-Allow-Method`: Must allow the method used by your API. Also confirm that your API supports preflight requests, a valid response to the OPTIONS HTTP method.
- **Headers allowed** header: `Access-Control-Allow-Headers`: If the API requires any header, you must include it in the API configuration or your authorization server.

Connecting the component to the API in Tanzu Developer Portal

When the API-producing component is added to the Tanzu Developer Portal catalog, it is helpful to connect the component to the API. By connecting the two entities, the catalog graph visualizes

this relationship for users. To connect the entities you must add the `.spec.providesApis` in your component.yaml where you can list all the APIs it provides using string reference format. Ensure that you follow the `namespace/name` format instead of `name`, otherwise the namespace defaults to the component's namespace, typically `default`. The name is the auto-registered API's name. VMware recommends updating the `component.yaml` with the API name after the auto-registration finishes.

The following is an example of a `component.yaml` with string entity reference to an API:

```
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: petclinic
  namespace: petclinic-systems
  description: Petstore
spec:
  type: service
  lifecycle: dev
  owner: team-petclinic
  providesApis:
    - internal/petclinic-api-dev
```

For more information, see the API documentation plug-in in Tanzu Developer Portal.

API Curation (alpha)

This topic tells you how to use API Curation to expose several standalone APIs as one API for API Auto Registration.

Caution

The API Curation feature is in alpha and is intended for evaluation and test purposes only. VMware discourages using API Curation in a production environment.

Overview

To unlock the maximum power of API curation, VMware recommends using a supported route provider. Without this a supported route provider, the generated API specifications do not have a functional server URL set for testing the curated API. You must manually create the routing resources to route traffic to each referenced API to match with the aggregated API specifications. With the route provider integration, these routing concerns are taken care of automatically.

Prerequisites

A successful API curation requires the following prerequisites:

- (Optional) Spring Cloud Gateway for Kubernetes is installed.
- (Optional) `SpringCloudGateway` resources created with the matching `groupId` and `version`.
- One or more `APIDescriptor`s in the ready state.
- Create a `CuratedAPIDescriptor` resource referencing ready `APIDescriptor`s.

You can get the aggregated API specifications from the OpenAPI endpoint from the controller.

(Optional) Install route provider and create gateway resources

You can optionally install a route provider and create gateway resources.

Setup Spring Cloud Gateway integration

Using Spring Cloud Gateway for Kubernetes (SCG) integration as a route provider is optional. To install SCG, see Install Spring Cloud Gateway for Kubernetes.

For SCG v2.1.0 and later, if you enabled TLS on SCG, or installed it in a custom namespace, you must overwrite `route_provider.spring_cloud_gateway.scg_openapi_service_url` in your API Auto Registration values file.

```
route_provider:
  spring_cloud_gateway:
    scg_openapi_service_url: "http://scg-openapi-service.spring-cloud-gateway.svc.clus
ter.local" # default value
```

If you are using an earlier version of SCG, consider upgrading or see the following table to understand the impact:

Capability with SCG v2.1.0 and later	Behavior before SCG v2.1.0
The default value `scg_openapi_service_url` is sufficient if using the default SCG installation	You must overwrite the value `scg_openapi_service_url` with `http://scg-operator.tap-install.svc.cluster.local` if using the default SCG installation.
Matching SCG updates API metadata automatically and the generated OpenAPI specifications reflect the metadata	API metadata annotations are added or updated, but the API specifications exposed from SCG OpenAPI endpoint do not reflect that

Create SpringCloudGateway resource

You can create, using GitOps or manually, an `SpringCloudGateway` CR and apply it in the cluster you choose.

The following is an example SCG resource:

```
apiVersion: "tanzu.vmware.com/v1"
kind: SpringCloudGateway
metadata:
  name: test-api-curation
spec:
  api:
    version: 1.2.3
    groupId: test-api-curation
    serverUrl: https://my-curated-api.mydomain.com
    cors:
      allowedOrigins:
        - "http://api-portal.mydomain.com"
      allowedMethods:
        - "GET"
        - "PUT"
        - "POST"
      allowedHeaders:
        - '*'
```

The `groupId` and `version` matches on SCG when it reconciles for `CuratedAPIDescriptor`. After finding a match, SCG uses `serverUrl` as the baseURL of the curated API.

Create APIDescriptors for curation

To create `APIDescriptor` resources, see API Auto Registration Usage Guide. If any of the referenced `APIDescriptor` are not ready, the `CuratedAPIDescriptor` keeps retrying until all the

referenced `APIDescriptor`s are ready. If the API specifications from any of the `APIDescriptor` update, the controller selects the changes on the next reconciliation loop.

Note By default, the update might take up to 10 minutes, including maximum 5 minutes to refresh the `APIDescriptor`, and maximum 5 minutes to refresh `CuratedAPIDescriptor`. You can shorten the wait time by configuring `sync_period` in the AAR values file.

Create CuratedAPIDescriptor custom resource

Using your GitOps process, or manually, you can create an CuratedAPIDescriptor CR and apply it in the cluster you choose. Specify all the required fields for an CuratedAPIDescriptor CR to reconcile.

For information about CuratedAPIDescriptors, see CuratedAPIDescriptor explained.

You can view the readiness of the applied CuratedAPIDescriptors by running:

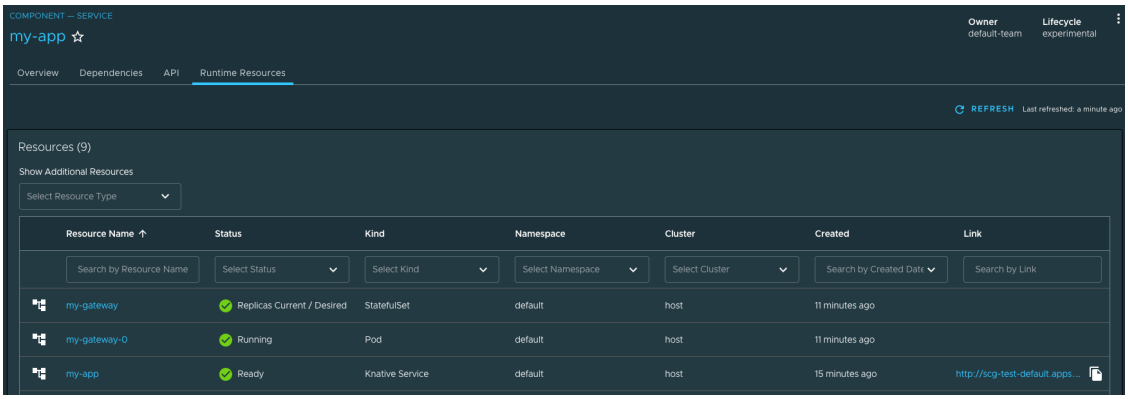
```
kubectl get curatedapidescriptors -A

NAMESPACE           NAME         GROUPID            VERSION   STATUS   CURATED API SPE
C URL
my-apps             petstore     cute-api-group     1.2.3     Ready    http://AAR-CONT
ROLLER-FQDN/openapi/my-apps/petstore
```

View the auto-registered API within Tanzu Developer Portal

After you the CuratedAPIDescriptor you created is in the `Ready` state, navigate to Tanzu Developer Portal to view the automatically registered APIs. You will see one new API for the CuratedAPIDescriptor and the other APIs generated from APIDescriptors.

Additionally, you will see a component registered for the curated API. If you chose to use Spring Cloud Gateway as the route provider in Setup Spring Cloud Gateway integration earlier, this component can show any gateway resources with the matching `app.kubernetes.io/part-of={PART_OF}` label under the **Runtime Resources**. To make this connection, add the `app.kubernetes.io/part-of={PART_OF}` label to the `SpringCloudGateway` instance you create. For more information, see the Spring Cloud Gateway documentation.



Retrieve curated API specifications

The API Auto Registration controller offers an endpoint to retrieve all the generated API specifications for the curated APIs in the cluster.

To retrieve curated API specifications:

1. Find the HTTPProxy that's created to access the endpoint:

```
kubectl get httpproxy api-auto-registration-controller -n api-auto-registration
```

The following is an example output:

```
NAME                                FQDN                                TLS SECRET
STATUS   STATUS DESCRIPTION
api-auto-registration-controller   api-auto-registration.tap.domain  api-auto-r
egistration-cert   valid    Valid HTTPProxy
```

2. To get all the curated API specifications with curl, use FQDN with an http scheme:

   ```
   curl http(s)://AAR-CONTROLLER-FQDN/openapi
   ```

   Where `AAR-CONTROLLER-FQDN` is the AAR FQDN controller you want the specifications for.

3. Retrieve specifications for `CuratedAPIDescriptor` from a specific namespace by specifying the following path:

   ```
   curl http(s)://AAR-CONTROLLER-FQDN/openapi/NAMESPACE
   ```

   Where:

   - `NAMESPACE` is the namespace that you want to retrieve specifications from.

4. Retrieve specifications for a specific `CuratedAPIDescriptor` by `namespace` and `name`:

   ```
   curl http(s)://AAR-CONTROLLER-FQDN/openapi/NAMESPACE/NAME
   ```

   Where:

   - `NAMESPACE` is the namespace of the `CuratedAPIDescriptor` that you want to use.
   - `NAME` is name of the `CuratedAPIDescriptor` that you want to use.

5. Retrieve specifications for a specific `groupId` and `version` combination by specifying query parameters:

   ```
   curl http(s)://AAR-CONTROLLER-FQDN/openapi?groupId=GROUP-ID&version=VERSION
   curl http(s)://AAR-CONTROLLER-FQDN/openapi/NAMESPACE?groupId=GROUP-ID&version=V
   ERSION
   ```

   Where:

   - `NAMESPACE` is the namespace that you want to retrieve specifications from.
   - `VERSION` is the version you want the specifications for.
   - `GROUP-ID` is group ID you want the specifications for.
   - `AAR-CONTROLLER-FQDN` is the AAR-CONTROLLER-FQDN you want to query.

6. Add the curated APIs to an API portal for display by configuring the source URL locations of an existing API portal. You can add all your curated APIs by using the unfiltered URL `http(s)://AAR-CONTROLLER-FQDN/openapi` or the filtered URL with query parameters to add a specific curated API of your choice. See Configuring API portal for VMware Tanzu on Kubernetes.

7. This curated API is not automatically registered in Tanzu Developer Portal. You can do this manually by creating an `api.yaml` and adding it to the catalog. For information about the structure of the definition file for an API entity, see the API documentation plug-in in Tanzu Developer Portal or the Backstage Kind: API documentation.

Troubleshoot API Auto Registration

This topic contains ways that you can troubleshoot API Auto Registration.

Debug API Auto Registration

This section includes commands for debugging or troubleshooting the APIDescriptor Custom Resource (commonly known as CR).

1. Get the details of CRs.

   ```
   kubectl get apidescriptor NAME -n NAMESPACE -owide
   kubectl get curatedapidescriptor NAME -n NAMESPACE -owide
   ```

   Where:

   - `NAME` is the name of the CR you want to debug.
   - `NAMESPACE` is the namespace associated with the CR you want to debug.

2. Find the status of the CRs.

   ```
   kubectl get apidescriptor NAME -n NAMESPACE -o jsonpath='{.status.conditions}'
   kubectl get curatedapidescriptor NAME -n NAMESPACE -o jsonpath='{.status.condit
   ions}'
   ```

   Where:

   - `NAME` is the name of the CR you want to debug.
   - `NAMESPACE` is the namespace associated with the CR you want to debug.

3. Find the generated SpringCloudGatewayRouteConfig (commonly known as SCGRC) and SpringCloudGatewayMapping (commonly known as SCGM) resource associated with a specific APIDescriptor CR from curation.

   The generated Spring Cloud Gateway (commonly known as SCG) resources are placed in the same namespace as the CuratedAPIDescriptor, and the generated name has a prefix with the name of the CuratedAPIDescriptor. To see which APIDescriptor the resource was generated for, you can list generated SCG resources with additional labels by running:

   ```
   kubectl get scgrc -A -L apis.apps.tanzu.vmware.com/api-descriptor-name -L apis.
   apps.tanzu.vmware.com/api-descriptor-namespace
   kubectl get scgm -A -L apis.apps.tanzu.vmware.com/api-descriptor-name -L apis.a
   pps.tanzu.vmware.com/api-descriptor-namespace
   ```

   Sample output:

   ```
   NAMESPACE      NAME                AGE   API-DESCRIPTOR-NAME   API-DESCRIPTOR-N
   AMESPACE
   api-curation   petstore-4a8d2a4f   49s   turtle-api            turtle-ns
   api-curation   petstore-ce551c65   49s   cat-api               cat-ns
   api-curation   petstore-d0243a39   49s   dog-api               dog-ns
   ```

4. Read logs from the `api-auto-registration` controller.

   ```
   kubectl -n api-auto-registration logs deployment.apps/api-auto-registration-con
   troller
   ```

5. Patch an APIDescriptor that is stuck in Deleting mode.

This might happen if the controller package is uninstalled before you clean up the APIDescriptor resources. You can reinstall the package and delete all the APIDescriptor resources first, or run for each stuck APIDescriptor resource.

```
kubectl patch apidescriptor API-DESCRIPTOR-NAME -p '{"metadata":{"finalizers":n
ull}}' --type=merge
```

Where `API-DESCRIPTOR-NAME` is the name of the API descriptor you want to patch.

Note

If you manually remove the finalizers from the APIDescriptor resources, you can have stale API entities within Tanzu Developer Portal that you must manually deregister.

6. Fix a `CuratedAPIDescriptor` that does not match with a SCG.

   This might happen if the `groupId` and version of the `CuratedAPIDescriptor` does not match any available `SpringCloudGateway` resource. You can remove the `"apis.apps.tanzu.vmware.com/route-provider": "spring-cloud-gateway"` annotation from your `CuratedAPIDescriptor` to skip SCG matching, or ensure that you have a matching SCG applied to the cluster.

APIDescriptor CRD shows message of `connection refused` but service is up and running

In Tanzu Application Platform v1.4 and later, if your workloads use ClusterIssuer for the TLS configuration or your API specifications location URL is secured using a custom CA, you might encounter the following message.

Your APIDescription CRD shows a status and message similar to:

```
    Message:               Get "https://spring-petclinic.example.com/v3/api-docs": dia
l tcp 12.34.56.78:443: connect: connection refused
    Reason:                FailedToRetrieve
    Status:                False
    Type:                  APISpecResolved
    Last Transition Time:  2022-11-28T09:59:13Z
```

This might be due to your workloads using a custom Ingress issuer. To solve this issue, either:

- Configure `ca_cert_data` following the instructions in Configure CA Cert Data.
- Deactivate TLS by setting `shared.ingress_issuer: ""`. VMware discourages this method. Deactivating TLS reduces your ability to test plug-in capability and iterate quickly.

Configure CA Cert Data

1. Obtain the PEM Encoded crt file for your ClusterIssuer or TLS setup. You use this to update the `api-auto-registration` package.
2. If you installed the API Auto Registration package through predefined profiles, you must update the `tap-values.yaml` and update the Tanzu Application Platform installation. Place the PEM encoded certificate into the `shared.ca_cert_data` key of the values file. See Install your Tanzu Application Platform profile. Run to update the package:

   ```
   tanzu package installed update tap -p tap.tanzu.vmware.com -v TAP-VERSION  --va
   lues-file tap-values.yaml -n tap-install
   ```

Where `TAP-VERSION` is the version of Tanzu Application Platform installed.

3. If you installed the API Auto Registration package as standalone, you must update the `api-auto-registration-values.yaml` and then update the package. Place the PEM encoded certificate into the `ca_cert_data` key of the values file. Run to update the package.

```
tanzu package installed update api-auto-registration --version API-AUTO-REGISTR
ATION-VERSION --namespace tap-install --values-file api-auto-registration-value
s.yaml
```

Where `API-AUTO-REGISTRATION-VERSION` is the version of API Auto Registration installed.

You can find the available api-auto-registration versions by running:

```
tanzu package available list -n tap-install | grep 'API Auto Registration'
```

APIDescriptor status shows `x509: certificate signed by unknown authority` but service is running

Your APIDescription CR shows a status and message similar to:

```
    Message:               Put "https://tap-gui.tap.my-cluster.tapdemo.vmware.com/api/
catalog/immediate/entities": x509: certificate signed by unknown authority
    Reason:                Error
    Status:                False
    Type:                  Ready
    Last Transition Time:  2022-11-28T09:59:13Z
```

This is the same issue as `connection refused` described earlier.

Unexpected content in generated specification

This issue happens when there are two or more `CuratedAPIDescriptor` with conflicting `groupId` and `version`.

If you create two `CuratedAPIDescriptor`s with the same `groupId` and `version` combination, both reconcile without an error. However, the following undesired behaviors can signal conflict:

- The `/openapi?groupId&version` endpoint returns more than one API specification.
- If you add both specifications to API portal, for example, `/openapi?groupId&version`, only one of them shows up in the API portal UI with a warning indicating that there is a conflict.
- If you add the route provider annotation for both of the `CuratedAPIDescriptor`s to use SCG, the generated API specification includes API routes from both `CuratedAPIDescriptor`s.

You can see the `groupId` and `version` information from all `CuratedAPIDescriptor`s to find conflicts by running:

```
$ kubectl get curatedapidescriptors -A

NAMESPACE           NAME         GROUPID            VERSION   STATUS   CURATED API SPE
C URL
my-apps             petstore     test-api-group     1.2.3     Ready    http://AAR-CONT
ROLLER-FQDN/openapi/my-apps/petstore
default             mystery      test-api-group     1.2.3     Ready    http://AAR-CONT
ROLLER-FQDN/openapi/default/mystery
```

Unsupported OpenAPI version

API Auto Registration only supports OpenAPI v3.0 and v2.0, formerly known as Swagger.

Overview of API portal for VMware Tanzu

This topic tells you how to use API portal for VMware Tanzu to find APIs you can use in your own applications.

Overview

You can view detailed API documentation and try out an API to meet your needs. API portal assembles its dashboard and detailed API documentation views by ingesting OpenAPI documentation from the source URLs. An API portal operator can add any number of OpenAPI source URLs in a single instance.

Getting started

To install the package without the predefined profiles of Tanzu Application Platform, see Install API portal.

For information about API portal for VMware Tanzu, see API portal for VMware Tanzu.

For information about configuring the package, see Configuring API portal for VMware Tanzu on Kubernetes.

API portal for VMware Tanzu supports:

- Authentication through Single Sign-On (SSO)
- API keys configuration and management
- Secure communication by using TLS

Overview of API portal for VMware Tanzu

This topic tells you how to use API portal for VMware Tanzu to find APIs you can use in your own applications.

Overview

You can view detailed API documentation and try out an API to meet your needs. API portal assembles its dashboard and detailed API documentation views by ingesting OpenAPI documentation from the source URLs. An API portal operator can add any number of OpenAPI source URLs in a single instance.

Getting started

To install the package without the predefined profiles of Tanzu Application Platform, see Install API portal.

For information about API portal for VMware Tanzu, see API portal for VMware Tanzu.

For information about configuring the package, see Configuring API portal for VMware Tanzu on Kubernetes.

API portal for VMware Tanzu supports:

- Authentication through Single Sign-On (SSO)
- API keys configuration and management
- Secure communication by using TLS

Install API portal for VMware Tanzu

This topic tells you how to install and update Tanzu API portal for VMware Tanzu from the Tanzu Application Platform (commonly known as TAP) package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install API portal. For more information about profiles, see Components and installation profiles.

Prerequisites

Before installing API portal:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.

Install

To install the API portal package:

1. Confirm what versions of API portal are available to install by running:

   ```
   tanzu package available list -n tap-install api-portal.tanzu.vmware.com
   ```

   For example:

   ```
   $ tanzu package available list api-portal.tanzu.vmware.com --namespace tap-inst
   all
   - Retrieving package versions for api-portal.tanzu.vmware.com...
     NAME                          VERSION  RELEASED-AT
     api-portal.tanzu.vmware.com  1.0.3    2021-10-13T00:00:00Z
   ```

2. (Optional) Gather values schema.

   ```
   tanzu package available get api-portal.tanzu.vmware.com/VERSION-NUMBER --values
   -schema --namespace tap-install
   ```

   Where `VERSION-NUMBER` is the version of the API Portal package listed earlier.

   For example:

   ```
   $ tanzu package available get api-portal.tanzu.vmware.com/1.0.3 --values-schema
   --namespace tap-install

   Retrieving package details for api-portal.tanzu.vmware.com/1.0.3...
   ```

3. (Optional) VMware recommends creating `api-portal-values.yaml`.

   To overwrite the default values when installing the package, create a `api-portal-values.yaml` file by following the values schema.

4. Install API portal by running:

   ```
   tanzu package install api-portal -n tap-install -p api-portal.tanzu.vmware.com
   -v VERSION-NUMBER --values-file api-portal-values.yaml
   ```

   Where `VERSION-NUMBER` is the version of the API Portal package listed earlier.

For example:

```
$ tanzu package install api-portal -n tap-install -p api-portal.tanzu.vmware.co
m -v 1.0.3 --values-file api-portal-values.yaml

/ Installing package 'api-portal.tanzu.vmware.com'
| Getting namespace 'api-portal'
| Getting package metadata for 'api-portal.tanzu.vmware.com'
| Creating service account 'api-portal-api-portal-sa'
| Creating cluster admin role 'api-portal-api-portal-cluster-role'
| Creating cluster role binding 'api-portal-api-portal-cluster-rolebinding'
/ Creating package resource
- Package install status: Reconciling


Added installed package 'api-portal' in namespace 'tap-install'
```

5. Verify the package installation by running:

   ```
   tanzu package installed get api-portal -n tap-install
   ```

   Verify that `STATUS` is `Reconcile succeeded`:

   ```
   kubectl get pods -n api-portal
   ```

Update the installation values for the `api-portal` package

To update the installation values for the `api-portal` package:

1. To overwrite the default values, create new values, or update the existing values, you need an `api-portal-values.yaml` file. If you do not already have an existing values file, you can extract the existing values by running:

   ```
   tanzu package installed get api-portal -n tap-install --values-file-output api-
   portal-values.yaml
   ```

   You can view the schema of the package:

   ```
   tanzu package available get apis.apps.tanzu.vmware.com/VERSION-NUMBER --values-
   schema --namespace tap-install
   ```

   Where `VERSION-NUMBER` is the version of the API Portal package listed in the earlier step.

   For example:

   ```
   tanzu package available get api-portal.tanzu.vmware.com/1.2.5 --values-schema -
   -namespace tap-install
   ```

2. Update the package by using the Tanzu CLI:

   ```
   tanzu package installed update api-auto-registration
   --package apis.apps.tanzu.vmware.com
   --namespace tap-install
   --version VERSION-NUMBER
   --values-file api-portal-values.yaml
   ```

   Where `VERSION-NUMBER` is the version of the API Portal package listed in the earlier step.

3. If you installed the API portal package as part of Tanzu Application Platform, you must update the `tap-values.yaml` and update the installation of Tanzu Application Platform. See Install your Tanzu Application Platform profile.

```
tanzu package installed update tap --package tap.tanzu.vmware.com --version VER
SION-NUMBER --values-file tap-values.yaml -n tap-install
```

Where `VERSION-NUMBER` is the version of the API Portal package listed in the earlier step.

Note

You can update API portal as part of upgrading Tanzu Application Platform. See Upgrading Tanzu Application Platform.

Overview of Application Accelerator

This topic tells you about the Application Accelerator component and architecture in Tanzu Application Platform (commonly known as TAP).

Overview

Application Accelerator for VMware Tanzu helps you bootstrap developing your applications and deploying them in a discoverable and repeatable way.

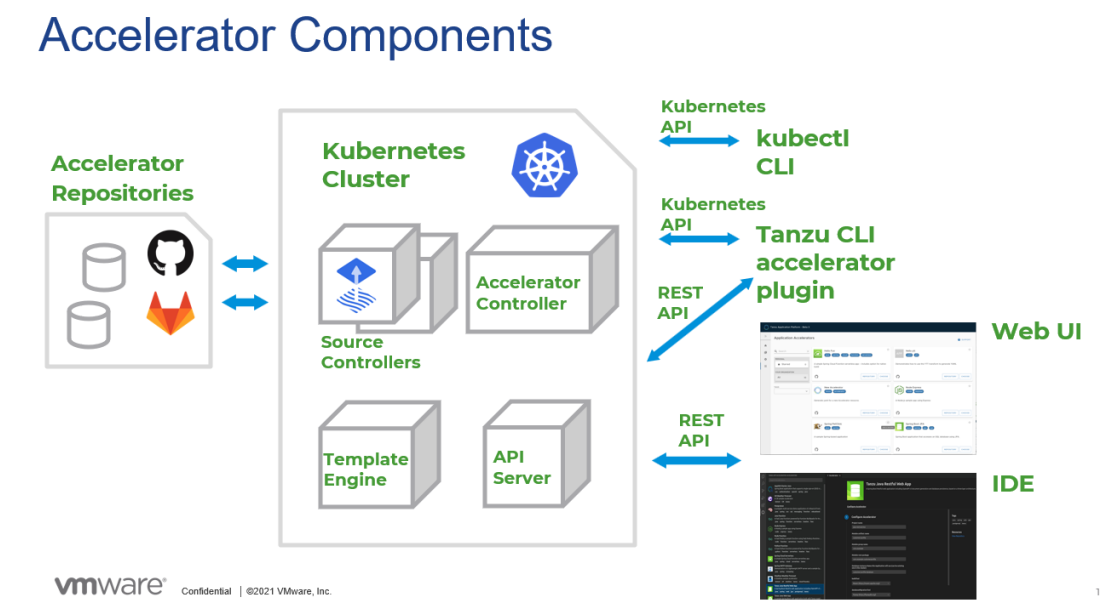
Enterprise Architects author and publish accelerator projects that provide developers and operators in their organization ready-made, conforming code and configurations.

Published accelerator projects are maintained in Git repositories. You can then use Application Accelerator to create new projects based on those accelerator projects.

The Application Accelerator UI enables you to discover available accelerators, configure them, and generate new projects to download.

Architecture

The following diagram of Accelerator components illustrates the Application Accelerator architecture.



How does Application Accelerator work?

Application Accelerator allows you to generate new projects from files in Git repositories. An `accelerator.yaml` file in the repository declares input options for the accelerator. This file also contains instructions for processing the files when you generate a new project.

Accelerator custom resources (CRs) control which repositories appear in both the Tanzu Application Platform Application Accelerator UI and in the Application Accelerator extension for VS Code. You can maintain CRs by using Kubernetes tools such as kubectl or by using the Tanzu CLI accelerator commands. The Accelerator controller reconciles the CRs with a Flux2 Source Controller to fetch files from GitHub or GitLab.

The Application Accelerator web UI gives you a searchable list of accelerators to choose from. After you select an accelerator, the UI presents text boxes for the options that are defined within the `accelerator.yaml` of the selected accelerator.

Application Accelerator sends the input values to the Accelerator Engine for processing. (Optional) The user can choose to have a new Git repository created as part of the project creation process. The Engine then returns the project in a ZIP file. If the project was generated using the Application Accelerator extension for VS Code, the project automatically be extracted to the directory location of your choice on your local machine. You can then open the project in your favorite integrated development environment (IDE) to develop further.

Next steps

Learn more about:

- Creating Accelerators

Overview of Application Accelerator

This topic tells you about the Application Accelerator component and architecture in Tanzu Application Platform (commonly known as TAP).

Overview

Application Accelerator for VMware Tanzu helps you bootstrap developing your applications and deploying them in a discoverable and repeatable way.

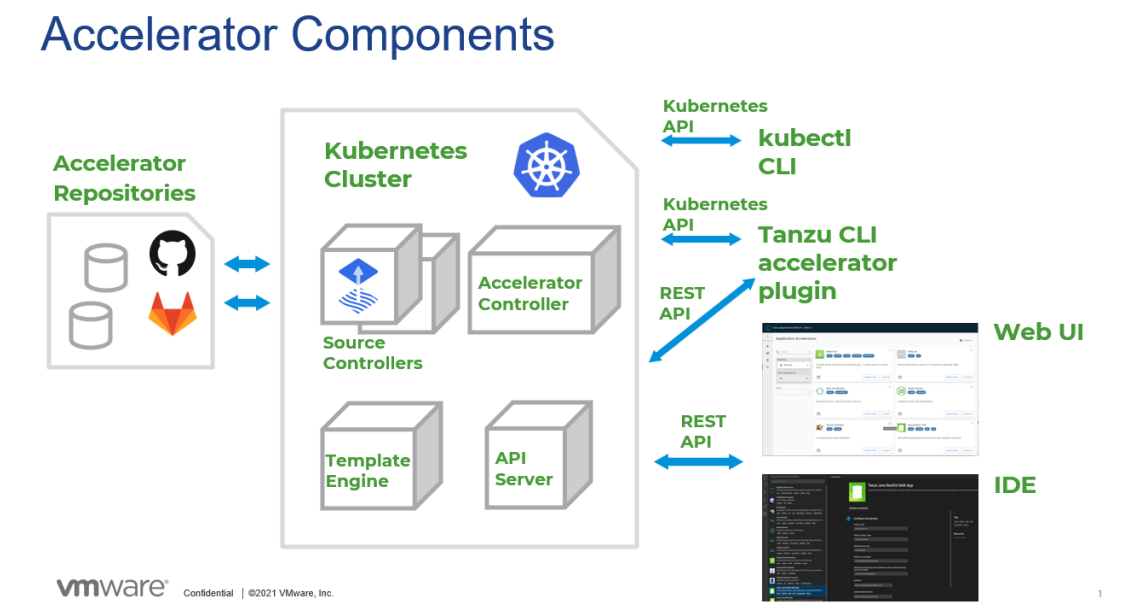
Enterprise Architects author and publish accelerator projects that provide developers and operators in their organization ready-made, conforming code and configurations.

Published accelerator projects are maintained in Git repositories. You can then use Application Accelerator to create new projects based on those accelerator projects.

The Application Accelerator UI enables you to discover available accelerators, configure them, and generate new projects to download.

Architecture

The following diagram of Accelerator components illustrates the Application Accelerator architecture.



How does Application Accelerator work?

Application Accelerator allows you to generate new projects from files in Git repositories. An `accelerator.yaml` file in the repository declares input options for the accelerator. This file also contains instructions for processing the files when you generate a new project.

Accelerator custom resources (CRs) control which repositories appear in both the Tanzu Application Platform Application Accelerator UI and in the Application Accelerator extension for VS Code. You can maintain CRs by using Kubernetes tools such as kubectl or by using the Tanzu CLI accelerator commands. The Accelerator controller reconciles the CRs with a Flux2 Source Controller to fetch files from GitHub or GitLab.

The Application Accelerator web UI gives you a searchable list of accelerators to choose from. After you select an accelerator, the UI presents text boxes for the options that are defined within the `accelerator.yaml` of the selected accelerator.

Application Accelerator sends the input values to the Accelerator Engine for processing. (Optional) The user can choose to have a new Git repository created as part of the project creation process. The Engine then returns the project in a ZIP file. If the project was generated using the Application Accelerator extension for VS Code, the project automatically be extracted to the directory location of your choice on your local machine. You can then open the project in your favorite integrated development environment (IDE) to develop further.

Next steps

Learn more about:

- Creating Accelerators

Install Application Accelerator

This topic tells you how to install Application Accelerator from the Tanzu Application Platform (commonly known as TAP) package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Application Accelerator. For more information about profiles, see About Tanzu Application Platform components and profiles.

Prerequisites

Before installing Application Accelerator:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- Install Flux SourceController on the cluster. See Install Flux CD Source Controller.
- Install Source Controller on the cluster. See Install Source Controller.

Install

To install Application Accelerator:

1. List version information for the package by running:

   ```
   tanzu package available list accelerator.apps.tanzu.vmware.com --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package available list accelerator.apps.tanzu.vmware.com --namespace tap-install

   - Retrieving package versions for accelerator.apps.tanzu.vmware.com...
     NAME                                VERSION  RELEASED-AT
     accelerator.apps.tanzu.vmware.com  1.4.0    2022-12-08 12:00:00 -0500 EST
   ```

2. (Optional) View the changes you can make to the default installation settings by running:

   ```
   tanzu package available get accelerator.apps.tanzu.vmware.com/VERSION-NUMBER \
     --values-schema \
     --namespace tap-install
   ```

   Where `VERSION-NUMBER` is the version of the Application Accelerator package listed earlier.

   For example:

   ```
   $ tanzu package available get accelerator.apps.tanzu.vmware.com/1.4.0 \
       --values-schema \
       --namespace tap-install
   ```

   For more information about values schema options, see the properties listed in Configure properties and resource use later.

3. Create a file named `app-accelerator-values.yaml` using the following example code:

   ```
   server:
     service_type: "LoadBalancer"
     watched_namespace: "accelerator-system"
   samples:
     include: true
   ```

4. Edit the values in your `app-accelerator-values.yaml` if needed, or leave the default values. You can add values you want from Configure properties and resource use.

5. Install the package by running:

```
tanzu package install app-accelerator \
  --package accelerator.apps.tanzu.vmware.com \
  --version VERSION-NUMBER \
  --namespace tap-install \
  --values-file app-accelerator-values.yaml
```

Where `VERSION-NUMBER` is the version of the Application Accelerator package listed earlier.

For example:

```
$ tanzu package install app-accelerator \
    --package accelerator.apps.tanzu.vmware.com \
    --version 1.4.0 \
    --namespace tap-install \
    --values-file app-accelerator-values.yaml

- Installing package 'accelerator.apps.tanzu.vmware.com'
| Getting package metadata for 'accelerator.apps.tanzu.vmware.com'
| Creating service account 'app-accelerator-tap-install-sa'
| Creating cluster admin role 'app-accelerator-tap-install-cluster-role'
| Creating cluster role binding 'app-accelerator-tap-install-cluster-rolebindin
g'
| Creating secret 'app-accelerator-tap-install-values'
- Creating package resource
- Package install status: Reconciling

 Added installed package 'app-accelerator' in namespace 'tap-install'
```

6. Verify the package install by running:

```
tanzu package installed get app-accelerator -n tap-install
```

For example:

```
$ tanzu package installed get app-accelerator -n tap-install

| Retrieving installation details for cc...
NAME:                    app-accelerator
PACKAGE-NAME:            accelerator.apps.tanzu.vmware.com
PACKAGE-VERSION:         1.4.0
STATUS:                  Reconcile succeeded
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`.

7. To see the IP address for the Application Accelerator API when the `server.service_type` is set to `LoadBalancer`, run:

```
kubectl get service -n accelerator-system
```

This lists an external IP address for use with the `--server-url` Tanzu CLI flag for the Accelerator plug-in `generate` & `generate-from-local` command.

For how to troubleshoot installation issues, see Troubleshoot Application Accelerator.

Configure properties and resource use

When you install the Application Accelerator, you can configure the following optional properties from within your `app-accelerator-values.yaml` configuration file:

Property	Default	Description
registry.secret_ref	registry.tanzu.vmware.com	The secret used for accessing the registry where the App-Accelerator images are located
server.service_type	ClusterIP	The service type for the acc-ui-server service including LoadBalancer, NodePort, or ClusterIP
server.watched_namespace	accelerator-system	The namespace the server watches for accelerator resources
server.engine_invocation_url	http://acc-engine.accelerator-system.svc.cluster.local/invocations	The URL to use for invoking the accelerator engine
engine.service_type	ClusterIP	The service type for the acc-engine service including LoadBalancer, NodePort, or ClusterIP
engine.max_direct_memory_size	32M	The maximum size for the Java -XX:MaxDirectMemorySize setting
samples.include	True	Option to include the bundled sample Accelerators in the installation
ingress.include	False	Option to include the ingress configuration in the installation
ingress.enable_tls	False	Option to include TLS for the ingress configuration
domain	tap.example.com	Top-level domain to use for ingress configuration, default is shared.ingress_domain
tls.secret_name	tls	The name of the secret
tls.namespace	tanzu-system-ingress	The namespace for the secret
telemetry.retain_invocation_events_for_no_days	30	The number of days to retain recorded invocation events resources
telemetry.record_invocation_events	true	The system records each engine invocation when generating files for an accelerator?
git_credentials.secret_name	git-credentials	The name to use for the secret storing Git credentials for accelerators
git_credentials.username	null	The user name to use in secret storing Git credentials for accelerators
git_credentials.password	null	The password to use in secret storing Git credentials for accelerators
git_credentials.ca_file	null	The CA certificate data to use in secret storing Git credentials for accelerators
managed_resources.enable	false	Whether to enable the App used to control managed accelerator resources
managed_resources.git.url	none	Required if managed_resources are enabled. Git repository URL containing manifests for managed accelerator resources
managed_resources.git.ref	origin/main	Required if managed_resources are enabled. Git ref to use for repository containing manifests for managed accelerator resources
managed_resources.git.sub_path	null	Git subPath to use for repository containing manifests for managed accelerator resources

Property	Default	Description
`managed_resources.git.secret_ref`	`git-credentials`	Secret name to use for repository containing manifests for managed accelerator resources

VMware recommends that you do not override the default setting for `registry.secret_ref`, `server.engine_invocation_url`, or `engine.service_type`. These properties are only used to configure non-standard installations.

The following table is the resource use configurations for the components of Application Accelerator.

Component	Resource requests	Resource limits
acc-controller	CPU: 100m memory: 20Mi	CPU: 100m memory: 30Mi
acc-server	CPU: 100m memory:20Mi	CPU: 100m memory: 30Mi
acc-engine	CPU: 500m memory: 1Gi	CPU: 500m memory: 2Gi

Configure Application Accelerator

This topic tells you about advanced configuration options available for Application Accelerator in Tanzu Application Platform (commonly known as TAP). This includes configuring Git-Ops style deployments of accelerators and configurations for use with non-public repositories and in air-gapped environments.

Accelerators are created either using the Tanzu CLI or by applying a YAML manifest using kubectl. Another option is Using a Git-Ops style configuration for deploying a set of managed accelerators.

Application Accelerator pulls content from accelerator source repositories using either the “Flux SourceController” or the “Tanzu Application Platform Source Controller” components. If the repository used is accessible anonymously from a public server, you do not have to configure anything additional. Otherwise, provide authentication as explained in Using non-public repositories. There are also options for making these configurations easier explained in Configuring tap-values.yaml with Git credentials secret

Using a Git-Ops style configuration for deploying a set of managed accelerators

To enable a Git-Ops style of managing resources used for deploying accelerators, there is a new set of properties for the Application Accelerator configuration. The resources are managed using a Carvel kapp-controller App in the `accelerator-system` namespace that watches a Git repository containing the manifests for the accelerators. This means that you can make changes to the manifests, or to the accelerators they point to, and the changes are reconciled and reflected in the deployed resources.

You can specify the following accelerator configuration properties when installing the Application Accelerator. The same properties are provided in the `accelerator` section of the `tap-values.yaml` file:

```
accelerator:
  managed_resources:
    enable: true
    git:
      url: GIT-REPO-URL
```

```
      ref: origin/main
      sub_path: null
      secret_ref: git-credentials
```

Where:

- `GIT-REPO-URL` is the URL of a Git repository that contains manifest YAML files for the accelerators that you want to have managed. The URL must start with `https://` or `git@`. You can specify a `sub_path` if necessary and also a `secret_ref` if the repository requires authentication. If not needed, then leave these additional properties out.

  For more information, see Configure tap-values.yaml with Git credentials secret and Creating a manifest with multiple accelerators and fragments in this topic.

Functional and Organizational Considerations

Any accelerator manifest that is defined under the `GIT-REPO-URL` and optional `sub_path` is selected by the kapp-controller app. If there are multiple manifests at the defined `GIT-REPO-URL`, they are all watched for changes and displayed to the user as a merged catalog.

For example: if you have two manifests containing multiple accelerator or fragment definitions, `manifest-1.yaml`, and `manifest-2.yaml`, on the same path in the organizational considerations. The resulting catalog is (`manifest-1.yaml` + `manifest-2.yaml`).

Examples for creating accelerators

A minimal example for creating an accelerator

A minimal example might look like the following manifest:

`spring-cloud-serverless.yaml`

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: spring-cloud-serverless
spec:
  git:
    url: https://github.com/vmware-tanzu/application-accelerator-samples
    subPath: spring-cloud-serverless
    ref:
      branch: main
```

This example creates an accelerator named `spring-cloud-serverless`. The `displayName`, `description`, `iconUrl`, and `tags` text boxes are populated based on the content under the `accelerator` key in the `accelerator.yaml` file found in the `main` branch of the Git repository at Application Accelerator Samples under the sub-path `spring-cloud-serverless`. For example:

`accelerator.yaml`

```
accelerator:
  displayName: Spring Cloud Serverless
  description: A simple Spring Cloud Function serverless app
  iconUrl: https://raw.githubusercontent.com/vmware-tanzu/application-accelerator-samp
les/main/icons/icon-cloud.png
  tags:
  - java
  - spring
  - cloud
  - function
  - serverless
```

```
  - tanzu
...
```

To create this accelerator with kubectl, run:

```
kubectl apply --namespace --accelerator-system --filename spring-cloud-serverless.yaml
```

Or, you can use the Tanzu CLI and run:

```
tanzu accelerator create spring-cloud-serverless --git-repo https://github.com/vmware-
tanzu/application-accelerator-samples.git --git-branch main --git-sub-path spring-clou
d-serverless
```

An example for creating an accelerator with customized properties

You can specify the `displayName`, `description`, `iconUrl`, and `tags` text boxes and this overrides any values provided in the accelerator's Git repository. The following example explicitly sets those text boxes and the `ignore` text box:

`my-spring-cloud-serverless.yaml`

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: my-spring-cloud-serverless
spec:
  displayName: My Spring Cloud Serverless
  description: My own Spring Cloud Function serverless app
  iconUrl: https://raw.githubusercontent.com/vmware-tanzu/application-accelerator-samp
les/main/icons/icon-cloud.png
  tags:
    - spring
    - cloud
    - function
    - serverless
  git:
    ignore: ".git/, bin/"
    url: https://github.com/vmware-tanzu/application-accelerator-samples
    subPath: spring-cloud-serverless
    ref:
      branch: test
```

To create this accelerator with kubectl, run:

```
kubectl apply --namespace --accelerator-system --filename my-spring-cloud-serverless.y
aml
```

To use the Tanzu CLI, run:

```
tanzu accelerator create my-spring-cloud-serverless --git-repo https://github.com/vmwa
re-tanzu/application-accelerator-samples --git-branch main --git-sub-path spring-cloud
-serverless \
  --description "My own Spring Cloud Function serverless app" \
  --display-name "My Spring Cloud Serverless" \
  --icon-url https://raw.githubusercontent.com/vmware-tanzu/application-accelerator-sa
mples/main/icons/icon-cloud.png \
  --tags "spring,cloud,function,serverless"
```

Note

It is not possible to provide the `git.ignore` option with the Tanzu CLI.

Creating a manifest with multiple accelerators and fragments

You might have a manifest that contains multiple accelerators or fragments. For example:

`accelerator-collection.yaml`

```
---
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: spring-cloud-serverless
spec:
  git:
    url: https://github.com/vmware-tanzu/application-accelerator-samples
    subPath: spring-cloud-serverless
    ref:
      branch: main
---
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: tanzu-java-web-app
spec:
  git:
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
    subPath: tanzu-java-web-app
    ref:
      branch: main
```

For a larger example of this, see Sample Accelerators Main. Optionally, use this to create an initial catalog of accelerators and fragments during a fresh Application Accelerator install.

Configure `tap-values.yaml` with Git credentials secret

Note

For how to create a new OAuth Token for optional Git repository creation, see Create an Application Accelerator Git repository during project creation.

When deploying accelerators using Git repositories that requires authentication or are installed with custom CA certificates, you must provide some additional authentication values in a secret. The examples in the next section provide more details. This section describes how to configure a Git credentials secret that is used in later Git-based examples.

You can specify the following accelerator configuration properties when installing Application Accelerator. The same properties are provided in the `accelerator` section of the `tap-values.yaml` file:

```
accelerator:
  git_credentials:
    secret_name: git-credentials
    username: GIT-USER-NAME
    password: GIT-CREDENTIALS
    ca_file: CUSTOM-CA-CERT
```

Where:

- `GIT-USER-NAME` is the user name for authenticating with the Git repository.
- `GIT-CREDENTIALS` is the password or access token used for authenticating with the Git repository. VMware recommends using an access token for this.
- `CUSTOM-CA-CERT` is the certificate data needed when accessing the Git repository.

This is an example of this part of a `tap-values.yaml` configuration:

```
accelerator:
  git_credentials:
    secret_name: git-credentials
    username: testuser
    password: s3cret
    ca_file: |
      -----BEGIN CERTIFICATE-----
      .
      .
      .  < certificate data >
      .
      .
      -----END CERTIFICATE-----
```

You can specify the custom CA certificate data using the shared config value `shared.ca_cert_data` and it propagates to all components that can make use of it, including the App Accelerator configuration. The example earlier produces an output such as this using the shared value:

```
shared:
  ca_cert_data: |
    -----BEGIN CERTIFICATE-----
    .
    .
    .  < certificate data >
    .
    .
    -----END CERTIFICATE-----

accelerator:
  git_credentials:
    secret_name: git-credentials
    username: testuser
    password: s3cret
```

Using non-public repositories

For GitHub repositories that aren't accessible anonymously, you must provide credentials in a Secret.

- For HTTPS repositories the secret must contain user name and password fields. The password field can contain a personal access token instead of an actual password. For more information, see Fluxcd/source-controller basic access authentication.
- For HTTPS with self-signed certificates, you can add a `.data.caFile` value to the secret created for HTTPS authentication. For more information, see fluxcd/source-controller HTTPS Certificate Authority.
- For SSH repositories, the secret must contain identity, identity.pub, and known_hosts text boxes. For more information, see fluxcd/source-controller SSH authentication.
- For Image repositories that aren't publicly available, an image pull secret might be provided. For more information, see Kubernetes documentation on using imagePullSecrets.

Examples for a private Git repository

Example using http credentials

To create an accelerator using a private Git repository, first create a secret with the HTTP credentials.

Note

For better security, use an access token as the password.

```
kubectl create secret generic https-credentials \
    --namespace accelerator-system \
    --from-literal=username=USER \
    --from-literal=password=ACCESS-TOKEN
```

Verify that your secret was created by running:

```
kubectl get secret --namespace accelerator-system https-credentials -o yaml
```

The output is similar to:

```
apiVersion: v1
kind: Secret
metadata:
  name: https-credentials
  namespace: accelerator-system
type: Opaque
data:
  username: <BASE64>
  password: <BASE64>
```

After you created and verified the secret, you can create the accelerator by using the `spec.git.secretRef.name` property:

`private-acc.yaml`

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: private-acc
spec:
  displayName: private
  description: Accelerator using a private repository
  git:
    url: REPOSITORY-URL
    ref:
      branch: main
    secretRef:
      name: https-credentials
```

For https credentials, the `REPOSITORY-URL` must use `https://` as the URL scheme.

If you are using the Tanzu CLI, add the `--secret-ref` flag to your `tanzu accelerator create` command and provide the name of the secret for that flag.

Example using http credentials with self-signed certificate

To create an accelerator using a private Git repository with a self-signed certificate, create a secret with the HTTP credentials and the certificate.

Note

For better security, use an access token as the password.

```
kubectl create secret generic https-ca-credentials \
    --namespace accelerator-system \
    --from-literal=username=USER \
    --from-literal=password=ACCESS-TOKEN \
    --from-file=caFile=PATH-TO-CA-FILE
```

Verify that your secret was created by running:

```
kubectl get secret --namespace accelerator-system https-ca-credentials -o yaml
```

The output is similar to:

```
apiVersion: v1
kind: Secret
metadata:
  name: https-ca-credentials
  namespace: accelerator-system
type: Opaque
data:
  username: <BASE64>
  password: <BASE64>
  caFile: <BASE64>
```

After you have the secret created, you can create the accelerator by using the `spec.git.secretRef.name` property:

`private-acc.yaml`

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: private-acc
spec:
  displayName: private
  description: Accelerator using a private repository
  git:
    url: REPOSITORY-URL
    ref:
      branch: main
    secretRef:
      name: https-ca-credentials
```

Important

For https credentials, the `REPOSITORY-URL` must use `https://` as the URL scheme.

If you are using the Tanzu CLI, add the `--secret-ref` flag to your `tanzu accelerator create` command and provide the name of the secret for that flag.

Example using SSH credentials

To create an accelerator using a private Git repository, create a secret with the SSH credentials such as this example:

```
ssh-keygen -q -N "" -f ./identity
ssh-keyscan github.com > ./known_hosts
kubectl create secret generic ssh-credentials \
    --namespace accelerator-system \
    --from-file=./identity \
    --from-file=./identity.pub \
    --from-file=./known_hosts
```

If you have a key file already created, skip the `ssh-keygen` and `ssh-keyscan` steps and replace the values for the `kubectl create secret` command. Such as:

- `--from-file=identity=PATH-TO-YOUR-IDENTITY-FILE`
- `--from-file=identity.pub=PATH-TO-YOUR-IDENTITY.PUB-FILE`
- `--from-file=known_hosts=PATH-TO-YOUR-KNOWN-HOSTS-FILE`

Verify that your secret was created by running:

```
kubectl get secret --namespace accelerator-system ssh-credentials -o yaml
```

The output is similar to :

```
apiVersion: v1
kind: Secret
metadata:
  name: ssh-credentials
  namespace: accelerator-system
type: Opaque
data:
  identity: <BASE64>
  identity.pub: <BASE64>
  known_hosts: <BASE64>
```

To use this secret when creating an accelerator, provide the secret name in the `spec.git.secretRef.name` property:

`private-acc-ssh.yaml`

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: private-acc
spec:
  displayName: private
  description: Accelerator using a private repository
  git:
    url: REPOSITORY-URL
    ref:
      branch: main
    secretRef:
      name: ssh-credentials
```

When using SSH credentials, the `REPOSITORY-URL` must include the user name as part of the URL. For example: `ssh://user@example.com:22/repository.git`. For more information, see Flux documentation.

If you are using the Tanzu CLI, add the `--secret-ref` flag to your `tanzu accelerator create` command and provide the name of the secret for that flag.

Examples for a private source-image repository

If your registry uses a self-signed certificate then you must add the CA certificate data to the configuration for the “Tanzu Application Platform Source Controller” component. Add it under `source_controller.ca_cert_data` in your `tap-values.yaml` file that is used during installation.

`tap-values.yaml`

```
source_controller:
  ca_cert_data: |-
    -----BEGIN CERTIFICATE-----
    .
    .
    .  < certificate data >
    .
    .
    -----END CERTIFICATE-----
```

Example using image-pull credentials

To create an accelerator using a private source-image repository, create a secret with the image-pull credentials:

```
create secret generic registry-credentials \
    --namespace accelerator-system \
    --from-literal=username=USER \
    --from-literal=password=PASSWORD
```

Verify that your secret was created by running:

```
kubectl get secret --namespace accelerator-system registry-credentials -o yaml
```

The output is similar to:

```
apiVersion: v1
kind: Secret
metadata:
  name: registry-credentials
  namespace: accelerator-system
type: Opaque
data:
  username: <BASE64>
  password: <BASE64>
```

After you have the secret created, you can create the accelerator by using the `spec.git.secretRef.name` property:

`private-acc.yaml`

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: private-acc
spec:
  displayName: private
  description: Accelerator using a private repository
  source:
    image: "registry.example.com/test/private-acc-src:latest"
    imagePullSecrets:
    - name: registry-credentials
```

If you are using the Tanzu CLI, add the `--secret-ref` flag to your `tanzu accelerator create` command and provide the name of the secret for that flag.

Configure ingress timeouts when some accelerators take longer to generate

If Tanzu Application Platform is configured to use an ingress for Tanzu Developer Portal and the Accelerator Server, then it might detect a timeout during accelerator generation.

This can happen if the accelerator takes a longer time to generate than the default timeout. When this happens, Tanzu Developer Portal appears to continue to run for an indefinite period. In the IDE extension, it shows a `504` error. To mitigate this, you can increase the timeout value for the HTTPProxy resources used for the ingress by applying secrets with overlays to edit the HTTPProxy resources.

Configure an ingress timeout overlay secret for each HTTPProxy

For Tanzu Developer Portal, create the following overlay secret in the `tap-install` namespace:

```
apiVersion: v1
kind: Secret
metadata:
  name: patch-tap-gui-timeout
  namespace: tap-install
stringData:
  patch.yaml: |
    #@ load("@ytt:overlay", "overlay")
    #@overlay/match by=overlay.subset({"kind": "HTTPProxy", "metadata": {"name": "tap-
gui"}})
    ---
    spec:
      routes:
        #@overlay/match by=overlay.subset({"services": [{"name": "server"}]})
        #@overlay/match-child-defaults missing_ok=True
        - timeoutPolicy:
            idle: 30s
            response: 30s
```

For Accelerator Server (used for IDE extension), create the following overlay secret in the `tap-install` namespace:

```
apiVersion: v1
kind: Secret
metadata:
  name: patch-accelerator-timeout
  namespace: tap-install
stringData:
  patch.yaml: |
    #@ load("@ytt:overlay", "overlay")
    #@overlay/match by=overlay.subset({"kind": "HTTPProxy", "metadata": {"name": "acce
lerator"}})
    ---
    spec:
      routes:
        #@overlay/match by=overlay.subset({"services": [{"name": "acc-server"}]})
        #@overlay/match-child-defaults missing_ok=True
        - timeoutPolicy:
            idle: 30s
            response: 30s
```

Apply the timeout overlay secrets in tap-values.yaml

Add the following `package_overlays` section to `tap-values.yaml` before installing or updating Tanzu Application Platform:

```
package_overlays:
- name: tap-gui
  secrets:
  - name: patch-tap-gui-timeout
- name: accelerator
  secrets:
  - name: patch-accelerator-timeout
```

Configuring skipping TLS verification for access to Source Controller

You can configure the Flux or Tanzu Application Platform Source Controller to use Transport Layer Security (TLS) and use custom certificates. In that case, configure the Accelerator System to skip the TLS verification for calls to access the sources by providing the following property in the `accelerator` section of the `tap-values.yaml` file:

```
sources:
  skip_tls_verify: true
```

Enabling TLS for Accelerator Server

To enable TLS for the Accelerator Server, the following properties must be provided in the `accelerator` section of the `tap-values.yaml` file:

```
server:
  tls:
    enabled: true
    key: SERVER-PRIVATE-KEY
    crt: SERVER-CERTIFICATE
```

Where:

- `SERVER-PRIVATE-KEY` is the pem encoded server private key.
- `SERVER-CERTIFICATE` is the pem encoded server certificate.

Here is a sample `tap-values.yaml` configuration with TLS enabled for Accelerators Server:

```
server:
  tls:
    enabled: true
    key: |
      -----BEGIN PRIVATE KEY-----
      .
      .  < private key data >
      .
      -----END PRIVATE KEY-----
    crt: |
      -----BEGIN CERTIFICATE-----
      .
      .  < certificate data >
      .
      -----END CERTIFICATE-----
```

Configuring skipping TLS verification of Engine calls for Accelerator Server

If you configure the Accelerator Engine to use TLS and use custom certificates, then you can configure the Accelerator Server to skip the TLS verification for calls to the Engine by providing the following property in the `accelerator` section of the `tap-values.yaml` file:

```
server:
  engine_skip_tls_verify: true
```

Enabling TLS for Accelerator Engine

To enable TLS for the Accelerator Engine, the following properties are provided in the `accelerator` section of the `tap-values.yaml` file:

```
engine:
  tls:
    enabled: true
    key: ENGINE-PRIVATE-KEY
    crt: ENGINE-CERTIFICATE
```

Where:

- `ENGINE-PRIVATE-KEY` is the pem encoded acc-engine private key.
- `ENGINE-CERTIFICATE` is the pem encoded acc-engine certificate.

Here is a sample `tap-values.yaml` configuration with TLS enabled for Accelerators Engine:

```
engine:
  tls:
    enabled: true
    key: |
      -----BEGIN PRIVATE KEY-----
      .
      .  < private key data >
      .
      -----END PRIVATE KEY-----
    crt: |
      -----BEGIN CERTIFICATE-----
      .
      .  < certificate data >
      .
      -----END CERTIFICATE-----
```

Next steps

- Creating accelerators

Create accelerators

This topic tells you how to create an accelerator in Tanzu Developer Portal.

An accelerator contains your conforming code and configurations that developers can use to create new projects that by default follow the standards defined in your accelerators.

Prerequisites

The following prerequisites are required to create an accelerator:

- Application Accelerator is installed. For information about installing Application Accelerator, see Installing Application Accelerator for VMware Tanzu.
- You can access Tanzu Developer Portal from a browser or use the Application Accelerator extension for VS Code.
    - For more information about Tanzu Developer Portal, see Overview of Tanzu Developer Portal.
    - For more information about Application Accelerator extension for VS Code, see Application Accelerator Visual Studio Code extension.
- kubectl is installed and authenticated with admin rights for your target cluster.

Getting started

You can use any Git repository to create an accelerator. You need the URL of the repository to create an accelerator.

For this example, the Git repository is `public` and contains a `README.md` file. These are options available when you create repositories on GitHub.

Use the following procedure to create an accelerator based on this Git repository:

1. Clone your Git repository.
2. Create a file named `accelerator.yaml` in the root directory of this Git repository.
3. Add the following content to the `accelerator.yaml` file:

    ```
    accelerator:
      displayName: Simple Accelerator
      description: Contains just a README
      iconUrl: https://images.freecreatives.com/wp-content/uploads/2015/05/smiley-5
    59124_640.jpg
      tags:
      - simple
      - getting-started
    ```

    Feel free to use a different icon if it uses a reachable URL.
4. Add the new `accelerator.yaml` file, commit this change and push to your Git repository.

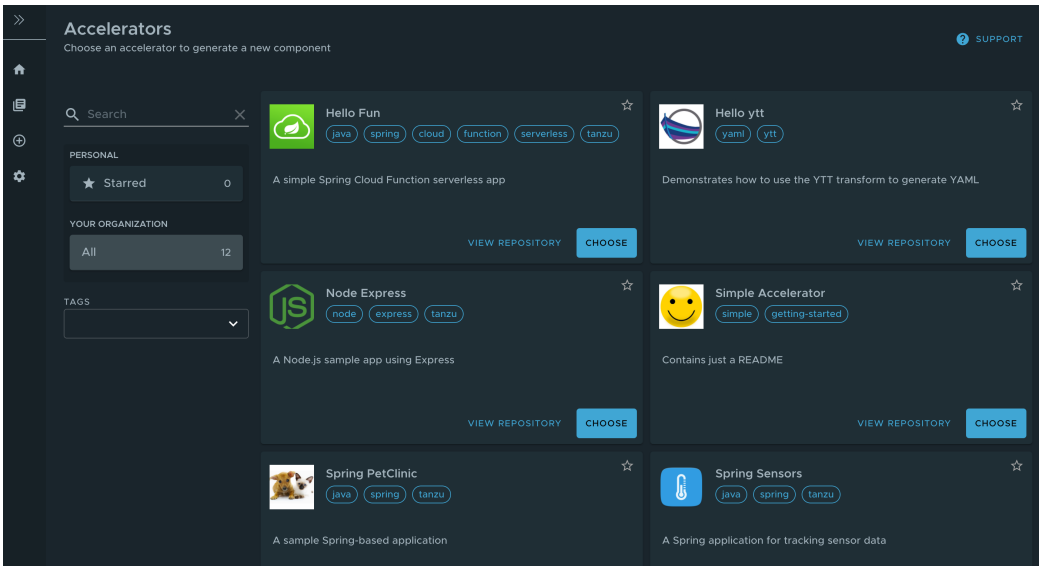
Publishing the new accelerator

1. To publish your new accelerator, run:

    ```
    tanzu accelerator create simple --git-repository ${GIT_REPOSITORY_URL} --git-br
    anch ${GIT_REPOSITORY_BRANCH}
    ```

    Where:

    - `GIT-REPOSITORY-URL` is the URL for your Git repository where the accelerator is located.
    - `GIT-REPOSITORY-BRANCH` is the name of the branch where you pushed the new `accelerator.yaml` file.
2. Refresh Tanzu Developer Portal or the Application Accelerator extension in VS Code to reveal the newly published accelerator. It might take a few seconds to refresh the catalog and add an entry for your new accelerator.



Alternatively, use the Tanzu CLI to create a separate manifest file and apply it to the cluster.

1. Create a `simple-manifest.yaml` file and add the following content, filling in with your Git repository and branch values.

   ```
   apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
   kind: Accelerator
   metadata:
     name: simple
     namespace: accelerator-system
   spec:
     git:
       url: YOUR-GIT-REPOSITORY-URL
       ref:
         branch: YOUR-GIT-BRANCH
   ```

2. To apply the `simple-manifest.yaml`, run this command in your terminal in the directory where you created this file:

   ```
   tanzu accelerator apply -f simple-manifest.yaml
   ```

Using accelerator fragments

Accelerator fragments are reusable accelerator components that can provide options, files, or transforms. They can be imported from accelerators using an `import` entry and the transforms from the fragment can be referenced in an `InvokeFragment` transform in the accelerator that is declaring the import. For additional details see InvokeFragment transform.

The accelerator samples include three fragments - `java-version`, `tap-initialize`, and `live-update`. See vmware-tanzu/application-accelerator-samples/fragments Git repository for the content of these fragments.

To discover what fragments are available to use, run:

```
tanzu accelerator fragment list
```

Look a the `java-version` fragment as an example. It contains the following `accelerator.yaml` file:

```
accelerator:
  options:
  - name: javaVersion
    inputType: select
```

```
    label: Java version to use
    choices:
    - value: "1.8"
      text: Java 8
    - value: "11"
      text: Java 11
    - value: "17"
      text: Java 17
    defaultValue: "11"
    required: true

engine:
  merge:
    - include: [ "pom.xml" ]
      chain:
      - type: ReplaceText
        regex:
          pattern: "<java.version>.*<"
          with: "'<java.version>' + #javaVersion + '<'"
    - include: [ "build.gradle" ]
      chain:
      - type: ReplaceText
        regex:
          pattern: "sourceCompatibility = .*"
          with: "'sourceCompatibility = ''' + #javaVersion + '''\n'"
    - include: [ "config/workload.yaml" ]
      chain:
      - type: ReplaceText
        condition: "#javaVersion == '17'"
        substitutions:
          - text: "spec:"
            with: "'spec:\n  build:\n    env:\n    - name: BP_JVM_VERSION\n      valu
e: \"17\"\n'"
```

This fragment contributes the following to any accelerator that imports it:

1. An option named `javaVersion` with three choices `Java 8`, `Java 11`, and `Java 17`
2. Three `ReplaceText` transforms:
    - If the accelerator has a `pom.xml` file, then what is specified for `<java.version>` is replaced with the chosen version.
    - If the accelerator has a `build.gradle` file, then what is specified for `sourceCompatibility` is replaced with the chosen version.
    - If the accelerator has a `config/workload.yaml` file, and the user selected “Java 17” then a build environment entry of BP_JVM_VERSION is inserted into the `spec:` section.

Deploying accelerator fragments

To deploy new fragments to the accelerator system, use the new `tanzu accelerator fragment create` CLI command or apply a custom resource manifest file with either `kubectl apply` or the `tanzu accelerator apply` commands.

The resource manifest for the `java-version` fragment looks like this:

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Fragment
metadata:
  name: java-version
  namespace: accelerator-system
spec:
  displayName: Select Java Version
```

```
  git:
    ref:
      tag: GIT_TAG_VERSION
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
    subPath: fragments/java-version
```

Where `GIT-TAG-VERSION` is the Git tag of the `java-version` fragment. For example, `tap-1.4.0` is a valid Git tag for the `java-version` fragment.

To create the fragment, save the above manifest in a `java-version.yaml` file) and run:

```
tanzu accelerator apply -f ./java-version.yaml
```

Note

The `accelerator apply` command can be used to apply both Accelerator and Fragment resources.

To avoid having to create a separate manifest file, run:

```
tanzu accelerator fragment create java-version \
  --git-repo https://github.com/vmware-tanzu/application-accelerator-samples.git \
  --git-tag ${GIT_TAG_VERSION} \
  --git-sub-path fragments/java-version
```

Where `GIT-TAG-VERSION` is the Git tag of the `java-version` fragment. For example,`tap-1.4.0` is a valid Git tag for the `java-version` fragment.

Now you can use this `java-version` fragment in an accelerator:

```
accelerator:
  displayName: Hello Fragment
  description: A sample app
  tags:
  - java
  - spring
  - cloud
  - tanzu

  imports:
  - name: java-version

engine:
  merge:
    - include: ["**/*"]
    - type: InvokeFragment
      reference: java-version
```

The earlier accelerator imports the `java-version` which, as seen earlier, provides an option to select the Java version to use for the project. It then instructs the engine to run the transforms provided in the fragment that updates the Java version used in `pom.xml` or `build.gradle` files from the accelerator.

For more detail on the use of fragments, see InvokeFragment transform.

Next steps

Learn how to:

- Write an accelerator.yaml.

- Configure accelerators with Accelerator Custom Resources.
- Manipulate files using Transforms.
- Use SpEL in the accelerator.yaml file.

Create accelerators

This topic tells you how to create an accelerator in Tanzu Developer Portal.

An accelerator contains your conforming code and configurations that developers can use to create new projects that by default follow the standards defined in your accelerators.

Prerequisites

The following prerequisites are required to create an accelerator:

- Application Accelerator is installed. For information about installing Application Accelerator, see Installing Application Accelerator for VMware Tanzu.
- You can access Tanzu Developer Portal from a browser or use the Application Accelerator extension for VS Code.
    - For more information about Tanzu Developer Portal, see Overview of Tanzu Developer Portal.
    - For more information about Application Accelerator extension for VS Code, see Application Accelerator Visual Studio Code extension.
- kubectl is installed and authenticated with admin rights for your target cluster.

Getting started

You can use any Git repository to create an accelerator. You need the URL of the repository to create an accelerator.

For this example, the Git repository is `public` and contains a `README.md` file. These are options available when you create repositories on GitHub.

Use the following procedure to create an accelerator based on this Git repository:

1. Clone your Git repository.
2. Create a file named `accelerator.yaml` in the root directory of this Git repository.
3. Add the following content to the `accelerator.yaml` file:

    ```
    accelerator:
      displayName: Simple Accelerator
      description: Contains just a README
      iconUrl: https://images.freecreatives.com/wp-content/uploads/2015/05/smiley-5
    59124_640.jpg
      tags:
      - simple
      - getting-started
    ```

    Feel free to use a different icon if it uses a reachable URL.
4. Add the new `accelerator.yaml` file, commit this change and push to your Git repository.

Publishing the new accelerator

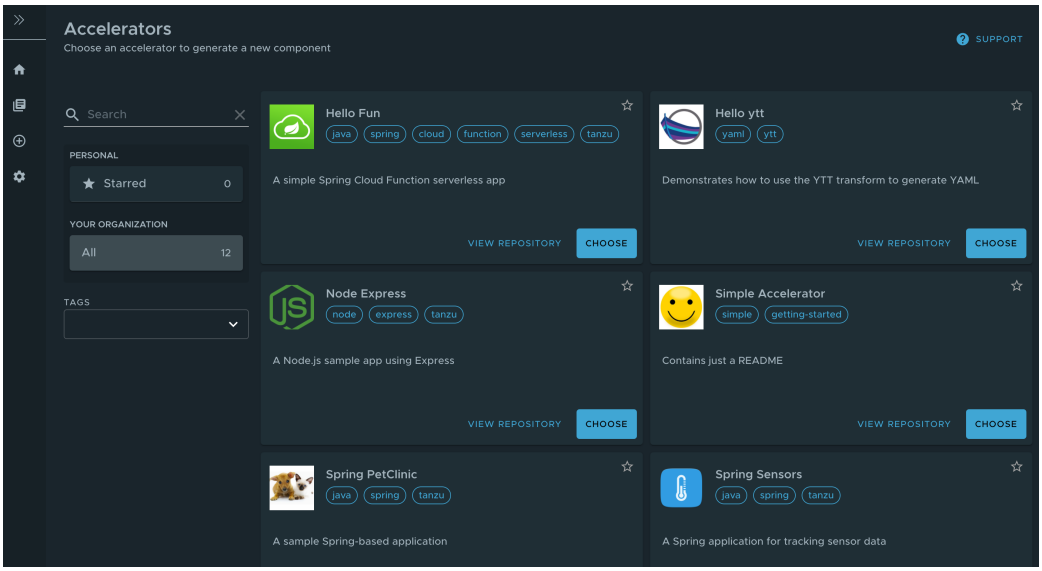
1. To publish your new accelerator, run:

```
tanzu accelerator create simple --git-repository ${GIT_REPOSITORY_URL} --git-br
anch ${GIT_REPOSITORY_BRANCH}
```

Where:

- `GIT-REPOSITORY-URL` is the URL for your Git repository where the accelerator is located.
- `GIT-REPOSITORY-BRANCH` is the name of the branch where you pushed the new `accelerator.yaml` file.

2. Refresh Tanzu Developer Portal or the Application Accelerator extension in VS Code to reveal the newly published accelerator. It might take a few seconds to refresh the catalog and add an entry for your new accelerator.



Alternatively, use the Tanzu CLI to create a separate manifest file and apply it to the cluster.

1. Create a `simple-manifest.yaml` file and add the following content, filling in with your Git repository and branch values.

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: simple
  namespace: accelerator-system
spec:
  git:
    url: YOUR-GIT-REPOSITORY-URL
    ref:
      branch: YOUR-GIT-BRANCH
```

2. To apply the `simple-manifest.yaml`, run this command in your terminal in the directory where you created this file:

```
tanzu accelerator apply -f simple-manifest.yaml
```

Using accelerator fragments

Accelerator fragments are reusable accelerator components that can provide options, files, or transforms. They can be imported from accelerators using an `import` entry and the transforms from the fragment can be referenced in an `InvokeFragment` transform in the accelerator that is declaring the import. For additional details see InvokeFragment transform.

The accelerator samples include three fragments - `java-version`, `tap-initialize`, and `live-update`. See vmware-tanzu/application-accelerator-samples/fragments Git repository for the content of these fragments.

To discover what fragments are available to use, run:

```
tanzu accelerator fragment list
```

Look a the `java-version` fragment as an example. It contains the following `accelerator.yaml` file:

```
accelerator:
  options:
  - name: javaVersion
    inputType: select
    label: Java version to use
    choices:
    - value: "1.8"
      text: Java 8
    - value: "11"
      text: Java 11
    - value: "17"
      text: Java 17
    defaultValue: "11"
    required: true

engine:
  merge:
    - include: [ "pom.xml" ]
      chain:
      - type: ReplaceText
        regex:
          pattern: "<java.version>.*<"
          with: "'<java.version>' + #javaVersion + '<'"
    - include: [ "build.gradle" ]
      chain:
      - type: ReplaceText
        regex:
          pattern: "sourceCompatibility = .*"
          with: "'sourceCompatibility = ''' + #javaVersion + '''\n'"
    - include: [ "config/workload.yaml" ]
      chain:
      - type: ReplaceText
        condition: "#javaVersion == '17'"
        substitutions:
          - text: "spec:"
            with: "'spec:\n  build:\n    env:\n    - name: BP_JVM_VERSION\n      valu
e: \"17\"'"
```

This fragment contributes the following to any accelerator that imports it:

1. An option named `javaVersion` with three choices `Java 8`, `Java 11`, and `Java 17`
2. Three `ReplaceText` transforms:
    - If the accelerator has a `pom.xml` file, then what is specified for `<java.version>` is replaced with the chosen version.
    - If the accelerator has a `build.gradle` file, then what is specified for `sourceCompatibility` is replaced with the chosen version.
    - If the accelerator has a `config/workload.yaml` file, and the user selected “Java 17” then a build environment entry of BP_JVM_VERSION is inserted into the `spec:` section.

Deploying accelerator fragments

To deploy new fragments to the accelerator system, use the new `tanzu accelerator fragment create` CLI command or apply a custom resource manifest file with either `kubectl apply` or the `tanzu accelerator apply` commands.

The resource manifest for the `java-version` fragment looks like this:

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Fragment
metadata:
  name: java-version
  namespace: accelerator-system
spec:
  displayName: Select Java Version
  git:
    ref:
      tag: GIT_TAG_VERSION
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
    subPath: fragments/java-version
```

Where `GIT-TAG-VERSION` is the Git tag of the `java-version` fragment. For example, `tap-1.4.0` is a valid Git tag for the `java-version` fragment.

To create the fragment, save the above manifest in a `java-version.yaml` file) and run:

```
tanzu accelerator apply -f ./java-version.yaml
```

Note

The `accelerator apply` command can be used to apply both Accelerator and Fragment resources.

To avoid having to create a separate manifest file, run:

```
tanzu accelerator fragment create java-version \
  --git-repo https://github.com/vmware-tanzu/application-accelerator-samples.git \
  --git-tag ${GIT_TAG_VERSION} \
  --git-sub-path fragments/java-version
```

Where `GIT-TAG-VERSION` is the Git tag of the `java-version` fragment. For example,`tap-1.4.0` is a valid Git tag for the `java-version` fragment.

Now you can use this `java-version` fragment in an accelerator:

```
accelerator:
  displayName: Hello Fragment
  description: A sample app
  tags:
  - java
  - spring
  - cloud
  - tanzu

  imports:
  - name: java-version

engine:
  merge:
    - include: ["**/*"]
```

```
    - type: InvokeFragment
      reference: java-version
```

The earlier accelerator imports the `java-version` which, as seen earlier, provides an option to select the Java version to use for the project. It then instructs the engine to run the transforms provided in the fragment that updates the Java version used in `pom.xml` or `build.gradle` files from the accelerator.

For more detail on the use of fragments, see InvokeFragment transform.

Next steps

Learn how to:

- Write an accelerator.yaml.
- Configure accelerators with Accelerator Custom Resources.
- Manipulate files using Transforms.
- Use SpEL in the accelerator.yaml file.

Create an accelerator.yaml file in Application Accelerator

This topic tells you how to use Application Accelerator to create an `accelerator.yaml` file in Tanzu Application Platform (commonly known as TAP).

By including an `accelerator.yaml` file in your Accelerator repository, you can declare input options that users fill in using a form in the UI. Those option values control processing by the template engine before it returns the zipped output files. For more information, see the Sample accelerator.

When there is no `accelerator.yaml`, the repository still works as an accelerator but the files are passed unmodified to users.

`accelerator.yaml` has two top-level sections: `accelerator` and `engine`.

Accelerator

This section documents how an accelerator is presented to users in the web UI. For example:

```
accelerator:
  displayName: Hello Fun
  description: A simple Spring Cloud Function serverless app
  iconUrl: https://raw.githubusercontent.com/vmware-tanzu/application-accelerator-samp
les/main/icons/icon-cloud.png
  tags:
  - java
  - spring

  options:
  - name: deploymentType
    inputType: select
    choices:
    - value: none
      text: Skip Kubernetes deployment
    - value: k8s-simple
      text: Kubernetes deployment and service
    - value: knative
      text: Knative service
    defaultValue: k8s-simple
    required: true
```

Accelerator metadata

These properties are in accelerator listings such as the web UI:

- **displayName**: A human-readable name.
- **description**: A more detailed description.
- **iconUrl**: A URL pointing to an icon image.
- **tags**: A list of tags used to filter accelerators.

Accelerator options

The list of options is passed to the UI to create input text boxes for each option.

The following option properties are used by both the UI and the engine.

- **name**: Each option must have a unique, camelCase name. The option value entered by a user is made available as a SPeL variable name. For example, `#deploymentType`.

  You can specify your own default name by including:

  ```
  options:
  - name: projectName
  label: Name
  inputType: text
  defaultValue: myname
  required: true
  ```

- **dataType**: Data types that work with the UI are:

  - `string`
  - `boolean`
  - `number`
  - Custom types defined in the accelerator `types` section
  - Arrays of these such as `[string]`, `[number]`, and so on.

  Most input types return a string, which is the default. Use Boolean values with `checkbox`.

- **defaultValue**: This literal value pre-populates the option. Ensure that it's type matches the dataType. For example, use `["text 1", "text 2"]` for the dataType `[string]`. Options without a `defaultValue` can trigger a processing error if the user doesn't provide a value for that option.

- **validationRegex**: When present, a regex validates the string representation of the option value *when set*. It doesn't apply when the value is blank. As a consequence, don't use the regex to enforce prerequisites. See **required** for that purpose.

  This regex is used in several layers in Application Accelerator, built using several technologies, for example, JavaScript and Java. So refrain from using "exotic" regex features. Also, the regex applies to portions of the value by default. That is, `[a-z ]+` matches `Hello world` despite the capital `H`. To apply it to the whole value (or just start/end), anchor it using `^` and `$`.

  Finally, backslashes in a YAML string using double quotes must be escaped, so to match a number, write `validationRegex: "\\d+"` or use another string style.

The following option properties are for UI purposes only.

- **label**: A human-readable version of the `name` identifying the option.
- **description**: A tooltip to accompany the input.

- **inputType**:
  - `text`: The default input type.
  - `textarea`: Single text value with larger input allowing line breaks.
  - `checkbox`: Ideal for Boolean values or multi-value selection from choices.
  - `select`: Single-value selection from choices using a drop-down menu.
  - `radio`: Alternative single-value selection from choices using buttons.
- **choices**: This is a list of predefined choices. Users can select from the list in the UI. Choices are supported by `checkbox`, `select`, and `radio`. Each choice must have a `text` property for the displayed text, and a `value` property for the value that the form returns for that choice. The list is presented in the UI in the same order as it is declared in `accelerator.yaml`.
- **required**: `true` forces users to enter a value in the UI.
- **dependsOn**: This is a way to control visibility by specifying the `name` and optional `value` of another input option. When the other option has a value exactly equal to `value`, or `true` if no `value` is specified, then the option with `dependsOn` is visible. Otherwise, it is hidden. Ensure that the value matches the dataType of the `dependsOn` option. For example, a multi-value option (`dataType = [string]`) such as a `checkbox` uses `[matched-value]` to trigger another option when `matched-value` (and only `matched-value`) is selected. See the following section for more information about `dependsOn`.

DependsOn and multi-value dataType

`dependsOn` tests for strict equality, even for multi-valued options. This means that a multi-valued option must not be used to trigger several other options unfolding, one for each value. Instead, use several single-valued options:

Instead of

```
options:
  - name: toppings
    dataType: [string]
    inputType: checkbox
    choices:
      - value: vegetables
        text: Vegetables
      - value: meat
        text: Meat
        ...
  - name: vegType
    dependsOn:
      name: toppings
      value: [vegetables] # or vegetables, this won't do what you want either
  - name: meatType
    dependsOn:
      name: toppings
      value: [meat]
  ...
```

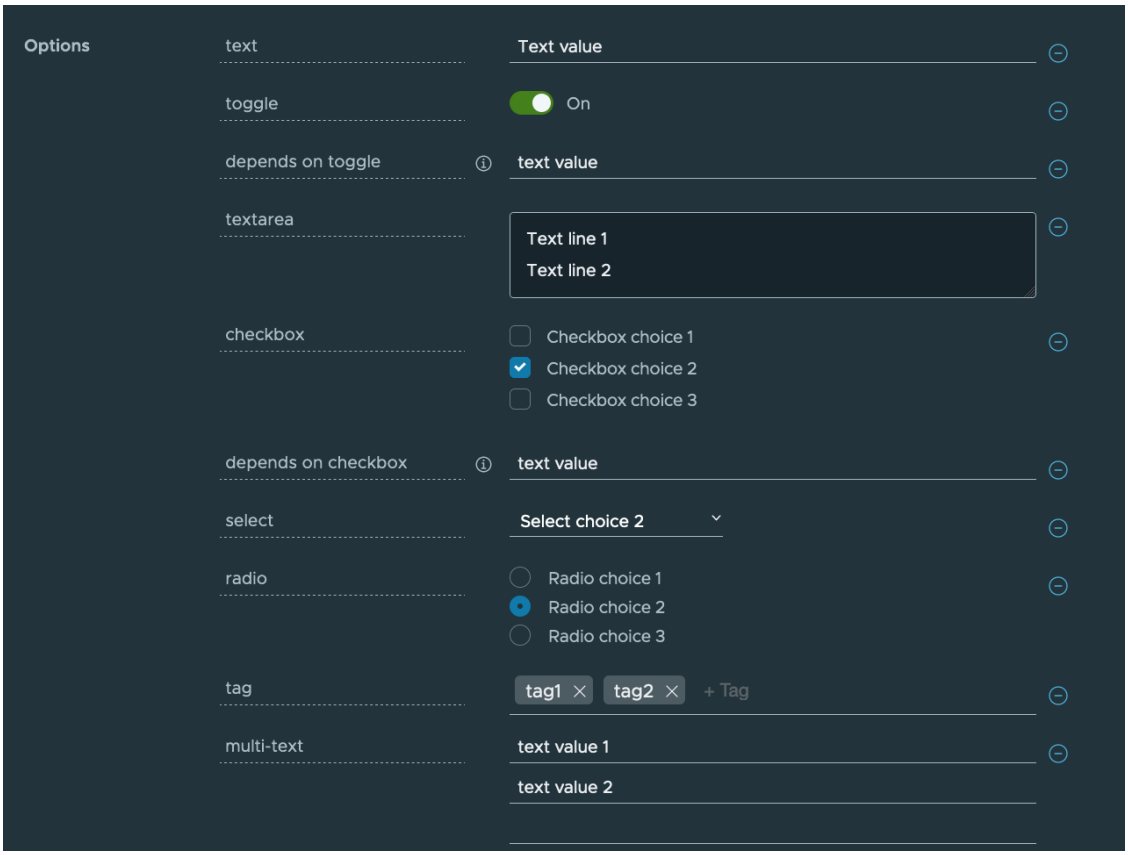
do this:

```
options:
  - name: useVeggies
    dataType: boolean
    inputType: checkbox
    label: Vegetables
  - name: useMeat
    dataType: boolean
```

```
    inputType: checkbox
    label: Meat
  - name: vegType
    dependsOn:
      name: useVeggies
      value: true
  - name: meatType
    dependsOn:
      name: useMeat
      value: true
  ...
```

Examples

The later screenshot and `accelerator.yaml` file snippet that follows demonstrates each `inputType`. You can also see the GitHub sample demo-input-types.




```
accelerator:
  displayName: Demo Input Types
  description: "Accelerator with options for each inputType"
  iconUrl: https://raw.githubusercontent.com/vmware-tanzu/application-accelerator-samp
les/main/icons/icon-cloud.png
  tags: ["demo", "options"]

  options:

  - name: text
    display: true
    defaultValue: Text value

  - name: toggle
    display: true
    dataType: boolean
    defaultValue: true
```

```
  - name: dependsOnToggle
    label: 'depends on toggle'
    description: Visibility depends on the value of the toggle option being true.
    dependsOn:
      name: toggle
    defaultValue: text value

  - name: textarea
    inputType: textarea
    display: true
    defaultValue: |
      Text line 1
      Text line 2

  - name: checkbox
    inputType: checkbox
    display: true
    dataType: [string]
    defaultValue:
      - value-2
    choices:
      - text: Checkbox choice 1
        value: value-1
      - text: Checkbox choice 2
        value: value-2
      - text: Checkbox choice 3
        value: value-3

  - name: dependsOnCheckbox
    label: 'depends on checkbox'
    description: Visibility depends on the checkbox option containing exactly value va
lue-2.
    dependsOn:
      name: checkbox
      value: [value-2]
    defaultValue: text value

  - name: select
    inputType: select
    display: true
    defaultValue: value-2
    choices:
      - text: Select choice 1
        value: value-1
      - text: Select choice 2
        value: value-2
      - text: Select choice 3
        value: value-3

  - name: radio
    inputType: radio
    display: true
    defaultValue: value-2
    choices:
      - text: Radio choice 1
        value: value-1
      - text: Radio choice 2
        value: value-2
      - text: Radio choice 3
        value: value-3

engine:
  type: YTT
```

Engine

The engine section describes how to take the files from the accelerator root directory and transform them into the contents of a generated project file.

The YAML notation here defines what is called a transform. A transform is a function on a set of files. It uses a set of files as input. It produces a modified set of files as output derived from this input.

Different types of transforms do different tasks:

- Filtering the set of files: that is, removing, or keeping files that match certain criteria.
- Changing the contents of files. For example, replacing some strings in the files.
- Renaming or moving files: that is, changing the paths of the files.

The notation also provides the composition operators `merge` and `chain`, which enable you to create more complex transforms by composing simpler transforms together.

The following is an example of what is possible. To learn the notation, see Introduction to transforms.

Engine example

```
engine:
  include:
    ["**/*.md", "**/*.xml", "**/*.gradle", "**/*.java"]
  exclude:
    ["**/secret/**"]
  let:
    - name: includePoms
      expression:
        "#buildType == 'Maven'"
    - name: includeGradle
      expression: "#buildType == 'Gradle'"
  merge:
    - condition:
        "#includeGradle"
      include: ["*.gradle"]
    - condition: "#includePoms"
      include: ["pom.xml"]
    - include: ["**/*.java", "README.md"]
      chain:
        - type: ReplaceText
          substitutions:
            - text: "Hello World!"
              with: "#greeting"
  chain:
    - type: RewritePath
      regex: (.*)simpleboot(.*)
      rewriteTo: "#g1 + #packageName + #g2"
    - type: ReplaceText
      substitutions:
        - text: simpleboot
          with: "#packageName"
  onConflict:
    Fail
```

Engine notation descriptions

This section describes the notations in the preceding example.

`engine` is the global transformation node. It produces the final set of files to be zipped and returned from the accelerator. As input, it receives all the files from the accelerator repository root. The properties in this node dictate how this set of files is transformed into a final set of files zipped as the accelerator result.

`engine.include` filters the set of files, retaining only those matching a list of path patterns. This ensures that that the accelerator only detects files in the repository that match the list of patterns.

`engine.exclude` further restricts which files are detected. The example ensures files in any directory called `secret` are never detected.

`engine.let` defines additional variables and assigns them values. These derived symbols function such as options, but instead of being supplied from a UI widget, they are computed by the accelerator itself.

`engine.merge` executes each of its children in parallel. Each child receives a copy of the current set of input files. These are files remaining after applying the `include` and `exclude` filters. Each of the children therefore produces a set of files. All the files from all the children are then combined, as if overlaid on top of each other in the same directory. If more than one child produces a file with the same path, the transform resolves the conflict by dropping the file contents from the earlier child and keeping the contents from the later child.

`engine.merge.chain` specifies additional transformations to apply to the set of files produced by this child. In the example, `ReplaceText` is only applied to Java files and `README.md`.

`engine.chain` applies transformation to all files globally. The chain has a list of child transformations. These transformations are applied after everything else in the same node. This is the global node. The children in a chain are applied sequentially.

`engine.onConflict` specifies how conflict is handled when an operation, such as merging, produces multiple files at the same path: - `Fail` raises an error when there is a conflict. - `UseFirst` keeps the contents of the first file. - `UseLast` keeps the contents of the last file. - `Append` keeps both by using `cat <first-file> <second-file>`.

Advanced accelerator use

Additional advanced features can be leveraged when writing an `accelerator.yaml`. For more information see, Creating dynamic parameters using custom types

Application Accelerator sample accelerator.yaml file

This topic provides you with a sample accelerator file to get you started writing your own accelerators in Tanzu Application Platform (commonly known as TAP).

```
accelerator:
  # The `accelerator` section serves to document how an accelerator is presented to th
e
  # user in the accelerator web UI.

  # displayName: a descriptive human-readable name. Make this short so as to look nice
  #              in a list of many accelerators shown to a user.
  displayName: Hello Spring Boot

  # description: a more detailed description that a user can see if they took a closer
  #              look at a particular accelerator.
  description: Simple Hello World Rest Service based on Spring Boot

  # iconUrl: Optional, a nice colorful, icon for your accelerator to make it stand out
visually.
  iconUrl: https://raw.githubusercontent.com/vmware-tanzu/application-accelerator-samp
les/main/icons/icon-cloud.png
```

```
  # tags: A list of classification tags. The UI allows users to search for accelerator
s based on tags
  tags:
    - Java
    - Spring
    - Function

  # options are parameters that can affect how the accelerator behaves.
  # The purpose of the options section is
  #   - to list all applicable options
  #   - describe each option in enough detail so that the UI can create
  #     a suitable input widget for it.
  options: # a list of options
    # a first option
    - name:
        greeting
        # name: each option must have a name.
        # This must be
        #   - camelCase
        #   - unique (i.e. no two options can have the same name)
        # This is like a variable used by the accelerator to refer to
        # and use the value during its execution.
        # This name is internal to your accelerator and is not shown to
        # the user.
      label:
        Greeting Message
        # A human readable version of the `name`. This is used to identify an
        # option to the user in the UI.
        # This should be short (so as not to look ugly in a ui with limited
        # space available for labeling the input widgets).
        # There are no limits on what characters can be used in the label (so spaces
        # are allowed).
      description:
        Greeting message displayed by the Hello World app.
        # An optional more detailed description / explanation that can be shown to
        # to the user in the UI when the short label alone might not be enough to unde
rstand
        # its purpose.
      dataType:
        string
        # type of data the accelerator expects during execution (this is
        # like the type of the 'variable'.
        # possible dataTypes are string, boolean, number or [string] (the latter meani
ng a
        # list of strings
      inputType:
        text
        # Related to the dataType but somewhat independent, this identifies the type
        # of widget shown in the ui. Available types are:
        # - text - the default
        # - textarea (single text value with larger input that allows linebreaks)
        # - checkbox - multivalue selection from choices
        # - select - single value selection from choices
        # - radio - alternative single value selection from choices
        # - tag - multivalue input ui for entering single-word tags
      required: true
      defaultValue: Hello Accelerator
    # second option:
    - name: packageName
      label: "Package Name"
      description: Name of Java package
      dataType: string
      inputType: text
      defaultValue: somepackage
    # another option:
```

```
    - name: buildType
      label: Build Type
      description: Choose whether to use Maven or Gradle to build the project.
      dataType: string
      inputType: select
      choices:
        - value: Maven
          text: Maven (pom.xml)
        - value: Gradle
          text: Gradle (build.gradle)

# The 'engine' section describes how to take the files from the accelerator
# repo root folder and 'transform' them into the contents of a generated project / zi
p.
# transformation operate on the files as a set and can do things like:
# - filtering the set of files (i.e. removing / keeping only files that match certain
criteria)
# - changing the contents of a file (e.g. replacing some strings in them)
# - renaming or moving files (changing the paths of the files)
engine:
  # this is the 'global' transformation node. It produces the final set of
  # files to be zipped and returned from the accelerator.
  # As input it receives all the files from the accelerator repo root.

  # The properties in this node dictate how this set of files is
  # transformed into a final set of files to zip up as the accelerator
  # result.

  include:
    ["**/*.md", "**/*.xml", "**/*.gradle", "**/*.java"]
    # This globally defined `include` filters the set of files
    # retaining only those matching a given list of path patterns.
    # This can ensure that only files in the repo matching the list of
    # patterns will be seen / considered by the accelerator.

  exclude:
    ["**/secret/**"]
    # This globally defined `exclude` further restricts what files are considered.
    # This example ensures files in any directory called `secret` are never considere
d.

  # Under 'let' you can define additional variables and assign them values
  # These 'derived symbols' function much like options, but instead of
  # being supplied from a UI widget, they are computed by the accelerator itself.
  let:
    - name: includePoms # name of a symbol, must be camelCase
      expression:
        "#buildType == 'Maven'" # <- SpEL expression given as a string. You must take
care to use
        # proper quotes to avoid yaml treating '#' as starting a comment.
    - name: includeGradle
      expression: "#buildType == 'Gradle'"
  merge: # This merge section executes each of its children 'in parallel'.
    # Each child receives a copy of the current set of input files.
    # (i.e. the files that are remaining after considering the `include` and `exclude
`.
    # Each of the children thus produces a set of files.
    # Merge then combines all the files from all the children, as if by overlaying the
m on top of each other
    # in the same directory. If more than one child produces a file with the same pat
h,
    # this 'conflict' is resolved by dropping the file contents from the earlier child
    # and keeping only the later one.
    # merge child 1: this child node wants to contribute 'gradle' files to the final r
esult
    - condition:
```

```
        "#includeGradle" # this child is deactivated if the Gradle option was not sele
cted by the user
        # A deactivated child doesn't contribute anything to the final result.
      include: ["*.gradle"] # this child only focusses on gradle files (ignoring all o
ther files)
    # merge child 2: this child wants to contribute 'pom' files to the final result
    - condition: "#includePoms"
      include: ["pom.xml"]
    # merge child 3: this child wants to contribute Java code and README.md to the fin
al result
    - include: ["**/*.java", "README.md"]
      # Using: chain you can specify additional transformations to be applied to the s
et
      # of files produced by this child (i.e. the `ReplaceText` below is only applied
to .java files and README.md)
      chain:
        - type: ReplaceText
          substitutions:
            - text: "Hello World!"
              with: "#greeting"
  chain:
    # Globally specified chain, works like the one `from merge child 3`. But because i
t is global, it
    # applies transformation to all files globally.
    #
    # The chain has a list of child transformations. These transformation are applied
after everything else
    # in the same node (here we are in the 'global node').
    #
    # The children in a chain are applied sequentially.
    - type: RewritePath
      regex: (.*)simpleboot(.*)
      rewriteTo: "#g1 + #packageName + #g2" # SpEL expression. You can use '#g1' and
'#g2' to reference 'match groups'
    - type: ReplaceText
      substitutions:
        - text: simpleboot
          with: "#packageName"
  onConflict:
    Fail # other values are `UseFirst`, `UseLast`, or `Append`
    # when merging (or really any operation) produces multiple files at the same path
    # this defines how that conflict is handled.
    # Fail: raise an error when conflict happens
    # UseFirst: keep the contents of the first file
    # UseLast: keep the contents of the last file
    # Append: keep both as by using `cat <first-file> <second-file>`).
```

Use transforms in Application Accelerator

This topic tells you about using transforms with Application Accelerator.

When the accelerator engine executes the accelerator, it produces a ZIP file containing a set of files. The purpose of the `engine` section is to describe precisely how the contents of that ZIP file is created.

```
accelerator:
  ...
engine:
  <transform-definition>
```

Why transforms?

When you run an accelerator, the contents of the accelerator produce the result. It is made up of subsets of the files taken from the accelerator `<root>` directory and its subdirectories. You can copy the files as is, or transform them in a number of ways before adding them to the result.

The YAML notation in the `engine` section defines a transformation that takes as input a set of files (in the `<root>` directory of the accelerator) and produces as output another set of files, which are put into the ZIP file.

Every transform has a `type`. Different types of transform have different behaviors and different YAML properties that control precisely what they do.

In the following example, a transform of type `Include` is a filter. It takes as input a set of files and produces as output a subset of those files, retaining only those files whose path matches any one of a list of `patterns`.

If the accelerator has something like this:

```
engine:
  type: Include
  patterns: ['**/*.java']
```

This accelerator produces a ZIP file containing all the `.java` files from the accelerator `<root>` or its subdirectories but nothing else.

Transforms can also operate on the contents of a file, instead of merely selecting it for inclusion.

For example:

```
type: ReplaceText
substitutions:
- text: hello-fun
  with: "#artifactId"
```

This transform looks for all instances of a string `hello-fun` in all its input files and replaces them with an `artifactId`, which is the result of evaluating a SpEL expression.

Combining transforms

From the preceding examples, you can see that transforms such as `ReplaceText` and `Include` are too primitive to be useful by themselves. They are meant to be the building blocks of more complex accelerators.

To combine transforms, provide two operators called `Chain` and `Merge`. These operators are recursive in the sense that they compose a number of child transforms to create a more complex transform. This allows building arbitrarily deep and complex trees of nested transform definitions.

The following example shows what each of these two operators does and how they are used together.

Chain

Because transforms are functions whose input and output are of the same type (a set of files), you can take the output of one function and feed it as input to another. This is what `Chain` does. In mathematical terms, `Chain` is function composition.

You might, for example, want to do this with the `ReplaceText` transform. Used by itself, it replaces text strings in all the accelerator input files. What if you wanted to apply this replacement to only a subset of the files? You can use an `Include` filter to select only a subset of files of interest and chain that subset into `ReplaceText`.

For example:

```
type: Chain
transformations:
- type: Include
  patterns: ['**/pom.xml']
- type: ReplaceText
  substitutions:
  - text: hello-fun
    with: "#artifactId"
```

Merge

Chaining `Include` into `ReplaceText` limits the scope of `ReplaceText` to a subset of the input files. It also eliminates all other files from the result.

For example:

```
engine:
  type: Chain
  transformations:
  - type: Include
    patterns: ['**/pom.xml']
  - type: ReplaceText
    substitutions:
    - text: hello-fun
      with: "#artifactId"
```

The preceding accelerator produces a ZIP file that only contains `pom.xml` files and nothing else.

What if you also wanted other files in that ZIP? Perhaps you want to include some Java files as well, but don't want to apply the same text replacement to them.

You might be tempted to write something such as:

```
engine:
  type: Chain
  transformations:
  - type: Include
    patterns: ['**/pom.xml']
  - type: ReplaceText
    ...
  - type: Include
    patterns: ['**/*.java']
```

However, that doesn't work. If you chain non-overlapping includes together like this, the result is an empty result set. The reason is that the first include retains only `pom.xml` files. These files are fed to the next transform in the chain. The second include only retains `.java` files, but because there are only `pom.xml` files left in the input, the result is an empty set.

This is where `Merge` comes in. A `Merge` takes the outputs of several transforms executed independently on the same input sourceset and combines or merges them together into a single sourceset.

For example:

```
engine:
  type: Merge
  sources:
  - type: Chain
    - type: Include
      patterns: ['**/pom.xml']
    - type: ReplaceText
      ...
```

```
  - type: Include
    patterns: ['**/*.java']
```

The preceding accelerator produces a result that includes both:

- The `pom.xml` files with some text replacements applied to them.
- Verbatim copies of all the `.java` files.

Shortened notation

It becomes cumbersome and verbose to combine transforms such as `Include`, `Exclude`, and `ReplaceText` with explicit `Chain` and `Merge` operators. Also, there is a common composition pattern to using them. Specifically, select an interesting subset using includes and excludes, apply a chain of additional transformations to the subset, and merge the result with the results of other transforms. That is why there is a transform known the `Combo` transform that combines `Include`, `Exclude`, `Merge`, and `Chain`.

For example:

```
type: Combo
include: ['**/*.txt', '**/*.md']
exclude: ['**/secret/*']
merge:
- <transform-definition>
- ...
chain:
- <transform-definition>
- ...
```

Each of the properties in this `Combo` transform is optional if you specify at least one.

Notice how each of the properties `include`, `exclude`, `merge`, and `chain` corresponds to the name of a type of transform, only spelled with lowercase letters.

If you specify only one of the properties, the `Combo` transform behaves exactly as if you used that type of transformation by itself.

For example:

```
merge: ...
```

Behaves the same as:

```
type: Merge
sources: ...
```

When you do specify multiple properties at the same time, the `Combo` transform composes them together and combines `Merge` and `Chain`.

For example:

```
include: ['**/*.txt', '**.md']
chain:
- type: ReplaceText
  ...
```

Is the same as:

```
type: Chain
transformations:
- type: Include
```

```
  patterns: ['**/*.txt', '**.md']
- type: Chain
  transformations:
  - type: ReplaceText
    ...
```

When you use all of the properties of `Combo` at once:

```
include: I
exclude: E
merge:
- S1
- S2
chain:
- T1
- T2
```

This is equivalent to:

```
type: Chain
transformations:
- type: Include
  patterns: I
- type: Exclude
  patterns: E
- type: Merge
  sources:
  - S1
  - S2
- T1
- T2
```

A Combo of one?

You can use the `Combo` as a convenient shorthand for a single type of annotation. However, though you can use it to combine multiple types, and though that is its main purpose, that doesn't mean you have to.

For example:

```
include: ["**/*.java"]
```

This is a `Combo` transform (remember, `type: Combo` is optional), but rather than combining multiple types of transforms, it only defines the `include` property. This makes it behaves exactly as an `Include` transform:

```
type: Include
patterns: ["**/*.java"]
```

It is usually more convenient to use a `Combo` transform to denote a single `Include`, `Exclude`, `Chain`, or `Merge` transform, because it is slightly shorter to write it as a `Combo` than writing it with an explicit `type:` property.

A common pattern with merge transforms

It is a common and useful pattern to use merges with overlapping contents to apply a transformation to a subset of files and then replace these changed files within a bigger context.

For example:

```
engine:
  merge:
  - include: ["**/*"]
  - include: ["**/pom.xml"]
    chain:
    - type: ReplaceText
        subsitutions: ...
```

The preceding accelerator copies all files from accelerator `<root>` while applying some text replacements only to `pom.xml` files. Other files are copied verbatim.

Here in more detail is how this works:

- **Transform A** is applied to the files from accelerator `<root>`. It selects all files, including `pom.xml` files.
- **Transform B** is *also* applied to the files from accelerator `<root>`. Again, `Merge` passes the same input independently to each of its child transforms. Transform B selects `pom.xml` files and replaces some text in them.

So both **Transform A** and **Transform B** output `pom.xml` files. The fact that both result sets contain the same file, and with different contents in them in this case, is a conflict that has to be resolved. By default, `Combo` follows a simple rule to resolve such conflicts: take the contents from the last child. Essentially, it behaves as if you overlaid both result sets one after another into the same location. The contents of the latter overwrite any previous files placed there by the earlier.

In the preceding example, this means that while both **Transform A** and **Transform B** produce contents for `pom.xml`, the contents from **Transform B** “wins.” So you get the version of the `pom.xml` that has text replacements applied to it rather than the verbatim copy from **Transform A**.

Conditional transforms

Every `<transform-definition>` can have a `condition` attribute.

```
  - condition: "#k8sConfig == 'k8s-resource-simple'"
    include: [ "kubernetes/app/*.yaml" ]
    chain:
      - type: ReplaceText
        substitutions:
        - text: hello-fun
          with: "#artifactId"
```

When a transform’s condition is `false`, that transform is deactivated. This means it is replaced by a transform that does nothing. However, doing nothing can have different meanings depending on the context:

- When in the context of a `Merge`, a deactivated transform behaves like something that returns an empty set. A `Merge` adds things together using a kind of union; adding an empty set to union essentially does nothing.
- When in the context of a `'Chain` however, a deactivated transform behaves like the `identity` function instead (that is, `lambda (x) => x`). When you chain functions together, a value is passed through all functions in succession. So each function in the chain has the chance to do something by returning a different modified value. If you are a function in a chain, to do nothing means to return the input you received unchanged as your output.

If a transform is deactivated in the context of your accelerator definition, it evaluates to false and is ignored. Your accelerator behaves as if you deleted or commented out that transform’s YAML text from the accelerator definition file.

The following examples illustrate both cases.

Conditional 'Merge' transform

This example, **transform A**, has a conditional transform in a `Merge` context:

```
merge:
  - condition: "#k8sConfig == 'k8s-resource-simple'"
    include: [ "kubernetes/app/*.yaml" ]
    chain:
      ...
  - include: [ "pom.xml" ]
    chain:
      ...
```

If the condition of **transform A** is `false`, it is replaced with an empty set because it is used in a `Merge` context. This has the same effect as if the whole of **transform A** was deleted or commented out:

```
merge:
  - include: [ "pom.xml" ]
    chain:
      ...
```

In this example, if the condition is `false`, only `pom.xml` file is in the result.

Conditional 'Chain' transform

In the following example, some conditional transforms are used in a `Chain` context:

```
merge:
- include: [ '**/*.json' ]
  chain:
  - type: ReplaceText
    condition: '#customizeJson'
    substitutions: ...
  - type: JsonPrettyPrint
    condition: '#prettyJson'
    indent: '#jsonIndent'
```

In the preceding example, both **transform A** and **transform B** are conditional and used in a `Chain` context. **Transform A** is chained after the `include` transform. Whereas **transform B** is chained after **transform A**. When either of these conditions is `false`, the corresponding transform behaves like the identity function. Namely, whatever set of files it receives as input is exactly what it returns as output.

For example, if **transform A**'s condition is `false`, it behaves as if **transform A** isn't there. **Transform A** is chained after `include` so it receives the `include`'s result, returns it unchanged, and this is passed to **transform B**. In other words, the result of the `include` is passed as is to **transform B**.

A small gotcha with using conditionals in merge transforms

As mentioned earlier, it is a useful pattern to use merges with overlapping contents. But you must be careful using this in combination with conditional transforms.

For example:

```
engine:
  merge:
  - include: ["**/*"]
  - include: ["**/pom.xml"]
    chain:
```

```
    - type: ReplaceText
      subsitutions: ...
```

If you only want to include pom files, if you select a `useMaven` option, when you add a 'condition' to **transform B** to deactivate it, the final result still contains `pom.xml` files.:

```
engine:
  merge:
  - include: "**/*"
  - condition: '#useMaven'
    include: ["**/pom.xml"]
    chain:
    - type: ReplaceText
      subsitutions: ...
```

This is because if a transform is deactivated in the context of your accelerator definition, it evaluates to false and is ignored. So when `#useMaven` is `false`, the example reduces to:

```
engine:
  merge:
  - include: ["**/*"]
```

This accelerator copies all files from accelerator `<root>`, including `pom.xml`.

There are several ways to avoid this. One is to ensure the `pom.xml` files are not included in **transform A** by explicitly excluding them:

```
  ...
  - include: ["**/*"]
    exclude: ["**/pom.xml"]
  ...
```

Another way is to apply the exclusion of `pom.xml` conditionally in a `Chain` after the main transform:

```
engine:
  merge:
  - include: ["**/*"]
  - include: ["**/pom.xml"]
    chain:
    - type: ReplaceText
        subsitutions: ...
  chain:
  - condition: '!#useMaven'
    exclude: ['**/pom.xml']
```

Merge conflict

The representation of the set of files upon which transforms operate is richer than what you can physically store on a file system. A key difference is that in this case, the set of files allows for multiple files with the same path to exist at the same time. When files are initially read from a physical file system, or a ZIP file, this situation does not arise. However, as transforms are applied to this input, it can produce results that have more than one file with the same path and yet different contents.

Earlier examples illustrated this happening through a `merge` operation. For example:

```
merge:
- include: ["**/*"]
- include: ["**/pom.xml"]
  chain:
```

```
  - type: ReplaceText
    subsitutions: ...
```

The result of the preceding `merge` is two files with path `pom.xml`, assuming there was a `pom.xml` file in the input. **Transform A** produces a `pom.xml` that is a verbatim copy of the input file. **Transform B** produces a modified copy with some text replaced in it.

It is impossible to have two files on a disk with the same path. Therefore, this conflict must be resolved before you can write the result to disk or pack it into a ZIP file.

As the example shows, merges are likely to give rise to these conflicts, so you might call this a "merge conflict." However, such conflicts can also arise from other operations. For example, `RewritePath`:

```
type: RewritePath
regex: '.*.md'
rewriteTo: "'docs/README.md'"
```

This example renames any `.md` file to `docs/README.md`. Assuming the input contains more than one `.md` file, the output contains multiple files with path `docs/README.md`. Again, this is a conflict, because there can only be one such file in a physical file system or ZIP file.

Resolving merge conflicts

By default, when a conflict arises, the engine doesn't do anything with it. Our internal representation for a set of files allows for multiple files with the same path. The engine carries on manipulating the files as is. This isn't a problem until the files must be written to disk or a ZIP file. If a conflict is still present at that time, an error is raised.

If your accelerator produces such conflicts, they must be resolved before writing files to disk. VMware provides the UniquePath transform. This transform allows you to specify what to do when more than one file has the same path. For example:

```
chain:
- type: RewritePath
  regex: '.*.md'
  rewriteTo: "'docs/README.md'"
- type: UniquePath
  strategy: Append
```

The result of the above transform is that all `.md` files are gathered up and concatenated into a single file at path `docs/README.md`. Another possible resolution strategy is to keep only the contents of one of the files. See Conflict Resolution.

Combo transform includes some convenient built-in support for conflict resolution. It automatically selects the `UseLast` strategy if none is explicitly supplied. You rarely, if ever, need to specify a conflict resolution strategy.

File ordering

As mentioned earlier, our set of files representation is richer than the files on a typical file system in that it allows for multiple files with the same path. Another way in which it is richer is that the files in the set are ordered. That is, a `FileSet` is more like an ordered list than an unordered set.

In most situations, the order of files in a `FileSet` doesn't matter. However, in conflict resolution it is significant. If you look at the preceding `RewritePath` example again, you might wonder about the order in which the various `.md` files are appended to each other. This ordering is determined by the order of the files in the input set.

So what is that order? In general, when files are read from disk to create a `FileSet`, you cannot assume a specific order. Yes, the files are read and processed in a sequential order, but the actual order is not well defined. It depends on implementation details of the underlying file system. The accelerator engine therefore does not ensure a specific order in this case. It only ensures that it preserves whatever ordering it receives from the file system, and processes files in accord with that order.

If you do not want the file order produced from reading directly from a file system and want to control the order of the sections in the `README.md` file, change the order of the merge children. `Merge` processes its children in order and reflects this order in the resulting output

For example:

```
chain:
  - merge:
      - include: ['README.md']
      - include: ['DEPLOYMENT.md']
        chain:
          - type: RewritePath
            rewriteTo: "'README.md'"
  - type: UniquePath
    strategy: Append
```

In this example, `README.md` from the first child of `merge` comes before `DEPLOYMENT.md` from the second child of `merge`.

Next steps

This introduction focused on an intuitive understanding of the `<transform-definition>` notation. This notation defines precisely how the accelerator engine generates new project content from the files in the accelerator root.

For more information, see:

- An exhaustive Reference of all built-in transform types
- A sample, commented accelerator.yaml to learn from a concrete example

Use custom types in Application Accelerator

This topic tells you how to declare new `types` in `accelerator.yaml`

Use these types for options declaration, in addition to the built-in types `string`, `number`, and `boolean`.

In `accelerator.yaml`, use the `types` entry (inside the top-level `accelerator` section) to define custom types.

The name must be an initial capital letter.

In the following example, the `struct` type definition is syntactically equivalent to a sequence of option definitions:

```
accelerator:
  options:
    ...
  types:
    - name: Task
      struct:
        - name: title
          dataType: string
          label: Title
```

```
          description: A sample title
        - name: details
          label: Task details
          description: Enter the task details
        - name: done
          dataType: boolean
          label: Done?
          defaultValue: false
```

This example creates a new *type* that is available for the `dataType` property of any option. For example,

```
accelerator:
  options:
    - name: myTask
      dataType: Task
  types:
    ...
```

UIs render similar to the following:

and associate the entered values to the `myTask` top-level name, resulting in the following example values submission (here represented using JSON notation):

```
{
  "myTask": { // Note the use of a nested object here
    "title": "Get job done!",
    "details": "Needs this asap",
    "done": false
  }
}
```

The type of the `myTask` value is `object` (in Javascript/JSON parlance) and `Map<String, ?>` when seen from the Java engine side.

The earlier example is technically possible with the custom types feature, but brings little benefit over having three options named to achieve the same result, for example, `myTaskTitle`, `myTaskDetails`, and `myTaskDone`. The value of custom types is when they are used in sequence types, allowing you to enter an unbounded list of structured data:

```
accelerator:
  options:
    - name: myTasks
      dataType: [Task]
  types:
    ...
```

Which might result in the following example submission (JSON):

```
{
  "myTasks": [ // Note the use of JSON array
    {  // with elem 0 being an object
      "details": "something",
      "done": true,
      "title": "The Title"
    },
    {  // and elem 1 as well, etc
      "details": "something else",
      "done": false,
      "title": "The other Title"
    }
  ]
}
```

Limitations

A `struct` custom type declaration is made of an ordered series of option definitions. The support and semantics for individual text boxes of option-definition-like elements when used in the type *declaration* are stated in the following example.

When *referencing* a custom type in an option definition, some previously valid properties of an option definition might become irrelevant or unsupported. This is stated in the following example:

```
accelerator:
  types:
    - name: MyType
      struct:
        - name: someField   # the "option name" will become a 'property' of the newly
created type
          dataType: string  # is the type of this single property. Typically, will be
a simple
                            # scalar type like string or number
          defaultValue: foo # supported and is the default if not overridden at usage
point by the option's defaultValue
          description: something # will become the description for the field's widget
          choices:              # supported
            - value: v
              text:  t
          validationRegex:      # validates that single property
          label:                # will become the "title" of the widget
          inputType:            # supported
          required:             # supported
          dependsOn:            # supported against other properties of THIS struct
    .. other fields
  options:
    - name: anOptionThatUsesACustomType
      dataType: MyType
      defaultValue: # supported, should then be an object (or array thereof)
      description:  # supported, is the description of the whole option (as opposed to
individual fields)
      label:        # supported, idem
      choices:      # NOT supported
        - value: v
          text:  t
      validationRegex: # NOT supported
      inputType:       # NOT supported
      required:        # technically supported, useless in practice
      dependsOn:       # OK to depend on another option
```

Interaction with SpEL

Everywhere that SpEL is used in the engine syntax, accelerator authors might use SpEL syntax for accessing properties or array elements:

```
#myTasks[2]['done']
```

Do not use array indexing either with a literal number or a variable, as the purpose of the list of the custom types feature is that you don't know the data length in advance. For more information about idiomatic uses of repeated structured data, see Loop Transform.

Interaction with Composition

Using composition alongside custom types has the following advantages/disadvantages:

- You might want to **leverage** types declared in an imported fragment
- There might be a type **name clash** between a host accelerator/fragment and an imported fragment, because the imported fragment author is unaware of how the fragment is to be used.

For more information about the syntax to customize the imported types names, see Use fragments in Application Accelerator.

Use fragments in Application Accelerator

This topic tells you how to use fragments in Application Accelerator.

Introduction

Despite their benefits, writing and maintaining, accelerators can become repetitive and verbose as new accelerators are added. Some create a project different from the next with similar aspects, requiring some form of copy-paste.

To alleviate this concern, Application Accelerators support a feature named Composition that allows the re-use of parts of an accelerator, called **fragments**.

Introducing fragments

A **fragment** looks exactly the same as an accelerator:

- It is made of a set of files.
- It contains an `accelerator.yaml` descriptor with options, declarations, and a root transform.

There are differences however. Namely:

- Fragments are declared to the system differently. They are filed as **fragment** custom resources.
- They deal with files differently. Because fragments deal with their own files and files from the accelerator using them, they use dedicated conflict resolution strategies (more on this later).

Fragments may be thought of as "functions" in programming languages. After being defined and referenced, they are "called" at various points in the main accelerator. The composition feature is designed with ease of use and "common use first" in mind, so these "functions" are typically called with as little noise as possible. You can also call them complex or different values.

Composition relies on two building blocks that play hand in hand:

- The `imports` section at the top of an accelerator manifest.
- The, `InvokeFragment` transform, to be used alongside any other transform.

| The `imports` section explained

To be usable in composition, a fragment MUST be *imported* into the dedicated section of an accelerator manifest:

```
accelerator:
  name: my-awesome-accelerator
  options:
    - name: flavor
      dataType: string
      defaultValue: Strawberry
  imports:
    - name: my-first-fragment
    - name: another-fragment
engine:
  ...
```

The effect of importing a fragment this way is twofold:

- It makes its files available to the engine (therefore importing a fragment is required).
- It exposes all of its options as options of the accelerator as if they were defined by the accelerator itself.

So in the earlier example, if the `my-first-fragment` fragment had the following `accelerator.yaml` file:

```
accelerator
  name: my-first-fragment
  options:
    - name: optionFromFragment
      dataType: boolean
      description: this option comes from the fragment
...
```

Then it is as if the `my-awesome-accelerator` accelerator defined it:

```
accelerator:
  name: my-awesome-accelerator
  options:
    - name: flavor
      dataType: string
      defaultValue: Strawberry
    - name: optionFromFragment
      dataType: boolean
      description: this option comes from the fragment
  imports:
    - name: my-first-fragment
    - name: another-fragment
engine:
  ...
```

All the metadata about options (type, default value, description, choices if applicable, *etc.*) come along when imported.

Because of this, users are prompted for a value for those options that come from fragments, as if they were options of the accelerator.

Using the `InvokeFragment` Transform

The second part at play in the composition is the `InvokeFragment` Transform.

As with any other transform, it may be used anywhere in the `engine` tree and receives files that are "visible" at that point. Those files, alongside those that make up the fragment, are made available to the fragment logic. If the fragment wants to choose between two versions of a file for a path, two new conflict resolution strategies are available: `FavorForeign` and `FavorOwn`.

The behavior of the `InvokeFragment` transform is very straightforward: after having validated options that the fragment expects (and maybe after having set default values for options that support one), it executes the whole transform of the fragment *as if it was written in place of* `InvokeFragment`.

See the `InvokeFragment` reference page for more explanations, examples, and configuration options. This topic now focuses on additional features of the `imports` section that are seldom used but still available to cover more complex use cases.

Back to the `imports` section

The complete definition of the `imports` section is as follows, with features in increasing order of "complexity":

```
accelerator:
  name: ...
  options:
    - name: ...
    ...
  imports:
    - name: some-fragment

    - name: another-fragment
      expose:
        - name: "*"
      exposeTypes:
        - name: "*"

    - name: yet-another-fragment
      expose:
        - name: someOption

        - name: someOtherOption
          as: aDifferentName
      exposeType:
        - name: SomeType

        - name: SomeOtherType
          as: ADifferentName
engine:
  ...
```

As shown earlier, the `imports` section calls a list of fragments to import. By default, all their options and types become options/type of the accelerator. Those options appear *after* the options defined by the accelerator, in the order the fragments are imported in.

It is even possible for a fragment to import another fragment, the semantics being the same as when an accelerator imports a fragment. This is a way to break apart a fragment even further if needed.

When importing a fragment, you can select which options of the fragment to make available as options of the accelerator. **This feature should only be used when a name clash arises in option names.**

The semantics of the `expose` block are as follows:

- For every `name`/`as` pair, don't use the original (`name`) of the option but instead, use the alias (`as`). Other metadata about the option is left unchanged.
- If the special `name: "*"` (which is NOT a legit option name usually) appears, all imported option names that are not remapped (the index at which the `*` appears in the YAML list is irrelevant) might be exposed with their original name.
- The default value for `expose` is `[{name: "*"}]`, that is, by default exposes all options with their original name.
- As soon as a single remap rule appears, the default is overridden. For example, to override some names AND expose the others unchanged, the `*` must be explicitly re-added.
- To explicitly un-expose ALL options from an imported fragment, an empty array may be used and overrides the default: `expose: []`.

Similarly, you can also select which custom types of the fragment to make available as types of the accelerator. **This feature should only be used when a name clash arises in types names.**

The semantics of the `exposeTypes` block are as follows:

- For every `name`/`as` pair, don't use the original (`name`) of the type but instead, use the alias (`as`). Options that used the original name are automatically "rewritten" to use the new name.
- If the special `name: "*"` appears, which is NOT usually a legit type name, all imported other type names that are not remapped are exposed with their original name. The index at which the `*` appears in the YAML list is irrelevant.
- The default value for `exposeTypes` is `[{name: "*"}]`, that is, by default exposes all types with their original name.
- As soon as a single remap rule appears, the default is overridden. For example, to override some names AND expose the others unchanged, the `*` must be explicitly re-added.
- To explicitly un-expose ALL types from an imported fragment, an empty array may be used, which overrides the default: `exposeTypes: []`.

Using `dependsOn` in the `imports` section

Lastly, as a convenience for the conditional use of fragments, you can make an exposed imported option *depend on* another option, as in the following example:

```
imports:
  - name: tap-initialize
    expose:
      - name: gitRepository
        as: gitRepository
        dependsOn:
          name: deploymentType
          value: workload
      - name: gitBranch
        as: gitBranch
        dependsOn:
          name: deploymentType
          value: workload
```

This plays well with the use of `condition`, as in the following example:

```
...
engine:
  ...
```

```
    type: InvokeFragment
    condition: "#deploymentType == 'workload'"
    reference: tap-initialize```
```

Discovering fragments using Tanzu CLI accelerator plug-in

Using the accelerator plug-in for Tanzu CLI, you can view a list of available fragments. Run:

```
tanzu accelerator fragment list
```

To see a list of available accelerator fragments. For example:

```
NAME                                READY   REPOSITORY
app-sso-client                      true    source-image: dev.registry.tanzu.vmware.c
om/app-accelerator/fragments/app-sso-client@sha256:ed5cf5544477d52d4c7baf3a76f71a11298
7856e77558697112e46947ada9241
java-version                        true    source-image: dev.registry.tanzu.vmware.c
om/app-accelerator/fragments/java-version@sha256:df99a5ace9513dc8d083fb5547e2a24770dfb
08ec111b6591e98497a329b969d
live-update                         true    source-image: dev.registry.tanzu.vmware.c
om/app-accelerator/fragments/live-update@sha256:c2eda015b0f811b0eeaa587b6f2c5410ac87d4
0701906a357cca0decb3f226a4
spring-boot-app-sso-auth-code-flow  true    source-image: dev.registry.tanzu.vmware.c
om/app-accelerator/fragments/spring-boot-app-sso-auth-code-flow@sha256:78604c96dd52697
ea0397d3933b42f5f5c3659cbcdc0616ff2f57be558650499
tap-initialize                      true    source-image: dev.registry.tanzu.vmware.c
om/app-accelerator/fragments/tap-initialize@sha256:7a3ae8f9277ef633200622dbf9d0f5a07de
a25351ac3dbf803ea2fa759e3baac
tap-workload                        true    source-image: dev.registry.tanzu.vmware.c
om/app-accelerator/fragments/tap-workload@sha256:8056ad9f05388883327d9bbe457e6a0122dc4
52709d179f683eceb6d848338d0
```

The `tanzu accelerator fragment get <fragment-name>` command shows all the options defined for the fragment and also any accelerators or other fragments that import this fragment. Run:

```
tanzu accelerator fragment get java-version
```

The following output is displayed:

```
name: java-version
namespace: accelerator-system
displayName: Select Java Version
ready: true
options:
- choices:
  - text: Java 8
    value: "1.8"
  - text: Java 11
    value: "11"
  - text: Java 17
    value: "17"
  defaultValue: "11"
  inputType: select
  label: Java version to use
  name: javaVersion
  required: true
artifact:
  message: Resolved revision: dev.registry.tanzu.vmware.com/app-accelerator/fragments/
java-version@sha256:df99a5ace9513dc8d083fb5547e2a24770dfb08ec111b6591e98497a329b969d
  ready: true
  url: http://source-controller-manager-artifact-service.source-system.svc.cluster.loc
al./imagerepository/accelerator-system/java-version-frag-97nwp/df99a5ace9513dc8d083fb5
```

```
547e2a24770dfb08ec111b6591e98497a329b969d.tar.gz
imports:
  None
importedBy:
  accelerator/java-rest-service
  accelerator/java-server-side-ui
  accelerator/spring-cloud-serverless
```

This shows the `options` and `importedBy` with a list of three accelerators that import this fragment.

Correspondingly, the `tanzu accelerator get <accelerator-name>` shows the fragments that an accelerator imports. Run:

```
tanzu accelerator get java-rest-service
```

The following output is shown:

```
name: java-rest-service
namespace: accelerator-system
description: A Spring Boot Restful web application including OpenAPI v3 document gener
ation and database persistence, based on a three-layer architecture.
displayName: Tanzu Java Restful Web App
iconUrl: ...abbreviated...
source:
  image: dev.registry.tanzu.vmware.com/app-accelerator/samples/java-rest-service@sha25
6:c098bb38b50d8bbead0a1b1e9be5118c4fdce3e260758533c38487b39ae0922d
  secret-ref: [{reg-creds}]
tags:
- java
- spring
- web
- jpa
- postgresql
- tanzu
ready: true
options:
- defaultValue: customer-profile
  inputType: text
  label: Module artifact name
  name: artifactId
  required: true
- defaultValue: com.example
  inputType: text
  label: Module group name
  name: groupId
  required: true
- defaultValue: com.example.customerprofile
  inputType: text
  label: Module root package
  name: packageName
  required: true
- defaultValue: customer-profile-database
  inputType: text
  label: Database Instance Name this Application will use (can be existing one in
    the cluster)
  name: databaseName
  required: true
- choices:
  - text: Maven (https://maven.apache.org/)
    value: maven
  - text: Gradle (https://gradle.org/)
    value: gradle
  defaultValue: maven
  inputType: select
  name: buildTool
```

```
  required: true
- choices:
  - text: Flyway (https://flywaydb.org/)
    value: flyway
  - text: Liquibase (https://docs.liquibase.com/)
    value: liquibase
  defaultValue: flyway
  inputType: select
  name: databaseMigrationTool
  required: true
- dataType: boolean
  defaultValue: false
  label: Expose OpenAPI endpoint?
  name: exposeOpenAPIEndpoint
- defaultValue: ""
  dependsOn:
    name: exposeOpenAPIEndpoint
  inputType: text
  label: System API Belongs To
  name: apiSystem
- defaultValue: ""
  dependsOn:
    name: exposeOpenAPIEndpoint
  inputType: text
  label: Owner of API
  name: apiOwner
- defaultValue: ""
  dependsOn:
    name: exposeOpenAPIEndpoint
  inputType: text
  label: API Description
  name: apiDescription
- choices:
  - text: Java 8
    value: "1.8"
  - text: Java 11
    value: "11"
  - text: Java 17
    value: "17"
  defaultValue: "11"
  inputType: select
  label: Java version to use
  name: javaVersion
  required: true
- dataType: boolean
  defaultValue: true
  dependsOn:
    name: buildTool
    value: maven
  inputType: checkbox
  label: Include TAP IDE Support for Java Workloads
  name: liveUpdateIDESupport
- defaultValue: dev.local
  dependsOn:
    name: liveUpdateIDESupport
  description: The prefix for the source image repository where source can be stored
    during development
  inputType: text
  label: The source image repository prefix to use when pushing the source
  name: sourceRepositoryPrefix
artifact:
  message: Resolved revision: dev.registry.tanzu.vmware.com/app-accelerator/samples/ja
va-rest-service@sha256:c098bb38b50d8bbead0a1b1e9be5118c4fdce3e260758533c38487b39ae0922
d
  ready: true
  url: http://source-controller-manager-artifact-service.source-system.svc.cluster.loc
```

```
al./imagerepository/accelerator-system/java-rest-service-acc-wcn8x/c098bb38b50d8bbead0
a1b1e9be5118c4fdce3e260758533c38487b39ae0922d.tar.gz
imports:
  java-version
  live-update
  tap-workload
```

The `imports` section at the end shows the fragments that this accelerator imports. The `options` section shows all options defined for this accelerator. This includes all options defined in the imported fragments, for example, the options for the Java version imported from the `java-version` fragment.

Transforms reference

This topic provides you with a list and brief description of the available Application Accelerator transforms in Tanzu Application Platform (commonly known as TAP).

Available transforms

You can use:

- Combo as a shortcut notation for many common operations. It combines the behaviors of many of the other transforms.
- Include to select files to operate on.
- Exclude to select files to operate on.
- Merge to work on subsets of inputs and to gather the results at the end.
- Chain to apply several transforms in sequence using function composition.
- Let to introduce new scoped variables to the model.
- InvokeFragment allows re-using various fragments across accelerators.
- ReplaceText to perform simple token replacement in text files.
- RewritePath to move files around using regular expression (regex) rules.
- OpenRewriteRecipe to apply Rewrite recipes, such as package rename.
- YTT to run the `ytt` tool on its input files and gather the result.
- UseEncoding to set the encoding to use when handling files as text.
- UniquePath to decide what to do when several files end up on the same path.
- Loop to iterate over a list and apply a transform for each element.
- Provenance to generate a manifest of the accelerator run.

See also

- Conflict Resolution

Transforms reference

This topic provides you with a list and brief description of the available Application Accelerator transforms in Tanzu Application Platform (commonly known as TAP).

Available transforms

You can use:

- Combo as a shortcut notation for many common operations. It combines the behaviors of many of the other transforms.
- Include to select files to operate on.
- Exclude to select files to operate on.
- Merge to work on subsets of inputs and to gather the results at the end.
- Chain to apply several transforms in sequence using function composition.
- Let to introduce new scoped variables to the model.
- InvokeFragment allows re-using various fragments across accelerators.
- ReplaceText to perform simple token replacement in text files.
- RewritePath to move files around using regular expression (regex) rules.
- OpenRewriteRecipe to apply Rewrite recipes, such as package rename.
- YTT to run the `ytt` tool on its input files and gather the result.
- UseEncoding to set the encoding to use when handling files as text.
- UniquePath to decide what to do when several files end up on the same path.
- Loop to iterate over a list and apply a transform for each element.
- Provenance to generate a manifest of the accelerator run.

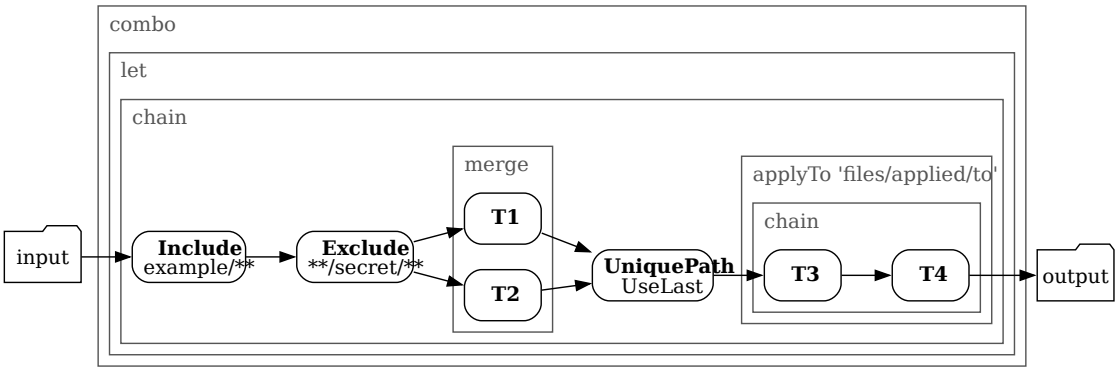
See also

- Conflict Resolution

Combo transform

This topic tells you about the Application Accelerator `Combo` transform in Tanzu Application Platform (commonly known as TAP).

The `Combo` transform combines the behaviors of Include, Exclude, Merge, Chain, UniquePath, and Let.



Syntax reference

Here is the full syntax of `Combo`:

```
type: Combo                    # This can be omitted, because Combo is the default trans
form type.
let:                          # See Let.
```

```
  - name: <string>
    expression: <SpEL expression>
  - name: <string>
    expression: <SpEL expression>
condition: <SpEL expression>
include: [<ant pattern>]    # See Include.
exclude: [<ant pattern>]    # See Exclude.
merge:                      # See Merge.
  - <m1-transform>
  - <m2-transform>
  - ...
chain:                    # See Chain.
  - <c1-transform>
  - <c2-transform>
  - ...
applyTo: [<ant pattern>]   # See Chain
onConflict: <conflict resolution> # See UniquePath.
```

Behavior

The `Combo` transform properties have default values, are optional, and you must use at least one property.

When you configure the `Combo` transform with all properties, it behaves as follows:

1. Applies the `include` as if it were the first element of a Chain. The default value is `['**']`; if not present, all files are retained.
2. Applies the `exclude` as if it were the second element of the chain. The default value is `[]`; if not present, no files are excluded. Only files that match the `include`, but are not excluded by the `exclude`, remain.
3. Feeds all those files as input to all transforms declared in the `merge` property, exactly as Merge does. The result of that `Merge`, which is the third transform in the big chain, is another set of files. If there are no elements in `merge`, the previous result is directly fed to the next step.
4. The result of the merge step is prone to generate duplicate entries for the same `path`. It's implicitly forwarded to a UniquePath check, configured with the `onConflict` strategy. The default policy is to retain files appearing later. The results of the transforms that appear later in the `merge` block "win" against results appearing earlier.
5. Passes that result as the input to the chain defined by the `chain` property. The combo chain is prolonged with the elements defined in `chain`. If there are no elements in `chain`, it's as if the previous result was used directly. If the `applyTo` property is set, it applies to the sub-chain (and that sub-chain only).
6. If the `let` property is defined in the `Combo`, the whole execution is wrapped inside a Let that exposes its derived symbols.

To recap in pseudo code, a giant `Combo` behaves like this:

```
Let(symbols, in:
    Chain(
        include,
        exclude,
        Chain(Merge(<m1-transform>, <m2-transform>, ...), UniquePath(onConflict)),
        Chain(<applyTo>, <c1-transform>, <c2-transform>, ...)
    )
)
```

You rarely use at any one time all the features that `Combo` offers. Yet `Combo` is a good way to author other common building blocks without having to write their `type: x` in full.

For example, this:

```
include: ['**/*.txt']
```

is a perfectly valid way to achieve the same effect as this:

```
type: Include
patterns: ['**/*.txt']
```

Similarly, this:

```
chain:
  - type: T1
    ...
  - type: T2
    ...
```

is often preferred over the more verbose:

```
type: Chain
transformations:
  - type: T1
    ...
  - type: T2
    ...
```

As with other transforms, the order of declaration of properties has no impact. For clarity, a convention that mimics the actual behavior is used, but the following applies **T1** and **T2** on all `.yaml` files even though it places the `include` section after the `merge` section.

```
merge:
  - type: T1
  - type: T2
include: ["*.yaml"]
```

In other words, `Combo` applies `include` filters before `merge` irrespective of the physical order of the keys in the YAML text. It's a good practice to place the `include` key before the `merge` key. This makes the accelerator definition more readable, but has no effect on its execution order.

Examples

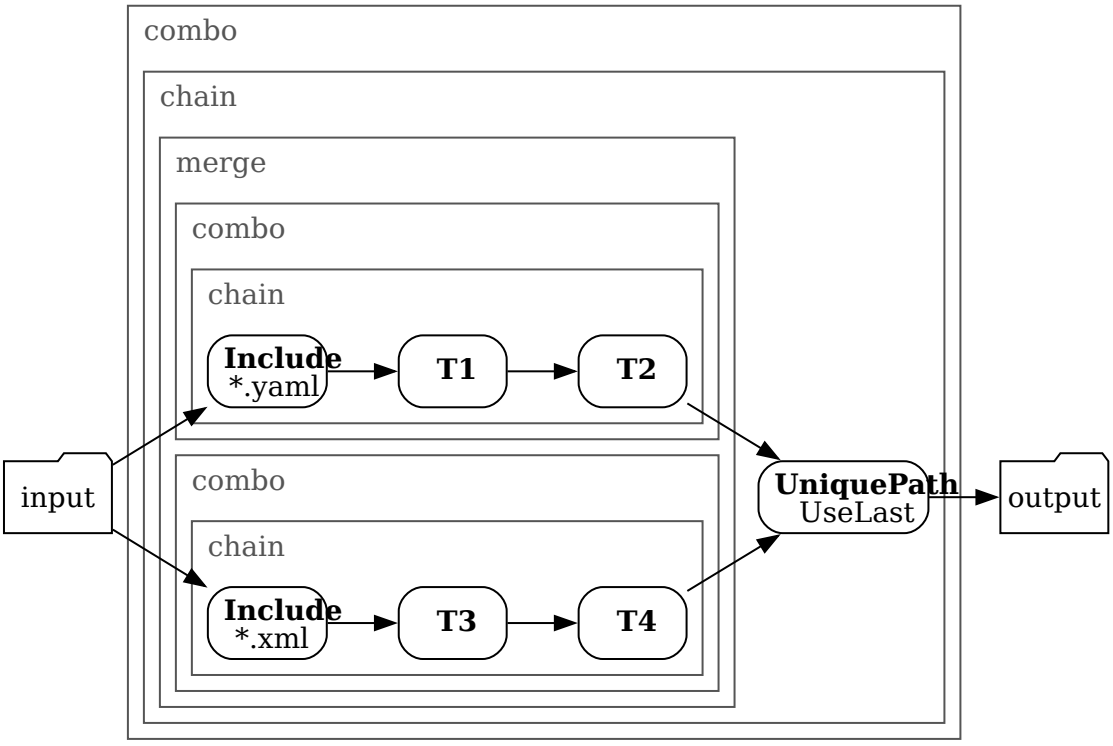
The following are typical use cases for `Combo`.

Example 1

To apply separate transformations to separate sets of files. For example, to all `.yaml` files and to all `.xml` files:

```
merge:                   # This uses the Merge syntax in a first Combo.
  - include: ['*.yaml']      # This actually nests a second Combo inside the first.
    chain:
      - type: T1
      - type: T2
  - include: ['*.xml']      # Here comes a third Combo, used as the 2nd child inside t
he first
    chain:
```

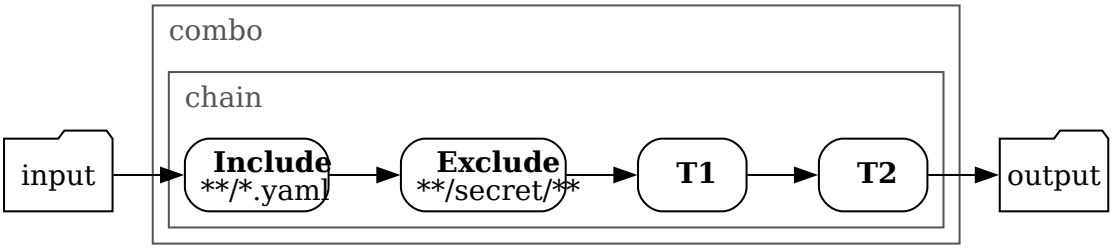
```
      - type: T3
      - type: T4
```



Example 2

To apply **T1** then **T2** on all `.yaml` files that are not in any `secret` directory:

```
include: ['**/*.yaml']
exclude: ['**/secret/**']
chain:
  - type: T1
    ..
  - type: T2
    ..
```



Include transform

This topic tells you about the Application Accelerator `Include` transform in Tanzu Application Platform (commonly known as TAP).

The `Include` transform retains files based on their `path`, letting in *only* those files whose path matches at least one of the configured `patterns`. The contents of files, and any of their other characteristics, are unaffected.

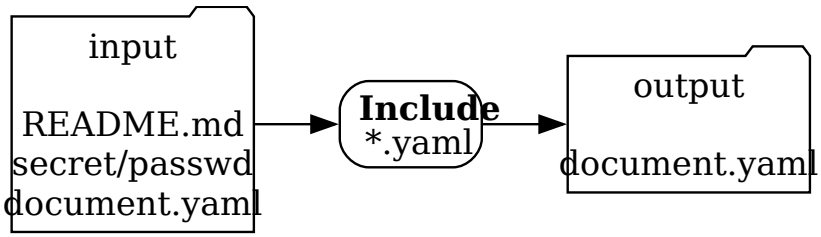
`Include` is a basic building block seldom used as is, which makes sense if composed inside a Chain or a Merge. It is often more convenient to leverage the shorthand notation offered by Combo.

Syntax reference

```
type: Include
patterns: [<ant pattern>]
condition: <SpEL expression>
```

Examples

```
type: Chain
transformations:
  - type: Include
    patterns: ["**/*.yaml"]
  - type: # At this point, only yaml files are affected
```



See also

- Exclude
- Combo

Exclude transform

This topic tells you about the Application Accelerator `Exclude` transform in Tanzu Application Platform (commonly known as TAP).

The `Exclude` transform retains files based on their `path`, allowing all files except ones with a path that matches at least one of the configured `patterns`. The contents of files, and any of their other characteristics are unaffected.

`Exclude` is a basic building block seldom used *as is*, which makes sense if composed inside a Chain or a Merge. It is often more convenient to leverage the shorthand notation offered by Combo.

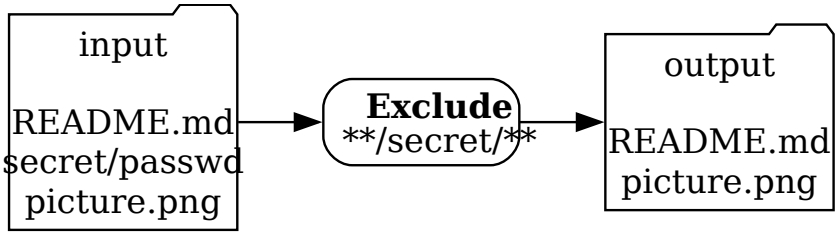
Syntax reference

```
type: Exclude
patterns: [<ant pattern>]
condition: <SpEL expression>
```

Examples

```
type: Chain
transformations:
  - type: Exclude
```

```
    patterns: ["**/secret/**"]
  - type: # At this point, no file matching **/secret/** is affected.
```



See also

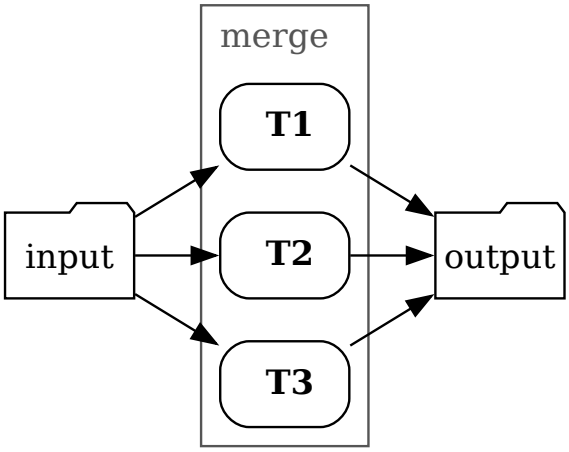
- Include
- Combo

Merge transform

This topic tells you about the Application Accelerator `Merge` transform in Tanzu Application Platform (commonly known as TAP).

The `Merge` transform feeds a copy of its input to several other transforms and merges the results together using set union.

A `Merge` of **T1**, **T2**, and **T3** applied to input **I** results in **T1(I)** ∪ **T2(I)** ∪ **T3(I)**.



An empty merge produces nothing (∅).

Syntax reference

```
type: Merge
sources:
  - <transform>
  - <transform>
  - <transform>
  - ...
condition: <SpEL expression>
```

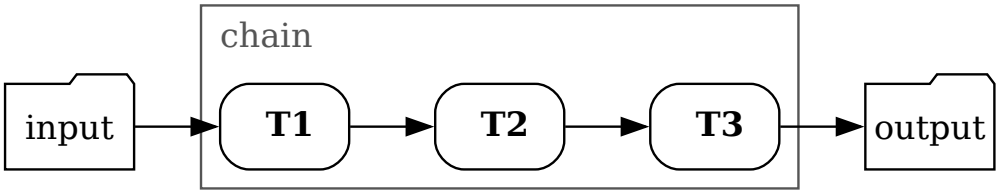
See also

- Combo is often used to express a `Merge` **and** other transformations in a shorthand syntax.

Chain transform

This topic tells you about the Application Accelerator `Chain` transform in Tanzu Application Platform (commonly known as TAP).

The `Chain` transform uses function composition to produce its final output.



Syntax reference

```
type: Chain
transformations:
  - <transform>
  - <transform>
  - <transform>
  - ...
applyTo: [<ant pattern>]
condition: <SpEL expression>
```

Behavior

A chain of **T1** then **T2** then **T3** first applies transform **T1**. It then applies **T2** to the output of **T1**, and finally applies **T3** to the output of that. In other words, **T3** to **T2** to **T1**.

An empty chain acts as function identity.

If the optional `applyTo` property is set, then the chained transformations are only applied to files with paths that match the `applyTo` patterns. Files with paths that don't match are left untouched and merged back with the other results to form the final result of the `Chain` transform.

Let transform

This topic tells you about the Application Accelerator `Let` transform in Tanzu Application Platform (commonly known as TAP).

The `Let` transform wraps another transform, creating a new scope that extends the existing scope.

SpEL expressions inside the `Let` can access variables from both the existing scope and the new scope.

Variables defined by the `Let` should not shadow existing variables. If they do, those existing variables won't be accessible.

Syntax reference

```
type: Let
symbols:
- name: <string>
  expression: <SpEL expression>
- ...
in: <transform> # <- new symbols are visible in here
```

Execution

The `Let` adds variables to the new scope by computation of SpEL expressions.

```
engine:
  let:
  - name: <string>
    expression: <SpEL expression>
  - ...
```

Both a `name` and an `expression` must define each symbol where:

- `name` must be a camelCase string name. If a let *symbol* happens to have the same name as a symbol already defined in the surrounding scope, then the local symbol shadows the symbol from the surrounding scope. This makes the variable from the surrounding scope inaccessible in the remainder of the `Let` but doesn't alter its original value.
- `expression` must be a valid SpEL expression expressed as a YAML string. Be careful when using the `#` symbol for variable evaluation, because this is the comment marker in YAML. So SpEL expressions in YAML must enclose strings in quotes or rely on block style. For more information about block style, see Block Style Productions.

Symbols defined in the `Let` are evaluated in the new scope in the order they are defined. This means that symbols lower in the list can make use of the variables defined higher in the list but not the other way around.

See also

- Combo provides a way to declare a `Let` scope and other transforms in a short syntax.

Loop transform

This topic tells you about the Application Accelerator `Loop` transform in Tanzu Application Platform (commonly known as TAP).

The `Loop` transform iterates over elements in a list and applies the provided transform for every element in that list.

When `doAsMerge` is used, a copy of the `Loop` transform's input is passed to each transform and the outputs from each transform are merged using a set union.

When `doAsChain` is used, each transform is executed sequentially, receiving the previous transform's output as its input. The first transform is to receive the `Loop` transform's input as its input.

Syntax reference

```
type: Loop
on: <SpEL expression>
var: <string>
index: <string>
doAsChain: <transform>
doAsMerge: <transform>
```

- `on` must be a SpEL expression that evaluates a list. This is the list of elements to be iterated over.
- `var` is the name of the variable to be assigned to the current element on each iteration. (optional)

- `index` is the variable's name to be assigned to the index of the current element on each iteration. (optional)
- `doAsMerge` is the transform to be executed for every element in the list, on a copy of the `Loop` transform's input.
- `doAsChain` is the transform to be executed for every element in the list, passing the output of the transform as input to the next transform.

Both `var` and `index` are optional.

Only one of the `doAsMerge` or `doAsChain` variables is to be used in a `Loop` transform.

Behavior

Consider the following when choosing `doAsMerge` or `doAsChain`:

`doAsMerge` executes the transform on the same input files for every iteration and merges the resulting outputs. It is best suited when a transform is executed multiple times on the same input and does not have conflicts.

`doAsChain` executes the transform on the initial input files once and then passes the resulting output to the second iteration and so on. It is best suited when a transform must detect any changes that occurred in the previous iteration.

Examples

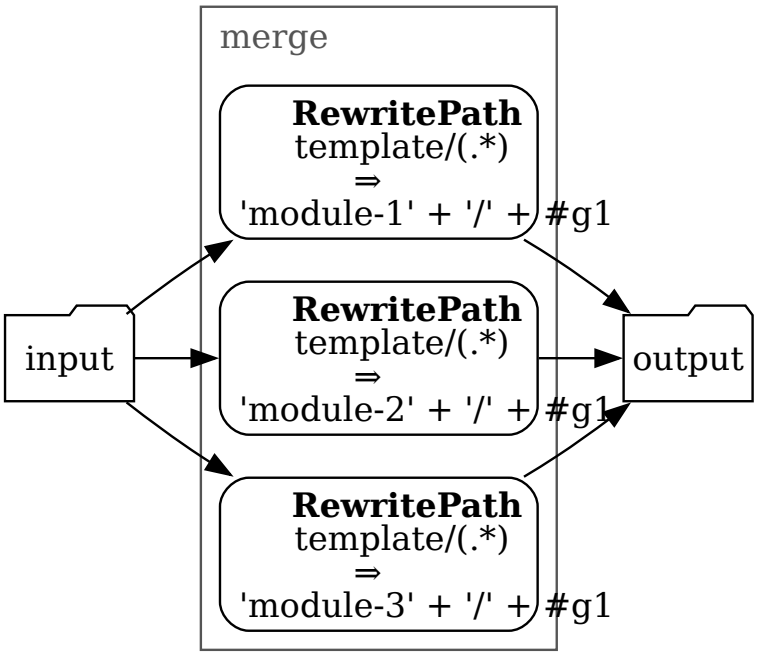
See the following examples using the `Loop` transform.

Example 1

Create a new directory for every module in `modules` (a list of strings) based on the contents of the "template" directory.

```
type: Loop
on: "#modules"
var: m
doAsMerge:
  type: RewritePath
  regex: "template/(.*)"
  rewriteTo: "#m + '/' + #g1"
```

The following diagram shows how this example behaves:

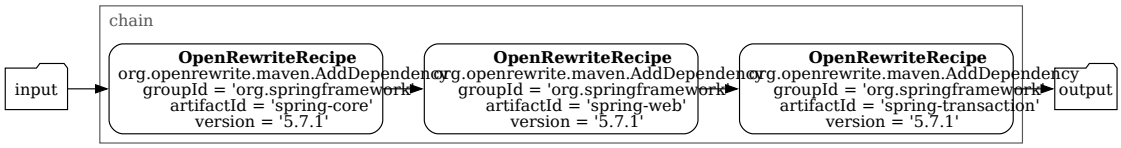


Example 2

Add every artifactId in `artifacts` (a list of strings) as a Spring dependency.

```
type: Loop
on: "#artifacts"
var: a
doAsChain:
  type: OpenRewriteRecipe
  recipe: org.openrewrite.maven.AddDependency
  options:
    groupId: "'org.springframework'"
    artifactId: "#a"
    version: "'5.7.1'"
```

The following diagram shows how this example behaves:



Example 3

You can use `Loop` in combination with custom types, for example:

```
accelerator:
  types:
    - name: MavenPlugin
      struct:
        - name: groupId
        - name: artifactId
        - name: version
  options:
    - name: pluginsToAdd
      dataType: [MavenPlugin] # End users will be able to enter a collection of GAV tu
ples
engine:
```

```
  include: [pom.xml]
  chain:
    - type: Loop
      on: pluginsToAdd # Iterate on the pluginsToAdd collection
      var: p           # The variable "p" will contain each tuple in turn
      doAsChain:       # Will apply the second execution to the result of the first, a
nd so on...
        type: OpenRewriteRecipe
        recipe: org.openrewrite.maven.AddPlugin
        options:
          groupId:    "#p['groupId']"
          artifactId: "#p['artifactId']"
          version:    "#p['version']"
```

For more information, see Using Custom Types.

InvokeFragment transform

This topic tells you about the Application Accelerator `InvokeFragment` transform in Tanzu Application Platform (commonly known as TAP).

The `InvokeFragment` performs transformations defined in an imported Fragment, allowing re-use across accelerators.

Syntax reference

```
type: InvokeFragment
reference: <imported-fragment>
let:  # See Let
  - name: <string>
    expression: <SpEL expression>
  ...
anchor: [<file path>]
```

Behavior

Assuming some fragment `my-fragment` has been imported in the accelerator (thus exposing the options it defines as options of the current accelerator), the following construct invokes `my-fragment`:

```
type: InvokeFragment
reference: my-fragment
```

This passes all input files (depending where this invocation sits in the “tree”) to the invoked fragment, which can then manipulate them alongside its own files. The result of the invocation becomes the result of this transform.

Variables

At the point of invocation, all currently defined variables are made visible to the invoked fragment. Therefore, if it was `import`-ed in the most straightforward manner, a fragment defining an option `myOption` is defining an option named `myOption` at the accelerator level, and the value provided by the user is visible at the time of invocation.

To override a value, or if an imported option has been exposed under a different name, or not at all, you can use a `let` construct when using `InvokeFragment`. This behaves as the Let transform: for the duration of the fragment invocation, the variables defined by `let` now have their newly defined values. Outside the scope of the invocation, the regular model applies.

Files

The set of files coming from the invoking accelerator and made visible to the fragment is the set of files that “reach” the point of invocation. For example, in the following case:

```
include: ["somedir/**"]
chain:
  - type: InvokeFragment
    reference: my-fragment
```

All files that the fragment invocation “sees” are files in the `somedir/` subdirectory. If the `my-fragment` has not been written accordingly, this can be problematic. Chances are that this re-usable fragment expects files to be present at the root of the project tree and work on them.

To better cope with this typical situation, the `InvokeFragment` transform exposes the optional `anchor` configuration property. Continuing with the earlier example, by using `anchor: somedir`, then all files coming from the current accelerator are exposed as if their `path` had the `somedir/` prefix removed. When it comes to gathering the result of the invocation though, all resulting files are re-introduced with a prefix prepended to their `path` (this applies to **all** files produced by the fragment, not just the ones originating from the accelerator).

The value of the `anchor` property must not start nor end with a slash (`/`) character.

Examples

The following is a full-featured example showcasing the interaction between the `imports` section and `InvokeFragment`:

```
accelerator:
  name: my-accelerator
  options:
    - name: someOption
      dataType: number
  imports:
    - name: my-fragment

engine:
  merge:
    - include: ["..."]
    - ...
    - chain:
        - include: ["**/pom.xml"]
        - type: InvokeFragment
          reference: my-fragment
```

Assuming `my-fragment` is defined as follows:

```
accelerator:
  name: my-fragment
  options:
    - name: indentationLevel
      dataType: number
      defaultValue: 2
transform:
  chain:
    - include: ["**/*.xml"]
    - type: SomeTransform
      ...
```

Then users will be presented with two options: `someOption` and `indentationLevel`, as if `indentationLevel` was defined in the host accelerator.

Moreover, the behavior of the calling accelerator is exactly as if the body of the fragment transform was inserted in-place of `InvokeFragment`:

```
accelerator:
  name: my-accelerator
  options:
    - name: someOption
      dataType: number
    - name: indentationLevel
      dataType: number
      defaultValue: 2

engine:
  merge:
    - include: ["..."]
    - ...
    - chain:
        - include: ["**/pom.xml"]
        - chain:
          - include: ["**/*.xml"]
          - type: SomeTransform
            ...
```

Now you can imagine some scenarios to better clarify all configuration properties.

If, for some reason, you don't want to use the value entered in the `indentationLevel` option for the fragment, but twice the value provided for `someOption`. The `InvokeFragment` block can be rewritten as follows:

```
    type: InvokeFragment
    reference: my-fragment
    let:
      - name: indentationLevel
        value: '2 * #someOption'
```

Finally, if the invocation in the accelerator looks like this:

```
engine:
  merge:
    - include: ["..."]
    - ...
    - chain:
        - include: ["**/README.md"]
        - type: InvokeFragment
          reference: my-fragment
```

Then there is zero visible effect, because this is forwarding only `README.md` files to the fragment and the fragment is itself using a filter on `*.xml` files.

See also

- Let
- RewritePath

ReplaceText transform

This topic tells you about the Application Accelerator `ReplaceText` transform in Tanzu Application Platform (commonly known as TAP).

The `ReplaceText` transform allows replacing one or several text tokens in files as they are being copied to their destination. The replacement values are the result of dynamic evaluation of SpEL expressions.

This transform is text-oriented and requires knowledge of how to interpret the stream of bytes that make up the file contents into text. All files are assumed to use `UTF-8` encoding by default, but you can use the UseEncoding transform upfront to specify a different charset to use on some files.

You can use `ReplaceText` transform in one of two ways:

- To replace several literal text tokens.
- To define the replacement behavior using a single regular expression, in which case the replacement SpEL expression can leverage the regex capturing group syntax.

Syntax reference

Syntax reference for replacing several literal text tokens:

```
type: ReplaceText
substitutions:
  - text: STRING
    with: SPEL-EXPRESSION
  - text: STRING
    with: SPEL-EXPRESSION
  - ..
condition: SPEL-EXPRESSION
```

Syntax reference for defining the replacement behavior using a *single* regular expression:

Regex is used to match the entire document. To match on a per line basis, enable multiline mode by including `(?m)` in the regex.

```
type: ReplaceText
regex:
  pattern: REGULAR-EXPRESSION
  with: SPEL-EXPRESSION
condition: SPEL-EXPRESSION
```

In both cases, the SpEL expression can use the special `#files` helper object. This enables the replacement string to consist of the contents of an accelerator file. See the following example.

Another set of helper objects are functions of the form `xxx2Yyyy()` where `xxx` and `yyy` can take the value `camel`, `kebab`, `pascal`, or `snake`. For example, `camel2Snake()` enables changing from camelCase to snake_case.

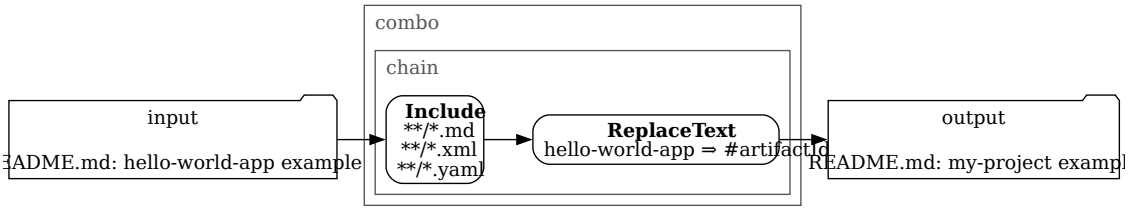
Examples

See the following examples using The `ReplaceText` transform.

Example 1

Replacing the hardcoded string `"hello-world-app"` with the value of variable `#artifactId` in all `.md`, `.xml`, and `.yaml` files.

```
include: ['**/*.md', '**/*.xml', '**/*.yaml']
chain:
  - type: ReplaceText
    substitutions:
      - text: "hello-world-app"
        with: "#artifactId"
```



Example 2

Replacing the hardcoded string `"hello-world-app"` with the value of variable `#artifactId` in the `README-fr.md` and `README-de.md` files, which are encoded using the `ISO-8859-1` charset:

```
include: ['README-fr.md', 'README-de.md']
chain:
  - type: UseEncoding
    encoding: 'ISO-8859-1'
  - type: ReplaceText
    substitutions:
      - text: "hello-world-app"
        with: "#artifactId"
```

Example 3

Similar to the preceding example, but making sure the value appears as kebab case, while the entered `#artifactId` is using camel case:

```
include: ['**/*.md', '**/*.xml', '**/*.yaml']
chain:
  - type: ReplaceText
    substitutions:
      - text: "hello-world-app"
        with: "#camel2Kebab(#artifactId)"
```

Example 4

Replacing the hardcoded string `"REPLACE-ME"` with the contents of file named after the value of the `#platform` option in `README.md`:

```
include: ['README.md']
chain:
  - type: ReplaceText
    substitutions:
      - text: "REPLACE-ME"
        with: "#files.contentsOf('snippets/install-' + #platform + '.md')"
```

Example 5

Replacing all occurrences of `apple` or `orange`, singular or plural, with the value of the accelerator option `#vegetable`, for example `'banana'`, keeping the trailing `'s'` when there was one.

```
include: ['README.md']
chain:
  - type: ReplaceText
    regex:
      pattern: "(apple|orange)(s)?"
      # This constructs a SpEL string containing eg 'banana$2' where $2
      # refers to the second capturing group (the optional 's')
      with: "#vegetable + '$2'"
```

See also

- UseEncoding

RewritePath transform

This topic tells you about the Application Accelerator `RewritePath` transform in Tanzu Application Platform (commonly known as TAP).

The `RewritePath` transform allows you to change the name and path of files without affecting their content.

Syntax reference

```
type: RewritePath
regex: <string>
rewriteTo: <SpEL expression>
matchOrFail: <boolean>
```

For each input file, `RewritePath` attempts to match its `path` by using the regular expression (regex) defined by the `regex` property. If the regex matches, `RewritePath` changes the `path` of the file to the evaluation result of `rewriteTo`.

`rewriteTo` is an expression that has access to the overall engine model and to variables defined by capturing groups of the regular expression. Both *named capturing groups* `(?<example>[a-z]*)` and regular *index-based* capturing groups are supported. `g0` contains the whole match, `g1` contains the first capturing group, and so on.

If the regex doesn't match, the behavior depends on the `matchOrFail` property:

- If set to `false`, which is the default, the file is left untouched.
- If set to `true`, an error occurs. This prevents misconfiguration if you expect all files coming in to match the regex. For more information about typical interactions between `RewritePath` and `Chain + Include`, see the following section, Interaction with Chain and Include.

The default value for `regex` is the following regular expression, which provides convenient access to some named capturing groups:

```
^(?<folder>.*/)?(?<filename>([^/]+?|)(?=(?<ext>\.[^/.]*)?)$)
```

Using `some/deep/nested/file.xml` as an example, the preceding regular expression captures:

- **folder:** The full folder path the file is in. In this example, `some/deep/nested/`.
- **filename:** The full name of the file, including extension *if present*. In this example, `file.xml`.
- **ext:** The last dot and extension in the filename, *if present*. In this example, `.xml`.

The default value for `rewriteTo` is the expression `#folder + #filename`, which doesn't rewrite paths.

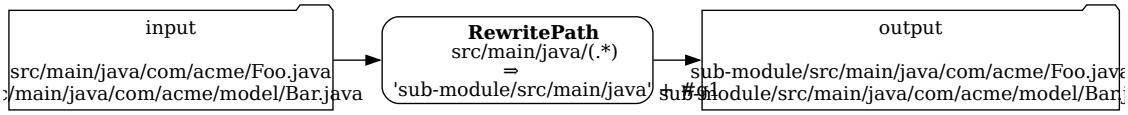
Examples

See the following examples using the `RewritePath` transform.

Example 1

The following moves all files from `src/main/java` to `sub-module/src/main/java`:

```
type: RewritePath
regex: src/main/java/(.*)
rewriteTo: "'sub-module/src/main/java' + #g1"   # 'sub-module/' + #g0 works too
```



Example 2

The following flattens all files found inside the `sub-path` directory and its subdirectories, and puts them into the `flattened` folder:

```
type: RewritePath
regex: sub-path/(.*/)*(?<filename>[^/]+)
rewriteTo: "'flattened' + #filename"   # 'flattened' + #g2 would work too
```

Example 3

The following turns all paths into lowercase:

```
type: RewritePath
rewriteTo: "#g0.toLowerCase()"
```

Interaction with Chain and Include

It's common to define pipelines that perform a `Chain` of transformations on a subset of files, typically selected by `Include/Exclude`:

```
- include: ["**/*.java"]
- chain:
    - # do something here
    - # and then here
```

If one of the transformations in the chain is a `RewritePath` operation, chances are you want the rewrite to apply to *all* files matched by the `Include`. For those typical configurations, you can set the `matchOrFail` guard to `true` to ensure the `regex` you provide indeed matches all files coming in.

See also

- Use UniquePath to ensure rewritten paths don't clash with other files, or to decide which path to select if they do clash.

OpenRewriteRecipe transform

This topic tells you about the Application Accelerator `OpenRewriteRecipe` transform in Tanzu Application Platform (commonly known as TAP).

The `OpenRewriteRecipe` transform allows you to apply any Open Rewrite **Recipe** to a set of files and gather the results.

The engine leverages v8.7.4 of Open Rewrite and parses Java files using the grammar for Java 17.

The following Open Rewrite Recipes are supported:

- Java recipes (v8.7.4)
- Maven recipes (v8.7.4)

- XML recipes (v8.7.4)
- YAML recipes (v8.7.4)
- JSON recipes (v8.7.4)
- Properties recipes (v8.7.4)
- Kubernetes recipes (v1.18.0)
- Python recipes (v1.1.2)
- Terraform recipes (v2.0.5)

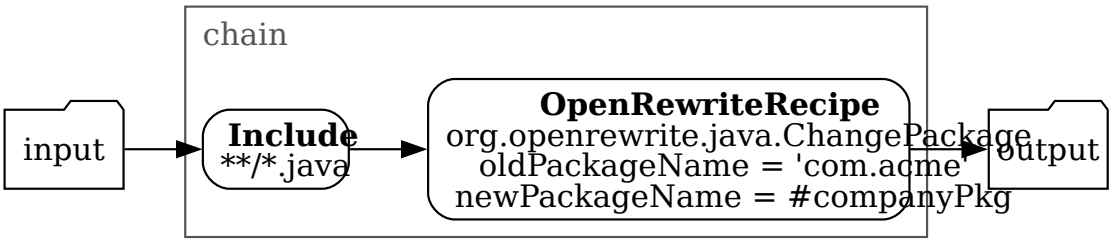
Syntax reference

```
type: OpenRewriteRecipe
recipe: <string>                  # Full qualified classname of the recipe
options:
  <string>: <SpEL expression>      # Keys and values depend on the class of the recipe
  <string>: <SpEL expression>      # Refer to the documentation of said recipe
  ...
```

Example

The following example applies the ChangePackage Recipe to a set of Java files in the `com.acme` package and moves them to the value of `#companyPkg`. This is more powerful than using RewritePath and ReplaceText, as it reads the syntax of files and correctly deals with imports, fully compared to non-fully qualified names, and so on.

```
chain:
  - include: ["**/*.java"]
  - type: OpenRewriteRecipe
    recipe: org.openrewrite.java.ChangePackage
    options:
      oldPackageName: "'com.acme'"
      newPackageName: "#companyPkg"
```



YTT transform

This topic tells you about the Application Accelerator `YTT` transform in Tanzu Application Platform (commonly known as TAP).

The `YTT` transform starts the YTT template engine as an external process.

Syntax reference

```
type: YTT
extraArgs: # optional
  - <SPEL-EXPRESSION-1>
```

```
  - <SPEL-EXPRESSION-2>
  - ...
```

The `YTT` transform's YAML notation does not require any parameters. When invoked without parameters, which is the typical use case, the YTT transform's input is determined entirely by two things only:

1. The input files fed into the transform.
2. The current values for options and derived symbols.

Execution

YTT is invoked as an external process with the following command line:

```
ytt -f <input-folder> \
    --data-values-file <symbols.json> \
    --output-files <output-folder> \
    <extra-args>
```

The `<input-folder>` is a temporary directory into which the input files are "materialized." That is, the set of files passed to the YTT transform as input is written out into this directory to allow the YTT process to read them.

The `<symbols.json>` file is a temporary JSON file, which the current option values and derived symbols are materialized in the form of a JSON map. This allows YTT templates in the `<input-folder>` to make use of these symbols during processing.

The `<output-folder>` is a fresh temporary directory that is empty at the time of invocation. In a typical scenario, upon completion, the output directory contains files generated by YTT.

The `<extra-args>` are additional command line arguments obtained by evaluating the SPEL expressions from the `extraArgs` attribute.

When the `ytt` process completes with a 0 exit code, this is considered a successful execution and the contents of the output directory is taken to be the result of the YTT transform.

When the `ytt` process completes with a non 0 exit code, the execution of the `YTT` transform is considered to have failed and an exception is raised.

Examples

See the following examples using the `YTT` transform.

Basic invocation

When you want to execute `ytt` on the contents of the entire accelerator repository, use the YTT transform as your only transform in the engine declaration.

```
accelerator:
  ...
engine:
  type: YTT
```

To do anything beyond calling YTT, compose YTT into your accelerator flow using merge or chain combinators. This is exactly the same as composing any other type of transform.

For example, when you want to define some derived symbols as well as merge the results from YTT with results from other parts of your accelerator transform, you can reference this example:

```
engine:
  let: # Define derived symbols visible to all transforms (including YTT)
  - name: theAnswer
    expression: "41 + 1"
  merge:
  - include: ["deploy/**.yml"] # select some yaml files to process with YTT
    chain: # Chain selected yaml files to YTT
    - type: YTT
  - ... include/generate other stuff to be merged alongside yaml generated by YTT...
```

The preceding example uses a combination of Chain and Merge. You can use either `Merge` or `Chain` or both to compose YTT into your accelerator flow. Which one you choose depends on how you want to use YTT as part of your larger accelerator.

Using `extraArgs`

The `extraArgs` passes additional command line arguments to YTT. This adds file marks. See File Marks in the Carvel documentation.

For example, the following runs YTT and renames the `foo/demo.yml` file in its output to `bar/demo.yml`.

```
engine:
  type: YTT
  extraArgs: ["'--file-mark'",  "'foo/demo.yml:path=bar/demo.yml'"]
```

The `extraArgs` attribute expects SPEL expressions. Take care to use proper escaping of literal strings using double and single quotes (that is, `“‘LITERAL-STRING’”).

UseEncoding transform

This topic tells you about the Application Accelerator `UseEncoding` transform in Tanzu Application Platform (commonly known as TAP).

When considering files in textual form, for example, when doing text replacement with the ReplaceText transform, the engine must decide which encoding to use.

By default, `UTF-8` is assumed. If any files must be handled differently, use the `UseEncoding` transform to annotate them with an explicit encoding.

`UseEncoding` returns an error if you apply encoding to files that have already been explicitly configured with a particular encoding.

Syntax reference

```
type: UseEncoding
encoding: <encoding>    # As recognized by the java java.nio.charset.Charset class
condition: <SpEL expression>
```

Supported encoding names include, for example, `UTF-8`, `US-ASCII`, and `ISO-8859-1`.

Example use

`UseEncoding` is typically used as an upfront transform to, for example, ReplaceText in a chain:

```
type: Chain   # Or using "Combo"
transformations:
  - type: UseEncoding
    encoding: ISO-8859-1
```

```
  - type: ReplaceText
    substitutions:
      - text: "hello"
        with: "#howToSayHello"
```

See also

- ReplaceText

UniquePath transform

This topic tells you about the Application Accelerator `UniquePath` transform in Tanzu Application Platform (commonly known as TAP).

You can use the `UniquePath` transform to ensure there are no `path` conflicts between files transformed. You can often use this at the tail of a Chain.

Syntax reference

```
type: UniquePath
strategy: <conflict resolution>
condition: <SpEL expression>
```

Examples

The following example concatenates the file that was originally named `DEPLOYMENT.md` to the file `README.md`:

```
chain:
  - merge:
      - include: ['README.md']
      - include: ['DEPLOYMENT.md']
        chain:
          - type: RewritePath
            rewriteTo: "'README.md'"
  - type: UniquePath
    strategy: Append
```

See also

- `UniquePath` uses a Conflict Resolution strategy to decide what to do when several input files use the same `path`.
- Combo implicitly embeds a `UniquePath` after the Merge defined by its `merge` property.

Conflict resolution

This topic tells you how to resolve conflicts that Application Accelerator transforms in Tanzu Application Platform (commonly known as TAP) might produce.

For example, if you're using Merge (or Combo's `merge` syntax) or RewritePath, a transform can produce several files at the same `path`. The engine then must take an action: Should it keep the last file? Report an error? Concatenate the files together?

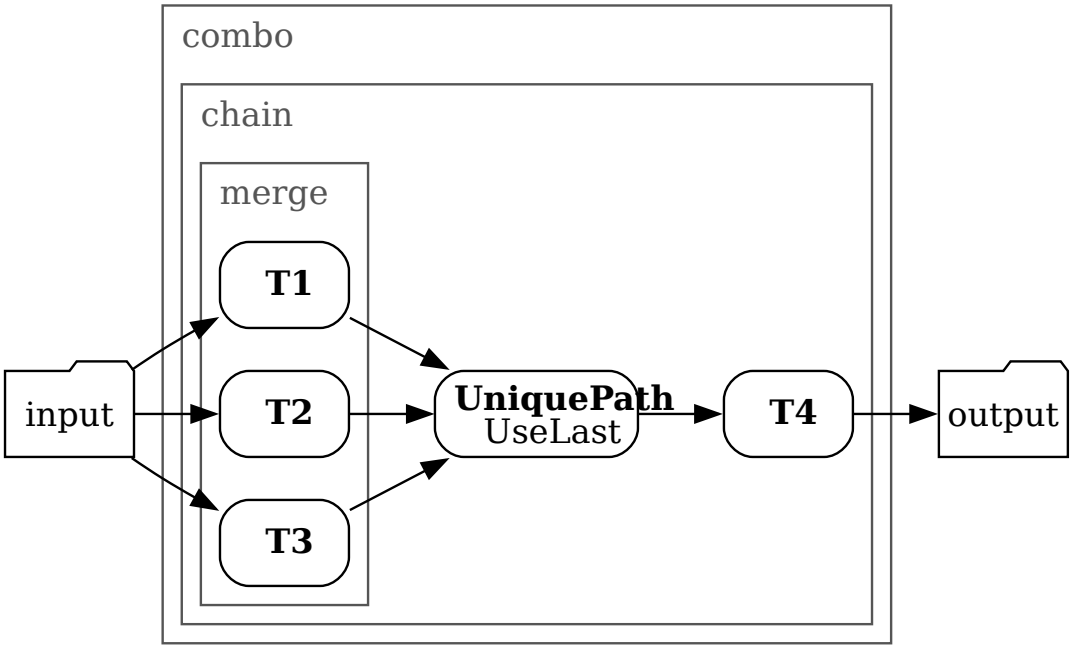
Syntax reference

Conflicts can arise for a number of reasons. You can avoid or resolve them by configuring transforms with a conflict resolution. For example:

- Combo uses UseLast by default, but you can configure it to do otherwise.
- You can explicitly end a transform Chain with a UniquePath, which by default uses Fail. This is customizable.

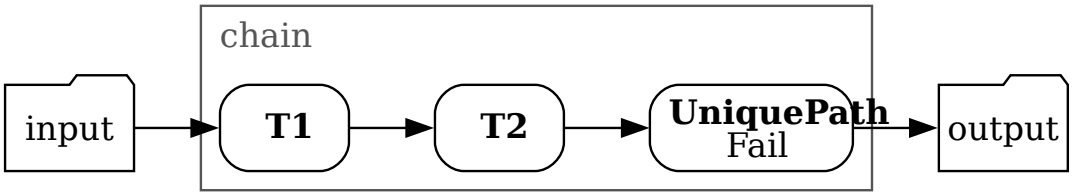
Combo

```
type: Combo      # often omitted
merge:
  - <transform>
  - <transform>
  - <transform>
chain:
  - <transform>
  - ...
onConflict: <conflict resolution>  # defaults to 'UseLast'
```



Chain

```
type: Chain      # or implicitly using Combo
transformations:
  - <transform>
  - <transform>
  - type: UniquePath
    strategy: <conflict resolution>  # defaults to 'Fail'
```



Available strategies

The following values and behaviors are available:

- `Fail`: Stop processing on the first file that exhibits `path` conflicts.
- `UseFirst`: For each conflicting file, the file produced first (typically by a transform appearing earlier in the YAML definition) is retained.
- `UseLast`: For each conflicting file, the file produced last (typically by a transform appearing later in the YAML definition) is retained.
- `Append`: The conflicting versions of files are concatenated (as if using `cat file1 file2 ...`), with files produced first appearing first.
- `FavorOwn`: Only makes sense in the context of composition. Selects the version of the file that comes from the current executing fragment if possible, falls back to the caller version otherwise.
- `FavorForeign`: Only makes sense in the context of composition. Selects the version of the file that was provided by the caller if present, falls back to the file originating from this fragment's fileset otherwise.
- `NWayDiff`: Try to merge the conflicting resources by applying patches computed against a common ancestor. The resulting resource has the attributes of the first conflicting resource.

See also

- Combo
- UniquePath

Provenance transform

This topic tells you about the Application Accelerator `Provenance` transform in Tanzu Application Platform (commonly known as TAP).

The `Provenance` transform is a special transform used to generate a file that provides details of the accelerator engine transform.

Syntax reference

```
type: Provenance
condition: <SpEL expression>
```

The `Provenance` transform is added as a child to the top-most transform, which is usually a `Merge` or a `Chain`, using a `Combo`.

Behavior

The `Provenance` transform ignores its input and outputs a single resource named `accelerator-info.yaml`. For example:

```
id: <unique GUID of invocation>
timestamp: <timestamp in RFC3339 format>
username: <captured username of user triggering the run>
source: <client environment from which accelerator was run>
accelerator:
  name: <name of registered accelerator>
  git:
```

```
    url: <git repository location>
    ref:
      branch: <branch name> or
      tag: <tag name> or
      commit: <specific requested commit>
    subPath: <optional subpath inside the repo>
    commit: <actual SHA the branch or tag pointed to>
fragments:
  - name: <name of registered fragment 1>
    git:
      url: <git repository location>
      ref:
        branch: <branch name> or
        tag: <tag name> or
        commit: <specific requested commit>
      subPath: <optional subpath inside the repo>
      commit: <actual SHA the branch or tag pointed to>
  - name: <name of registered fragment 2>
    git:
      url: <git repository location>
      ref:
        branch: <branch name> or
        tag: <tag name> or
        commit: <specific requested commit>
      subPath: <optional subpath inside the repo>
      commit: <actual SHA the branch or tag pointed to>
  - ...
options:
  - name: <option name>
    value: <option value>
  - name: <option name>
    value: <option value>
```

Note

Depending on the invocation scenario, some pieces of data might not be present.

Use SpEL with Application Accelerator

This topic tells you about some common Spring Expression Language (SpEL) use cases for the `accelerator.yaml` file in Application Accelerator.

For more information, see Spring Expression Language documentation.

Variables

You can reference all the values added as options in the `accelerator` section from the YAML file as variables in the `engine` section. You can access the value using the syntax `#<option name>`:

```
options:
  - name: foo
    dataType: string
    inputType: text
...
engine:
  - include: ["some/file.txt"]
    chain:
    - type: ReplaceText
      substitutions:
```

```
      - text: bar
        with: "#foo"
```

This sample replaces every occurrence of the text `bar` in the file `some/file.txt` with the contents of the `foo` option.

Implicit variables

Some variables are made available to the model by the engine, including:

- `artifactId` is a built-in value derived from the `projectName` passed in from the UI with spaces replaced by "_". If that value is empty, it is set to `app`.
- `files` is a helper object that currently exposes the `contentsOf(<path>)` method. For more information, see ReplaceText.
- `camel2Kebab` and other variations of the form `xxx2Yyyy` is a series of helper functions for dealing with changing case of words. For more information, see ReplaceText.

Conditionals

You can use Boolean options for conditionals in your transformations.

```
options:
  - name: numbers
    inputType: select
    choices:
    - text: First Option
      value: first
    - text: Seconf Option
      value: second
    defaultValue: first
...
engine:
  - include: ["some/file.txt"]
    condition: "#numbers == 'first'"
    chain:
    - type: ReplaceText
      substitutions:
      - text: bar
        with: "#foo"
```

This replaces the text only if the selected option is the first one.

Rewrite path concatenation

```
options:
  - name: renameTo
    dataType: string
    inputType: text
...
engine:
  - include: ["some/file.txt"]
    chain:
    - type: RewritePath
      rewriteTo: "'somewhere/' + #renameTo + '.txt'"
```

Regular expressions

Regular expressions allow you to use patterns as a matcher for strings. Here is a small example of what you can do with them:

```
options:
  - name: foo
    dataType: string
    inputType: text
    defaultValue: abcZ123
...
engine:
  - include: ["some/file.txt"]
    condition: "#foo.matches('[a-z]+Z\d+')"
    chain:
    - type: ReplaceText
      substitutions:
      - text: bar
        with: "#foo"
```

This example uses RegEx to match a string of letters that ends with a capital Z and any number of digits. If this condition is fulfilled, the text is replaced in the file, `file.txt`.

Dealing with string arrays

Options with a `dataType` of `[string]` come out as an array of strings.

To use them and for example format the result as a bulleted list, you can use the Java static String.join() method. For example:

```
accelerator:
  options:
    - name: meals
      dataType: [string]
      inputType: checkbox
      choices:
        - value: fish
        - value: chips
        - value: BLT
...
engine:
  type: ReplaceText
  substitutions:
  - text: recipe
    with: "' * ' + T(java.lang.String).join('\n * ', #meals)"
```

Accelerator custom resource definition

This topic tells you about the Application Accelerator custom resource definition.

Overview

The `Accelerator` custom resource definition (CRD) defines any accelerator resources to be made available to the Application Accelerator for VMware Tanzu system. It is a namespaced CRD, meaning that any resources created belong to a namespace. For the resource to be available to the Application Accelerator system, it must be created in the namespace that the Application Accelerator UI server is configured to watch.

The `Fragment` custom resource definition (CRD) defines any accelerator fragment resources to be made available to the Application Accelerator for VMware Tanzu system. It is a namespaced CRD, meaning that any resources created belong to a namespace. For the resource to be available to

the Application Accelerator system, it must be created in the namespace that the Application Accelerator UI server is configured to watch.

API definitions

The `Accelerator` CRD is defined with the following properties:

Property	Value
Name	Accelerator
Group	accelerator.apps.tanzu.vmware.com
Version	v1alpha1
ShortName	acc

Accelerator CRD Spec

The `Accelerator` CRD *spec* defined in the `AcceleratorSpec` type has the following fields:

Field	Description	Required/Optional
displayName	A short descriptive name used for an Accelerator.	Optional (*)
description	A longer description of an Accelerator.	Optional (*)
iconUrl	A URL for an image to represent the Accelerator in a UI.	Optional (*)
tags	An array of strings defining attributes of the Accelerator that can be used in a search.	Optional (*)
git	Defines the accelerator source Git repository.	Optional (***)
git.url	The repository URL, can be a HTTP/S or SSH address.	Optional (***)
git.gitImplementation	Determines which git client library to use. The default setting is to go-git. Valid values are (‘go-git’).	Optional (**)
git.ignore	Overrides the set of excluded patterns in the .sourceignore format (which is the same as .gitignore). If not provided, a default of `.git/` is used.	Optional (**)
git.interval	The interval at which to inquire for repository updates. If not provided the default setting is 10 minuets. There is an additional refresh interval (currently 10s) involved before accelerators can appear in the UI. There might be a 10s delay before changes are reflected in the UI.*	Optional (**)
git.ref	Git reference to checkout and monitor for changes, the default is main branch.	Optional (**)
git.ref.branch	The Git branch to checkout, the default is main.	Optional (**)
git.ref.commit	The Git commit SHA to checkout, if specified tag filters are ignored.	Optional (**)
git.ref.semver	The Git tag semver expression, takes precedence over tag.	Optional (**)
git.ref.tag	The Git tag to checkout, takes precedence over branch.	Optional (**)
git.secretRef	The secret name containing the Git credentials. For HTTPS repositories, the secret must contain user name and password fields. For SSH repositories, the secret must contain identity, identity.pub, and known_hosts fields.	Optional (**)
git.subPath	SubPath is the folder inside the git repository to consider as the root of the accelerator or fragment. Defaults at the root of the repository.	Optional

Field	Description	Required/Optional
source	Defines the source image repository.	Optional (***)
source.image	Image is a reference to an image in a remote registry.	Optional (***)
source.imagePullSecrets	ImagePullSecrets contains the names of the Kubernetes Secrets containing registry login information to resolve image metadata	Optional
source.interval	The interval at which to check for repository updates.	Optional
source.serviceAccountName	ServiceAccountName is the name of the Kubernetes ServiceAccount used to authenticate the image pull if the service account has attached pull secrets	Optional

The `Fragment` CRD is defined with the following properties:

Property	Value
Name	Fragment
Group	accelerator.apps.tanzu.vmware.com
Version	v1alpha1
ShortName	frag

Fragment CRD Spec

The `Fragment` CRD *spec* defined in the `FragmentSpec` type has the following fields:

Field	Description	Required/Optional
displayName	DisplayName is a short descriptive name used for a Fragment.	Optional
git	Defines the fragment source Git repository.	Required
git.url	The repository URL, can be a HTTP/S or SSH address.	Required
git.gitImplementation	Determines which git client library to use. The default setting is to go-git. Valid values are ('go-git', 'libgit2'). Deprecated: gitImplementation is deprecated. go-git is the only supported implementation.	Optional (**)
git.ignore	Overrides the set of excluded patterns in the .sourceignore format (which is the same as .gitignore). If not provided, a default of `.git/` is used.	Optional (**)
git.interval	The interval at which to inquire for repository updates. If not provided the default is 10 min.	Optional (**)
git.ref	Git reference to checkout and monitor for changes, the default is main branch.	Optional (**)
git.ref.branch	The Git branch to checkout, defaults to main.	Optional (**)
git.ref.commit	The Git commit SHA to checkout, if specified tag filters are ignored.	Optional (**)
git.ref.semver	The Git tag semver expression, takes precedence over tag.	Optional (**)
git.ref.tag	The Git tag to checkout, takes precedence over branch.	Optional (**)

Field	Description	Required/Optional
git.secretRef	The secret name containing the Git credentials. For HTTPS repositories, the secret must contain user name and password fields. For SSH repositories, the secret must contain identity, identity.pub, and known_hosts fields.	Optional (**)
git.subPath	SubPath is the directory inside the Git repository to consider as the root of the accelerator or fragment. Defaults at the root of the repository.	Optional

* Any optional text boxes marked with an asterisk (*) are populated from a text box of the same name in the `accelerator` definition in the `accelerator.yaml` file if that is present in the Git repository for the accelerator.

** Any fields marked with a double asterisk (**) are part of the Flux GitRepository CRD that is documented in the Flux Source Controller Git Repositories documentation.

*** Any fields marked with a triple asterisk (`***`) are optional but either `git` or `source` is required to specify the repository to use. If `git` is specified, the `git.url` is required, and if `source` is specified, `source.image` is required.

Excluding files

The `git.ignore` field defaults to `.git/`, which is different from the defaults provided by the Flux Source Controller GitRepository implementation. You can override this, and provide your own exclusions. For more information, see fluxcd/source-controller Excluding files.

Use a local Application Accelerator engine server

This topic tells you how to run a local Application Accelerator engine server that you can use for testing accelerators that you are authoring.

About running a local engine server

When you are authoring your accelerator, it might be convenient to generate a project based on the local files. This enables you to verify that the accelerator provides the defined options and generates the correct set of files.

With the local engine server, you can serve your accelerators with their fragments on `localhost`, including any changes you have locally. You can use the Visual Studio Code (VS Code) Tanzu App Accelerator extension or the Tanzu CLI Accelerator plug-in to generate new projects based on these local files. After you are satisfied with the new or modified accelerators and fragments, you can commit them to a Git repository and then publish them to a cluster to give others access.

Install the local engine server

To install the local engine server:

1. Sign in to VMware Tanzu Network.
2. Go to the Tanzu Application Platform product page.
3. Select your Tanzu Application Platform version from the release drop-down menu.
4. From the list of resources, select the file group named `Application Accelerator Engine Server-v1.9.0`. It contains four different ZIP archives, `macos-aarch64`, `macos-amd64`, `windows`, and `linux`. Download the ZIP file for your operating system and architecture.

   The ZIP file contains the local engine server and a Java runtime for running the server.

5. Extract the ZIP file to a local directory by using the `unzip` command or any other extraction tool.
6. For macOS users, you must give permission to open an app from an unidentified developer:
    1. In the Finder on your Mac, locate the directory where you extracted the downloaded ZIP file and expand the `acc-engine` directory.
    2. Control-open the following files:
        - Control-click `acc-engine/app/bin/ytt` and then click **Open**. This runs it in a terminal that you can close.
        - Control-click `acc-engine/app/bin/java` and then click **Open**. This runs it in a terminal that you can close.

    The app files you control-opened are saved as exceptions to your security settings. You can now run them without getting a verification message.

    > **Note**
    >
    > VMware plans to have these artifacts signed using an Apple developer account to avoid these extra steps.

7. Open a terminal window and change directory to `acc-engine` located inside the directory where you extracted the ZIP file.
8. Set an environment variable named `ACC_LOCAL_FILES` that points to a directory that contains the fragments and accelerators you want to use with the local engine server. There must be a directory named `accelerators` and one named `fragments`. Under these directories, you can provide your local accelerators and fragments. For example:

    ```
    workspace
    ├── accelerators
    │   └── hello-world
    │       ├── accelerator.yaml
    │       ├── ...
    └── fragments
        ├── build-wrapper-maven
        │   ├── accelerator.yaml
        │   ├── ...
        ├── java-version
        │   ├── accelerator.yaml
        │   ├── ...
    ```

    For more examples, see application-accelerator-samples in GitHub.

    - **For macOS and Linux:** Set the environment variable, for example:

        ```
        $ export ACC_LOCAL_FILES="$HOME/workspace"
        ```

    - **For Windows Powershell:** Set the environment variable, for example:

        ```
        $ $Env:ACC_LOCAL_FILES="$HOME\workspace"
        ```

Use the local engine server to generate projects

To use the local engine server:

1. Start the local engine server by running the `engine` script from the terminal:

```
./engine
```

2. The latest versions of the VS Code Tanzu App Accelerator extension, Tanzu Application Accelerator plug-in for IntelliJ, and the Tanzu CLI Accelerator plug-in have settings to use the local engine server instead of the regular cluster endpoints.

    - **For the VS Code Tanzu App Accelerator extension:**

      Under **Settings** > **Extensions** > **Tanzu Application Accelerator**, select the **Use Local Server instead of Developer Portal** check box.

      The extension can then show accelerators from the local engine server that you started. Use them in the same way that you use accelerators loaded from Tanzu Developer Portal. For more information, see Use the extension.

    - **For the Tanzu Application Accelerator plug-in for IntelliJ:**

      Under **IntelliJ IDEA** > **Preferences** > **Tools** > **Tanzu Application Accelerator**, select the **Use Local Server instead of Developer Portal** check box.

      The plug-in can then show accelerators from the local engine server that you started. Use them in the same way that you use accelerators loaded from Tanzu Developer Portal. For more information, see Use the plug-in.

    - **For the Tanzu CLI Accelerator plug-in:**

      For the `list`, `get`, and `generate` commands, use the `--local-server` flag instead of `--server-url`.

Test accelerators in Application Accelerator

This topic tells you how to test an updated accelerator, or fragment that is not registered in your Tanzu Application Platform (commonly known as TAP) cluster.

Generating a project from local sources

When you are authoring your accelerator, you can test it before committing any changes.

With the `tanzu accelerator generate-from-local` command, you can run your accelerator (or fragment), including any changes you have locally, specify a set of options and view the generated project.

You can run the accelerator using the components on your Tanzu Application Platform cluster, without impacting the state of the Tanzu Application Platform cluster.

To do so, ensure that you have the following prerequisites:

- The Tanzu CLI is installed, with the Application Accelerator plug-in. For details about installing the Tanzu CLI and plug-ins, see Tanzu CLI.
- The server URL is pointing to the Tanzu Application Platform cluster you want to test with. For details about setting the server URL, see Application Accelerator CLI plug-in overview.

For example, to use the accelerator that is located at the path `workspace/java-rest`:

```
tanzu accelerator generate-from-local --accelerator-path java-rest=workspace/java-rest
--fragment-names tap-workload,java-version --options '{"projectName":"test"}' --output
-dir generated-project
```

This generates the project in the local directory `generated-project`, using the accelerator located at `workspace/java-rest`, the fragments `tap-workload` and `java-version` which are assumed to be already registered in the Tanzu Application Platform cluster and the option `projectName` set to `test`.

For example, to use the fragment named `java-version` that is located at the path `workspace/version`:

```
tanzu accelerator generate-from-local --accelerator-name java-rest --fragment-paths ja
va-version=workspace/version --fragment-names tap-workload --options '{"projectNam
e":"test"}' --output-dir generated-project
```

This generates the project in the local directory `generated-project`, using the accelerator `java-rest` and the fragment `tap-workload` which are assumed to be already registered in the Tanzu Application Platform cluster, the fragment named `java-version` located at `workspace/version`, and the option `projectName` set to `test`.

For more information about the `generate-from-local` command, see the Tanzu CLI Command Reference documentation.

No changes are made to the Tanzu Application Platform cluster that is provided with the server URL. No new accelerators/fragments are registered or modified. A Tanzu Application Platform cluster is required to ensure that there is consistency between the version that is used for testing and the version that is used when the accelerator is registered. Furthermore, it allows using registered fragments and accelerators as dependencies for the local accelerator/fragment.

CI/CD Pipeline

As you iterate on an accelerator, you can have some automated assertions run before any changes to the accelerator are accepted.

The process for generating a project from the committed source files is the same as described earlier.

When the generated project is available, you can run various assertions on it:

```
cd generated-project
test -f build.gradle
./gradlew test
```

If you have multiple assertions, you might choose to run a predefined script:

```
cd generated-project
../assertions/validate-generate-project.sh
```

You might choose to generate multiple projects from the same accelerator, providing different options for each and running different assertions on each generated project.

(Optional) Getting the Tanzu CLI in a CI/CD pipeline

If the Tanzu CLI is already available in your CI/CD pipeline you can skip this section.

VMware provides an example script that is agnostic to the CI/CD system it is running on. The script requires a variable named `TANZU_REFRESH_TOKEN` which holds a personal VMware Tanzu Network refresh token. To generate such a token see How to Authenticate. The script also uses `curl` and `jq`.

The script downloads artifacts compatible with Tanzu Application Platform version v1.4 and a Linux operating system. Update the script to suit the Tanzu Application Platform version and OS that you are using.

```
#!/bin/bash

# Get access token using personal Tanzu Network refresh token
# See https://network.tanzu.vmware.com/docs/api#how-to-authenticate
```

```
ACCESS_TOKEN=$(curl -X POST https://network.tanzu.vmware.com/api/v2/authentication/acc
ess_tokens -d '{"refresh_token":"'"$TANZU_REFRESH_TOKEN"'"}' | jq -r ".access_token")

# Download bundle
# See https://docs.vmware.com/en/VMware-Tanzu-Application-Platform/1.7/tap/install-tan
zu-cli.html#cli-and-plugin
# Update url to download desired version
mkdir -p $HOME/tanzu
curl -L -X GET https://network.tanzu.vmware.com/api/v2/products/tanzu-application-plat
form/releases/1205491/product_files/1352407/download -H "Authorization: Bearer $ACCESS
_TOKEN" --output bundle.tar

# Unpack bundle
export TANZU_CLI_NO_INIT=true
export VERSION=v0.25.0 # Update to desired version
tar -xvf bundle.tar -C $HOME/tanzu
cd $HOME/tanzu

# Install CLI
# Update to use desired OS
sudo install cli/core/$VERSION/tanzu-core-linux_amd64 /usr/local/bin/tanzu

# Install plugins
tanzu plugin install --local cli accelerator
```

Track life cycle using Provenance transform

This topic tells you about the Application Accelerator `Provenance` transform in Tanzu Application Platform, commonly known as TAP.

Overview

Use the `Provenance` transform to track the life cycle of generated projects.

The `Provenance` transform generates a file that provides details of the accelerator engine transform.

The `Provenance` transform provides traceability and visibility for the generation of an application from an accelerator. The following information is embedded into a file that is part of the generated project:

- Which accelerator was used to bootstrap the project
- Which version of the accelerator was used
- When the application was bootstrapped
- Who bootstrapped the application

For more information about the structure of the file and how to enable application bootstrapping provenance, see Provenance transform.

Integration with AMR

Application Accelerator integrates with Artifact Metadata Repository (AMR).

When you generate an application with an accelerator, an event that contains the same information captured by the `Provenance` transform is sent to the AMR store. The relevant data saved to AMR is `AppAcceleratorRuns` (alpha) and `AppAcceleratorFragments` (alpha).

AppAcceleratorRuns (alpha)

The `AppAcceleratorRuns` data model represents a generation of a project from an accelerator. An `accelerator.yaml` file in the repository declares input options for the accelerator. This file contains instructions for processing the files when you generate a new project. This information is sent to the CloudEvent Handler to store `AppAcceleratorRuns`.

There is one `AppAcceleratorRuns` for each invocation of an accelerator, including version information about which accelerator was used. Each `AppAcceleratorRuns` data entry has a unique `guid`. The `AppAcceleratorRuns` data entry contains information about the Git repository including, `AppAcceleratorRepoURL`, `AppAcceleratorRevision`, and `AppAcceleratorSubpath`. Multiple `AppAcceleratorFragments` entries can point to the same `AppAcceleratorRuns` entry. `AppAcceleratorRuns` are associated with one `AppAcceleratorSource`, also known as `Commit`.

AppAcceleratorFragments (alpha)

The `AppAcceleratorFragments` Accelerator fragments are reusable accelerator components that can provide options, files, or transforms. You can import accelerators using an `import` entry. An `InvokeFragment` transform references the transforms from the fragment in the accelerator that declares the import. The `AppAcceleratorFragments` data model represents the information of a fragment in the accelerator app. The Observer sends this information to the Cloud Event Handler to store `AppAcceleratorFragments`.

Each `AppAcceleratorFragment` data entry stores information about source Git repository: `AppAcceleratorFragmentSourceRepoURL`, `AppAcceleratorFragmentSourceRevision`, and `AppAcceleratorFragmentSourceSubpath`. You can associate an `AppAcceleratorFragment` to an `AppAcceleratorRuns`. You can point a `AppAcceleratorFragmentSource` (also known as `Commit`) to one `AppAcceleratorFragment`.

There is one instance of AppAcceleratorFragment for each named fragment used by the running accelerator.

Querying the data

When invocations are recorded, use the GraphQL capability to query the system about accelerator use and gain insights about generated applications. For more information, see Run query with GraphQL

For more information about how to use the Artifact Metadata Repository, and some sample queries relevant to `AppAcceleratorRuns`, see the Artifact Metadata Repository.

Use the Application Accelerator Visual Studio Code extension

This topic describes how to use the Application Accelerator Visual Studio Code extension to explore and generate projects from the defined accelerators in Tanzu Application Platform (commonly known as TAP) using VS Code.

The Application Accelerator Visual Studio Code extension lets you explore and generate projects from the defined accelerators in Tanzu Application Platform using VS Code.

Dependencies

- To use the VS Code extension, the extension must access the Tanzu Developer Portal URL. For information about how to retrieve the Tanzu Developer Portal URL, see Retrieving the URL for the Tanzu Developer Portal.
- (Optionally) To use Git repository provisioning during project creation in the VS Code extension, you must enable GitHub repository creation in the Application Accelerator plug-

in. For more information, see Create an Application Accelerator Git repository during project creation.

Install the extension

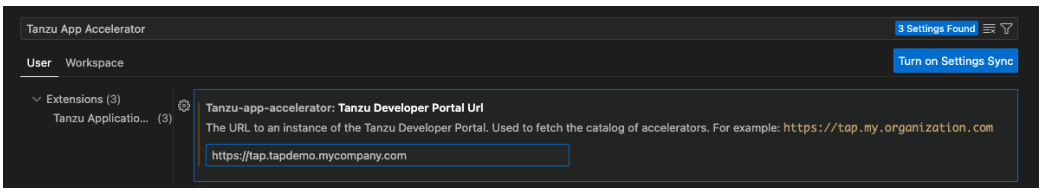
Use the following steps to install the Application Accelerator Visual Studio Code extension:

1. Open Visual Studio Code.
2. Open the command palette and enter `Extensions`.
3. Click **Extensions: Install Extensions**.
4. The **Extensions** view opens on the left side of your screen. In the search box, enter `Tanzu`.
5. Click **Tanzu App Accelerator**.
6. Click **Install**.

Configure the extension

Before using the extension, you need follow the next steps:

1. Go to VS Code settings - click **Code > Preferences > Settings > Extensions > Tanzu App Accelerator**.
2. Look for the setting `Tap Gui Url`.
3. Add the Tanzu Application Platform GUI URL.



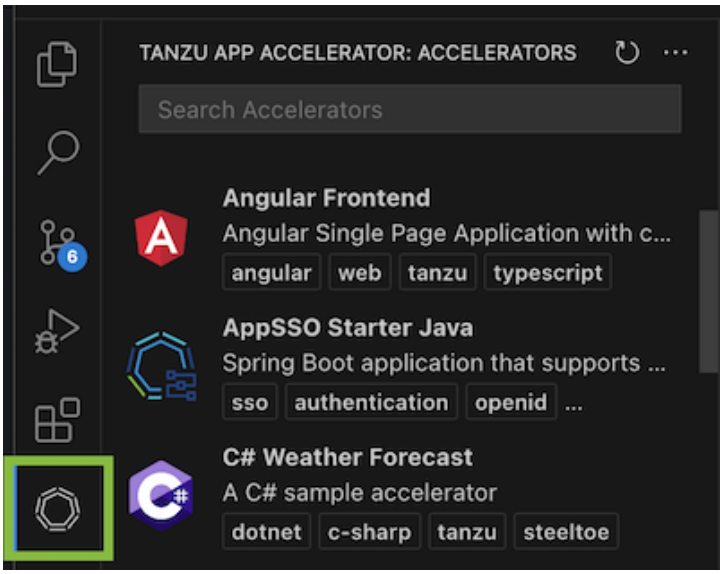
An example URL: `https://tap-gui.myclusterdomain.myorg.com`. If you have access to the Tanzu Application Platform cluster that is running the Tanzu Developer Portal, you can run the following command to determine the fully-qualified domain name:

```
kubectl get httpproxy tap-gui -n tap-gui
```

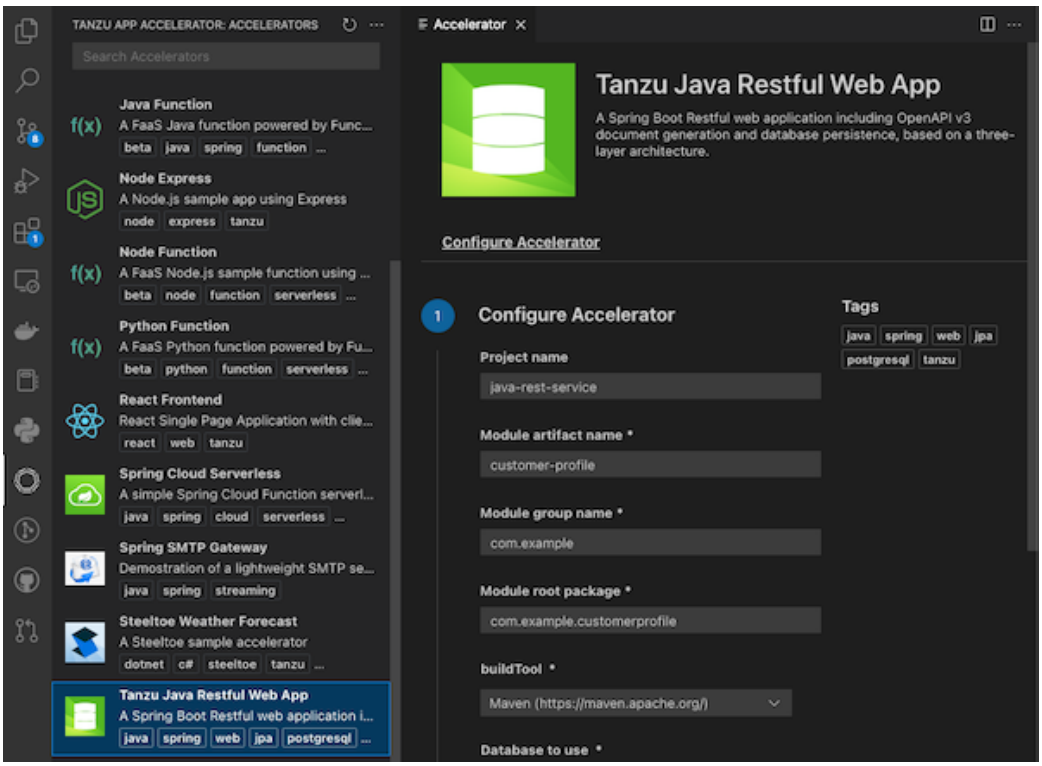
Use the extension

To use the VS Code extension:

1. Click the Tanzu Application Accelerator extension icon in the Activity Bar to explore the defined accelerators.



2. Choose any of the defined accelerators, fill in the options under **Configure Accelerator**, and click **Next**.



3. Review your settings and click **Generate Project**.

Retrieve the URL for the Tanzu Developer Portal

If you have access to the Tanzu Application Platform cluster that is running Tanzu Developer Portal, run the following command to determine the fully-qualified domain name:

```
kubectl get httpproxy tap-gui -n tap-gui
```

Example output:

```
NAME      FQDN                                    TLS SECRET     STATUS   STATUS DES
CRIPTION
```

```
tap-gui   tap-gui.tap.tapdemo.myorg.com             tap-gui-cert   valid    Valid HTTP
Proxy
```

Download and install self-signed certificates from the Tanzu Developer Portal

To enable the Application Accelerator extension for VS Code to communicate with a Tanzu Developer Portal instance that is secured using TLS, you must download and install the certificates locally.

Prerequisites

yq is required to process the YAML output.

Procedure

1. Find the name of the Tanzu Developer Portal certificate. The name of the certificate might look different to the following example.

   ```
   kubectl get secret -n cert-manager
   ```

   ```
   NAME                                          TYPE
   DATA   AGE
   canonical-registry-secret                     kubernetes.io/dockerconfigjson
   1      18d
   cert-manager-webhook-ca                       Opaque
   3      18d
   postgres-operator-ca-certificate              kubernetes.io/tls
   3      18d
   tanzu-sql-with-mysql-operator-ca-certificate  kubernetes.io/tls
   3      18d
   tap-ingress-selfsigned-root-ca                kubernetes.io/tls
   3      18d <------- This is the certificate that is needed
   ```

2. Download the certificate:

   ```
   kubectl get secret -n cert-manager tap-ingress-selfsigned-root-ca -o yaml | yq
   '.data."ca.crt"' | base64 -d > ca.crt
   ```

3. Install the certificate on your local system and fully restart any applications that uses the certificate. After restarting, the application uses the certificate to communicate with the endpoints using TLS. For more information, see Installing a root CA certificate in the trust store in the Ubuntu documentation.

   macOS
   Run:

   ```
   sudo security add-trusted-cert -d -r trustRoot -k /Library/Keychains/System.k
   eychain ca.crt
   ```

   Windows
   Complete the following steps:

   1. Use Windows Explorer to navigate to the directory where the certificate was downloaded and click on the certificate.
   2. In the Certificate window, click **Install Certificate...**.
   3. Change the **Store Location** from **Current User** to **Local Machine**. Click **Next**.

4. Select **Place all certificates in the following store**, click **Browse**, and select **Trusted Root Certification Authorities**
5. Click **Finish**.
6. A pop-up window stating **The import was successful.** is displayed.

Use the Application Accelerator IntelliJ plug-in

This topic tells you about the Application Accelerator IntelliJ plug-in. The plug-in is used to explore and generate projects from the defined accelerators in Tanzu Application Platform (commonly called TAP) using IntelliJ.

Dependencies

The plug-in must have access to the Tanzu Developer Portal URL. For information about how to retrieve the Tanzu Developer Portal URL, see Retrieving the URL for the Tanzu Developer Portal later in this topic.

Install the plug-in

The VMware Tanzu Application Accelerator plug-in for IntelliJ is available from the JetBrains Marketplace.

To install the plug-in from the JetBrains Marketplace:

1. Open IntelliJ.
2. Open the command palette, enter `Plugins`, and click **Plugins**.
3. Select the **Marketplace** tab in the **Plugins Settings** dialog box.
4. In the search box, enter `Tanzu`.
5. Click **Tanzu Application Accelerator** then click **Install**.

Update the plug-in

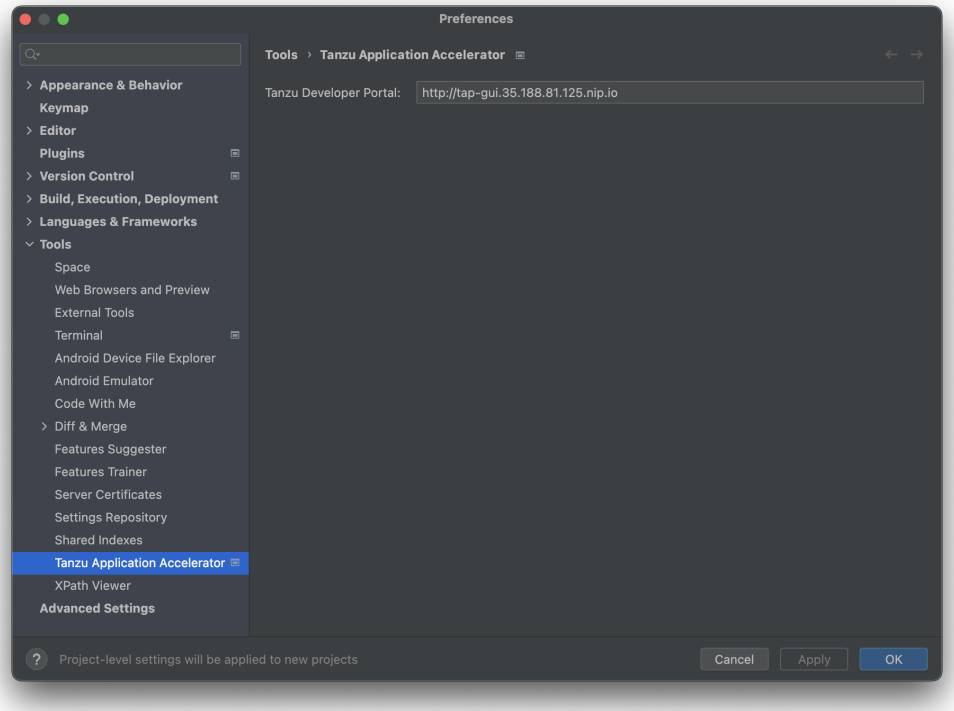
To update to a later version, repeat the steps in Install the plug-in. You do not need to uninstall your current version.

Configure the plug-in

Before using the plug-in, you must enter the Tanzu Developer Portal URL in the IntelliJ Preferences:

1. Go to the IntelliJ menu, select **IntelliJ IDEA > Preferences** > **Tools** > **Tanzu Application Accelerator**.
2. Add the Tanzu Developer Portal URL. For example, `https://tap-gui.myclusterdomain.myorg.com`. If you have access to the Tanzu Application Platform cluster that is running the Tanzu Developer Portal, run the following command to determine the fully-qualified domain name:

```
kubectl get httpproxy tap-gui -n tap-gui
```

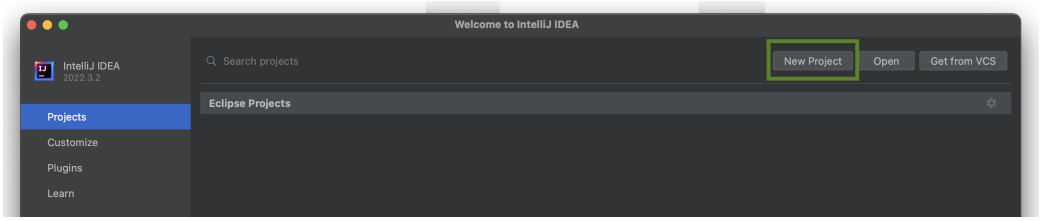


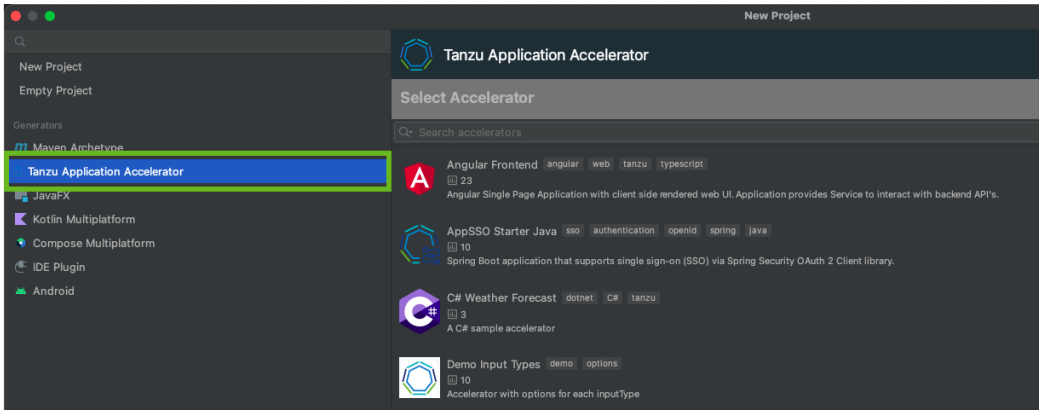
3. Click **Apply** and **OK**.

Use the plug-in

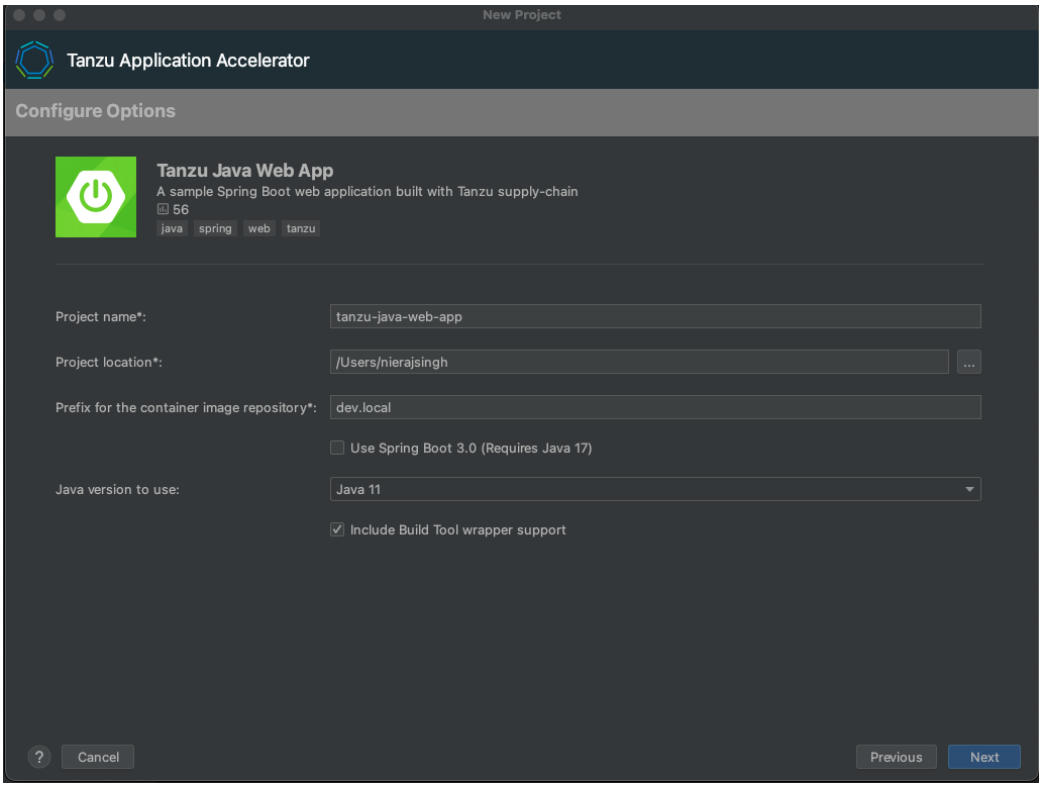
To use the IntelliJ plug-in:

1. To explore the defined accelerators, select **New Project**, then select **Tanzu Application Accelerator**.

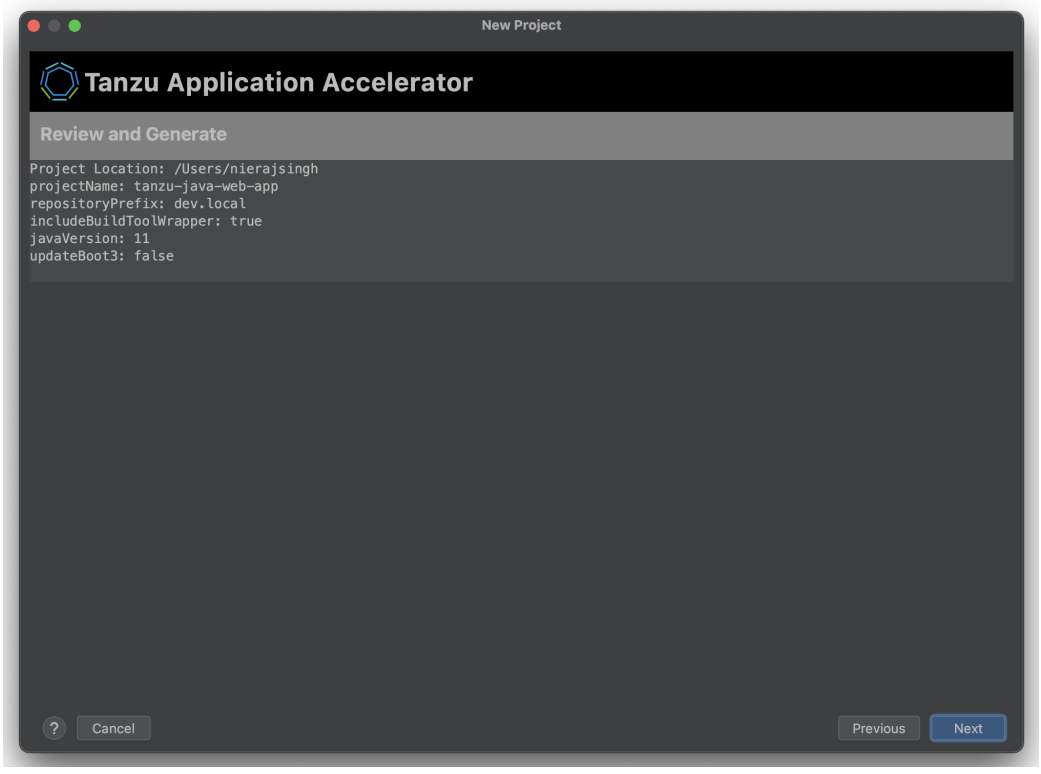




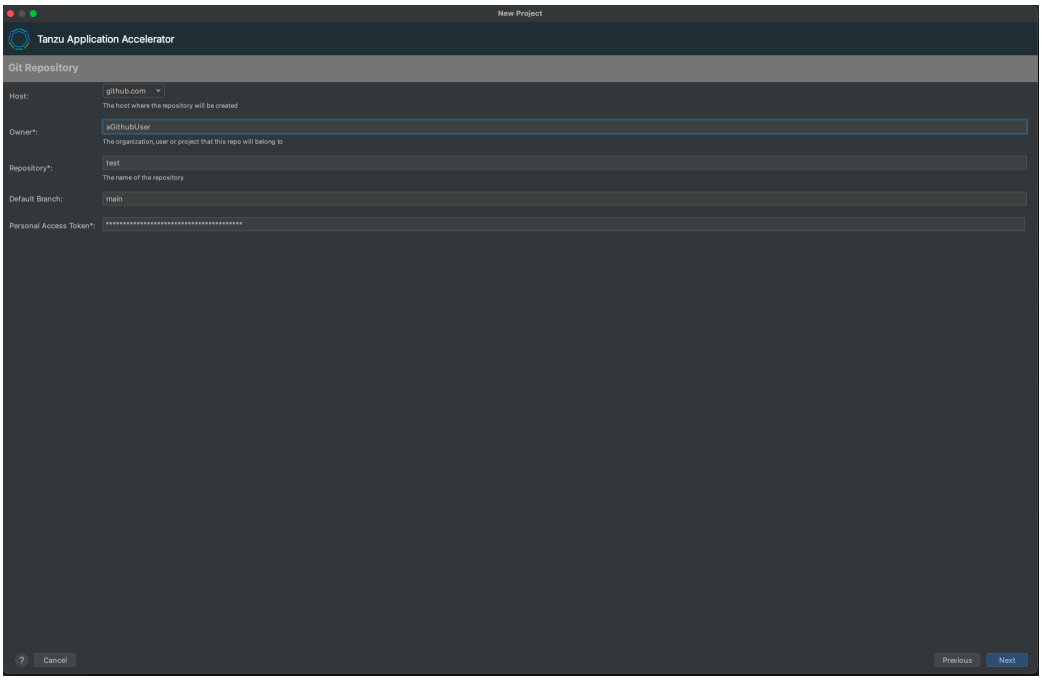
2. Choose one of the defined accelerators and configure the options.



3. Click `Next` to go to the optional step `create git repository`.



4. Fill the fields required to create the Git repository, a personal access token from the Git provider is required, this will be stored in a secured location for future use, click `Next` to go to the review step.



Note: This is an optional step, the values can be left blank if a repository isn't required.

5. Click `Next` to download the project. If the project is downloaded successfully, the `Create` button is enabled and you can now create and open the project.

Retrieving the URL for the Tanzu Developer Portal

If you have access to the Tanzu Application Platform cluster that is running the Tanzu Application Platform GUI, run the following command to determine the fully-qualified domain name:

```
kubectl get httpproxy tap-gui -n tap-gui
```

The result is similar to:

```
NAME      FQDN                                  TLS SECRET     STATUS   STATUS DES
CRIPTION
tap-gui   tap-gui.tap.tapdemo.myorg.com         tap-gui-cert   valid    Valid HTTP
Proxy
```

Download and Install Self-Signed Certificates

To enable communication between the Application Accelerator plug-in and a Tanzu Application Platform GUI instance that is secured using TLS, you must download and install the certificates locally.

Prerequisites

yq is required to process the YAML output.

Procedure

1. Find the name of the Tanzu Developer Portal certificate. The name of the certificate might look different to the following example.

   ```
   kubectl get secret -n cert-manager
   ```

```
NAME                                            TYPE
DATA   AGE
canonical-registry-secret                       kubernetes.io/dockerconfigjson
1      18d
cert-manager-webhook-ca                         Opaque
3      18d
postgres-operator-ca-certificate                kubernetes.io/tls
3      18d
tanzu-sql-with-mysql-operator-ca-certificate    kubernetes.io/tls
3      18d
tap-ingress-selfsigned-root-ca                  kubernetes.io/tls
3      18d <------- This is the certificate that is needed
```

2. Download the certificate:

```
kubectl get secret -n cert-manager tap-ingress-selfsigned-root-ca -o yaml | yq
'.data."ca.crt"' | base64 -d > ca.crt
```

3. Install the certificate on your local system and restart any applications that use the certificate. After restarting, the application uses the certificate to communicate with the endpoints using TLS.

   macOS
   Run:

   ```
   sudo security add-trusted-cert -d -r trustRoot -k /Library/Keychains/System.key
   chain ca.crt
   ```

   For more information, see Installing a root CA certificate in the trust store in the Ubuntu documentation.

   Windows
   Complete the following steps:

   1. Use Windows Explorer to navigate to the directory where the certificate was downloaded and select the certificate.
   2. In the Certificate window, click **Install Certificate...**.
   3. Change the **Store Location** from **Current User** to **Local Machine**. Click **Next**.
   4. Select **Place all certificates in the following store**, click **Browse**, and select **Trusted Root Certification Authorities**
   5. Click **Finish**.
   6. A pop-up window stating **The import was successful.** is displayed.

Uninstall the plug-in

To uninstall the VMware Tanzu Application Accelerator plug-in for IntelliJ:

1. Open the **Preferences** pane and go to **Plugins**.
2. Select the extension, click the gear icon, and click **Uninstall**.
3. Restart IntelliJ.

Application Accelerator best practices

The following topics tells you about best practices for authoring accelerators and fragments.

- **Best practices for using Accelerators**

  A collection of best practices for authoring accelerators.

- **Best practices for using Fragments**

  A collection of best practices for authoring fragments.

Best practices for using accelerators

This topic tells you about the benefits, and design considerations for accelerators.

Benefits of using an accelerator

There are several good reasons to develop accelerators:

- If you're repeatedly using the same application architecture for new applications.
- To enforce standardization of technology stacks and application setups throughout your organization.
- To share best practices around application architecture, application, and test setup.

Design considerations

Each accelerator must have only one base technology stack, combined with related tooling, and one target architecture. For example, if you use both Spring Boot and C# .NET Core applications in your target environment, you must set up two separate accelerators. Mixing multiple technology stacks and multiple target architectures makes both the directory structure and acceleratory.YAML unreadable.

The scope of your accelerator must align with your different types of deployments. For example, back-end API, front-end UI, business service, and so on.

Choose OpenRewrite-based transformation over ReplaceText-based transformation when possible. OpenRewrite-based transformations understand the semantics of the files they work on, for example, Maven pom.xml or Java source files. OpenRewrite-based transformations also provide more accurate and robust modifications. As a last resort, ReplaceText supports a regex mode. When used with capturing groups in the replacement string, ReplaceText allows most modifications.

Housekeeping rules

VMware has found that the following rules keep the set of accelerators clear and findable for end users:

- Use an intuitive name and short description that reflects the accelerators purpose. The word 'accelerator' must not be in the name.
- Use an appropriate and intuitive icon.
- Use tags that reflect language, framework, and type of service. For example, database, messaging, and so on. This helps when searching for an accelerator by tags. Tag names must use lowercase letters, consist of [a-z0-9+#] separated by [-], and not exceed 63 characters.
- Accelerators must expose options to allow configuring an accelerator for different use cases instead of creating multiple similar accelerators.

- Options must be straightforward, the description of each clearly stating the role it plays in the accelerator. Options must have default values when appropriate.
- Options must be short so that they are easy to navigate. Make options conditional on other options as appropriate.
- Free text options that have limitations on their values must ensure these limitations are met by a regular expression-based validation. This validation ensures early feedback on invalid user input.
- Generated application skeletons must have a detailed README file that describes the function and structure of a generated application. It must provide detailed information about how developers can build and deploy a generated application of the accelerator and how to use it.

Tests

Application skeleton

An accelerator that generates an application skeleton without a good test suite for the different layers of the application promotes bad behavior. It could result in code running in production without testing.

Tests you could use for the application skeleton:

- An overall application test that bootstraps the application to see if it comes online.
- A test per layer of the application. For example, presentation layer, business layer, and data layer. These tests can be unit tests that leverage stubbing or mocking frameworks.
- An integration test per layer of the application, especially the presentation and data layer. For example, you can provide an integration test with some database interaction by using test containers.

Best practices for using fragments

This topic tells you about the benefits, and design considerations for fragments.

Benefits of using Fragment

A fragment is a partial accelerator. It can do the same transformations as an accelerator, but it cannot run on its own. It’s always part of the calling (host) accelerator.

Developing a fragment is useful in the following situations:

- When you must update a version of an element of a technology stack in multiple locations. For example, when the Java Development Kit (JDK) version must be updated in the build tool configuration, the buildpack configuration, and in the deployment options.
- To add a consistent cross-cutting concern to a set of accelerators. For example, logging, monitoring, or support for a certain type of deployment or framework.
- To add integration with some technology to a generated application skeleton. For example, certain database support, support for a messaging middleware, or integration with an email provider.

Design considerations

Developing and maintaining a fragment is complex. The following is a list of design considerations:

- The fragment you develop must work with all possible syntax and format variations. For example, dependency in a `Gradle` build.gradle.kts can have the following forms:
    - `implementation('org.springframework.boot:spring-boot-starter')`
    - `implementation("org.springframework.boot:spring-boot-starter")`
    - `implementation(group = "org.springframework.boot", name= "spring-boot-starter")`
    - `implementation(group = 'org.springframework.boot', name= 'spring-boot-starter')`
    - `implementation(name= "spring-boot-starter", group = "org.springframework.boot")`
- The fragment can be used in multiple accelerator contexts and its behavior must result in a compilable and deployable application skeleton.
- Testing a fragment in isolation is more difficult than testing an accelerator. Testing takes more time because all the combinations must be tested from an accelerator perspective.
- When flexibly reusing fragments in different combinations, each fragment must cover a small, cohesive function. Fragments must follow these two UNIX principles:
    - Small is beautiful.
    - Each fragment does one thing well.
- Keep the files the fragment changes to a minimum. Only change the files that are related to the same technology stack for the same purpose.
- The design of both the accelerator and fragment is limited by the technology stack and the target deployment technology chosen for the accelerator. For example, to create a fragment for standardizing logging, you must create one fragment per base technology stack.

Housekeeping rules

Fragments are used by accelerator authors. VMware has found that the following guidelines keep fragments understandable and reusable.

- Give fragments an intuitive name and short description that reflects their purpose. Do not include “fragment” in the name.
- Fragments must expose options to allow configuring the output of execution.
- Each fragment must contain a README file explaining the additional functions the fragment adds to a generated application skeleton. List any options expected by this fragment. Also describe how this fragment can be included in a host accelerator. Be sure to state any known limitations or use cases not covered. For example, if the fragment supports Maven and Gradle as build tools but only Groovy DSL of Gradle is supported, the README file must include this information.
- If a fragment must provide additional documentation to end users, it can either be added to a README-X file of the generated application skeleton or append a section to the host’s README.

Versioning

Fragments might require the use of versioning. For example, if accelerator `acc-1` imports fragment `frag`, and accelerator `acc-2` also uses fragment `frag`, then care must be taken in the contract that

the fragment exposes. Changing its behavior for the benefit of `acc-1` might break its use in `acc-2`.

To resolve this, you can track the versions of the fragment and document with numbers when there is an incompatible change. VMware recommends that you use semantic versioning to track the contract for fragments.

To give you more flexibility with version control, Application Accelerator does not manage versioning. VMware recommends that you track the semantic version of fragments in the fragment name. For example, if fragment `frag` changes its contract in a way that is not compatible with previous accelerators that use it, it becomes `frag-v2` and is regarded as a different fragment.

You can store fragment versions where you want, but version control systems such as Git are a good choice.

Troubleshoot Application Accelerator

This topic provides troubleshooting steps for development, accelerator authorship, and operations issues in Application Accelerator.

Installation issues

Depending on the error output, you can take the following actions to troubleshoot installation problems.

Verify installed packages

The package might be already installed. Verify this by running:

```
tanzu package installed list -n tap-install
```

Look for any package called `accelerator.apps.tanzu.vmware.com`.

Look at resource events

The error might be within the custom resources such as accelerator, Git repository, fragment, and so on. Find these errors by using the Kubernetes command line interface tool (kubectl).

Here is an example using the custom resource `accelerator`:

1. Check for errors. For example, review the custom resource `accelerator` by running:

   ```
   kubectl get acc -n accelerator-system
   ```

   Note items in the output with a `READY` status `False`:

   ```
   NAME                        READY   REASON      AGE
   appsso-starter-java         True    Ready       5h2m
   where-for-dinner            True    Ready       5h2m
   java-function               True    Ready       5h2m
   java-rest-service           True    Ready       5h2m
   java-server-side-ui         True    Ready       5h2m
   node-express                True    Ready       5h2m
   node-function               False   Not-Ready   5h2m
   python-function             True    Ready       5h2m
   spring-cloud-serverless     True    Ready       5h2m
   spring-smtp-gateway         True    Ready       5h2m
   tanzu-java-web-app          True    Ready       5h2m
   tap-initialize              True    Ready       5h2m
   weatherforecast-csharp      True    Ready       5h2m
   weatherforecast-steeltoe    True    Ready       5h2m
   ```

2. To see more information about any error events you found, run:

   ```
   kubectl get acc node-function -n accelerator-system -o yaml
   ```

   Look at the event section for more information about the error.

Development issues

Failure to generate a new project

`URI is not absolute` error

The `generate` command fails with the following error:

```
% tanzu accelerator generate test --server-url https://accelerator.example.com
Error: there was an error generating the accelerator, the server response was: "URI is
not absolute"

Use:
  tanzu accelerator generate [flags]

Examples:
  tanzu accelerator generate <accelerator-name> --options '{"projectName":"test"}'

Flags:
  -h, --help                  help for generate
      --options string        options JSON string
      --options-file string   path to file containing options JSON string
      --output-dir string     directory that the zip file will be written to
      --server-url string     the URL for the Application Accelerator server

Global Flags:
      --context name      name of the kubeconfig context to use (default is current-co
ntext defined by kubeconfig)
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)

there was an error generating the accelerator, the server response was: "URI is not ab
solute"

Error: exit status 1

✖  exit status 1
```

This indicates that the accelerator resource requested is not in a `READY` state. Review the instructions in the When Accelerator ready column is false section or contact your system admin.

Accelerator authorship issues

General tips

Speed up the reconciliation of the accelerator

Set the `git.interval` to make the accelerator reconcile sooner. The default interval is 10 minutes, which is too long when developing an accelerator.

You can set this when using the YAML manifest:

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
```

```
  name: test-accelerator
spec:
  git:
    url: https://github.com/trisberg/test-accelerator
    ref:
      branch: main
    interval: 10s
```

You can also set this when creating the accelerator resource. To do so from the Tanzu CLI, run:

```
tanzu accelerator create test-accelerator --git-repo https://github.com/trisberg/test-accelerator --git-branch main --interval 10s
```

Use a source image with local accelerator source directory

You don't have to use a Git repository when developing an accelerator. You can create an accelerator based on content in a local directory using `--local-path` when creating the accelerator resource.

Push the local path content to an OCI image by running:

```
tanzu accelerator create test-accelerator --local-path . --source-image REPO-PREFIX/test-accelerator --interval 10s
```

Where `REPO-PREFIX` is your own repository prefix. Use a repository that the deployed Application Accelerator system can access.

The interval is 10s so that you can push changes to the source-image repository and get faster reconcile time for the accelerator resource. When you have made changes to your accelerator source, push those changes by running:

```
tanzu accelerator push --local-path . --source-image REPO-PREFIX/test-accelerator
```

Where `REPO-PREFIX` is your own repository prefix. Use a repository that is accessible to the deployed Application Accelerator system.

Expression evaluation errors

Expression evaluation errors include:

- Expression `evaluated to null`, such as:

  ```
  Could not read response from accelerator: java.lang.IllegalArgumentException: Expression '#mytestexp' evaluated to null
  ```

  In most cases, a typo in the variable name causes this error. Compare the expression with the defined options or any variables declared with `let`.

- `could not parse SpEL expression`, such as:

  ```
  Could not read response from accelerator: Error reading manifest:could not parse SpEL expression at [Source: (InputStreamReader); line: 65, column: 1] (through reference chain: com.vmware.tanzu.accelerator.engine.manifest.Manifest["engine"]->com.vmware.tanzu.accelerator.engine.transform.transforms.Combo["let"]->java.util.ArrayList[0]->com.vmware.tanzu.accelerator.engine.transform.transforms.Let$DerivedSymbol["expression"])
  ```

  In most cases, an error in a `let` expression causes this error. Review the error message and, for more information, see SpEL samples.

- `SpelEvaluationException`, such as:

```
Could not read response from accelerator: org.springframework.expression.spel.S
pelEvaluationException: EL1007E: Property or field 'test' cannot be found on nu
ll
```

In most cases, an error in a transform expression causes this error. Review the error message and, for more information, see SpEL samples.

Operations issues

Accelerator persists in Tanzu Developer Portal after deletion

If an accelerator still displays in the Tanzu Developer Portal after it is deleted using the `tanzu accelerator delete` command, complete the following steps to delete:

1. Navigate to your instance of the Tanzu Developer Portal.
2. Search for the accelerator which should be deleted and select it.
3. On the top right of the window, click the three dots, and select **Unregister Template**.

The accelerator is not longer displayed in the Tanzu Developer Portal Accelerator Catalog.

Check status of accelerator resources

Verify the status of accelerator resources by using kubectl or the Tanzu CLI:

- From kubectl, run:

  ```
  kubectl get accelerators.accelerator.apps.tanzu.vmware.com -n accelerator-syste
  m
  ```

- From the Tanzu CLI, run:

  ```
  tanzu accelerator list
  ```

Verify that the `READY` status is `true` for all accelerators.

When Accelerator ready column is blank

1. View the status of `accelerator-system` by running:

   ```
   kubectl get deployment -n accelerator-system
   ```

   Example output:

   ```
   NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
   acc-engine                        1/1     1            1           3d5h
   acc-server                        1/1     1            1           2d1h
   accelerator-controller-manager    0/1     1            0           3d5h
   ```

2. View the logs for any component with no Pods available by running:

   ```
   kubectl logs deployment/COMPONENT-NAME/ -n accelerator-system -p
   ```

   Where `COMPONENT-NAME` is the component with no pods you retrieved in the previous step.

   - If the log has the following error then the Flux CD source-controller is not installed:

     ```
     2021-11-18T20:55:18.963Z ERROR setup problem running manager {"error": "f
     ailed to wait for accelerator caches to sync: no matches for kind \"GitRe
     pository\" in version \"source.toolkit.fluxcd.io/v1beta1\""}
     ```

- If the log has the following error, the Tanzu Application Platform source-controller is not installed:

  ```
  2021-11-18T20:50:10.557Z ERROR setup problem running manager {"error": "f
  ailed to wait for accelerator caches to sync: no matches for kind \"Image
  Repository\" in version \"source.apps.tanzu.vmware.com/v1alpha1\""}
  ```

When Accelerator ready column is false

View the `REASON` column for non-ready accelerators. Run:

```
kubectl get accelerators.accelerator.apps.tanzu.vmware.com -n accelerator-system
```

REASON: `GitRepositoryResolutionFailed`

For example:

```
$ kubectl get accelerators.accelerator.apps.tanzu.vmware.com -n accelerator-system
NAME        READY   REASON                          AGE
more-fun    False   GitRepositoryResolutionFailed   28s
```

1. View the resource status. Run:

   ```
   kubectl get -oyaml accelerators.accelerator.apps.tanzu.vmware.com -n accelerato
   r-system hello-fun
   ```

2. Read `status.conditions.message` near the end of the output to learn the likely cause of failure. For example:

   ```
   status:
     address:
       url: http://accelerator-engine.accelerator-system.svc.cluster.local/invocat
   ions
     artifact:
       message: 'unable to clone ''https://github.com/vmware-tanzu/application-acc
   elerator-samples'',
         error: couldn''t find remote ref "refs/heads/test"'
       ready: false
       url: ""
     conditions:
     - lastTransitionTime: "2021-11-18T21:05:47Z"
       message: |-
         failed to resolve GitRepository
         unable to clone 'https://github.com/vmware-tanzu/application-accelerator-
   samples', error: couldn't find remote ref "refs/heads/test"
       reason: GitRepositoryResolutionFailed
       status: "False"
       type: Ready
     description: Test-git
     observedGeneration: 1
   ```

   In this example, `couldn't find remote ref "refs/heads/test"` reveals that the branch or tag specified doesn't exist.

   Another common problem is that the Git repository doesn't exist. For example:

   ```
   status:
     address:
       url: http://accelerator-engine.accelerator-system.svc.cluster.local/invocat
   ions
     artifact:
   ```

```
    message: 'unable to clone ''https://github.com/vmware-tanzu/application-acc
elerator-sampl'',
      error: authentication required'
    ready: false
    url: ""
  conditions:
  - lastTransitionTime: "2021-11-18T21:09:52Z"
    message: |-
      failed to resolve GitRepository
      unable to clone 'https://github.com/vmware-tanzu/application-accelerator-
sampl', error: authentication required
    reason: GitRepositoryResolutionFailed
    status: "False"
    type: Ready
  description: Test-git
  observedGeneration: 1
```

An error message about failed authentication might display because the Git repository doesn't exist. For example:

```
unable to clone 'https://github.com/vmware-tanzu/application-accelerator-samp
l', error: authentication required
```

REASON: `GitRepositoryResolutionPending`

For example:

```
$ kubectl get accelerators.accelerator.apps.tanzu.vmware.com -n accelerator-system
NAME        READY   REASON                            AGE
more-fun    False   GitRepositoryResolutionPending    28s
```

1. See the resource status. Run:

   ```
   kubectl get -oyaml accelerators.accelerator.apps.tanzu.vmware.com -n accelerato
   r-system hello-fun
   ```

2. Locate `status.conditions` at the end of the output. For example:

   ```
   status:
     address:
       url: http://accelerator-engine.accelerator-system.svc.cluster.local/invocat
   ions
     artifact:
       message: ""
       ready: false
       url: ""
     conditions:
     - lastTransitionTime: "2021-11-18T20:17:38Z"
       message: GitRepository not yet resolved
       reason: GitRepositoryResolutionPending
       status: "False"
       type: Ready
     description: Test-git
     observedGeneration: 1
   ```

3. Verify that the Flux system is running and that the `READY` column has `1/1`. Run:

   ```
   kubectl get -n flux-system deployment/source-controller
   ```

   Example output:

```
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
source-controller   1/1     0            0           5d4h
```

REASON: `ImageRepositoryResolutionPending`

For example:

```
$ kubectl get accelerators.accelerator.apps.tanzu.vmware.com -n accelerator-system
NAME        READY   REASON                            AGE
more-fun    False   ImageRepositoryResolutionPending   28s
```

1. See the resource status. Run:

   ```
   kubectl get -oyaml accelerators.accelerator.apps.tanzu.vmware.com -n accelerato
   r-system hello-fun
   ```

2. Locate `status.conditions` at the end of the output. For example:

   ```
   $ kubectl get -oyaml accelerators.accelerator.apps.tanzu.vmware.com -n accelera
   tor-system more-fun

   apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
   kind: Accelerator
   metadata:
     annotations:
       kubectl.kubernetes.io/last-applied-configuration: |
         {"apiVersion":"accelerator.apps.tanzu.vmware.com/v1alpha1","kind":"Accele
   rator","metadata":{"annotations":{},"name":"more-fun","namespace":"accelerator-
   system"},"spec":{"description":"Test-image","source":{"image":"trisberg/more-fu
   n-source"}}}
     creationTimestamp: "2021-11-18T20:32:36Z"
     generation: 1
     name: more-fun
     namespace: accelerator-system
     resourceVersion: "605401"
     uid: 407b565d-14aa-44fe-ad8d-c9b3c3a7e5ce
   spec:
     description: Test-image
     source:
       image: trisberg/more-fun-source
   status:
     address:
       url: http://accelerator-engine.accelerator-system.svc.cluster.local/invocat
   ions
     artifact:
       message: ""
       ready: false
       url: ""
     conditions:
     - lastTransitionTime: "2021-11-18T20:32:36Z"
       message: ImageRepository not yet resolved
       reason: ImageRepositoryResolutionPending
       status: "False"
       type: Ready
     description: Test-image
     observedGeneration: 1
   ```

3. Verify that Tanzu Application Platform source-controller system is running and the `READY` column has `1/1`. Run:

   ```
   kubectl get -n source-system deployment/source-controller-manager
   ```

Expected output:

```
NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
source-controller-manager     1/1     0            0           5d5h
```

Overview of Application Configuration Service for VMware Tanzu

Application Configuration Service (commonly known as App Config Service) provides a Kubernetes-native experience to enable the runtime configuration of existing Spring applications that were previously leveraged by using Spring Cloud Config Server.

Spring Cloud Config Server was an essential component in microservices architectures for providing runtime configuration to Spring Boot applications.

Spring Cloud Config Server did this by enabling configuration management to be hosted in Git repositories on different branches and in directories that could be used to generate runtime configuration properties for applications.

Application Configuration Service is compatible with the existing Git repository configuration management approach. It filters runtime configuration for any application by using slices that produce secrets.

For more information about Application Configuration Service, see the Application Configuration Service for VMware Tanzu documentation.

Overview of Application Configuration Service for VMware Tanzu

Application Configuration Service (commonly known as App Config Service) provides a Kubernetes-native experience to enable the runtime configuration of existing Spring applications that were previously leveraged by using Spring Cloud Config Server.

Spring Cloud Config Server was an essential component in microservices architectures for providing runtime configuration to Spring Boot applications.

Spring Cloud Config Server did this by enabling configuration management to be hosted in Git repositories on different branches and in directories that could be used to generate runtime configuration properties for applications.

Application Configuration Service is compatible with the existing Git repository configuration management approach. It filters runtime configuration for any application by using slices that produce secrets.

For more information about Application Configuration Service, see the Application Configuration Service for VMware Tanzu documentation.

Install Application Configuration Service for VMware Tanzu

This topic tells you how to install Application Configuration Service for VMware Tanzu (commonly known as App Config Service) from the Tanzu Application Platform (commonly known as TAP) package repository.

Prerequisites

Before installing Application Configuration Service, complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.

Install

To install Application Configuration Service on a compliant Kubernetes cluster:

1. List version information for the package by running:

   ```
   tanzu package available list application-configuration-service.tanzu.vmware.com
   --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package available list application-configuration-service.tanzu.vmware.c
   om --namespace tap-install
   - Retrieving package versions for application-configuration-service.tanzu.vmwar
   e.com...
   NAME                                                VERSION  RELEASED-AT
   application-configuration-service.tanzu.vmware.com  2.0.0    2023-03-08 19:00:0
   0 -0500 EST
   ```

2. (Optional) Make changes to the default installation settings by running:

   ```
   tanzu package available get application-configuration-service.tanzu.vmware.com/
   VERSION-NUMBER \
   --values-schema --namespace tap-install
   ```

   Where `VERSION-NUMBER` is the number you discovered previously. For example, `2.0.0`.

3. Install the package by running:

   ```
   tanzu package install application-configuration-service \
   --package application-configuration-service.tanzu.vmware.com \
   --version VERSION -n tap-install \
   --values-file values.yaml
   ```

   Where `VERSION` is the version you want, such as `2.0.0`. Using a `values.yaml` file is optional.

   For example:

   ```
   $ tanzu package install application-configuration-service \
   --package application-configuration-service.tanzu.vmware.com \
   --version 2.0.0 -n tap-install

   Installing package 'application-configuration-service.tanzu.vmware.com'
   Getting package metadata for 'application-configuration-service.tanzu.vmware.co
   m'
   Creating service account 'application-configuration-service-tap-install-sa'
   Creating cluster admin role 'application-configuration-service-tap-install-clus
   ter-role'
   Creating cluster role binding 'application-configuration-service-tap-install-cl
   uster-rolebinding'
   Creating package resource
   Waiting for 'PackageInstall' reconciliation for 'application-configuration-serv
   ice'
   'PackageInstall' resource install status: Reconciling
   'PackageInstall' resource install status: ReconcileSucceeded

   Added installed package 'application-configuration-service'
   ```

4. Verify that you installed the package by running:

   ```
   tanzu package installed get application-configuration-service -n tap-install
   ```

   For example:

```
$ tanzu package installed get application-configuration-service -n tap-install

NAME:                    application-configuration-service
PACKAGE-NAME:            application-configuration-service.tanzu.vmware.com
PACKAGE-VERSION:         2.0.0
STATUS:                  Reconcile succeeded
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Overview of Application Live View

Application Live View is a lightweight insights and troubleshooting tool for app developers and app operators that helps you to look inside running applications. It is based on the concept of Spring Boot Actuators.

The application provides information from inside the running processes using endpoints, in this case, HTTP endpoints. Application Live View uses those endpoints to get and interact with the data from apps.

Value proposition

Application Live View is a diagnostic tool for developers to manage and analyze runtime characteristics of containerized apps. In addition, it provides a Kubernetes-native feel for developers to manage their apps in a Kubernetes environment more effectively.

Intended audience

This documentation is intended for developers and operators to visualize the actuator information of their running apps on Application Live View for VMware Tanzu. This documentation helps developers to monitor and troubleshoot apps in development, staging, and production environments. It is also intended to help app operators to deploy and administer containerized apps in a Kubernetes environment.

Supported application platforms

You can extend Application Live View to support multiple app platforms, including, but not limited to, Spring Boot, Spring Cloud Gateway, and Steeltoe.

Multicloud compatibility

Using Tanzu platform, you can integrate Application Live View to monitor apps running across on-premises, public clouds, and edge. The platform provides a centralized view to manage apps across cloud environments, which accelerates developer productivity and reduces time-to-market.

Deployment

A connector is a component responsible for discovering and registering multiple apps with Application Live View running on a Kubernetes cluster. The connector is installed as a Kubernetes Deployment by default, but you can switch to DaemonSet or Namespace-scoped mode if required.

Overview of Application Live View

Application Live View is a lightweight insights and troubleshooting tool for app developers and app operators that helps you to look inside running applications. It is based on the concept of Spring

Boot Actuators.

The application provides information from inside the running processes using endpoints, in this case, HTTP endpoints. Application Live View uses those endpoints to get and interact with the data from apps.

Value proposition

Application Live View is a diagnostic tool for developers to manage and analyze runtime characteristics of containerized apps. In addition, it provides a Kubernetes-native feel for developers to manage their apps in a Kubernetes environment more effectively.

Intended audience

This documentation is intended for developers and operators to visualize the actuator information of their running apps on Application Live View for VMware Tanzu. This documentation helps developers to monitor and troubleshoot apps in development, staging, and production environments. It is also intended to help app operators to deploy and administer containerized apps in a Kubernetes environment.

Supported application platforms

You can extend Application Live View to support multiple app platforms, including, but not limited to, Spring Boot, Spring Cloud Gateway, and Steeltoe.

Multicloud compatibility

Using Tanzu platform, you can integrate Application Live View to monitor apps running across on-premises, public clouds, and edge. The platform provides a centralized view to manage apps across cloud environments, which accelerates developer productivity and reduces time-to-market.

Deployment

A connector is a component responsible for discovering and registering multiple apps with Application Live View running on a Kubernetes cluster. The connector is installed as a Kubernetes Deployment by default, but you can switch to DaemonSet or Namespace-scoped mode if required.

Install Application Live View

This topic tells you how to install Application Live View from the Tanzu Application Platform (commonly known as TAP) package repository.

Overview

Application Live View includes four packages you must install. The following table lists these packages and shows the Tanzu Application Platform profiles each package is included in.

Package	Profiles	Details
Application Live View APIServer (`apiserver.appliveview.tanzu.vmware.com`)	Full, Iterate, Run	Installed in the `appliveview-tokens-system` namespace

Package	Profiles	Details
Application Live View back end (`backend.appliveview.tanzu.vmware.com`)	Full, View	Installed with Tanzu Developer Portal in the `app-live-view` namespace
Application Live View connector (`connector.appliveview.tanzu.vmware.com`)	Full, Iterate, Run	Installed as a DaemonSet in the `app-live-view-connector` namespace
Application Live View conventions (`conventions.appliveview.tanzu.vmware.com`)	Full, Iterate, Build	Installed in the `app-live-view-conventions` namespace

For more information about these packages, see Application Live View internal architecture.

Note

Follow the steps in this topic if you do not want to use a profile to install Application Live View. For more information about profiles, see About Tanzu Application Platform components and profiles.

Prerequisites

Before installing Application Live View:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- Install Cartographer Conventions, which is bundled with Supply Chain Choreographer as of v0.5.3. To install, see Installing Supply Chain Choreographer. For more information, see Cartographer Conventions.

Install Application Live View

You can install Application Live View in single cluster or multicluster environment:

- `Single cluster`: All Application Live View components are deployed in a single cluster. The user can access Application Live View plug-in information of the applications across all the namespaces in the Kubernetes cluster. This is the default mode of Application Live View.
- `Multicluster`: In a multicluster environment, the Application Live View back end component is installed only once in a single cluster and exposes a RSocket registration for the other clusters using Tanzu shared ingress. Each cluster continues to install the connector as a DaemonSet. The connectors are configured to connect to the central instance of the Application Live View back end.

The improved security and access control secures the communication between the Application Live View components. For more information, see Configure security and access control in Application Live View.

Install Application Live View back end

To install Application Live View back end:

1. List version information for the package by running:

```
tanzu package available list backend.appliveview.tanzu.vmware.com --namespace t
```

```
ap-install
```

For example:

```
$ tanzu package available list backend.appliveview.tanzu.vmware.com --namespace
tap-install

- Retrieving package versions for backend.appliveview.tanzu.vmware.com...
  NAME                                 VERSION        RELEASED-AT
  backend.appliveview.tanzu.vmware.com  1.9.0          2024-03-22T00:00:00Z
```

2. Create the file `app-live-view-backend-values.yaml` using the following information:

Single-cluster environment
For a single-cluster environment, the Application Live View back end is exposed through the Kubernetes cluster service.

By default, ingress is deactivated for Application Live View back end. You do not have to change your `app-live-view-backend-values.yaml` file in this step.

Multicluster environment
For a multicluster environment, set the flag `ingressEnabled` to `true` for the Application Live View back end to be exposed on the ingress domain.

```
appliveview:
  ingressEnabled: true
```

Profile install using shared ingress domain key
If you are using a Tanzu Application Platform profile installation and the top-level key `shared.ingress_domain` is set in the `tap-values.yaml`, the back end is automatically exposed through the shared ingress.

To override the shared ingress for Application Live View in a multicluster environment, use the following values:

```
appliveview:
  ingressEnabled: true
  ingressDomain: ${INGRESS-DOMAIN}
```

Where `INGRESS-DOMAIN` is the top-level domain you use for the `tanzu-shared-ingress` service's external IP address. The `appliveview` subdomain is prepended to the value provided.

3. Configure TLS in your `app-live-view-backend-values.yaml` file:

Activate TLS with self-signed certificate
To enable TLS for Application Live View back end using a self-signed certificate:

1. Create the `app-live-view` namespace and the TLS secret for the domain. You must do this before installing the Tanzu Application Platform packages in the cluster so that the HTTPProxy is updated with the TLS secret. To create a TLS secret, run:

   ```
   kubectl create -n app-live-view secret tls alv-cert --cert=CERT-FILE -
   -key=KEY-FILE
   ```

   Where:

- `SECRET-NAME` is the name you want for the TLS secret for the domain, for example, `alv-cert`.
- `CERT-FILE` is a .crt file that contains the PEM encoded server certificate.
- `KEY-FILE` is a .key file that contains the PEM encoded server private key.

2. Provide the following properties in your `app-live-view-backend-values.yaml`:

```
appliveview:
  ingressEnabled: true
  tls:
    namespace: "NAMESPACE"
    secretName: "SECRET-NAME"
```

Where:

- `NAMESPACE` is the targeted namespace of TLS secret for the domain.
- `SECRET-NAME` is the name of TLS secret for the domain.

You can edit the values to suit your project needs or leave the default values as is.

When `ingressEnabled` is `true`, the HTTPProxy object is created in the cluster.

3. Verify the HTTPProxy object with the TLS secret by running:

```
kubectl get httpproxy -A
```

Expected output:

```
NAMESPACE       NAME          FQDN                                TLS S
ECRET               STATUS   STATUS DESCRIPTION
app-live-view   appliveview   appliveview.192.168.42.55.nip.io   app-l
ive-view/alv-cert   valid    Valid HTTPProxy
```

Activate TLS using ClusterIssuer

To enable TLS for Application Live View back end using ClusterIssuer:

1. Set the `ingressEnabled` key to `true` for TLS to be enabled on Application Live View back end using ClusterIssuer. This key is set to `false` by default.

```
appliveview:
  ingressEnabled: true
```

TLS is then automatically enabled on Application Live View back end using the shared ClusterIssuer. The `appliveview-cert` certificate is generated by default and its issuerRef points to the `.ingress_issuer` value. The `ingress_issuer` key consumes the value `shared.ingress_issuer` from `tap-values.yaml` by default if you don't specify the `ingress_issuer` in `tap-values.yaml`.

When `ingressEnabled` is `true`, HTTPProxy object is created in the cluster and also `appliveview-cert` certificate is generated by default in the `app_live_view` namespace. Here, the secretName `appliveview-cert` stores this certificate.

2. To verify the HTTPProxy object with the secret, run:

```
kubectl get httpproxy -A
```

Expected output:

```
NAMESPACE        NAME           FQDN                                  TLS S
ECRET          STATUS   STATUS DESCRIPTION
app-live-view    appliveview    appliveview.192.168.42.55.nip.io    appli
veview-cert    valid    Valid HTTPProxy
```

3. To verify the Application Live View pages in a multicluster setup, set the appropriate connector configuration in your run cluster as listed in Install Application Live View connector later in this topic.

Deactivate TLS

By default, Tanzu Application Platform installs and uses a self-signed CA as its ingress issuer for all components. If you don't specify the `ingress_issuer` in `tap-values.yaml`, the `ingress_issuer` key consumes the value `shared.ingress_issuer` from `tap-values.yaml`. This automatically enables TLS validation on Application Live View back end.

To deactivate TLS validation on Application Live View back end do one of the following:

- While VMware discourages it, you can deactivate the ingress issuer by setting `shared.ingress_issuer:` to `""` in `tap-values.yaml`. For example:

  ```
  ...
      shared:
        ingress_issuer: ""
  ```

- Set `ingress_issuer` to `""` in `app-live-view-backend-values.yaml`. For example:

  ```
  appliveview:
    ...
    ingress_issuer: ""
  ```

4. (Optional) View additional changes you can make in your `app-live-view-backend-values.yaml` file by running:

   ```
   tanzu package available get backend.appliveview.tanzu.vmware.com/VERSION-NUMBER
   \
     --values-schema \
     --namespace tap-install
   ```

   Where `VERSION-NUMBER` is the version of the package listed earlier. For example, `1.9.0`.

   For example:

   ```
   $ tanzu package available get backend.appliveview.tanzu.vmware.com/1.9.0 \
       --values-schema \
       --namespace tap-install

     KEY                      DEFAULT          TYPE        DESCRIPTION
     tls.namespace            <nil>            string      The targeted namespace
   for secret consumption by the HTTPProxy.

     tls.secretName           <nil>            string      The name of secret for
   consumption by the HTTPProxy.

     ingressDomain            tap.example.com  string      Domain to be used by th
   e HTTPProxy ingress object. The "appliveview"
                                                          subdomain will be prepe
   nded to the value provided. For example:
                                                          "example.com" would bec
   ome "appliveview.example.com".
     ingressEnabled           false            boolean     Flag for whether or not
   ```

```
to create an HTTPProxy for ingress.

  ingress_issuer                              string      Cluster issuer to be us
ed in App Live View Backend.
  kubernetes_distribution                     string      Kubernetes distribution
that this package is being installed on. Accepted
                                                          values: ['''',''openshi
ft'']
  kubernetes_version                          string      Optional: The Kubernete
s Version. Valid values are '1.24.*', or ''

  server.tls.crt                              string      TLS cert file
  server.tls.enabled       false              boolean     Flag to enable tls on b
ackend
  server.tls.key                              string      TLS key file
```

5. Install the Application Live View back end package by running:

```
tanzu package install appliveview \
  --package backend.appliveview.tanzu.vmware.com \
  --version VERSION-NUMBER \
  --namespace tap-install \
  --values-file app-live-view-backend-values.yaml
```

Where `VERSION-NUMBER` is the version of the package listed earlier. For example, `1.9.0`.

For example:

```
$ tanzu package install appliveview \
    --package backend.appliveview.tanzu.vmware.com \
    --version 1.9.0 \
    --namespace tap-install \
    --values-file app-live-view-backend-values.yaml

- Installing package 'backend.appliveview.tanzu.vmware.com'
| Getting namespace 'tap-install'
| Getting package metadata for 'backend.appliveview.tanzu.vmware.com'
| Creating service account 'appliveview-tap-install-sa'
| Creating cluster admin role 'appliveview-tap-install-cluster-role'
| Creating cluster role binding 'appliveview-tap-install-cluster-rolebinding'
| Creating package resource
| Package install status: Reconciling

Added installed package 'appliveview' in namespace 'tap-install'
```

The Application Live View back end component is deployed in `app-live-view` namespace by default.

6. Verify the Application Live View back end package installation by running:

```
tanzu package installed get appliveview -n tap-install
```

For example:

```
$ tanzu package installed get appliveview -n tap-install

\ Retrieving installation details for appliveview...
NAME:                    appliveview
PACKAGE-NAME:            backend.appliveview.tanzu.vmware.com
PACKAGE-VERSION:         1.9.0
STATUS:                  Reconcile succeeded
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`.

Install Application Live View connector

To install Application Live View connector:

1. List version information for the package by running:

```
tanzu package available list connector.appliveview.tanzu.vmware.com --namespace
tap-install
```

For example:

```
$ tanzu package available list connector.appliveview.tanzu.vmware.com --namespa
ce tap-install

- Retrieving package versions for connector.appliveview.tanzu.vmware.com...
  NAME                                    VERSION        RELEASED-AT
  connector.appliveview.tanzu.vmware.com  1.9.0          2024-03-22T00:00:00Z
```

2. Create the file `app-live-view-connector-values.yaml` using the following details:

Single-cluster environment
For a single-cluster environment, the Application Live View connector connects to the `cluster-local` Application Live View back end to register the applications.

By default, ingress is deactivated for connector. You do not have to change your `app-live-view-connector-values.yaml` file in this step.

Multicluster environment
For a multicluster environment, set the flag `ingressEnabled` to `true` for the Application Live View connector to connect to the Application Live View back end by using the ingress domain. For example:

```
appliveview_connector:
  backend:
    ingressEnabled: true
```

Profile install using shared ingress domain key
If you are using a Tanzu Application Platform profile installation and the top-level key `shared.ingress_domain` is set in the `tap-values.yaml`, the Application Live View connector and Application Live View back end are configured to communicate through ingress. The Application Live View connector then uses the `shared.ingress_domain` to reach the back end.

To override the shared ingress for Application Live View in a multicluster environment, use the following values:

```
appliveview_connector:
  backend:
    host: appliveview.INGRESS-DOMAIN
```

Where `INGRESS-DOMAIN` is the top-level domain the Application Live View back end exposes by using `tanzu-shared-ingress` for the connectors in other clusters to reach the Application Live View back end. Prepend the `appliveview` subdomain to the provided value.

3. Configure TLS in your `app-live-view-connector-values.yaml`. Choose a tab depending on how you configured TLS in Install Application Live View back end earlier.

Configure TLS with self-signed certificate

The `backend.sslDeactivated` is set to `false` by default. Set the CA certificate for the ingress domain in the `backend.caCertData` key for SSL validation as follows:

```
appliveview_connector:
  backend:
    ...
    caCertData: |-
      -----BEGIN CERTIFICATE-----
      MIIGMzCCBBugAwIBAgIJALHHzQjxM6wMMA0GCSqGSIb3DQEBDQUAMGcxCzAJBgNV
      BAgMAk1OMRQwEgYDVQQHDAtNaW5uZWFwb2xpczEPMA0GA1UECgwGVk13YXJlMRMw
      -----END CERTIFICATE-----
```

Configure TLS using ClusterIssuer

To enable TLS using ClusterIssuer:

1. Retrieve the certificate from the HTTPProxy secret by running the following command in the view cluster:

   ```
   kubectl get secret appliveview-cert -n app-live-view -o yaml |  yq '.d
   ata."ca.crt"' | base64 -d
   ```

2. Set the following connector configuration in the run cluster:

   ```
   appliveview_connector:
     backend:
       ingressEnabled: true
       sslDeactivated: false
       host: appliveview.INGRESS-DOMAIN
       caCertData: |-
         -----BEGIN CERTIFICATE-----
         MIIGMzCCBBugAwIBAgIJALHHzQjxM6wMMA0GCSqGSIb3DQEBDQUAMGcxCzAJBgNV
         BAgMAk1OMRQwEgYDVQQHDAtNaW5uZWFwb2xpczEPMA0GA1UECgwGVk13YXJlMRMw
         -----END CERTIFICATE-----
   ```

   Where:

   - `caCertData` is the certificate you retrieved from the HTTPProxy secret exposed by the Application Live View back end in the view cluster.
   - `host` is the backend host in the view cluster.

Deactivate TLS

If TLS is not enabled for the `INGRESS-DOMAIN` in the Application Live View back end, set `backend.sslDeactivated` to `true`. For example:

```
appliveview_connector:
  backend:
    ...
    sslDeactivated: true
```

Note

The `sslDisabled` key is deprecated and has been renamed to `sslDeactivated`.

You can edit the values to suit your project needs or leave the default values as is.

Using the HTTP proxy either on 80 or 443 based on SSL config exposes the back end service running on port 7000. The connector connects to the back end on port 80/443 by default. Therefore, you are not required to explicitly configure the `port` field.

4. (Optional) View additional changes you can make in your `app-live-view-connector-values.yaml` file by running:

```
tanzu package available get connector.appliveview.tanzu.vmware.com/VERSION-NUMB
ER \
  --values-schema \
  --namespace tap-install
```

Where `VERSION-NUMBER` is the version of the package listed earlier. For example, `1.9.0`.

For example:

```
$ tanzu package available get connector.appliveview.tanzu.vmware.com/1.9.0 \
    --values-schema \
    --namespace tap-install

  KEY                                  DEFAULT             TYPE        DESCRIP
TION
  backend.caCertData                     cert-in-pem-format  string    CA Cert
Data for ingress domain
  backend.host                           <nil>               string    Domain
to be used to reach the application live view backend. Prepend
                                                                       "appliv
eview" subdomain to the value if you are using shared ingress. For
                                                                       exampl
e: "example.com" would become "appliveview.example.com".
  backend.ingressEnabled                 false               boolean   Flag fo
r the connector to connect to ingress on backend

  backend.port                           <nil>               number    Port to
reach the application live view backend
  backend.sslDeactivated                 false               boolean   Flag fo
r whether or not to deactivate ssl
  backend.sslDisabled                    false               boolean   The key
sslDisabled has been deprecated in TAP 1.4.0 and will be removed in TAP
                                                                       1.X+Y.0
of TAP, please migrate to the key sslDeactivated
  connector.deployment.enabled           true               boolean   Flag for
the connector to run in deployment mode
  connector.deployment.replicas            1                number    Number
of replicas of connector pods at any given time
  connector.namespace_scoped.enabled     false               boolean   Flag fo
r the connector to run in namespace scope
  connector.namespace_scoped.namespace   default             string    Namespa
ce to deploy connector
  kubernetes_distribution                                    string    Kuberne
tes distribution that this package is being installed on. Accepted
                                                                       values:
['''',''openshift'']
  kubernetes_version                                         string    Optiona
l: The Kubernetes Version. Valid values are '1.24.*', or ''

  activateAppLiveViewSecureAccessControl                     boolean   Optiona
l: Configuration required to enable Secure Access Connection between App
                                                                       Live Vi
ew components
  activateSensitiveOperations                                boolean   Optiona
l: Configuration to allow connector to execute sensitive operations on a
```

```
                                                                    running
application
```

5. By default, the connector is deployed as a Kubernetes Deployment. You can override the default settings to run the connector in DaemonSet mode or a Namespace-scope mode. For more information, see Connector deployment modes in Application Live View.

6. Install the Application Live View connector package by running:

   ```
   tanzu package install appliveview-connector \
     --package connector.appliveview.tanzu.vmware.com \
     --version VERSION-NUMBER \
     --namespace tap-install \
     --values-file app-live-view-connector-values.yaml
   ```

   Where `VERSION-NUMBER` is the version of the package listed earlier. For example, `1.9.0`.

   For example:

   ```
   $ tanzu package install appliveview-connector \
       --package connector.appliveview.tanzu.vmware.com \
       --version 1.9.0 \
       --namespace tap-install \
       --values-file app-live-view-connector-values.yaml

   | Installing package 'connector.appliveview.tanzu.vmware.com'
   | Getting namespace 'tap-install'
   | Getting package metadata for 'connector.appliveview.tanzu.vmware.com'
   | Creating service account 'appliveview-connector-tap-install-sa'
   | Creating cluster admin role 'appliveview-connector-tap-install-cluster-role'
   | Creating cluster role binding 'appliveview-connector-tap-install-cluster-role
   binding'
   - Creating package resource
   / Package install status: Reconciling

   Added installed package 'appliveview-connector' in namespace 'tap-install'
   ```

   The connector is configured to connect to the central instance of the back end. The Application Live View connector component is deployed in `app-live-view-connector` namespace by default.

7. Verify the `Application Live View connector` package installation by running:

   ```
   tanzu package installed get appliveview-connector -n tap-install
   ```

   For example:

   ```
   $ tanzu package installed get appliveview-connector -n tap-install

   | Retrieving installation details for appliveview-connector...
   NAME:                    appliveview-connector
   PACKAGE-NAME:            connector.appliveview.tanzu.vmware.com
   PACKAGE-VERSION:         1.9.0
   STATUS:                  Reconcile succeeded
   CONDITIONS:              [{ReconcileSucceeded True  }]
   USEFUL-ERROR-MESSAGE:
   ```

   Verify that `STATUS` is `Reconcile succeeded`.

Install Application Live View conventions

To install Application Live View conventions:

1. List version information for the package by running:

```
tanzu package available list conventions.appliveview.tanzu.vmware.com --namespa
ce tap-install
```

For example:

```
$ tanzu package available list conventions.appliveview.tanzu.vmware.com --names
pace tap-install

- Retrieving package versions for conventions.appliveview.tanzu.vmware.com...
  NAME                                       VERSION        RELEASED-AT
  conventions.appliveview.tanzu.vmware.com   1.9.0          2024-03-22T00:00:00Z
```

2. (Optional) View the changes you can make to the default installation settings by running:

```
tanzu package available get conventions.appliveview.tanzu.vmware.com/VERSION-NU
MBER \
  --values-schema \
  --namespace tap-install
```

Where `VERSION-NUMBER` is the version of the package listed earlier. For example, `1.9.0`.

For example:

```
$ tanzu package available get conventions.appliveview.tanzu.vmware.com/1.9.0 \
    --values-schema \
    --namespace tap-install

  KEY                                DEFAULT             TYPE     DESCRIPTION
  kubernetes_distribution                                string   Kubernetes dist
ribution that this package is installed on. Accepted values: ['''',''openshif
t''].
  kubernetes_version                                     string   Optional: The K
ubernetes Version. Valid values are '1.24.*', or ''.
```

3. (Optional) Create a file named `app-live-view-conventions-values.yaml` to override the default installation settings using the information output in the previous step.

4. Install the Application Live View conventions package.

Default values

To install Application Live View conventions with the default settings, run:

```
tanzu package install appliveview-conventions \
  --package conventions.appliveview.tanzu.vmware.com \
  --version VERSION-NUMBER \
  --namespace tap-install
```

Where `VERSION-NUMBER` is the version of the package listed earlier. For example, `1.9.0`.

For example:

```
$ tanzu package install appliveview-conventions \
    --package conventions.appliveview.tanzu.vmware.com \
    --version 1.9.0 \
    --namespace tap-install

- Installing package 'conventions.appliveview.tanzu.vmware.com'
| Getting namespace 'tap-install'
| Getting package metadata for 'conventions.appliveview.tanzu.vmware.com'
| Creating service account 'appliveview-conventions-tap-install-sa'
| Creating cluster admin role 'appliveview-conventions-tap-install-cluster-ro
```

```
le'
| Creating cluster role binding 'appliveview-conventions-tap-install-cluster-
rolebinding'
- Creating package resource
\ Package install status: Reconciling

Added installed package 'appliveview-conventions' in namespace 'tap-install'
```

Overriding values

To install Application Live View conventions while overriding the default settings, run:

```
tanzu package install appliveview-conventions \
  --package conventions.appliveview.tanzu.vmware.com \
  --version VERSION-NUMBER \
  --namespace tap-install \
  --values-file app-live-view-conventions-values.yaml
```

Where `VERSION-NUMBER` is the version of the package listed earlier. For example, `1.9.0`.

5. Verify the package install for Application Live View conventions package by running:

   ```
   tanzu package installed get appliveview-conventions -n tap-install
   ```

   For example:

   ```
   $ tanzu package installed get appliveview-conventions -n tap-install

   | Retrieving installation details for appliveview-conventions...
   NAME:                    appliveview-conventions
   PACKAGE-NAME:            conventions.appliveview.tanzu.vmware.com
   PACKAGE-VERSION:         1.9.0
   STATUS:                  Reconcile succeeded
   CONDITIONS:              [{ReconcileSucceeded True  }]
   USEFUL-ERROR-MESSAGE:
   ```

   Verify that `STATUS` is `Reconcile succeeded`.

Install Application Live View APIServer

To install Application Live View APIServer:

1. List version information for the package by running:

   ```
   tanzu package available list apiserver.appliveview.tanzu.vmware.com --namespace
   tap-install
   ```

   For example:

   ```
   $ tanzu package available list apiserver.appliveview.tanzu.vmware.com --namespa
   ce tap-install

   - Retrieving package versions for apiserver.appliveview.tanzu.vmware.com...
     NAME                                    VERSION       RELEASED-AT
     apiserver.appliveview.tanzu.vmware.com  1.9.0         2024-03-22T00:00:00Z
   ```

2. (Optional) View the changes you can make to the default installation settings by running:

   ```
   tanzu package available get apiserver.appliveview.tanzu.vmware.com/VERSION-NUMB
   ER \
   ```

```
  --values-schema \
  --namespace tap-install
```

Where `VERSION-NUMBER` is the version of the package listed earlier. For example, `1.9.0`.

For example:

```
$ tanzu package available get apiserver.appliveview.tanzu.vmware.com/1.9.0 \
    --values-schema \
    --namespace tap-install \

  KEY                               DEFAULT             TYPE     DESCRIPTION
  kubernetes_distribution                               string  Kubernetes dist
ribution that this package is installed on. Accepted values: ['''',''openshif
t''].
  kubernetes_version                                    string  Optional: The K
ubernetes Version. Valid values are '1.24.*', or ''.
```

3. (Optional) Create a file named `app-live-view-apiserver-values.yaml` to override the default installation settings using the information output in the previous step.
4. Install the Application Live View APIServer package.

Default values

To install Application Live View APIServer with the default settings, run:

```
tanzu package install appliveview-apiserver \
  --package apiserver.appliveview.tanzu.vmware.com \
  --version VERSION-NUMBER \
  --namespace tap-install
```

Where `VERSION-NUMBER` is the version of the package listed earlier. For example, `1.9.0`.

For example:

```
$ tanzu package install appliveview-apiserver \
    --package apiserver.appliveview.tanzu.vmware.com \
    --version 1.9.0 \
    --namespace tap-install

- Installing package 'apiserver.appliveview.tanzu.vmware.com'
| Getting namespace 'tap-install'
| Getting package metadata for 'apiserver.appliveview.tanzu.vmware.com'
- Creating package resource
\ Package install status: Reconciling

Added installed package 'appliveview-apiserver' in namespace 'tap-install'
```

Overriding values

To install Application Live View APIServer while overriding the default settings, run:

```
tanzu package install appliveview-apiserver \
  --package apiserver.appliveview.tanzu.vmware.com \
  --version VERSION-NUMBER \
  --namespace tap-install \
  --values-file app-live-view-apiserver-values.yaml
```

Where `VERSION-NUMBER` is the version of the package listed earlier. For example, `1.9.0`.

5. Verify the package install for Application Live View APIServer package by running:

```
tanzu package installed get appliveview-apiserver -n tap-install
```

For example:

```
$ tanzu package installed get appliveview-apiserver -n tap-install

| Retrieving installation details for appliveview-apiserver...
NAME:                    appliveview-apiserver
PACKAGE-NAME:            apiserver.appliveview.tanzu.vmware.com
PACKAGE-VERSION:         1.9.0
STATUS:                  Reconcile succeeded
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`.

The Application Live View UI plug-in is part of Tanzu Developer Portal. To access the Application Live View UI, see Application Live View in Tanzu Application Platform GUI.

Deprecation notice for the sslDisabled key

The `appliveview_connector.backend.sslDisabled` key has been replaced by `appliveview_connector.backend.sslDeactivated`.

Connector deployment modes in Application Live View

This topic tells you about the different ways in which you can deploy the connector in Tanzu Application Platform (commonly known as TAP).

Deploy the connector as a regular deployment

This is the default mode of Application Live View connector in Tanzu Application Platform. When the connector is running as a regular deployment, it observes all the applications across all the namespaces in the cluster and registers with the back end.

Deploy the connector as a Kubernetes DaemonSet

When deployed as a Kubernetes DaemonSet the connector discovers applications across all the namespaces running in a worker node of a Kubernetes cluster.

To run the connector as a DaemonSet:

1. Change the default installation settings for Application Live View connector by running:

   ```
   tanzu package available get connector.appliveview.tanzu.vmware.com/VERSION-NUMB
   ER --values-schema --namespace tap-install
   ```

   Where `VERSION-NUMBER` is the version of the package listed. For example, `1.9.0`.

   For example:

   ```
   $ tanzu package available get connector.appliveview.tanzu.vmware.com/1.9.0 --va
   lues-schema --namespace tap-install
    KEY                                  DEFAULT             TYPE        DESCRIPT
   ION
    kubernetes_version                                       string      Optiona
   l: The Kubernetes Version. Valid values are '1.24.*', or ''.

    backend.sslDeactivated               false               boolean     Flag for
   ```

```
whether to disable SSL.
 backend.caCertData                    cert-in-pem-format  string      CA Cert
Data for ingress domain.
 backend.host                          <nil>               string      Domain u
sed to reach the Application Live View back end. Prepend
                                                                       "applive
view" subdomain to the value if you use shared ingress. For
                                                                       example:
"example.com" becomes "appliveview.example.com".
 backend.ingressEnabled                false               boolean     Flag for
the connector to connect to ingress on back end.

 backend.port                          <nil>               number      Port to
reach the Application Live View back end.
 connector.deployment.enabled          true                boolean      Flag fo
r the connector to run in deployment mode
 connector.deployment.replicas           1                 number      Number o
f replicas of connector pods at any given time
 connector.namespace_scoped.enabled    false               boolean     Flag for
the connector to run in namespace scope.
 connector.namespace_scoped.namespace  default             string      Namespac
e to deploy connector.
 kubernetes_distribution                                   string      Kubernet
es distribution that this package is being installed on. Accepted
                                                                       values:
[''''],''openshift''].
 activateAppLiveViewSecureAccessControl                    boolean     Optiona
l: Configuration required to enable Secure Access Connection between App Live V
iew components.
 activateSensitiveOperations                               boolean     Optiona
l: Configuration to allow connector to execute sensitive operations on a runnin
g application.
```

For more information about values schema options, see the properties listed earlier.

2. Override the deployment mode for the connector by using the following configuration values for the `app-live-view-connector-values.yaml`:

```
appliveview_connector:
  connector:
    deployment:
      enabled: false
      replicas: 1
```

By default, `connector.deployment.enabled` is set to `true`.

The `connector.deployment.replicas` key specifies the number of replicas of connector pods running at any given time.

Deploy the connector in namespace-scoped mode

When the connector is deployed in namespace-scoped mode, it discovers applications within the namespace of a Kubernetes cluster.

To deploy the connector in namespace-scoped mode, use the following configuration values for the connector in `app-live-view-connector-values.yaml`:

```
appliveview_connector:
  connector:
    namespace_scoped:
      enabled: true
      namespace: "NAMESPACE"
```

Where `NAMESPACE` specifies the namespace for deploying the connector

Configure security and access control in Application Live View

This topic tells you how to enable improved security and access control in Application Live View in Tanzu Application Platform (commonly known as TAP). Improved security and access control in Application Live View secures the REST API exposed by the Application Live View back end.

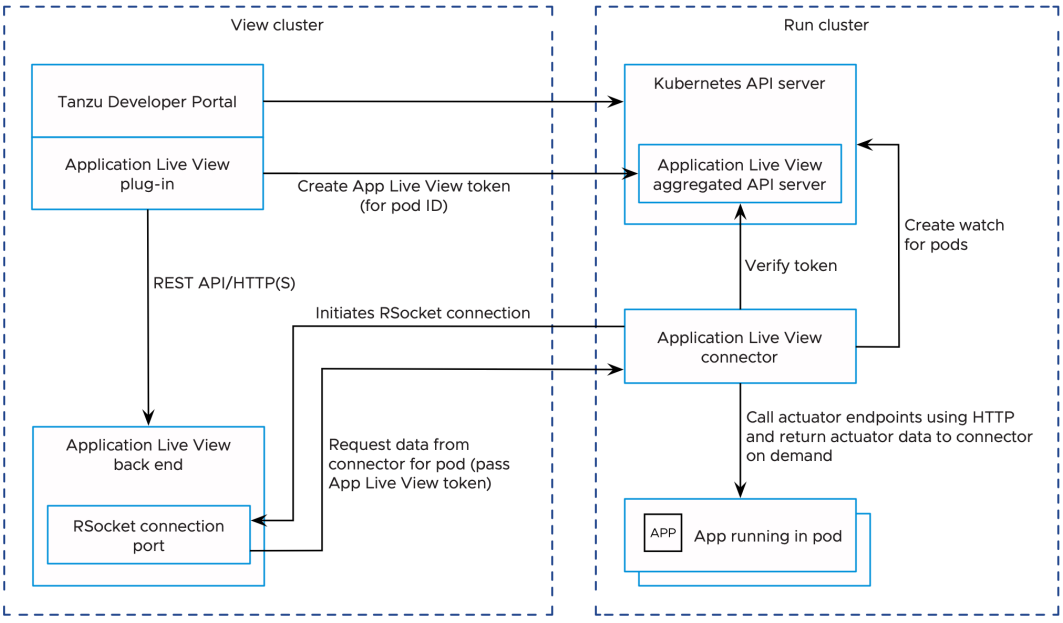
For more information about Application Live View packages, see Install Application Live View.

Security and access control overview

There is one instance of Application Live View back end installed per View profile. Multiple users access this back-end API to fetch actuator data for different applications. All the REST API calls to the back end are secured. A token must be passed to the Application Live View back end on each call to the REST API to fetch actuator data. This token is obtained from Application Live View APIServer.

The Application Live View APIServer generates a unique token upon access validation of a user to a pod. The Application Live View back end passes this token to the Application Live View connector when requesting the actuator data. The Application Live View connector verifies this token by calling the Application Live View APIServer and proxies the actuator data only if the token is valid.

The Application Live View UI plug-in part of Tanzu Developer Portal uses the preceding approach to securely query for the actuator data for a pod. It requests a token from Application Live View APIServer and passes it in the subsequent calls to the back end. This ensures that actuator data from the running application is fetched only if the user is authorized to see the live information for the pod.



The Application Live View UI plug-in relies on Tanzu Developer Portal authentication and authorization to access the Application Live View APIServer and fetch the Application Live View tokens.

The Tanzu Developer Portal controls the access to Kubernetes resources based on user roles and permissions for each of the remote clusters. For more information, see View runtime resources on authorization-enabled clusters.

For more information about how to set up unrestricted remote cluster visibility, see Viewing resources on multiple clusters in Tanzu Developer Portal.

Prerequisites

1. You install the Application Live View APIServer package (`apiserver.appliveview.tanzu.vmware.com`) in Tanzu Application Platform. For more information, see Install Application Live View APIServer.
2. Assign users necessary roles and permissions for the Kubernetes clusters. For more information about managing role-based access control, see Assign roles and permissions on Kubernetes clusters

For example: If you are using Service Account to view resources on a cluster in Tanzu Developer Portal, verify that the `ClusterRole` has rules to access and request tokens from the Application Live View APIServer.

```
- apiGroups: ['appliveview.apps.tanzu.vmware.com']
  resources:
  - resourceinspectiongrants
  verbs: ['get', 'watch', 'list', 'create']
```

For more information, see Set up a Service Account to view resources on a cluster.

Note

With the Service Account approach to view resources on a cluster, every user with the Service Account Token with access to view the pods is able to see the live information in Application Live View.

Configure improved security

The improved security feature is enabled by default for Application Live View.

In a Tanzu Application Platform profile install, the Application Live View connector (`connector.appliveview.tanzu.vmware.com`) and the Tanzu Application Platform GUI (`tap-gui.tanzu.vmware.com`) are automatically configured to enable the secure communication between Application Live View components.

You can control this feature by setting the top-level key `shared.activateAppLiveViewSecureAccessControl` in the `tap-values.yaml`.

For example:

```
shared:
    activateAppLiveViewSecureAccessControl: true
```

To override the security feature at the individual component level, take the following steps.

Application Live View connector

1. (Optional) Change the default installation settings for Application Live View connector by running:

```
tanzu package available get connector.appliveview.tanzu.vmware.com/VERSION-NUMB
ER --values-schema --namespace tap-install
```

Where `VERSION-NUMBER` is the version of the package listed. For example, `1.9.0`.

For example:

```
$ tanzu package available get connector.appliveview.tanzu.vmware.com/1.9.0 --va
lues-schema --namespace tap-install
  KEY                                  DEFAULT             TYPE        DESCRIP
TION
  kubernetes_version                                       string      Optiona
l: The Kubernetes Version. Valid values are '1.24.*', or ''.

  backend.sslDeactivated               false               boolean     Flag fo
r whether to disable SSL.
  backend.caCertData                   cert-in-pem-format  string      CA Cert
Data for ingress domain.
  backend.host                         <nil>               string      Domain
used to reach the Application Live View back end. Prepend
                                                                       "appliv
eview" subdomain to the value if you use shared ingress. For
                                                                       exampl
e: "example.com" becomes "appliveview.example.com".
  backend.ingressEnabled               false               boolean     Flag fo
r the connector to connect to ingress on back end.

  backend.port                         <nil>               number      Port to
reach the Application Live View back end.
  connector.deployment.enabled           true              boolean     Flag for
the connector to run in deployment mode
  connector.deployment.replicas            1               number      Number
of replicas of connector pods at any given time
  connector.namespace_scoped.enabled   false               boolean     Flag fo
r the connector to run in namespace scope.
  connector.namespace_scoped.namespace  default            string      Namespa
ce to deploy connector.
  kubernetes_distribution                                  string      Kuberne
tes distribution that this package is being installed on. Accepted
                                                                       values:
[''''],''openshift''].
  activateAppLiveViewSecureAccessControl                   boolean     Optiona
l: Configuration required to enable Secure Access Connection between App Live V
iew components.
  activateSensitiveOperations                              boolean     Optiona
l: Configuration to allow connector to execute sensitive operations on a runnin
g application.
```

For more information about values schema options, see the properties listed earlier.

2. Create `app-live-view-connector-values.yaml` with the following details:

   To override the default security settings for connector, use the following values:

   ```
   activateAppLiveViewSecureAccessControl: false
   ```

   By default, activateAppLiveViewSecureAccessControl is set to `true`.

   The `activateSensitiveOperations` key activates/deactivates the execution of sensitive operations, such as editing environment variables, downloading heap dump data, and changing log levels for the applications in the cluster. It is set to `false` by default.

   To enable the sensitive operations, set `activateSensitiveOperations` to `true`.

```
activateSensitiveOperations: true
```

To control permissions to execute sensitive operations at a user level, see Authorize a user to execute sensitive operations.

3. Install the Application Live View connector package by running:

```
tanzu package install appliveview-connector \
  --package connector.appliveview.tanzu.vmware.com \
  --version VERSION-NUMBER \
  --namespace tap-install \
  --values-file app-live-view-connector-values.yaml
```

Where `VERSION-NUMBER` is the version of the package listed. For example, `1.9.0`.

For example:

```
$ tanzu package install appliveview-connector \
    --package connector.appliveview.tanzu.vmware.com \
    --version 1.9.0 \
    --namespace tap-install \
    --values-file app-live-view-connector-values.yaml
| Installing package 'connector.appliveview.tanzu.vmware.com'
| Getting namespace 'tap-install'
| Getting package metadata for 'connector.appliveview.tanzu.vmware.com'
| Creating service account 'appliveview-connector-tap-install-sa'
| Creating cluster admin role 'appliveview-connector-tap-install-cluster-role'
| Creating cluster role binding 'appliveview-connector-tap-install-cluster-role
binding'
- Creating package resource
/ Package install status: Reconciling

Added installed package 'appliveview-connector' in namespace 'tap-install'
```

4. Verify the `Application Live View connector` package installation by running:

```
tanzu package installed get appliveview-connector -n tap-install
```

For example:

```
tanzu package installed get appliveview-connector -n tap-install
5s
| Retrieving installation details for appliveview-connector...
NAME:                    appliveview-connector
PACKAGE-NAME:            connector.appliveview.tanzu.vmware.com
PACKAGE-VERSION:         1.9.0
STATUS:                  Reconcile succeeded
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`.

Application Live View UI plug-in

The Application Live View UI plug-in is part of Tanzu Developer Portal. To override the default security settings for the Application Live View UI plug-in, take the following steps.

1. (Optional) Make changes to the default installation settings by running:

```
tanzu package available get tap-gui.tanzu.vmware.com/VERSION-NUMBER --values-sc
hema --namespace tap-install
```

Where `VERSION-NUMBER` is the number you discovered previously. For example, `1.4.6`.

For more information about values schema options, see the individual product documentation.

2. Create `tap-gui-values.yaml` and paste in the following code:

```
ingressEnabled: true
ingressDomain: "INGRESS-DOMAIN"
app_config:
  catalog:
    locations:
      - type: url
        target: https://GIT-CATALOG-URL/catalog-info.yaml
  appLiveView:
    activateAppLiveViewSecureAccessControl: false
```

Where:

- `INGRESS-DOMAIN` is the subdomain for the host name that you point at the `tanzu-shared-ingress` service's External IP address.
- `GIT-CATALOG-URL` is the path to the `catalog-info.yaml` catalog definition file from either the included Blank catalog (provided as an additional download named "Blank Tanzu Developer Portal Catalog") or a Backstage-compliant catalog that you've already built and posted on the Git infrastructure specified in Add Tanzu Developer Portal integrations.

3. Install the package by running:

```
tanzu package install tap-gui \
  --package tap-gui.tanzu.vmware.com \
  --version VERSION \
  --namespace tap-install \
  --values-file tap-gui-values.yaml
```

Where `VERSION` is the version that you want. For example, `1.4.6`.

For example:

```
$ tanzu package install tap-gui \
    --package tap-gui.tanzu.vmware.com \
    --version 1.4.6 \
    --namespace tap-install \
    --values-file tap-gui-values.yaml
- Installing package 'tap-gui.tanzu.vmware.com'
| Getting package metadata for 'tap-gui.tanzu.vmware.com'
| Creating service account 'tap-gui-default-sa'
| Creating cluster admin role 'tap-gui-default-cluster-role'
| Creating cluster role binding 'tap-gui-default-cluster-rolebinding'
| Creating secret 'tap-gui-default-values'
- Creating package resource
- Package install status: Reconciling

 Added installed package 'tap-gui' in namespace 'tap-install'
```

4. Verify that the package installed by running:

```
tanzu package installed get tap-gui -n tap-install
```

For example:

```
$ tanzu package installed get tap-gui -n tap-install
| Retrieving installation details for cc...
NAME:                    tap-gui
PACKAGE-NAME:            tap-gui.tanzu.vmware.com
PACKAGE-VERSION:         1.4.6
STATUS:                  Reconcile succeeded
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`.

5. To access Tanzu Developer Portal, use the service you exposed in the `service_type` field in the values file.

Authorize a user to execute sensitive operations

You can limit the access to execute sensitive operations to specific users. This secures operations such as editing environment variables, downloading heap dump data, and changing log levels for applications.

To authorize a user to execute these sensitive operations:

1. Create a `ClusterRole` with a rule that specifies the `execute` verb for the `SensitiveOperationsAccess` resource and a corresponding `RoleBinding` to bind it to a user:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: alv-execute-sensitive-op-role
rules:
- apiGroups: ['appliveview.apps.tanzu.vmware.com']
  resources:
  - sensitiveoperationsaccesses
  verbs: ['execute']
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: alice-alv-execute-sensitive-op-role-binding
  namespace: dev-team-1
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: alv-execute-sensitive-op-role
subjects:
- kind: User
  name: "alice@example.com"
  apiGroup: rbac.authorization.k8s.io
```

2. If you are using service account to view resources on a cluster in Tanzu Developer Portal, create a `ClusterRole` with a rule that specifies the `execute` verb for the `SensitiveOperationsAccess` resource and a corresponding `RoleBinding` to bind it to the `tap-gui` service account:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: alv-execute-sensitive-op-role
rules:
- apiGroups: ['appliveview.apps.tanzu.vmware.com']
  resources:
  - sensitiveoperationsaccesses
```

```
  verbs: ['execute']
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: alv-execute-sensitive-op-role-binding
  namespace: dev-team-1
subjects:
- kind: ServiceAccount
  namespace: tap-gui
  name: tap-gui-viewer
roleRef:
  kind: ClusterRole
  name: alv-execute-sensitive-op-role
  apiGroup: rbac.authorization.k8s.io
```

3. (Optional) Edit the tap-gui cluster role `k8s-reader` to add a rule that specifies the `execute` verb for the `SensitiveOperationsAccess` resource in a single cluster setup:

```
kubectl edit clusterrole k8s-reader -n tap-gui
```

4. Add the following rule to the cluster role:

```
- apiGroups: ['appliveview.apps.tanzu.vmware.com']
  resources:
  - sensitiveoperationsaccesses
  verbs: ['execute']
```

Note

Executing sensitive operations is deactivated by default. You can enable it at the cluster level by setting the `activateSensitiveOperations: true` key in the appliveview_connector config or at the individual user level by assigning the roles and permissions mentioned earlier.

Enabling Spring Boot apps for Application Live View

This topic for developers tells you how to configure a Spring Boot app for observation by Application Live View within Tanzu Application Platform (commonly known as TAP).

Enable Spring Boot apps

For Application Live View to interact with a Spring Boot app within Tanzu Application Platform, add the `spring-boot-starter-actuator` module dependency.

Add the maven dependency in `pom.xml` as follows:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

Add the following plugin configuration in `pom.xml`:

```
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
  <executions>
```

```
    <execution>
      <goals>
        <goal>build-info</goal>
      </goals>
      <configuration>
        <additionalProperties>
          <spring.boot.version>${project.parent.version}</spring.boot.version>
        </additionalProperties>
      </configuration>
    </execution>
  </executions>
</plugin>
```

Add the preceding configuration to generate `build-info.properties` into your Spring Boot application. This information is then used to display the Spring Boot version that the app uses in Application Live View.

To enable Application Live View for Spring Boot apps, Spring Boot conventions automatically sets the Application Live View labels onto the PodSpec. For more information about the labels automatically set by Spring Boot conventions, see Enable Application Live View for Spring Boot applications.

Enable Spring Boot 3 apps

For Application Live View to interact with a Spring Boot 3 app within Tanzu Application Platform, add the `spring-boot-starter-actuator` module dependency.

Add the maven dependency in `pom.xml` as follows:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

Add the following plugin configuration in `pom.xml`:

```
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
  <executions>
    <execution>
      <goals>
        <goal>build-info</goal>
      </goals>
      <configuration>
        <additionalProperties>
          <spring.boot.version>${project.parent.version}</spring.boot.version>
        </additionalProperties>
      </configuration>
    </execution>
  </executions>
</plugin>
```

Add the preceding configuration to generate `build-info.properties` into your Spring Boot application. This information is then used to display the Spring Boot version that the app uses in Application Live View.

To enable Application Live View for Spring Boot 3 apps, Spring Boot conventions automatically sets the Application Live View labels onto the PodSpec. For more information about the labels automatically set by Spring Boot conventions, see Enable Application Live View for Spring Boot applications.

Here is an example of creating a workload for a Spring Boot 3 Application:

```
tanzu apps workload create spring-boot-3 --git-repo https://github.com/martinlippert/sb3-demo.git --git-branch main --annotation autoscaling.knative.dev/min-scale=1 --yes --label app.kubernetes.io/part-of=tanzu-java-web-app --type web --build-env "BP_JVM_VERSION=17" --label apps.tanzu.vmware.com/auto-configure-actuators="true"
```

Enable Spring Cloud Gateway apps

For Application Live View to interact with a Spring Cloud Gateway app within Tanzu Application Platform, add the `spring-boot-starter-actuator` and `spring-cloud-starter-gateway` module dependency.

Add the maven dependencies in `pom.xml` as follows:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-gateway</artifactId>
</dependency>
```

To enable Application Live View on the Spring Cloud Gateway Tanzu Application Platform workload, Spring Boot conventions automatically applies labels on the workload, such as `tanzu.app.live.view.application.flavours: spring-boot_spring-cloud-gateway` and `tanzu.app.live.view: true`, based on the Spring Cloud Gateway image metadata.

Here is an example of creating a workload for a Spring Cloud Gateway Application:

```
tanzu apps workload create tanzu-scg-web-app --git-repo https://github.com/ksankaranara-vmw/gs-gateway.git --git-branch main --type web --label app.kubernetes.io/part-of=tanzu-scg-web-app --yes --annotation autoscaling.knative.dev/min-scale=1
```

Workload image NOT built with Tanzu Build Service

If your application image is NOT built with Tanzu Build Service, to enable Application Live View on `Spring Boot` Tanzu Application Platform workload, use the following command. For example:

```
tanzu apps workload create boot-app --type web --app boot-app --image <IMAGE NAME> --annotation autoscaling.knative.dev/min-scale=1 --yes --label tanzu.app.live.view=true --label tanzu.app.live.view.application.name=boot-app --label tanzu.app.live.view.application.flavours=spring-boot
```

If your application image is NOT built with Tanzu Build Service, to enable Application Live View on `Spring Cloud Gateway` Tanzu Application Platform workload, use the following command. For example:

```
tanzu apps workload create scg-app --type web --app scg-app --image <IMAGE NAME> --annotation autoscaling.knative.dev/min-scale=1 --yes --label tanzu.app.live.view=true --label tanzu.app.live.view.application.name=scg-app --label tanzu.app.live.view.application.flavours=spring-boot_spring-cloud-gateway
```

If your application image is NOT built with Tanzu Build Service, to enable Application Live View on `Steeltoe` Tanzu Application Platform workload, use the following command. For example:

```
tanzu apps workload create steeltoe-app --type web --app steeltoe-app --image <IMAGE NAME> --annotation autoscaling.knative.dev/min-scale=1 --yes --label tanzu.app.live.vie
```

```
w=true --label tanzu.app.live.view.application.name=steeltoe-app --label tanzu.app.liv
e.view.application.flavours=steeltoe
```

Enabling Spring Boot apps for Application Live View

This topic for developers tells you how to configure a Spring Boot app for observation by Application Live View within Tanzu Application Platform (commonly known as TAP).

Enable Spring Boot apps

For Application Live View to interact with a Spring Boot app within Tanzu Application Platform, add the `spring-boot-starter-actuator` module dependency.

Add the maven dependency in `pom.xml` as follows:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

Add the following plugin configuration in `pom.xml`:

```
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
  <executions>
    <execution>
      <goals>
        <goal>build-info</goal>
      </goals>
      <configuration>
        <additionalProperties>
          <spring.boot.version>${project.parent.version}</spring.boot.version>
        </additionalProperties>
      </configuration>
    </execution>
  </executions>
</plugin>
```

Add the preceding configuration to generate `build-info.properties` into your Spring Boot application. This information is then used to display the Spring Boot version that the app uses in Application Live View.

To enable Application Live View for Spring Boot apps, Spring Boot conventions automatically sets the Application Live View labels onto the PodSpec. For more information about the labels automatically set by Spring Boot conventions, see Enable Application Live View for Spring Boot applications.

Enable Spring Boot 3 apps

For Application Live View to interact with a Spring Boot 3 app within Tanzu Application Platform, add the `spring-boot-starter-actuator` module dependency.

Add the maven dependency in `pom.xml` as follows:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

Add the following plugin configuration in `pom.xml`:

```
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
  <executions>
    <execution>
      <goals>
        <goal>build-info</goal>
      </goals>
      <configuration>
        <additionalProperties>
          <spring.boot.version>${project.parent.version}</spring.boot.version>
        </additionalProperties>
      </configuration>
    </execution>
  </executions>
</plugin>
```

Add the preceding configuration to generate `build-info.properties` into your Spring Boot application. This information is then used to display the Spring Boot version that the app uses in Application Live View.

To enable Application Live View for Spring Boot 3 apps, Spring Boot conventions automatically sets the Application Live View labels onto the PodSpec. For more information about the labels automatically set by Spring Boot conventions, see Enable Application Live View for Spring Boot applications.

Here is an example of creating a workload for a Spring Boot 3 Application:

```
tanzu apps workload create spring-boot-3 --git-repo https://github.com/martinlippert/s
b3-demo.git --git-branch main --annotation autoscaling.knative.dev/min-scale=1 --yes -
-label app.kubernetes.io/part-of=tanzu-java-web-app --type web --build-env "BP_JVM_VER
SION=17" --label apps.tanzu.vmware.com/auto-configure-actuators="true"
```

Enable Spring Cloud Gateway apps

For Application Live View to interact with a Spring Cloud Gateway app within Tanzu Application Platform, add the `spring-boot-starter-actuator` and `spring-cloud-starter-gateway` module dependency.

Add the maven dependencies in `pom.xml` as follows:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-gateway</artifactId>
</dependency>
```

To enable Application Live View on the Spring Cloud Gateway Tanzu Application Platform workload, Spring Boot conventions automatically applies labels on the workload, such as `tanzu.app.live.view.application.flavours: spring-boot_spring-cloud-gateway` and `tanzu.app.live.view: true`, based on the Spring Cloud Gateway image metadata.

Here is an example of creating a workload for a Spring Cloud Gateway Application:

```
tanzu apps workload create tanzu-scg-web-app --git-repo https://github.com/ksankaranar
a-vmw/gs-gateway.git --git-branch main --type web --label app.kubernetes.io/part-of=ta
```

```
nzu-scg-web-app --yes --annotation autoscaling.knative.dev/min-scale=1
```

Workload image NOT built with Tanzu Build Service

If your application image is NOT built with Tanzu Build Service, to enable Application Live View on `Spring Boot` Tanzu Application Platform workload, use the following command. For example:

```
tanzu apps workload create boot-app --type web --app boot-app --image <IMAGE NAME> --a
nnotation autoscaling.knative.dev/min-scale=1 --yes --label tanzu.app.live.view=true -
-label tanzu.app.live.view.application.name=boot-app --label tanzu.app.live.view.appli
cation.flavours=spring-boot
```

If your application image is NOT built with Tanzu Build Service, to enable Application Live View on `Spring Cloud Gateway` Tanzu Application Platform workload, use the following command. For example:

```
tanzu apps workload create scg-app --type web --app scg-app --image <IMAGE NAME> --ann
otation autoscaling.knative.dev/min-scale=1 --yes --label tanzu.app.live.view=true --l
abel tanzu.app.live.view.application.name=scg-app --label tanzu.app.live.view.applicat
ion.flavours=spring-boot_spring-cloud-gateway
```

If your application image is NOT built with Tanzu Build Service, to enable Application Live View on `Steeltoe` Tanzu Application Platform workload, use the following command. For example:

```
tanzu apps workload create steeltoe-app --type web --app steeltoe-app --image <IMAGE N
AME> --annotation autoscaling.knative.dev/min-scale=1 --yes --label tanzu.app.live.vie
w=true --label tanzu.app.live.view.application.name=steeltoe-app --label tanzu.app.liv
e.view.application.flavours=steeltoe
```

Enable Spring Native apps for Application Live View

This topic for developers tells you how to configure a Spring Native app to be observed by Application Live View within Tanzu Application Platform (commonly known as TAP).

Configure a Spring Native application

To make Application Live View interact with a Spring Native app within Tanzu Application Platform:

1. Add the `spring-boot-starter-actuator` module dependency for Maven in `pom.xml` as follows:

   ```
   <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
   </dependency>
   ```

2. Add a native profile that includes the `native-maven-plugin` for the build phase in `pom.xml`, as follows:

   ```
   <profiles>
    <profile>
        <id>native</id>
        <build>
          <plugins>
              <plugin>
                <groupId>org.graalvm.buildtools</groupId>
                  <artifactId>native-maven-plugin</artifactId>
                </plugin>
   ```

```
        </plugins>
    </build>
 </profile>
</profiles>
```

Important

For Maven-based projects, you must add the `org.graalvm.buildtools` build plug-in and configure it to run when the native profile is active.

3. Add the following configuration in `pom.xml` to generate `build-info.properties` in your Spring Boot application. Application Live View uses this information to display the Spring Boot version that the app uses.

```
<plugin>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-maven-plugin</artifactId>
<executions>
 <execution>
   <goals>
     <goal>build-info</goal>
   </goals>
   <configuration>
     <additionalProperties>
       <spring.boot.version>${project.parent.version}</spring.boot.version>
     </additionalProperties>
   </configuration>
 </execution>
</executions>
</plugin>
```

Create a native workload

Create a workload for the Spring Native application similar to the following example:

```
$ tanzu apps workload create spring-cloud-serverless \
--git-repo https://github.com/vudayani-vmw/spring-cloud-serverless --git-branch main \
--type web --label apps.tanzu.vmware.com/auto-configure-actuators=true \
--label app.kubernetes.io/part-of=spring-cloud-serverless --yes \
--annotation autoscaling.knative.dev/min-scale=1 \
--build-env "BP_JVM_VERSION=17" --build-env "BP_NATIVE_IMAGE=true" \
--build-env "BP_MAVEN_BUILD_ARGUMENTS= -Pnative -Dmaven.test.skip=true \
--no-transfer-progress package \
-Dspring-boot.aot.jvmArguments='-Dmanagement.endpoint.health.probes.add-additional-pat
hs='true' \
-Dmanagement.endpoint.health.show-details='always' \
-Dmanagement.endpoints.web.base-path='/actuator' \
-Dmanagement.endpoints.web.exposure.include='*' \
-Dmanagement.health.probes.enabled='true' \
-Dmanagement.server.port=8081 -Dserver.port=8080' "
```

Where:

- The required build arguments, such as `BP_JVM_VERSION`, `BP_NATIVE_IMAGE`, and `BP_MAVEN_BUILD_ARGUMENTS`, are configured to build a native executable file as shown in the example. The build environment variables are required to enable a native build profile and native image-building.
- The override configuration options at runtime provide AOT (Ahead-Of-Time) configuration to the build process by using the `spring-boot.aot.jvmArguments` option. For Gradle builds,

the `spring-boot.aot.jvmArguments` is set as part of the `BP_GRADLE_ADDITIONAL_BUILD_ARGUMENTS` build environment variable.

These build environment parameters do not directly set the runtime values. The build environment parameters, provided as part of `spring-boot.aot.jvmArguments`, are only used as build-time signals. These signals indicate to generate code that allows configuration options to be set at runtime.

Configure at runtime

There are two methods to override the configuration at runtime. You can, by default, let Spring Boot conventions configure automatically, or you can use environment variables to configure manually.

Configure automatically (default)

Automatic configuration by Spring Boot conventions is the default method. Spring Boot conventions add the necessary environment variables and labels to the workload `PodSpec` based on the `apps.tanzu.vmware.com/auto-configure-actuators` flag. When the `apps.tanzu.vmware.com/auto-configure-actuators` is set to `true`, Spring Boot conventions adds the following environment variables to the native workload `PodSpec`:

```
Spec:
  Containers:
    Env:
      Name:   MANAGEMENT_SERVER_PORT
      Value:  8081
      Name:   MANAGEMENT_ENDPOINT_HEALTH_PROBES_ADD_ADDITIONAL_PATHS
      Value:  true
      Name:   MANAGEMENT_ENDPOINT_HEALTH_SHOW_DETAILS
      Value:  always
      Name:   MANAGEMENT_ENDPOINTS_WEB_BASE_PATH
      Value:  /actuator
      Name:   MANAGEMENT_ENDPOINTS_WEB_EXPOSURE_INCLUDE
      Value:  *
      Name:   MANAGEMENT_HEALTH_PROBES_ENABLED
      Value:  true
      Name:   SERVER_PORT
      Value:  8080
      Name:   SERVER_SHUTDOWN_GRACE_PERIOD
      Value:  24s
```

Configure manually

You can manually override runtime configuration settings by providing environment variables. To customize the configuration, set the relevant environment variables in accordance with your requirements. The following example overrides `MANAGEMENT_SERVER_PORT` as `8088`, and it overrides `SERVER_PORT` as `8082`.

```
tanzu apps workload create spring-cloud-serverless \
--git-repo https://github.com/vudayani-vmw/spring-cloud-serverless --git-branch main
--type web \
--label tanzu.app.live.view.application.flavours=spring-boot_spring-native \
--label apps.tanzu.vmware.com/auto-configure-actuators=true --label \
app.kubernetes.io/part-of=spring-cloud-serverless --label apps.tanzu.vmware.com/has-
tests=true \
--yes --annotation autoscaling.knative.dev/min-scale=1 --build-env "BP_JVM_VERSION=1
7" \
--build-env "BP_NATIVE_IMAGE=true" --build-env "BP_MAVEN_BUILD_ARGUMENTS= -Pnative \
-Dmaven.test.skip=true --no-transfer-progress package \
-Dspring-boot.aot.jvmArguments='-Dmanagement.endpoint.health.probes.add-additional-p
aths='true' \
-Dmanagement.endpoint.health.show-details='always' -Dmanagement.endpoints.web.base-p
```

```
ath='/actuator' \
-Dmanagement.endpoints.web.exposure.include='*' -Dmanagement.health.probes.enabled
='true' \
-Dmanagement.server.port=8081 -Dserver.port=8080' " --env MANAGEMENT_SERVER_PORT=808
8 \
--env SERVER_PORT=8082
```

The resulting `PodSpec` is:

```
Spec:
  Containers:
    Env:
      Name:   MANAGEMENT_SERVER_PORT
      Value:  8088
      Name:   SERVER_PORT
      Value:  8082
      Name:   MANAGEMENT_ENDPOINT_HEALTH_PROBES_ADD_ADDITIONAL_PATHS
      Value:  true
      Name:   MANAGEMENT_ENDPOINT_HEALTH_SHOW_DETAILS
      Value:  always
      Name:   MANAGEMENT_ENDPOINTS_WEB_BASE_PATH
      Value:  /actuator
      Name:   MANAGEMENT_ENDPOINTS_WEB_EXPOSURE_INCLUDE
      Value:  *
      Name:   MANAGEMENT_HEALTH_PROBES_ENABLED
      Value:  true
      Name:   SERVER_SHUTDOWN_GRACE_PERIOD
      Value:  24s
```

Spring Boot conventions also set the Application Live View labels on the `PodSpec` to enable Application Live View.

Verify the applied labels and annotations

Verify the applied labels and annotations by running:

```
kubectl get podintents.conventions.carto.run WORKLOAD-NAME -o yaml
```

Where `WORKLOAD-NAME` is the name of the deployed workload. For example, `spring-cloud-serverless`.

Expected output of the Spring Boot workload:

```
...
Labels:       app.kubernetes.io/component=intent
              app.kubernetes.io/part-of=spring-cloud-serverless
              apps.tanzu.vmware.com/auto-configure-actuators=true
              apps.tanzu.vmware.com/has-tests=true
              apps.tanzu.vmware.com/workload-type=web
              carto.run/cluster-template-name=convention-template
              carto.run/resource-name=config-provider
              carto.run/supply-chain-name=source-to-url
              carto.run/template-kind=ClusterConfigTemplate
              carto.run/template-lifecycle=mutable
              carto.run/workload-name=spring-cloud-serverless
              carto.run/workload-namespace=default
              tanzu.app.live.view.application.flavours=spring-boot_spring-native
...
```

Enable Steeltoe apps for Application Live View

This topic for developers tells you how to extend .NET Core Apps to Steeltoe apps and enable Application Live View on Steeltoe workloads within Tanzu Application Platform (commonly known as TAP).

Application Live View supports Steeltoe .NET apps with .NET core runtime version `v6.0.8`.

Extend .NET Core Apps to Steeltoe Apps

A .NET Core application can be extended to a Steeltoe application by adding independent NuGet packages.

To enable the Actuators on a .NET Core App:

1. Add a PackageReference to your `.csproj` file:

   ```
   <PackageReference Include="Steeltoe.Management.EndpointCore" Version="$(Steelto
   eVersion)" />
   ```

   Note

   The PackageReference is expected to change to `Steeltoe.Management.Endpoint` from version Steeltoe 4.0 onwards.

2. Call the extension `AddAllActuators` in your `Program.cs` file:

   ```
   builder.WebHost.AddAllActuators();
   ```

3. (Optional) You can add app-specific configurations, such as the following.

   To expose all management actuator endpoints except `env` endpoint, add the following configuration to your `appsettings.json` file:

   ```
   {
     "Management": {
       "Endpoints": {
         "Actuator":{
           "Exposure": {
             "Include": [ "*" ],
             "Exclude": [ "env" ]
           }
         }
       }
     }
   }
   ```

   To enable logging, add the following configuration to your `appsettings.json` file:

   ```
   {
     "Logging": {
       "LogLevel": {
         "Default": "Information",
         "Microsoft": "Warning",
         "Steeltoe": "Warning",
         "Sample": "Information"
       }
     }
   }
   ```

   To enable heapdump, add the following configuration to your `appsettings.json` file:

```
{
  "Management": {
    "Endpoints": {
      "HeapDump": {
        "HeapDumpType": "Normal"
      }
    }
  }
}
```

Enable Application Live View on Steeltoe Tanzu Application Platform workload

You can enable Application Live View to interact with a Steeltoe app within Tanzu Application Platform.

To enable Application Live View on the Steeltoe Tanzu Application Platform workload, the Application Live View convention service automatically applies labels on the workload, such as `tanzu.app.live.view.application.flavours: steeltoe` and `tanzu.app.live.view: true`, based on the Steeltoe image metadata.

Here's an example of creating a workload for a Steeltoe Application:

```
tanzu apps workload create steeltoe-app --type web --git-repo https://github.com/vmware-tanzu/application-accelerator-samples --sub-path weatherforecast-steeltoe --git-branch main --annotation autoscaling.knative.dev/min-scale=1 --yes --label app.kubernetes.io/part-of=sample-app
```

If your application image is NOT built with Tanzu Build Service, to enable Application Live View on Steeltoe Tanzu Application Platform workload, use the following command. For example:

```
tanzu apps workload create steeltoe-app --type web --app steeltoe-app --image IMAGE-NAME --annotation autoscaling.knative.dev/min-scale=1 --yes --label tanzu.app.live.view=true --label tanzu.app.live.view.application.name=steeltoe-app --label tanzu.app.live.view.application.flavours=steeltoe
```

Where `IMAGE-NAME` is the name of your application image.

Note

Thread metrics is available in SteeltoeVersion `3.2.*`. To enable the Threads page in the Application Live View UI, add the following configuration to your `.csproj` file:

```
<PropertyGroup>
    <SteeltoeVersion>3.2.*</SteeltoeVersion>
</PropertyGroup>
```

Application Live View convention server

This topic gives you information about Application Live View convention, which provides a Webhook handler for Cartographer Conventions.

Role of Application Live View convention

Application Live View conventions works in conjunction with core Cartographer Conventions. It enhances Tanzu PodIntents with metadata such as labels, annotations, or app properties. This

metadata allows Application Live View, specifically the connector, to discover app instances so that Application Live View can access the actuator data from those workloads.

Note

Application Live View conventions now supports only Steeltoe applications. Spring Boot conventions supports both Spring Boot and Spring Cloud Gateway applications. For more information about Spring Boot conventions, see Enable Application Live View with Spring Boot apps

To run Application Live View with Steeltoe apps, the Application Live View convention recognizes PodIntents and adds the following metadata labels:

- `tanzu.app.live.view: "true"`: Enables the connector to observe application pod.
- `tanzu.app.live.view.application.name: APPLICATION-NAME`: Identifies the app name to be used internally by Application Live View.
- `tanzu.app.live.view.application.actuator.port: ACTUATOR-PORT`: Identifies the port on the pod at which the actuators are available for Application Live View.
- `tanzu.app.live.view.application.flavours: steeltoe`: Exposes the framework flavor of the app.

These metadata labels allow Application Live View to identify pods that are enabled for Application Live View. The metadata labels also tell the Application Live View connector what kind of app it is, and on which port the actuators are accessible for Application Live View.

Description of metadata labels

If a workload resource explicitly defines a label under `metadata.labels` in the `workload.yaml`, then the convention service detects the presence of that label and respects its value. When deploying a workload using Tanzu Application Platform, you can override the labels listed in the following table using the `Workload` YAML.

Metadata	Default	Type	Description
`tanzu.app.live.view`	`true`	Label	When deploying a workload in Tanzu Application Platform, this label is set to `true` as default across the supply chain.
`tanzu.app.live.view.application.name`	`steeltoe-app`	Label	When deploying a workload in Tanzu Application Platform, this label is set to `steeltoe-app` if the container image metadata does not contain the app name. Otherwise, the label is set to the app name from container image metadata.
`tanzu.app.live.view.application.flavours`	`steeltoe`	Label	When deploying a Spring Boot workload in Tanzu Application Platform, this label is set to `steeltoe` as default across the supply chain.

Verify the applied labels and annotations

You can verify the applied labels and annotations by running:

```
kubectl get podintents.conventions.carto.run WORKLOAD-NAME -o yaml
```

Where `WORKLOAD-NAME` is the name of the deployed workload, for example `steetoe-app`.

Expected output for Steeltoe workload:

```
apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
  creationTimestamp: "2022-11-14T09:56:53Z"
  generation: 1
  labels:
    app.kubernetes.io/component: intent
    app.kubernetes.io/part-of: sample-app
    apps.tanzu.vmware.com/workload-type: web
    carto.run/cluster-template-name: convention-template
    carto.run/resource-name: config-provider
    carto.run/supply-chain-name: source-to-url
    carto.run/template-kind: ClusterConfigTemplate
    carto.run/workload-name: steeltoe-app
    carto.run/workload-namespace: default
  name: steeltoe-app
  namespace: default
  ownerReferences:
  - apiVersion: carto.run/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: Workload
    name: steeltoe-app
    uid: 97897399-807a-4815-9693-fb06bb4bc1ed
  resourceVersion: "428904"
  uid: 0c74e045-075c-4af3-beef-b092b951be7f
spec:
  serviceAccountName: default
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/min-scale: "1"
        developer.conventions/target-containers: workload
      labels:
        app.kubernetes.io/component: run
        app.kubernetes.io/part-of: sample-app
        apps.tanzu.vmware.com/workload-type: web
        carto.run/workload-name: steeltoe-app
    spec:
      containers:
      - image: dev.registry.tanzu.vmware.com/app-live-view/test/steeltoe-app-default@s
ha256:c8ea14d8714ec31ad978085ebff43d15679613a0c12df37812adf22cb47b5232
        name: workload
        resources: {}
        securityContext:
          runAsUser: 1000
      serviceAccountName: default
status:
  conditions:
  - lastTransitionTime: "2022-11-14T09:56:57Z"
    message: ""
    reason: Applied
    status: "True"
    type: ConventionsApplied
  - lastTransitionTime: "2022-11-14T09:56:57Z"
    message: ""
    reason: ConventionsApplied
    status: "True"
    type: Ready
  observedGeneration: 1
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/min-scale: "1"
        conventions.carto.run/applied-conventions: |-
          spring-boot-convention/auto-configure-actuators-check
```

```
          spring-boot-convention/app-live-view-appflavour-check
          appliveview-sample/app-live-view-connector-steeltoe
          appliveview-sample/app-live-view-appflavours-steeltoe
        developer.conventions/target-containers: workload
      labels:
        app.kubernetes.io/component: run
        app.kubernetes.io/part-of: sample-app
        apps.tanzu.vmware.com/workload-type: web
        carto.run/workload-name: steeltoe-app
        tanzu.app.live.view: "true"
        tanzu.app.live.view.application.flavours: steeltoe
        tanzu.app.live.view.application.name: steeltoe-app
    spec:
      containers:
      - image: dev.registry.tanzu.vmware.com/app-live-view/test/steeltoe-app-default@s
ha256:c8ea14d8714ec31ad978085ebff43d15679613a0c12df37812adf22cb47b5232
        name: workload
        resources: {}
        securityContext:
          runAsUser: 1000
      serviceAccountName: default
```

In your output:

- `status.template.metadata.labels` shows the list of applied labels by Application Live View convention server.
- `status.template.metadata.annotations` shows the list of applied annotations by Application Live View convention server.

Custom configuration for the connector

This topic for developers tells you how to custom configure an app or workload for Application Live View.

The connector component is responsible for discovering the app and registering it with Application Live View. Labels from the app PodSpec are used to discover the app and configure the connector to access the actuator data of the app.

Usually, Application Live View conventions applies the necessary configuration automatically. To deactivate the convention and configure the app and the workload manually, the list of labels in the following table gives you an overview of the options:

Label Name	Mandatory	Type	Default	Significance
`tanzu.app.live.view`	true	Boolean	None	Toggle to activate or deactivate pod discovery
`tanzu.app.live.view.application.name`	true	String	None	Application name
`tanzu.app.live.view.application.port`	false	Integer	`8080`	Application port
`tanzu.app.live.view.application.path`	false	String	`/`	Application context path
`tanzu.app.live.view.application.actuator.port`	false	Integer	`8080`	Application actuator port
`tanzu.app.live.view.application.actuator.path`	false	String	`/actuator`	Actuator context path
`tanzu.app.live.view.application.protocol`	false	http / https	`http`	Protocol scheme

Label Name	Mandatory	Type	Default	Significance
`tanzu.app.live.view.application.actuator.health.port`	false	Integer	`8080`	Health endpoint port
`tanzu.app.live.view.application.flavours`	false	Comma separated string	`spring-boot,spring-cloud-gateway`	Application flavors

You can add connector labels in the app `Workload` or override labels that the convention applies, such as `tanzu.app.live.view` and `tanzu.app.live.view.application.name`. If you do not want Application Live View to observe your app, you can override the existing label `tanzu.app.live.view: "false"`.

Configure the developer workload in Tanzu Application Platform

The following YAML is an example of a Spring PetClinic workload that overrides the connector label to `tanzu.app.live.view: "false"`:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: spring-petclinic
  namespace: default
  labels:
    tanzu.app.live.view: "false"
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/workload-type: web
  annotations:
    autoscaling.knative.dev/minScale: "1"
spec:
  source:
    git:
      ref:
        branch: main
      url: https://github.com/kdvolder/spring-petclinic
```

Deploy the workload

To deploy the workload, run:

```
kapp -y deploy -n default -a workloads -f workloads.yaml
```

Verify the label has propagated through the Supply Chain

To verify the label:

1. Verify that the workload build is successful by ensuring that `SUCCEEDED` is set to `True`:

   ```
   kubectl get builds
   NAME                          IMAGE
   SUCCEEDED
   spring-petclinic-build-1      dev.registry.tanzu.vmware.com/app-live-view/test/s
   pring-petclinic-default@sha256:9db2a8a8e77e9215239431fd8afe3f2ecdf09cce8afac565
   dad7b5f0c5ac0cdf     True
   ```

2. Verify the PodIntent of your workload by ensuring `status.template.metadata.labels` shows the newly added label has propagated through the Supply Chain:

```
kubectl get podintents.conventions.carto.run spring-petclinic -oyaml

status:
  conditions:
  - lastTransitionTime: "2021-12-03T15:14:33Z"
    status: "True"
    type: ConventionsApplied
  - lastTransitionTime: "2021-12-03T15:14:33Z"
    status: "True"
    type: Ready
  observedGeneration: 3
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/minScale: "1"
        boot.spring.io/actuator: http://:8080/actuator
        boot.spring.io/version: 2.5.6
        conventions.carto.run/applied-conventions: |-
          appliveview-sample/app-live-view-connector-boot
          appliveview-sample/app-live-view-appflavours-boot
          appliveview-sample/app-live-view-systemproperties
          spring-boot-convention/spring-boot
          spring-boot-convention/spring-boot-graceful-shutdown
          spring-boot-convention/spring-boot-web
          spring-boot-convention/spring-boot-actuator
          spring-boot-convention/service-intent-mysql
        developer.conventions/target-containers: workload
        kapp.k14s.io/identity: v1;default/carto.run/Workload/spring-petclinic;c
arto.run/v1alpha1
        kapp.k14s.io/original: '{"apiVersion":"carto.run/v1alpha1","kind":"Work
load","metadata":{"annotations":{"autoscaling.knative.dev/minScale":"2"},"label
s":{"app.kubernetes.io/part-of":"tanzu-java-web-app","apps.tanzu.vmware.com/wor
kload-type":"web","kapp.k14s.io/app":"1638455805474051000","kapp.k14s.io/associ
ation":"v1.5a9384bd7b93ca74ef494c4dec2caa4b","tanzu.app.live.view":"false"},"na
me":"spring-petclinic","namespace":"default"},"spec":{"source":{"git":{"ref":
{"branch":"main"},"url":"https://github.com/ksankaranara-vmw/spring-petclini
c"}}}}'
        kapp.k14s.io/original-diff-md5: 58e0494c51d30eb3494f7c9198986bb9
        services.conventions.carto.run/mysql: mysql-connector-java/8.0.27
      labels:
        app.kubernetes.io/component: run
        app.kubernetes.io/part-of: tanzu-java-web-app
        apps.tanzu.vmware.com/workload-type: web
        carto.run/workload-name: spring-petclinic
        conventions.carto.run/framework: spring-boot
        kapp.k14s.io/app: "1638455805474051000"
        kapp.k14s.io/association: v1.5a9384bd7b93ca74ef494c4dec2caa4b
        services.conventions.carto.run/mysql: workload
        tanzu.app.live.view: "false"
        tanzu.app.live.view.application.flavours: spring-boot
        tanzu.app.live.view.application.name: petclinic
```

3. Verify the ConfigMap was created for the app by ensuring `metadata.labels` shows the newly added label has propagated through the Supply Chain:

```
kubectl describe configmap spring-petclinic
Name:         spring-petclinic
Namespace:    default
Labels:       carto.run/cluster-supply-chain-name=source-to-url
              carto.run/cluster-template-name=config-template
              carto.run/resource-name=app-config
              carto.run/template-kind=ClusterConfigTemplate
              carto.run/workload-name=spring-petclinic
              carto.run/workload-namespace=default
Annotations:  <none>
```

```
Data
====
delivery.yml:
----
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: spring-petclinic
  labels:
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/workload-type: web
    kapp.k14s.io/app: "1638455805474051000"
    kapp.k14s.io/association: v1.5a9384bd7b93ca74ef494c4dec2caa4b
    tanzu.app.live.view: "false"
    app.kubernetes.io/component: run
    carto.run/workload-name: spring-petclinic
```

4. Verify the running Knative application pod by ensuring `labels` shows the newly added label on the Knative application pod:

```
kubectl get pods -o yaml spring-petclinic-00002-deployment-77dbb85c65-cf7rn | g
rep labels
    kapp.k14s.io/original: '{"apiVersion":"carto.run/v1alpha1","kind":"Workloa
d","metadata":{"annotations":{"autoscaling.knative.dev/minScale":"1"},"labels":
{"app.kubernetes.io/part-of":"tanzu-java-web-app","apps.tanzu.vmware.com/worklo
ad-type":"web","kapp.k14s.io/app":"1638455805474051000","kapp.k14s.io/associati
on":"v1.5a9384bd7b93ca74ef494c4dec2caa4b","tanzu.app.live.view":"false"},"nam
e":"spring-petclinic","namespace":"default"},"spec":{"source":{"git":{"ref":{"b
ranch":"main"},"url":"https://github.com/ksankaranara-vmw/spring-petclini
c"}}}}'
```

You can add or override the connector in the `Workload` of your Knative app.

Custom configuration for application actuator endpoints

This topic for developers tells you how to configure the Application Live View connector component to access actuator endpoints for custom settings, such as a different base path. By default, the actuator endpoint for an application is exposed on `/actuator`.

The following table describes the actuator configuration scenarios and the associated labels to use, assuming that the app runs on port `8080`:

management.server.base-path	management.server.port	management.endpoints.web.base-path	server.servlet.context.path	Comments	Connector Configuration
None	None	None	None	Actuators endpoints available at `localhost:8080/actuator`	`tanzu.app.live.view.application.actuator.path=actuator`, `tanzu.app.live.view.application.actuator.port=8080`

management.server.base-path	management.server.port	management.endpoints.web.base-path	server.servlet.context.path	Comments	Connector Configuration
`/path`	`8082`	`/`	None	Actuator endpoints available at `localhost:8082/path`	`tanzu.app.live.view.application.actuator.path=path`, `tanzu.app.live.view.application.actuator.port=8082`
`/path`	`8082`	`/manage/actuator`	None	Actuator endpoints available at `localhost:8082/path/manage/actuator`	`tanzu.app.live.view.application.actuator.path=path/manage/actuator`, `tanzu.app.live.view.application.actuator.port=8082`
None	None	`/`	None	Actuators are deactivated to avoid conflicts	None
None	None	`/manage`	None	Actuator endpoints available at `/manage`	`tanzu.app.live.view.application.actuator.path=manage`, `tanzu.app.live.view.application.actuator.port=8080`
`/path`	`8082`	None	None	Actuator endpoints available at `localhost:8082/path/actuator`	`tanzu.app.live.view.application.actuator.path=path/actuator`, `tanzu.app.live.view.application.actuator.port=8082`
`/`	`8082`	None	None	Actuator endpoints available at `localhost:8082/actuator`	`tanzu.app.live.view.application.actuator.path=actuator`, `tanzu.app.live.view.application.actuator.port=8082`

management.server.base-path	management.server.port	management.endpoints.web.base-path	server.servlet.context.path	Comments	Connector Configuration
None	None	None	`/api`	Actuator endpoints available at `localhost:8080/api/actuator`	`tanzu.app.live.view.application.actuator.path=api/actuator`, `tanzu.app.live.view.application.actuator.port=8080`
`/path`	`8082`	None	`/api`	Actuator endpoints available at `localhost:8082/path/actuator`	`tanzu.app.live.view.application.actuator.path=path/actuator`, `tanzu.app.live.view.application.actuator.port=8082`
`/path`	`8082`	`/manage`	`/api`	Actuator endpoints available at `localhost:8082/path/manage`	`tanzu.app.live.view.application.actuator.path=path/manage`, `tanzu.app.live.view.application.actuator.port=8082`
`/path`	None	`/manage`	`/api`	Actuator endpoints available at `localhost:8080/api/manage`	`tanzu.app.live.view.application.actuator.path=api/manage`, `tanzu.app.live.view.application.actuator.port=8080`
`/path`	None	`/`	`/api`	Actuators are deactivated to avoid conflicts	None
`/path`	`8082`	`/`	`/api`	Actuator endpoints available at `localhost:8082/path`	`tanzu.app.live.view.application.actuator.path=path`, `tanzu.app.live.view.application.actuator.port=8082`

management.server.base-path	management.server.port	management.endpoints.web.base-path	server.servlet.context.path	Comments	Connector Configuration
None	None	`/manage`	`/api`	Actuator endpoints available at `localhost:8080/api/manage`	`tanzu.app.live.view.application.actuator.path=api/manage`, `tanzu.app.live.view.application.actuator.port=8080`

Scaling Knative apps in Tanzu Application Platform

This topic tells you how to use Application Live View when scaling Knative apps or developer workloads in Tanzu Application Platform (commonly known as TAP).

Application Live View is focused on monitoring apps for a `live` window and does not apply to any of the apps that are scaled down to zero. The intended behavior for Knative apps is to keep track of revisions to allow you to rollback easily, but also scale all of the unused revision instances down to zero to keep resource consumption low.

You can configure Knative apps to set `autoscaling.knative.dev/minScale` to `1` so that Application Live View can still observe app instance. This ensures that there is at least one instance of the latest revision, while still scaling down the older instances.

You can configure any app in Tanzu Application Platform using the `Workload` resource. To scale a Knative app, add the annotation `autoscaling.knative.dev/minScale` in the `Workload` and set it to the value you want. For Application Live View to observe an app and have at least one instance of the latest revision, set `autoscaling.knative.dev/minScale = "1"`.

The annotations or labels in the `Workload` get propagated through the Tanzu Application Platform supply chain as follows:

Workload > PodIntent > ConfigMap > Push Config > to repository/registry > git-repository/imagerepository picks the Config from repository/registry > kapp-ctrl deploys and knative runs the config > final pod running on the Kubernetes cluster.

Configure the developer workload in Tanzu Application Platform

The following YAML is an example `Workload` that adds the annotation `autoscaling.knative.dev/minScale = "1"` to set the minimum scale for the `spring-petclinic` app:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: spring-petclinic
  namespace: default
  labels:
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/workload-type: web
  annotations:
    autoscaling.knative.dev/minScale: "1"
spec:
  source:
    git:
      ref:
```

```
    branch: main
  url: https://github.com/kdvolder/spring-petclinic
```

Deploy the workload

To deploy the workload, run:

```
kapp -y deploy -n default -a workloads -f workloads.yaml
```

Verify the annotation has propagated through the Supply Chain

To verify the annotation:

1. Verify that the workload build is successful by ensuring that `SUCCEEDED` is set to `True`:

   ```
   kubectl get builds
   NAME                          IMAGE
   SUCCEEDED
   spring-petclinic-build-1      dev.registry.tanzu.vmware.com/app-live-view/test/s
   pring-petclinic-default@sha256:9db2a8a8e77e9215239431fd8afe3f2ecdf09cce8afac565
   dad7b5f0c5ac0cdf     True
   ```

2. Verify the PodIntent of your workload by ensuring `status.template.metadata.annotations` shows the newly added annotation has propagated through the Supply Chain:

   ```
   kubectl get podintents.conventions.carto.run spring-petclinic -oyaml

   status:
     conditions:
     - lastTransitionTime: "2021-12-03T15:14:33Z"
       status: "True"
       type: ConventionsApplied
     - lastTransitionTime: "2021-12-03T15:14:33Z"
       status: "True"
       type: Ready
     observedGeneration: 3
     template:
       metadata:
         annotations:
           autoscaling.knative.dev/minScale: "1"
   ```

3. Verify the ConfigMap was created for the app by ensuring `spec.template.metadata.annotations` shows the newly added annotation has propagated through the Supply Chain:

   ```
   kubectl describe configmap spring-petclinic
   Name:         spring-petclinic
   Namespace:    default
   Labels:       carto.run/cluster-supply-chain-name=source-to-url
                 carto.run/cluster-template-name=config-template
                 carto.run/resource-name=app-config
                 carto.run/template-kind=ClusterConfigTemplate
                 carto.run/workload-name=spring-petclinic
                 carto.run/workload-namespace=default
   Annotations:  <none>

   Data
   ====
   delivery.yml:
   ```

```
---
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: spring-petclinic
  labels:
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/workload-type: web
    kapp.k14s.io/app: "1638455805474051000"
    kapp.k14s.io/association: v1.5a9384bd7b93ca74ef494c4dec2caa4b
    tanzu.app.live.view: "false"
    app.kubernetes.io/component: run
    carto.run/workload-name: spring-petclinic
spec:
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/minScale: "1"
```

4. Verify the running Knative application pod by ensuring `annotations` shows the newly added annotation on the Knative application pod:

```
kubectl get pods -o yaml spring-petclinic-00002-deployment-77dbb85c65-cf7rn | g
rep annotations
  annotations:
    kapp.k14s.io/original: '{"apiVersion":"carto.run/v1alpha1","kind":"Workloa
d","metadata":{"annotations":{"autoscaling.knative.dev/minScale":"1"},"labels":
{"app.kubernetes.io/part-of":"tanzu-java-web-app","apps.tanzu.vmware.com/worklo
ad-type":"web","kapp.k14s.io/app":"1638455805474051000","kapp.k14s.io/associati
on":"v1.5a9384bd7b93ca74ef494c4dec2caa4b","tanzu.app.live.view":"false"},"nam
e":"spring-petclinic","namespace":"default"},"spec":{"source":{"git":{"ref":{"b
ranch":"main"},"url":"https://github.com/ksankaranara-vmw/spring-petclini
c"}}}}'
```

Your Knative app is now set to a minimum scale of one so that Application Live View can observe the instance of the app.

Application Live View on OpenShift

Application Live View must run with a custom SecurityContextConstraint (SCC) to enable compliance with restricted Kubernetes Pod Security Standards on OpenShift. Tanzu Application Platform configures the following SCC for Application Live View back end, Application Live View connector, and Application Live View convention service when you configure the `kubernetes_distribution: openshift` key in the `tap-values.yaml` file.

The following is a `SecurityContextConstraints` specification for Application Live View connector:

```
---
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  name: appliveview-connector-restricted-with-seccomp
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: false
allowPrivilegedContainer: false
allowedCapabilities: null
defaultAddCapabilities: null
fsGroup:
  type: MustRunAs
```

```
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities:
  - ALL
runAsUser:
  type: MustRunAsNonRoot
seLinuxContext:
  type: MustRunAs
supplementalGroups:
  type: RunAsAny
volumes:
  - configMap
  - downwardAPI
  - emptyDir
  - persistentVolumeClaim
  - projected
  - secret
seccompProfiles:
  - runtime/default
```

The preceding `SecurityContextConstraints` specification is applicable to Application Live View back end and Application Live View convention service as well.

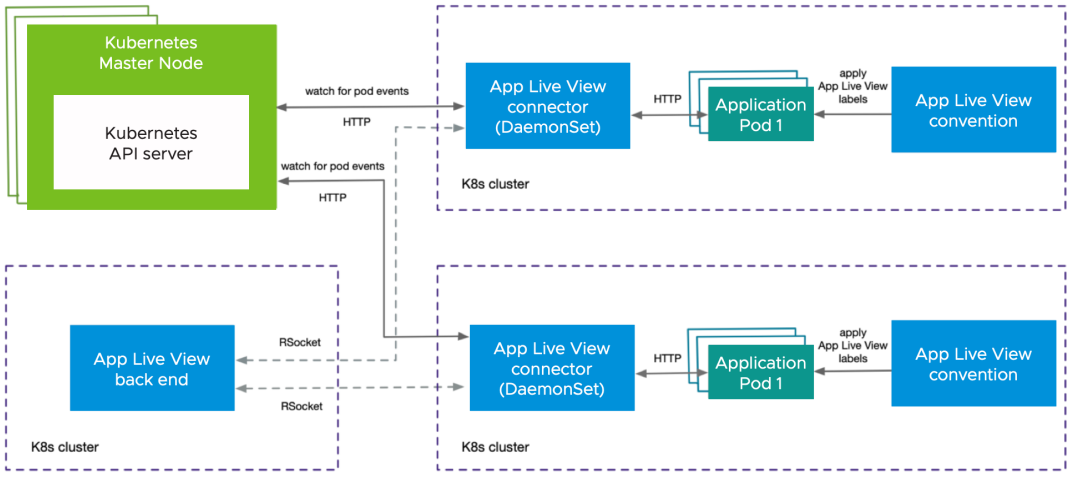
Support for polyglot apps with Application Live View

Application Live View currently supports Spring Boot, Spring Cloud Gateway, and Steeltoe apps.

- To enable Application Live View on Spring Boot and Spring Cloud Gateway apps, see Enable Application Live View for Spring Boot apps.
- To enable Application Live View on Steeltoe apps, see Enable Application Live View for Steeltoe apps.

Application Live View internal architecture

This topic describes the architecture of Application Live View and its components. You can deploy this system on a Kubernetes stack and use it to monitor containerized apps on hosted cloud platforms or on-premises.



Component overview

Application Live View includes the following components as shown in the architecture diagram:

- **Application Live View back end**

  Application Live View back end is the central server component that contains a list of registered apps. It provides a REST API that fetches the actuator data for the applications. The Application Live View UI plug-in, as part of Tanzu Developer Portal, queries this back-end REST API to get live actuator information for the pod.

- **Application Live View connector**

  Application Live View connector is the component responsible for discovering the app pods running on the Kubernetes cluster and registering the instances to the Application Live View back end for it to be observed. The Application Live View connector is also responsible for proxying the actuator queries to the app pods running in the Kubernetes cluster. The actuator data is then displayed in the Application Live View UI plug-in as part of Tanzu Developer Portal.

  You can deploy Application Live View connector in three modes:

  - `Deployment`: Deploy as a Kubernetes Deployment to discover apps running across all namespaces of a Kubernetes cluster. This is the default mode of Application Live View connector.
  - `Cluster access`: Deploy as a Kubernetes DaemonSet to discover apps across all namespaces running in a worker node of a Kubernetes cluster.
  - `Namespace scoped`: Deploy as a Kubernetes Deployment to discover apps running within a namespace across worker nodes of Kubernetes cluster.

- **Application Live View convention server**

  This component provides a webhook handler for the Tanzu convention controller. The webhook handler is registered with Tanzu convention controller. The webhook handler detects supply-chain workloads running a Spring Boot. Such workloads are annotated automatically to enable Application Live View to monitor them. Download and install the Application Live View conventions Webhook component with Tanzu Application Platform.

- **Application Live View APIServer**

  Application Live View APIServer generates a unique token when a user receives access validation to a pod. The Application Live View connector component verifies the token against the Application Live View APIServer before proxying the actuator data from the application. This ensures that the actuator data is secured and only the user who has valid access to view the live information for the pod can retrieve the data.

Design flow

As illustrated in the diagram, the applications run by the user are registered with Application Live View back end by using Application Live View connector. After the application is registered, the Application Live View back end offers the ability to serve actuator data from that registered application through its REST API. Application Live View back end proxies the call to the connector for querying actuator endpoint information.

Application Live View connector, which is a lean model, uses specific labels to discover apps across cluster or namespace. Application Live View connector serves as the connection between running applications and Application Live View back end. Application Live View connector communicates with the Kubernetes API server requesting events for pod creation and termination, and then filters out the events to find the pod of interest by using labels. Then Application Live View connector registers the filtered app instances with Application Live View back end.

Application Live View back end and Application Live View connector communicate through a bidirectional RSocket channel. Application Live View connector is implemented as a Java/Spring

Boot application and runs as a native executable file (Spring Native using GraalVM). Application Live View connector runs as a Deployment by default in the cluster.

Application Live View conventions identifies PodIntents for pods that can serve actuator data and annotates the PodSpec with application-specific labels. Those labels are used by the Application Live View connector to identify running pods that can serve actuator data. Application Live View conventions reads the image metadata to determine the application-specific labels applied on the PodSpec.

Troubleshoot Application Live View

This topic provides information to help you troubleshoot Application Live View.

App is not visible in Application Live View UI

Symptom:

Your app is not visible in the Application Live View UI.

Solution:

The connector component is responsible for discovering the app and registering it with Application Live View.

To troubleshoot, confirm the following:

1. The app must be a Spring Boot Application.
2. Confirm that an instance of a connector is located in the same namespace as your app.

   ```
   kubectl get pods -n NAMESPACE | grep connector
   ```

   Where `NAMESPACE` is the name of the namespace that your app is located in.
3. Confirm that the actuator endpoints are enabled for your app as follows:

   ```
   management.endpoints.web.exposure.include: "*"
   ```
4. Confirm that you have included the following labels within your app deployment YAML file:

   ```
   tanzu.app.live.view="true"
   tanzu.app.live.view.application.name="APP-NAME"
   ```

   Where `APP-NAME` is the name of your app.
5. Confirm that the Convention Service workload YAML file does not contain property `management.endpoints.web.exposure.include` overrides.

See also:

- App is not visible in Application Live View UI with actuator endpoints enabled
- The UI does not show any information for an app with actuator endpoints exposed at root

---

App is not visible in Application Live View UI with actuator endpoints enabled

Symptom:

Your app is not visible in Application Live View UI, but the actuator endpoints are enabled.

Solution:

To troubleshoot:

1. Check the port on which actuator endpoints are configured. By default, they are enabled on the application port. If they are configured on a port different from the application port, set the labels in your app deployment YAML file as follows:

   ```
   tanzu.app.live.view.application.port: "APPLICATION-PORT"
   tanzu.app.live.view.application.actuator.port: "ACTUATOR-PORT"
   ```

   Where:

   - `APPLICATION-PORT` is the application port.
   - `ACTUATOR-PORT` is the actuator port.

2. Check the path on which the app and actuator endpoints are configured. If they are configured on a different paths, set the labels in your app deployment YAML file as follows:

   ```
   tanzu.app.live.view.application.path: "APPLICATION-PATH"
   tanzu.app.live.view.application.actuator.path: "ACTUATOR-PATH"
   ```

   Where:

   - `APPLICATION-PATH` is the application port.
   - `ACTUATOR-PATH` is the actuator port.

The UI does not show any information for an app with actuator endpoints exposed at root

Symptom:

Your app has actuator endpoints exposed at root and the UI does not show any information.

Cause:

Application Live View cannot display the app details when the app is exposing the actuator endpoint on root (`/`) . This is due to conflict in the actuator context path and app default context path.

No information shown on the Health page

Symptom:

The app shows up in Application Live View UI, but the **Health** page does not show details of health.

Solution:

The information exposed by the health endpoint depends on the `management.endpoint.health.show-details` property. This must be set to `always` as as follows:

```
management.endpoint.health.show-details: "always"
```

Stale information in Application Live View

Symptom:

You can find your app in the UI, but it is an old instance that no longer exists while the new instance doesn't show up yet.

Solution:

To troubleshoot:

1. View the Application Live View connector pod logs to see if the connector is sending updates to the back end.
2. Delete the connector pod to recreate it by running:

   ```
   kubectl -n app-live-view-connector delete pods -l=name=application-live-view-connector
   ```

Unable to find CertificateRequests in Application Live View convention

Symptom:

The certificate request is missing for certificate `app-live-view-conventions/appliveview-webhook-cert`.

Solution:

To troubleshoot:

1. Run `kubectl get certificaterequest -A` to see if the certificate request is missing for Application Live View convention.
2. Delete the secret `appliveview-webhook-cert` that corresponds to the certificate in the `app-live-view-conventions` namespace by running:

   ```
   kubectl delete secret appliveview-webhook-cert -n app-live-view-conventions
   ```

   This recreates the certificate request and updates the corresponding certificate.

No live information for pod with ID

Symptom:

In Tanzu Developer Portal, you receive the error `No live information for pod with id`.

Cause:

This might happen because of stale information in Application Live View because it is an old instance that no longer exists while the new instance doesn't show up yet.

Solution:

The workaround is to delete the connector pod so it is re-created by running:

```
kubectl -n app-live-view-connector delete pods -l=name=application-live-view-connector
```

Cannot override the actuator path in the labels

Symptom:

You are unable to override the actuator path in the labels as part of the workload deployment.

Cause:

The changes to add or override the labels or annotations in the `Workload` are in progress. The changes from the `Workload` must be propagated up through the supply chain for the PodIntent to

see the new changes.

Cannot configure SSL in appliveview-connector

Symptom:

This might be because `sslDeactivated` flag in the values YAML file does not accept values without quotes.

Cause:

The `sslDeactivated` Boolean flag is treated as a string in the Kubernetes YAML file.

Solution:

You must specify the value within double quotation marks for the configuration to be picked up.

Verify the labels in your workload YAML file

To verify that the labels in your workload YAML file are working:

1. Verify the app live view convention webhook is running properly by running:

   ```
   kubectl get pods -n app-live-view | grep webhook
   ```

2. Verify the conventions controller is running properly by running:

   ```
   kubectl get pods -n conventions-system
   ```

3. Verify that the conventions are applied properly to the PodSpec by running:

   ```
   kubectl get podintents.conventions.carto.run WORKLOAD-NAME -oyaml
   ```

   Where `WORKLOAD-NAME` is the name of your workload.

   If everything works correctly, the status will contain a transformed template that includes the labels added as part of your workload YAML file. For example:

   ```
   status:
   conditions:
   - lastTransitionTime: "2021-10-26T11:26:35Z"
     status: "True"
     type: ConventionsApplied
   - lastTransitionTime: "2021-10-26T11:26:35Z"
     status: "True"
     type: Ready
   observedGeneration: 1
   template:
     metadata:
       annotations:
         conventions.carto.run/applied-conventions: |-
           appliveview-sample/app-live-view-connector
           appliveview-sample/app-live-view-appflavours
           appliveview-sample/app-live-view-systemproperties
       labels:
         tanzu.app.live.view: "true"
         tanzu.app.live.view.application.flavours: spring-boot
         tanzu.app.live.view.application.name: petclinic
     spec:
       containers:
       - env:
         - name: JAVA_TOOL_OPTIONS
   ```

```
        value: -Dmanagement.endpoint.health.show-details=always -Dmanagement.en
dpoints.web.exposure.include=*
    image: index.docker.io/kdvolder/alv-spring-petclinic:latest@sha256:1aa7bd22
8137471ea38ce36cbf5ffcd629eabeb8ce047f5533b7b9176ff51f98
    name: workload
    resources: {}
```

Override labels set by the Application Live View convention service

It is not possible to override the labels set by the Application Live View convention service for the workload deployment in Tanzu Application Platform. The labels `tanzu.app.live.view`, `tanzu.app.live.view.application.flavours` and `tanzu.app.live.view.application.name` cannot be overridden. The default values set by the Application Live View convention server are used.

However, if you want to override `management.endpoints.web.exposure.include` or `management.endpoint.health.show-details`, you can override these environment properties in `application.properties` or `application.yml` in the Spring Boot Application before deploying the workload in Tanzu Application Platform. Environment properties updated in your app take precedence over the default values set by Application Live View convention server.

Configure labels when management.endpoints.web.base-path and management.server.port are set

If the custom actuator context path is configured as follows:

```
management.endpoints.web.base-path=/manage
management.server.port=8085
```

Configure the connector as follows:

```
tanzu.app.live.view.application.actuator.path=/manage   (manage is the custom actuator
path set on the application)
tanzu.app.live.view.application.actuator.port=8085   (8085 is the custom management se
rver port set on the application)
```

Uninstall Application Live View

This topic tells you how to uninstall Application Live View from Tanzu Application Platform (commonly known as TAP).

To uninstall the Application Live View back end, Application Live View connector, and Application Live View convention server, run:

```
tanzu package installed delete appliveview -n tap-install
tanzu package installed delete appliveview-connector -n tap-install
tanzu package installed delete appliveview-conventions -n tap-install
```

Overview of Application Single Sign-On for VMware Tanzu® 5.0

Application Single Sign-On for VMware Tanzu® (AppSSO) provides APIs for curating and consuming a “Single Sign-On as a service” offering on Tanzu Application Platform.

To get started with Application Single Sign-On, see Get started with Application Single Sign-On.

With AppSSO, Service Operators can configure and deploy authorization servers. Application Operators can then secure their Workloads with these authorization servers to provide Single Sign-On to their end-users.

AppSSO allows integrating authentication and authorization decisions early in the software development and release life cycle. It provides a seamless transition for workloads from development to production when including Single Sign-On solutions in your software.

It's easy to get started with AppSSO, deploy an authorization server with static test users, and eventually progress to multiple authorization servers of production-grade scale with token key rotation, multiple upstream identity providers, configured secure storage, and client restrictions.

AppSSO's authorization server is based on the Spring Authorization Server project. For more information, see Spring documentation.

Document organization

The Application Single Sign-On component documentation consists of the following subsections that relate to what you want to achieve:

- Get started: To understand the basics of getting started with Application Single Sign-On.
- How-to guides: To find a set of steps to solve a specific problem acting as a certain user persona.
- Concepts: To gain a deeper understanding of Application Single Sign-On.
- Reference: To find specific information such as Application Single Sign-On's APIs.

Tutorials and concepts are of most relevance when studying, while how-to guides and reference material are of most use while working.

The following is a selection of useful topics on offer:

For application developers:

- Application Single Sign-On for App Operators

For service operators:

- Application Single Sign-On for Service Operators

For platform operators:

- Application Single Sign-On for Platform Operators

For everyone:

- Levels of consumption for Application Single Sign-On

Overview of Application Single Sign-On for VMware Tanzu® 5.0

Application Single Sign-On for VMware Tanzu® (AppSSO) provides APIs for curating and consuming a "Single Sign-On as a service" offering on Tanzu Application Platform.

To get started with Application Single Sign-On, see Get started with Application Single Sign-On.

With AppSSO, Service Operators can configure and deploy authorization servers. Application Operators can then secure their Workloads with these authorization servers to provide Single Sign-On to their end-users.

AppSSO allows integrating authentication and authorization decisions early in the software development and release life cycle. It provides a seamless transition for workloads from development to production when including Single Sign-On solutions in your software.

It's easy to get started with AppSSO, deploy an authorization server with static test users, and eventually progress to multiple authorization servers of production-grade scale with token key rotation, multiple upstream identity providers, configured secure storage, and client restrictions.

AppSSO's authorization server is based on the Spring Authorization Server project. For more information, see Spring documentation.

Document organization

The Application Single Sign-On component documentation consists of the following subsections that relate to what you want to achieve:

- Get started: To understand the basics of getting started with Application Single Sign-On.
- How-to guides: To find a set of steps to solve a specific problem acting as a certain user persona.
- Concepts: To gain a deeper understanding of Application Single Sign-On.
- Reference: To find specific information such as Application Single Sign-On's APIs.

Tutorials and concepts are of most relevance when studying, while how-to guides and reference material are of most use while working.

The following is a selection of useful topics on offer:

For application developers:

- Application Single Sign-On for App Operators

For service operators:

- Application Single Sign-On for Service Operators

For platform operators:

- Application Single Sign-On for Platform Operators

For everyone:

- Levels of consumption for Application Single Sign-On

Application Single Sign-On concepts

This topic tells you about the following concepts about Application Single Sign-On (commonly called AppSSO):

- The three levels of AppSSO consumption
- Grant types
- Token signatures

Levels of consumption for Application Single Sign-On

This topic tells you about the three levels of consuming Application Single Sign-On (commonly called AppSSO) services and explains the when and why for selecting a specific level over another.

VMware recommends using `ClassClaim` to consume an Application Single Sign-On service. However, there might be situations where the lower level `WorkloadRegistration` or `ClientRegistration` are a better fit.

At its core, the process of consuming Application Single Sign-On involves obtaining client credentials for an authorization server and loading them into a running workload. This process consists of the following steps:

1. Define your environment-independent OAuth2 client configurations, for example, client authentication method, scopes, and so on.
2. Define your OAuth2 client's redirect URIs.
3. Specify the authorization server that you want credentials for.
4. Create a resource that expresses your configuration.
5. Mount the client credentials into a workload.

Each of the following levels gradually takes away some of these steps by distributing them across APIs. As a result, each persona becomes responsible only for the tasks within their domain:

- **Platform operators** manage the installations of Tanzu Application Platform and Application Single Sign-On.
- **Service operators** curate and manage Application Single Sign-On services.
- **Application operators** consume Application Single Sign-On services from their workloads.

Level 1: ClientRegistration

The lowest-level and most general client API Application Single Sign-On offers is `ClientRegistration`. It holds all relevant OAuth2 client configurations. However, it requires fully qualified redirect URIs and targets its host `AuthServer` by using a label selector.

A hypothetical, fully-configured `ClientRegistration` is provided as follows:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClientRegistration
metadata:
  name: my-clientregistration
  namespace: my-namespace
spec:
  authServerSelector:
    matchLabels:
      sso.apps.tanzu.vmware.com/env: staging
      sso.apps.tanzu.vmware.com/ldap: ""
  redirectURIs:
    - https://profile.shop.staging.example.com/login
    - https://profile.shop.example.com/login
    - http://profile.shop.dev.example.com/login
  scopes:
    - name: openid
    - name: email
    - name: profile
    - name: roles
    - name: coffee.make
      description: bestows the ultimate power
  authorizationGrantTypes:
    - client_credentials
    - authorization_code
    - refresh_token
  clientAuthenticationMethod: client_secret_basic
  requireUserConsent: true
```

To specify redirect URIs for an application running on Tanzu Application Platform, you must know its scheme and FQDN in advance. For example, your redirect URI must be `https://profile.shop.example.com/login`. However, the various components of a redirect URI are controlled by multiple personas such as platform operators and application operators.

In most cases, platform operators control how FQDNs are templated and whether TLS is used. For the redirect URI, platform operators have full control over all the elements in

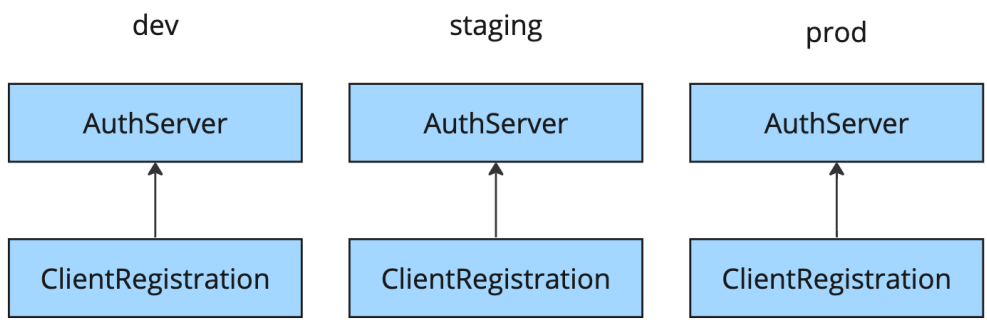
`https://profile.shop.example.com`, which include TLS, the domain name template, and the top-level ingress domain. These configurations can vary across different environments. Therefore, in an alternative environment, setting `https://profile.shop.staging.example.com` can be the appropriate choice.

Application operators control the application's code and its paths. Specifically, they are responsible for managing the `/login` path within the redirect URI. This path is unlikely to change and remains the same regardless of the target environment.

As a result, in a given environment, application operators might not know the FQDN and scheme. In such cases, they must seek assistance from platform operators to obtain this information. On the other hand, platform operators aim to change settings without being coupled to the application operator's configuration.

A `ClientRegistration` must uniquely identify an `AuthServer` by using a label selector. Service operators are in charge of managing `AuthServer`. The labels for a resource are not required to be consistent across environments. However, this can create complications for application operators. Label selectors are considered an advanced concept, and application operators might not be familiar with the specific labels associated with their desired `AuthServers`. It might be worth considering restricting the visibility of `AuthServers` to application operators because it falls within the domain of service operators.

All these factors make it challenging for application operators to use the same `ClientRegistration` across different environments.



In conclusion, although `ClientRegistration` offers flexibility, it is also complex and not easily transferable between different environments. It combines the responsibilities of multiple personas, making it a less straightforward solution.

Level 2: WorkloadRegistration

A higher-level abstraction over `ClientRegistration` is `WorkloadRegistration`. It is similar to `ClientRegistration` except for one major difference: it templates redirect URIs.

Instead of providing full redirect URIs, a `WorkloadRegistration` receives absolute redirect paths. Platform operators configure the template for redirect URIs while installing Tanzu Application Platform, which includes Application Single Sign-On. They configure this template to match the template for workload domains.

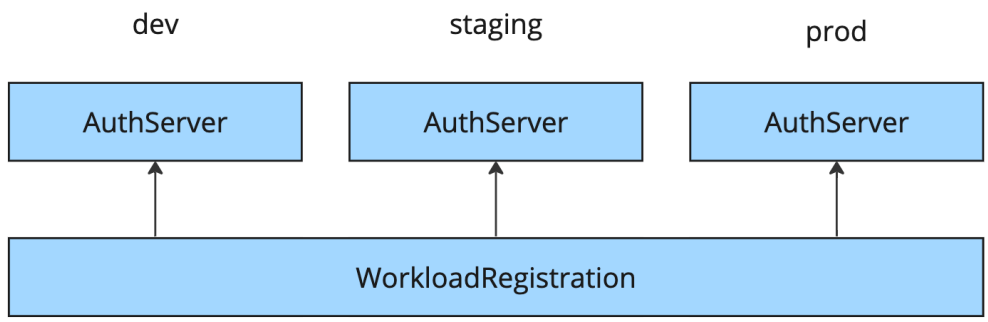
The following is a hypothetical `WorkloadRegistration` example. It is similar to the earlier `ClientRegistration` example, except that it specifies the redirect paths. In its truncated `status`, you can observe the templated redirect URIs.

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: WorkloadRegistration
metadata:
  name: my-workloadregistration
  namespace: my-namespace
spec:
  authServerSelector:
    matchLabels:
      sso.apps.tanzu.vmware.com/env: staging
      sso.apps.tanzu.vmware.com/ldap: ""
  redirectPaths:
    - /login
  workloadRef:
    name: my-workload
    namespace: my-namespace
  scopes:
    - name: openid
    - name: email
    - name: profile
    - name: roles
    - name: coffee.make
      description: bestows the ultimate power
  authorizationGrantTypes:
    - client_credentials
    - authorization_code
    - refresh_token
  clientAuthenticationMethod: client_secret_basic
  requireUserConsent: true
status:
  workloadDomainTemplate: "{{.Name}}.{{.Namespace}}.{{.Domain}}"
  redirectURIs:
    - https://my-workload.my-namespace.example.com/login
```

The additional `spec.workloadRef` provides templates for the redirect URIs.

Templating redirect URIs template decouples the application operators from the platform operators. Now the application operator only needs to provide the absolute redirect paths, which are consistent across environments. The platform operators can configure domain templates, ingress domains, and TLS as they see fit, and rest assured that settings are updated without interruption.

However, `WorkloadRegistration` still requires matching an `AuthServer` by the label selector. That means application operators and service operators are still coupled.



In summary, `WorkloadRegistration` is less flexible, but it is portable across environments when redirect URIs can be templated. However, it still mixes the concerns of personas.

Level 3: ClassClaim (recommended)

The final level is to obtain client credentials by claiming them from an Application Single Sign-On service. Unlike the previous levels that directly interacted with `AuthServer` resources, this level abstracts this part away with Services Toolkit's APIs. This eliminates the last remaining coupling between application operators and service operators.

You can pair Application Single Sign-On's `ClusterWorkloadRegistrationClass` with an `AuthServer` as a claimable service. These two APIs allow the service operators to manage the entire life cycle of an Application Single Sign-On service offering.

A `ClusterWorkloadRegistrationClass` exposes an `AuthServer` as a claimable service by creating a Services Toolkit `ClusterInstanceClass` and defining a blueprint `WorkloadRegistration`. This blueprint allows service operators to record the correct label selector for the `AuthServer`, which eliminates application operators' concerns.

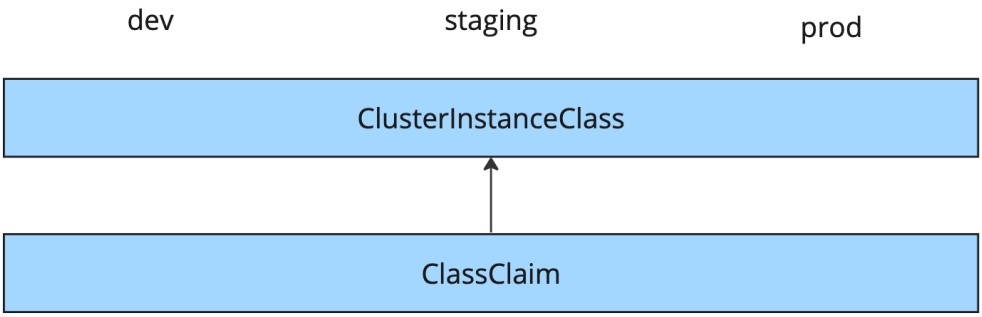
Credentials for a service are requested by using Services Toolkit's general-purpose `ClassClaim` API. A `ClassClaim` identifies a `ClusterInstanceClass` and it carries parameters that further describe the request. For Application Single Sign-On services, the parameters are essentially the trimmed `spec` of a `WorkloadRegistration`.

With the Tanzu Service CLI, application operators can discover and consume services in a self-service style. Commonly, this is how service operators provide all the services required for application teams to run their applications. This includes databases, queues, in-memory stores, and single sign-on by Application Single Sign-On.

The following is a hypothetical `ClassClaim` for an Application Single Sign-On service called `sso`:

```
---
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ClassClaim
metadata:
  name: my-client-credentials
  namespace: my-namespace
spec:
  classRef:
    name: sso
  parameters:
    workloadRef:
      name: sample-workload
    redirectPaths:
      - /login
    scopes:
      - name: openid
      - name: email
      - name: profile
      - name: roles
      - name: coffee.make
        description: bestows the ultimate power
    authorizationGrantTypes:
      - client_credentials
      - authorization_code
      - refresh_token
    clientAuthenticationMethod: client_secret_basic
    requireUserConsent: true
```

This level completely decouples all three personas by providing them with APIs to fulfill their jobs.



In summary, `ClassClaim` is less flexible but it is portable across environments when the redirect URIs can be templated. It completely decouples the concerns of personas. Additionally, offering a single resource for application operators to manage.

Summary

As an application operator, if your workload and its associated resources must be deployed across multiple environments, `ClassClaim` offers you the highest degree of portability, as long as the redirect URIs of your workload can be templated. In this case, consuming an Application Single Sign-On service only requires a single resource, a `ClassClaim`.

If you require control over the template for your workload's redirect URIs, `WorkloadRegistration` offers the desired flexibility. However, this flexibility comes with the cost of matching an `AuthServer` with a label selector. In situations where you want to consume the Application Single Sign-On service with such a setup, it requires multiple resources: a `WorkloadRegistration` and a `ResourceClaim`.

If your workload's redirect URIs cannot be templated and portability is not a concern, `ClientRegistration` offers the necessary flexibility. However, this flexibility comes at the cost of having to match an `AuthServer` with a label selector. Depending on your setup, consuming AppSSO requires multiple resources: a `ClientRegistration` and a `ResourceClaim`.

In conclusion, VMware recommend to use `ClassClaim` whenever possible.

Configure grant types

This topic tells you how to configure grant types for Application Single Sign-On (commonly called AppSSO).

Apps use grant types or flows to get an access token on behalf of a user. If not included, the default grant type is `['client_credentials']`. You must include these grant types in the `authorizationGrantTypes` property list in the Client Registration.

To register a client/application, apply the `yaml` with your specifications to your cluster `kubectl apply -f <path-to-your-yaml>`.

Topics

- Client Credentials Grant
- Authorization Code Grant

Client Credentials Grant Type

This grant type allows an application to get an access token for resources about the client itself, rather than a user.

Dynamic Client Registration (via `ClientRegistration` custom resource):

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClientRegistration
metadata:
  name: <your client name>
spec:
  authorizationGrantTypes:
    - client_credentials
  # ...
```

Note

Ensure that you are able to retrieve a token through your setup

1. Apply your ClientRegistration

   ```
   kubectl apply -f <path-to-the-clientregistration-yaml>
   ```

2. Verify your `ClientRegistration` was created

   ```
   kubectl get clientregistrations
   ```

   –> you should see a `ClientRegistration` with the name you provided

3. Verify your Secret was created

   ```
   kubectl get secrets
   ```

   –> you should see a Secret with that same name you provided for the `ClientRegistration`

4. Get the client secret and decode it

   ```
   kubectl get secret <your-client-registration-name> -o jsonpath="{.data.client-secret}" | base64 -d
   ```

5. Get the client id (or get it from your configuration)

   ```
   kubectl get secret <your-client-registration-name> -o jsonpath="{.data.client-id}" | base64 -d
   ```

6. Request token

   ```
   curl -X POST <AUTH-DOMAIN>/oauth2/token \
    -d "grant_type=client_credentials" \
    -u "YOUR_CLIENT_ID:DECODED_CLIENT_SECRET"
   ```

Authorization Code Grant Type

This grant type allows clients to exchange this code for access tokens.

Dynamic Client Registration (via `ClientRegistration` custom resource):

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClientRegistration
metadata:
  name: <your client name>
```

```
spec:
  authorizationGrantTypes:
    - authorization_code
  scopes:
     - openid
  # ...
```

Note

Ensure that you are able to retrieve a token through your setup

Ensure there is an Identity Provider configured

1. Get your authserver's label name

   ```
   kubectl get authserver sso4k8s -o jsonpath="{.metadata.labels.name}"
   ```

2. Apply this sample ClientRegistration (read more about ClientRegistrations

   The following is an example ClientRegistration that will work in this setup. The required scopes are `openid, email, profile, roles`. The redirect URI here has been set to match that of `oauth2-proxy`.

   ```
   apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
   kind: ClientRegistration
   metadata:
    name: oauth2-proxy-client
    namespace: <your-namespace>
   spec:
    authServerSelector:
    matchLabels:
      name: <your-authserver-label-name>
    authorizationGrantTypes:
      - client_credentials
      - authorization_code
    requireUserConsent: false
    redirectURIs:
      - http://127.0.0.1:4180/oauth2/callback
    scopes:
      - name: openid
      - name: email
      - name: profile
      - name: roles
   ```

   ```
   kubectl apply -f <path-to-the-clientregistration-yaml>
   ```

3. Verify your `ClientRegistration` was created

   ```
   kubectl get clientregistrations
   ```

   –> you should see a `ClientRegistration` with the name you provided

4. Verify your Secret was created

   ```
   kubectl get secrets
   ```

   –> you should see a Secret with that same name you provided for the `ClientRegistration`

5. Get the client secret and decode it

```
CLIENT_SECRET=$(kubectl get secret <your-client-registration-name> -o jsonpath
="{.data.client-secret}" | base64 -d)
```

6. Get the client id (or get it from your configuration)

```
CLIENT_ID=$(kubectl get secret <your-client-registration-name> -o jsonpath="{.d
ata.client-id}" | base64 -d)
```

7. Get the issuer uri

```
ISSUER_URI=$(kubectl get secret <your-client-registration-name> -o jsonpath="{.
data.issuer-uri}" | base64 -d)
```

8. Use the oauth2-proxy to spin up a quick trial run of the configured Authserver and run it with docker.

```
docker run -p 4180:4180 --name oauth2-proxy bitnami/oauth2-proxy:latest \
--oidc-issuer-url "$ISSUER_URI" \
--client-id "$CLIENT_ID" \
--insecure-oidc-skip-issuer-verification true \
--client-secret "$CLIENT_SECRET" \
--cookie-secret "0000000000000000" \
--http-address "http://:4180" \
--provider oidc \
--scope "openid email profile roles" \
--email-domain='*' \
--insecure-oidc-allow-unverified-email true \
--upstream "static://202" \
--oidc-groups-claim "roles" \
--oidc-email-claim "sub" \
--redirect-url "http://127.0.0.1:4180/oauth2/callback"
```

> **Note**
>
> Ensure that your issuer URL does not resolve to `127.0.0.1`.

9. Check your browser at `127.0.0.1:4180` to see if your configuration allows you to sign in.

   You should see a message that says “Authenticated”.

About token signatures

This topic tells you about the concept of token signatures.

Token signature 101

Token signature keys are used by an `AuthServer` to sign JSON Web Tokens (JWTs), produce a JWS Signature and attach it to the JOSE Header of a JWT. The client application can then verify the JWT signature.

A private key signs a JWT. A public key verifies the signature of a signed JWT.

The sign-and-verify mechanism serves multiple security purposes:

- **Authenticity**: signature verification ensures that the issuer of the JWT is from a source that is advertised.
- **Integrity**: signature verification ensures that the JWT has not been altered in transit or during its issued lifetime. Integrity is a foundational pillar of the CIA triad concept in

Information Security.

- **Non-repudiation**: signature verification ensures that the authorization server that signed the JWT cannot deny that they have signed it after its issuance (granted that the signing key that signed the JWT is available).

AppSSO only supports the `RS256` algorithm for signing tokens. For more information, see JSON Web Algorithms (JWA) documentation.

Token signature of an `AuthServer`

You must configure token signatures for `AuthServer`. An `AuthServer` receives its keys under `spec.tokenSignature`. For example:

```
spec:
  tokenSignature:
    signAndVerifyKeyRef:
      name: sample-token-signing-key
    extraVerifyKeyRefs:
      - name: sample-token-verification-key-1
      - name: sample-token-verification-key-2
```

There can only be **one** token signing key `spec.tokenSignature.signAngVerifyKeyRef` at any given time, and arbitrarily many token verification keys `spec.tokenSignature.extraVerifyKeyRefs`. The token signing key is used to sign and verify actively issued JWTs in circulation, whereas token verification keys are used to verify issued JWTs signatures. Token verification keys are thought to be previous token signing keys but have been rotated into verify only mode as a rotation mechanism measure, and can potentially be slated for eviction at a predetermined time.

The `AuthServer` serves its public keys at `{spec.issuerURI}/oauth2/jwks`. For example:

```
$ curl -s authserver-sample.default/oauth2/jwks | jq
{
  "keys": [
    {
      "kty": "RSA",
      "e": "AQAB",
      "kid": "sample-token-signing-key",
      "n": "0iCinir7sWKZE_3QXq4eTub_GU-lvdAKFI9dzDlwX7XZwwSERuzzQQ_Fs7i9djMl5bpv2ma_3Z
B-j2W9pR9ZIa3nqBI29AHqx2zmVQ8w-GxPDGRMkBdMOWNwyDQGIRlQnJFpXRoSQ5_viM9gYA56WthkDghrupGU
iB_zqGFYlgnz7sd4lC-thgEkDi9vY68DLIFdsXOQIXFqakyEIo43n_0vg6JRGQW1LU_32Ok6OgA3r6bYcE8VQh
JW3sE1qOSFcP0JrPA3YgmTNuDV6GoCLZeMxDdMDKdDcH5UgERLQe1qMMKwlMCeKamOWgo9eBvcFnWNR0I_MJV6
F14U1WbIcQ"
    },
    {
      "kty": "RSA",
      "e": "AQAB",
      "kid": "sample-token-verification-key-1",
      "n": "wc7uOACU62Yu_zKT9YrI4v-_X3L47nbVlcByi4UTVhg8o001OkiYAPAEoDCEHnDg_54gTWxe3h
DRcOJrd72PkTAaxH8aFdikoyakRVG9NvAPbcfzvI8R8plepUbs1U7TPPDEDARm_fZX6QdVyz0CTSafrz-yktTA
DxJhYPgvFLeHq7g7RouB1szTWDCM1haoxKa4960_x9meghNn87z0uF3cAd7TM_k3capYnxNOUT5g1vjJ05Vk14
JUl4R294OpMXPCGcFuvu9auXeBqXyKxxTAnLkDdNrgtT0FJHwnh4RGnrNqjYZOwlRvGbzwQ7du97aU2-qgbKkJ
rWYZWcw2bQ"
    },
    {
      "kty": "RSA",
      "e": "AQAB",
      "kid": "sample-token-verification-key-2",
      "n": "qELrLiaD-IVp_nthVn2EsLuShtU9ovyVIPkLVf47AqKogPV2frE_6Sv8k7Zim-SgDXfjLEg-UG
lQrb4KFm_WkaK2Uf6PCapiBnMi1Q5P8qC0WC5LT6XyPY1exCQbMrEsyd89oS0sKxgoc3Qv0XV24jGYiWQyJ7I0
Rub_QEldGM_dSlfbI-1Qt_U6Ll22OEc1D6P1A3MdDrgbur6N7ZemxlKI26-OAdlbNi0u-lFNj3Ss-pfTVi_fD2
hAajRRmc4tmHejQjH36M4F1NSW_gTbb6VX5EerVuDwSCCK0EuGvhcb1hg6kYEoO-qws54AQ0PywBXT5qksCMBm
mzjP6qO4Ow"
```

```
    }
  ]
}
```

> **Caution**
>
> Changes to `spec.tokenSignature.signAngVerifyKeyRef` have immediate effects.

As a service operator, you have control over which keys are used for certain purposes. For more information, see Manage token signature keys for Application Single Sign-On.

Get Started with Application Single Sign-On

This topic tells you about concepts important to getting started with Application Single Sign-On (commonly called AppSSO).

Use this topic to learn how to:

1. Curate an AppSSO service offering.
2. Claim credentials.
3. Deploy your workload.

After completing these steps, you can proceed with securing a workload. For more information, see Secure a workload with Application Single Sign-On.

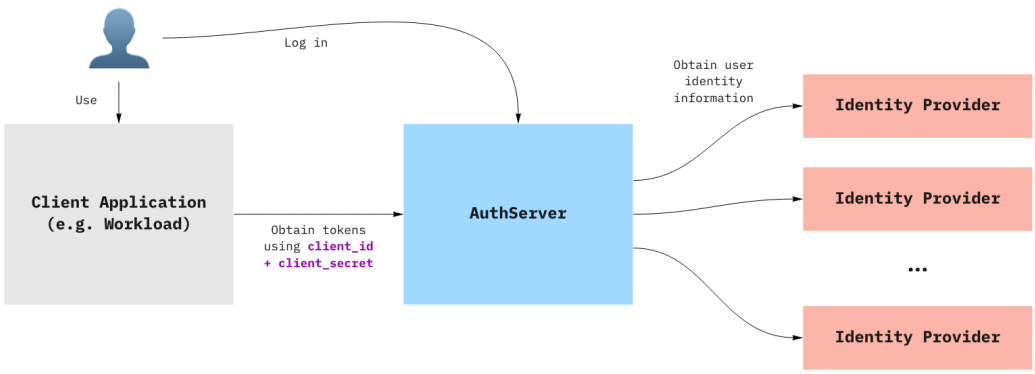
Prerequisites

To get start with Application Single Sign-On, you must:

- Install Application Single Sign-On on your Tanzu Application Platform cluster. For more information, see Install Application Single Sign-On.
- Install the Tanzu CLI on your machine and connect to a Tanzu cluster.

Key concepts

At the core of Application Single Sign-On is the concept of the Authorization Server, outlined by the AuthServer custom resource. Service Operators create those resources to provision running Authorization Servers, which are OpenID Connect Providers. They issue ID Tokens to the client applications, which contain identity information about the end user, such as email, first name, last name and so on.



The following steps outline how a client application uses an `AuthServer` to authenticate an end-user:

1. The end-user visits the client application.
2. The client application redirects the end-user to the `AuthServer`, with an OAuth2 request.
3. The end-user logs in with the `AuthServer` by using an external identity provider, for example, Google or Azure AD.
    1. The identity providers are set up by Service Operators.
    2. `AuthServer`s use various protocols to obtain identity information about the user, such as OpenID Connect, SAML, or LDAP, which might require additional redirects.
4. The `AuthServer` redirects the end-user to the client application with an authorization code.
5. The client application communicates with the `AuthServer` to receive an `id_token`.
    1. The client application does not know how the `AuthServer` collected identity information. It only receives the identity information as an `id_token`.

ID Tokens are JSON Web Tokens containing standard claims about the identity of the user, for example, name or email, and standard claims about the token itself, for example, “expires at” or “audience”. Here is an example of an `id_token` issued by an Authorization Server:

```
{
        "iss": "https://appsso.example.com",
        "sub": "213435498y",
        "aud": "my-client",
        "nonce": "fkg0-90_mg",
        "exp": 1656929172,
        "iat": 1656928872,
        "name": "Jane Doe",
        "given_name": "Jane",
        "family_name": "Doe",
        "email": "jane.doe@example.com",
        "roles": [
                "developer",
                "org-user"
        ]
}
```

`roles` claim is included in an `id_token` only if user roles are mapped and the `roles` scope is requested. For more information about mapping for OpenID Connect, LDAP and SAML, see:

- OpenID external groups mapping
- LDAP external groups mapping
- SAML (experimental) external groups mapping

ID Tokens are signed by the `AuthServer` by using Token signature keys. Client applications can verify their validity by using the `AuthServer`’s public keys.

Curate an AppSSO service offering

This section tells you how to provision an `AuthServer` for Application Single Sign-On. Use this topic to learn how to:

1. Discover the existing Application Single Sign-On service offerings in your cluster.
2. Set up your first `ClusterUnsafeTestLogin`.
3. Ensure the `AuthServer` is running and users can log in.

Prerequisites

You must install and correctly configure Application Single Sign-On on your Tanzu Application Platform cluster.

Application Single Sign-On is installed with the `run`, `iterate`, and `full` profiles. No extra steps are required.

To verify Application Single Sign-On is installed on your cluster, run:

```
tanzu package installed list -A | grep "sso.apps.tanzu.vmware.com"
```

For more information about the Application Single Sign-On installation, see Install Application Single Sign-On.

Discover the existing Application Single Sign-On service offerings

The Application Single Sign-On login servers are a consumable service offering in Tanzu Application Platform. The `ClusterWorkloadRegistrationClass` represents these service offerings.

In your Kubernetes cluster, run the following command:

```
tanzu service class list
```

If there is not a login offering you want to connect to, you must create your own.

Caution

The `AuthServer` example uses an unsafe testing-only identity provider. Never use it in production environments. For more information about the identity providers, see Identity providers.

Set up your first `ClusterUnsafeTestLogin`

In a non-poduction environment, `ClusterUnsafeTestLogin` is the recommended way to get started with Application Single Sign-On.

```
cat <<EOF | kubectl apply -f -
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClusterUnsafeTestLogin
metadata:
  name: my-login
EOF
```

Verify that your `AuthServer` is running

To see the service offering, run:

```
tanzu service class list
```

Expect to see the following output:

```
NAME                          DESCRIPTION
my-login  Login by AppSSO - user:password - UNSAFE FOR PRODUCTION!
```


Caution

> This login offering is not safe for production because it hard codes the user and password.

You can wait for the `ClusterUnsafeTestLogin` to be ready by running:

```
kubectl wait --for=condition=Ready clusterUnsafeTestLogin my-login
```

Alternatively, you can inspect your `ClusterUnsafeTestLogin` like any other resource:

```
kubectl get clusterunsafetestlogin.sso.apps.tanzu.vmware.com --all-namespaces
```

Expect to see the following output:

```
NAME       ISSUER URI                           STATUS
my-login   http://unsafe-my-login.appsso.<...>  Ready
```

You can visit the login page by using the `ISSUER URI`.

Claim credentials

Now that you have an Application Single Sign-On service offering. The next step is to create a `ClassClaim`, which creates consumable credentials for your workload, and allows your workload to connect to the login service offering by using the credentials.

Select your preferred login offering from the available options:

```
tanzu service class list
```

If there are none available, you can create one yourself:

```
tanzu service class-claim create my-workload \
  --class my-login \
  --parameter workloadRef.name=appsso-starter-java \
  --parameter redirectPaths='["/login/oauth2/code/appsso-starter-java"]'
```

The `redirectPaths` is the login redirect within your application. This example deploys a minimal Spring application, so you can use the Spring Security path.

Verify the status of your `ClassClaim` by running:

```
tanzu service class-claim list
```

Deploy an application with AppSSO

This section tells you how to deploy a minimal Kubernetes application that is protected by Application Single Sign-On (commonly called AppSSO) by using the credentials that tanzu service class-claim creates.

For more information about how a Client application uses an AuthServer to authenticate an end user, see the Overview of Application Single Sign-On.

Deploy a minimal application

Follow these steps to deploy a minimal Spring Boot application:

1. List the available accelerators by running:

```
tanzu accelerator list
```

Expect to see an accelerator named `appsso-starter-java`. This is the accelerator to use.

2. Create a project by running:

```
tanzu accelerator generate appsso-starter-java --server-url YOUR-TAP-SERVER-URI
--options '{"appssoOfferingName": "my-login"}'
```

This command creates a zip file.

3. Unzip the file by running:

```
unzip appsso-starter-java.zip
```

4. Make the following changes in `config/workload.yaml`:

    1. The reference to the `serviceClaims` is the `ServiceClaim` you generated in the previous step. Replace `appsso-starter-java` in the `serviceClaims` section with `my-workload`.
    2. There is a reference to a remote version of your repository. You can push this repository to your preferred Git repository and provide the reference here.

    The following code sample claimes the `ServiceClaim` from your appliction:

```
---
spec:
  serviceClaims:
    - name: YOUR-SERVICE-CLAIM
      ref:
        apiVersion: services.apps.tanzu.vmware.com/v1alpha1
        kind: ClassClaim
        name: YOUR-SERVICE-CLAIM
```

5. Deploy the workload by running:

```
kubectl apply -f config/workload.yaml
```

The `ServiceClaim` connects your application to the `AppSSO AuthServer`.

See the workload section of the Tanzu Application Platform Portal where you can find the URL for your workload at `https://tap-gui.YOUR-TAP-CLUSTER-DOMAIN`.

Application Single Sign-On how-to guides

This subsection describes the how-to guides for specific persona working with Application Single Sign-On (commonly called AppSSO):

- Platform operators
- Service operators
- Application operators
- Troubleshoot Application Single Sign-on

Application Single Sign-On for Platform Operators

The topics in this section tell you how to manage the Application Single Sign-On (commonly called AppSSO) package installation and what it installs. Use this section to learn:

- Install AppSSO
- Upgrades
- Uninstall AppSSO

Application Single Sign-On for Platform Operators

The topics in this section tell you how to manage the Application Single Sign-On (commonly called AppSSO) package installation and what it installs. Use this section to learn:

- Install AppSSO
- Upgrades
- Uninstall AppSSO

Install Application Single Sign-On

This topic tells you how to install Application Single Sign-On for VMware Tanzu (commonly called AppSSO) from the Tanzu Application Platform (commonly called TAP) package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Application Single Sign-On. For more information about profiles, see Components and installation profiles.

What's inside

The AppSSO package installs the following resources:

- The `appsso` namespace with a deployment of the AppSSO controller and services for webhooks.
- RBAC
- APIs

Prerequisites

Before installing Application Single Sign-On, ensure that you have Tanzu Application Platform installed on your Kubernetes cluster.

The `sso.apps.tanzu.vmware.com` package has these dependencies:

- `cert-manager.tanzu.vmware.com`: Required at installation time and runtime. For more information see, Install cert-manager.
- `crossplane.tanzu.vmware.com`: Required at installation time and runtime. For more information see, Install Crossplane.
- `service-bindings.tanzu.vmware.com`: Required at runtime. For more information see, Install Service Bindings.

Important

Installation time dependencies must be present on the cluster at the time the Application Single Sign-On package is being applied.

Runtime dependencies don't have to be present when the Application Single Sign-On package is being applied, but they are required eventually for it to function fully.

Install Application Single Sign-On for VMware Tanzu

1. Learn more about the AppSSO package by running:

```
tanzu package available get sso.apps.tanzu.vmware.com --namespace tap-install
```

2. Install the AppSSO package by running:

```
tanzu package install appsso \
 --namespace tap-install \
 --package sso.apps.tanzu.vmware.com \
 --version 5.0
```

3. Confirm the package has reconciled by running:

```
tanzu package installed get appsso --namespace tap-install
```

See also

- To configure the Application Single Sign-On package to meet your needs, see Package configuration for Application Single Sign-On.
- To upgrade to a later version of the package, see Upgrade Application Single Sign-On.
- When deployed on an OpenShift cluster, additional OpenShift-specific resources are installed. For more information, see Application Single Sign-On for OpenShift clusters.

Upgrade Application Single Sign-On

This topic tells you how to upgrade Application Single Sign-On (commonly called AppSSO).

The `AppSSO` package is upgraded as part of your `TAP` package installation.

For migrating your resources in between versions, see the Migration guides.

If you installed the `AppSSO` package on its own, and not as part of `TAP`, you can upgrade it individually by running:

```
tanzu package installed update PACKAGE-INSTALLATION-NAME -p sso.apps.tanzu.vmware.com
-v 5.0 --values-file PATH-TO-YOUR-VALUES-YAML -n YOUR-INSTALL-NAMESPACE
```

Note

You can also upgrade Application Single Sign-On as part of upgrading Tanzu Application Platform as a whole. See Upgrade Tanzu Application Platform for more information.

Migration guides

`v3.0.0` to `v3.1.0`

VMware recommends that you recreate your `AuthServers` after upgrading your Application Single Sign-On to `v3.1.0` with the following changes:

- Migrate field `.spec.identityProviders[*].openid.claimMappings["roles"]` to `.spec.identityProviders[*].openid.roles.fromUpstream.claim`.
- Migrate field `.spec.identityProviders[*].ldap.group.roleAttribute` to `.spec.identityProviders[*].ldap.roles.fromUpstream.attribute`.
- Migrate field `.spec.identityProviders[*].ldap.group.search` to `.spec.identityProviders[*].ldap.roles.fromUpstream.search`.
- Migrate field `.spec.identityProviders[*].saml.claimMappings["roles"]` to `.spec.identityProviders[*].saml.roles.fromUpstream.attribute`.

(Optional) If you plan to run Spring Boot 3 based `Workload`s, you must perform the following migration tasks in your existing `ClientRegistration` resources:

- Migrate `.spec.clientAuthenticationMethod` values.
- Migrate the existing value `post` to `client_secret_post` or migrate the existing value `basic` to `client_secret_basic`.

`v2.0.0` to `v3.0.0`

VMware recommends that you recreate your `AuthServers` after upgrading your Application Single Sign-On to `v3.0.0` with the following changes:

- Migrate the field `.spec.tls.disabled` to `.spec.tls.deactivated`.

`v1.0.0` to `v2.0.0`

VMware recommends that you recreate your `AuthServers` after upgrading your Application Single Sign-On to `v2.0.0` with the following changes:

- Migrate from `.spec.issuerURI` to `.spec.tls`:

    1. Configure one of `.spec.tls.{issuerRef, certificateRef, secretRef}`. See Issuer URI & TLS for more information.
    2. (Optional) Disable TLS with `.spec.tls.disabled`.
    3. Remove `.spec.issuerURI`.
    4. Delete your `AuthServer`-specific `Service` and ingress resources.
    5. Apply your `AuthServer`. You can find its issuer URI in `.status.issuerURI`.
    6. Update the redirect URIs in your upstream identity providers.

    Application Single Sign-On templates your issuer URI and enables TLS. When using the newer `.spec.tls`, a custom `Service` and an ingress resource are no longer required.

    It is not recommended to continue using `.spec.issuerURI` in Application Single Sign-On `v2.0.0`. To use `.spec.issuerURI` in `v2.0.0`, you must provide a `Service` and an ingress resource as in `v1.0.0`.

- If you use the `internalUnsafe` identity provider to migrate the existing users by replacing the bcrypt hash through the plaintext equivalent, you can still use the existing bcrypt passwords by prefixing them with `{bcrypt}`:

    ```
    ---
    apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
    kind: AuthServer
    metadata:
    ```

```
 # ...
spec:
 identityProviders:
   - name: internal
     internalUnsafe:
       users:
         # v1.0
         - username: test-user-1
           password: $2a$10$201z9o/tHlocFsHFTo0plukh03ApBYe4dRiXcqeyRQH6CNNtS8j
WK # bcrypt-encoded "password"
           # ...
         # v2.0
         - username: "test-user-1"
           password: "{bcrypt}$2a$10$201z9o/tHlocFsHFTo0plukh03ApBYe4dRiXcqeyRQ
H6CNNtS8jWK" # same bcrypt hash, with {bcrypt} prefix
         - username: "test-user-2"
           password: "password" # plaintext
 # ...
```

Uninstall Application Single Sign-On

This topic tells you how to uninstall Application Single Sign-On (commonly called AppSSO).

Delete the AppSSO package by running:

```
tanzu package installed delete appsso --namespace tap-install
```

To permanently delete and exclude AppSSO package from your Tanzu Application Platform install, edit your Tanzu Application Platform values file by including the following configuration:

```
excluded_packages:
  - sso.apps.tanzu.vmware.com
```

For more information, navigate to Exclude packages from a Tanzu Application Platform profile.

Application Single Sign-On for Service Operators

The following topics tell you how to configure a fully operational authorization server for Application Single Sign-On (commonly called AppSSO).

`AuthServer` represents the request for an OIDC authorization server. It results in the deployment of an authorization server backed by Redis. A Redis with mTLS is either automatically deployed for `AuthServer` or credentials to external storage can be provided. You can configure the labels with which clients can select an `AuthServer`, the namespaces it allows clients from, its issuer URI, its token signature keys, identity providers, and further details for its deployment.

`ClusterWorkloadRegistrationClass` exposes an `AuthServer` as a ready-to-claim service offering. Application operators can discover this offering and claim credentials. The mechanisms for this are provided by Services Toolkit. This is the recommended way for offering and consuming AppSSO.

If you just want to get started in a non-production environment, `ClusterUnsafeTestLogin` is a zero-config API that produces an unsafe, ready-to-claim AppSSO service offering. It is a higher-level alternative to the combination of `AuthServer` and `ClusterWorkloadRegistrationClass`.

For a full explanation of the available APIs, refer to the API reference.

The following sections outline the essential steps to configure a fully operational, ready-to-claim AppSSO service offering:

- Configure an unsafe test login
- Annotations and labels

- Issuer URI and TLS
- TLS scenario guides
- CA certificates
- Configure workloads to trust a custom CA
- Identity providers
- Configure authorization
- Public clients and CORS
- Token settings
- Token signatures
- Session settings
- Storage
- AuthServer readiness
- Scale AuthServer
- Curate an AppSSO service offering

Application Single Sign-On for Service Operators

The following topics tell you how to configure a fully operational authorization server for Application Single Sign-On (commonly called AppSSO).

`AuthServer` represents the request for an OIDC authorization server. It results in the deployment of an authorization server backed by Redis. A Redis with mTLS is either automatically deployed for `AuthServer` or credentials to external storage can be provided. You can configure the labels with which clients can select an `AuthServer`, the namespaces it allows clients from, its issuer URI, its token signature keys, identity providers, and further details for its deployment.

`ClusterWorkloadRegistrationClass` exposes an `AuthServer` as a ready-to-claim service offering. Application operators can discover this offering and claim credentials. The mechanisms for this are provided by Services Toolkit. This is the recommended way for offering and consuming AppSSO.

If you just want to get started in a non-production environment, `ClusterUnsafeTestLogin` is a zero-config API that produces an unsafe, ready-to-claim AppSSO service offering. It is a higher-level alternative to the combination of `AuthServer` and `ClusterWorkloadRegistrationClass`.

For a full explanation of the available APIs, refer to the API reference.

The following sections outline the essential steps to configure a fully operational, ready-to-claim AppSSO service offering:

- Configure an unsafe test login
- Annotations and labels
- Issuer URI and TLS
- TLS scenario guides
- CA certificates
- Configure workloads to trust a custom CA
- Identity providers
- Configure authorization
- Public clients and CORS

- Token settings
- Token signatures
- Session settings
- Storage
- AuthServer readiness
- Scale AuthServer
- Curate an AppSSO service offering

Configure an unsafe test login

This topic for service operators describes how you can get started with Application Single Sign-On for VMware Tanzu (commonly called AppSSO) in a non-production environment by using `ClusterUnsafeTestLogin`.

`ClusterUnsafeTestLogin` is a zero-config API that produces an unsafe, ready-to-claim AppSSO service offering. When you create a `ClusterUnsafeTestLogin`, you get a simple `AuthServer` and a `ClusterWorkloadRegistrationClass` for it. The `AuthServer` has a single login `user:password` and is configured to work without extra configuration.

Caution

`ClusterUnsafeTestLogin` is not safe for production. For production, use `AuthServer` and `ClusterWorkloadRegistrationClass`.

Configure a `ClusterUnsafeTestLogin`

The `ClusterUnsafeTestLogin` resource takes zero configuration except a `name`.

To configure a `ClusterUnsafeTestLogin` create a YAML file as follows:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClusterUnsafeTestLogin
metadata:
  name: demo
```

Use the unsafe test login

After applying the `ClusterUnsafeTestLogin` resource, application operators can discover credentials for it by running:

```
tanzu service class list
```

Example output:

```
NAME  DESCRIPTION
demo  Login by AppSSO - user:password - UNSAFE FOR PRODUCTION!
```

For how to application operators can claim credentials, see Claim credentials for an Application Single Sign-On service offering.

Annotations and labels for AppSSO

This topic tells you how to configure annotations and labels for Application Single Sign-On (commonly called AppSSO).

An `AuthServer` is selectable by `ClientRegistration` through labels. The namespace an `AuthServer` allows `ClientRegistrations` from is controlled with an annotation.

Labels

`ClientRegistrations` select an `AuthServer` with `spec.authServerSelector`. Therefore, an `AuthServer` must have a set of labels that uniquely identifies it amongst all `AuthServer`. A `ClientRegistration` must match only one `AuthServer`. Registration fails if multiple or no `AuthServer` resources are matched.

For example:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  labels:
    env: dev
    ldap: True
    saml: True
# ...
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  labels:
    env: prod
    saml: True
# ...
```

Allowing client namespaces

`AuthServer` optionally controls from which namespace (one of more) it allows `ClientRegistrations` with the annotation:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  annotations:
    sso.apps.tanzu.vmware.com/allow-client-namespaces: "my-apps"
```

To allow `ClientRegistrations` only from a restricted set of namespaces, you must set this annotation. Its value is a comma-separated list of allowed `Namespaces`, for example, `"app-team-red,app-team-green"`. If the annotation is missing, the default value is `*`, denoting that all client namespaces are allowed.

VMware recommends explicitly restricting to only workload-related namespaces to narrow the scope of the `AuthServer` operation.

Unsafe configuration

`AuthServer` enforces secure and production-ready configuration. However, sometimes it is required to opt-out those constraints, for example, when deploying `AuthServer` on an iterate cluster.

Caution

Allowing **unsafe** is not recommended for production.

Unsafe identity provider

The `InternalUnsafe` identity provider cannot be used unless explicitly allowed by including the annotation `sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider` as follows:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
spec:
  identityProviders:
    - name: static-users
      internalUnsafe:
      # ...
```

If the annotation is not present and an `InternalUnsafe` identity provider is configured the `AuthServer` will not apply.

Unsafe issuer URI

It's not possible to use a plain HTTP issuer URI, unless it's explicitly allowed by including the annotation `sso.apps.tanzu.vmware.com/allow-unsafe-issuer-uri` as follows:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-issuer-uri: ""
spec:
  issuerURI: http://this.is.unsafe
```

If the annotation is not present and a plain HTTP issuer URI is configured, the `AuthServer` does not apply.

Issuer URI and TLS for AppSSO

This topic tells you how to configure the issuer URI and TLS for Application Single Sign-On (commonly called AppSSO).

Overview

An `AuthServer` entry point for its clients and their end-users is called *issuer URI*. AppSSO will template the issuer URI and create a TLS-enabled `Ingress` for it. For this purpose, your platform operator configures the domain name and template. Once you created and `AuthServer` you can find the actual URL in `.status.issuerURI`.

You can configure whether and how to obtain a TLS certificate for the issuer URI by using `.spec.tls`. Unless TLS is deactivated, HTTPS is enforced. For example, requests for `http://` are redirected to `https://`. You can observe the TLS configuration in `.status.tls`.

If AppSSO is installed with a default issuer, you can omit `AuthServer.spec.tls` and a TLS certificate is obtained automatically. This is the recommended approach for TLS.

For example:

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: login
  namespace: services
  # ...
spec:
# ...
status:
  issuerURI: https://login.services.example.com
  tls:
    issuerRef:
      name: my-default-issuer
      kind: ClusterIssuer
      group: cert-manager.io
  # ...
```

This `AuthServer` is reachable at its templated issuer URI `https://login.services.example.com` and serves a TLS certificate obtained from `my-default-issuer`.

Learn how to configure TLS for your `AuthServer`:

- Issuer URI and TLS for AppSSO
  - Overview
  - Configure TLS by using a (Cluster)Issuer
  - Configure TLS by using a Certificate
  - Configure TLS by using a Secret
  - Deactivate TLS (unsafe)
  - Allow `Workloads` to trust a custom CA `AuthServer`

There are many use-cases that pertain to TLS use. To find out which scenario applies to you and how to configure it, see TLS scenario guides.

If your `AuthServer` obtains a certificate from a custom CA, you can enable App Operators to trust it. See Allow Workloads to trust a custom CA AuthServer for more information.

Configure TLS by using a (Cluster)Issuer

You can obtain a TLS certificate for your `AuthServer` by referencing a `cert-manager.io/v1/Issuer` or `cert-manager.io/v1/ClusterIssuer`. This enables AppSSO to retrieve a `cert-manager.io/v1/Certificate` from the issuer and apply it to the `Ingress` configuration.

The composition of an `AuthServer` and a self-signed `Issuer` looks as follows:

```
---
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: my-selfsigned-issuer
  namespace: services
spec:
  selfSigned: { }

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: login
  namespace: services
  # ...
```

```
spec:
  tls:
    issuerRef:
      name: my-selfsigned-issuer
      # 'kind: Issuer' can be omitted. It is the default.
```

The composition of an `AuthServer` and a *self-signed* `ClusterIssuer` for looks as follows:

```
---
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: my-selfsigned-cluster-issuer
spec:
  selfSigned: { }

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: login
  namespace: services
  # ...
spec:
  tls:
    issuerRef:
      name: my-selfsigned-cluster-issuer
      kind: ClusterIssuer
```

Confirm that your `AuthServer` serves a TLS certificate from the specified issuer by visiting its `{.status.issuerURI}`.

For more information about cert-manager and its APIs. see cert-manager documentation.

Configure TLS by using a Certificate

In order to configure TLS for your `AuthServer` using a `cert-manager.io/v1/Certificate` you must know what its templated issuer URI will be. You can infer it from the AppSSO package's domain template.

The composition of an `AuthServer` and a `Certificate` looks as follows:

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: login
  namespace: services
spec:
  dnsNames:
    - login.services.example.com
  issuerRef:
    name: my-cluster-issuer
    kind: ClusterIssuer
  secretName: login-cert

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: login
  namespace: services
  # ...
spec:
  tls:
```

```
    certificateRef:
      name: login
```

Confirm that your `AuthServer` serves the specified Certificate by visiting its `{.status.issuerURI}`.

For more information about cert-manager and its APIs. see cert-manager documentation.

Configure TLS by using a Secret

If you don't want to use cert-manager.io's APIs or you have a raw TLS certificate in a TLS `Secret`, you can compose it with your `AuthServer` by referencing it. The certificate must be for the issuer URI that will be templated for the `AuthServer`. You can infer it from the AppSSO package's domain template.

The composition of an `AuthServer` and TLS `Secret` looks as follows:

```
apiVersion: v1
kind: Secret
metadata:
  name: my-tls-cert
  namespace: services
type: kubernetes.io/tls
data:
  tls.key: # ...
  tls.crt: # ...

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: login
  namespace: services
  # ...
spec:
  tls:
    secretRef:
      name: my-tls-cert
```

Deactivate TLS (unsafe)

If you deactivate TLS autoconfiguration, `AuthServer` only works over plain HTTP. You must deactivate TLS with the `sso.apps.tanzu.vmware.com/allow-unsafe-issuer-uri: ""` annotation.

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: login
  namespace: services
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-issuer-uri: ""
  # ...
spec:
  tls:
    deactivated: true
```


Caution

Deactivating TLS is unsafe and not recommended for production.

Allow `Workloads` to trust a custom CA `AuthServer`

If your `AuthServer` obtains a certificate from a custom CA, its consumers do not trust it by default. You can enable App Operators' `Workloads` to trust your `AuthServer` by exporting a `ca-certificates` service binding `Secret` to their `Namespace`.

A composition of `SecretTemplate` and `SecretExport` are a way to achieve this. If your custom CA's TLS `Secret` is present in the namespace `my-certs`, then you can provide a `ca-certificates` service binding `Secret` like so:

```
---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretTemplate
metadata:
  name: ca-cert
  namespace: my-certs
spec:
  inputResources:
    - name: my-custom-ca
      ref:
        apiVersion: v1
        kind: Secret
        name: my-custom-ca
  template:
    data:
      ca.crt: $(.my-custom-ca.data.tls\.crt)
    stringData:
      type: ca-certificates

---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretExport
metadata:
  name: ca-cert
  namespace: my-certs
spec:
  toNamespace: "*"
```

This templates a `ca-certificates` service binding `Secret` which `Workload` can claim to trust the custom CA. It does not contain the CA's private key and is generally safe to share.

However, be careful, this example exports to all namespace on the cluster. If this does not comply with your policies, then adjust the target namespaces if required.

For more information about secretgen-controller and its APIs, see secretgen-controller documentation in GitHub.

TLS scenario guides for AppSSO

This topic tells you how to obtain a TLS certificate in different scenarios for Application Single Sign-On (commonly called AppSSO).

Overview

`AuthServer` is a piece of security infrastructure. It is imperative to configure TLS for it, so that its issuer URI's scheme is `https://`.

`AuthServer.spec.tls` accommodates different scenarios for obtaining a TLS certificate. Select the scenario that matches your case.

The recommended path is to install AppSSO with a default issuer. In that case, you can omit `AuthServer.spec.tls` and a TLS certificate is obtained automatically.

Prerequisites

Each of the scenarios requires that the AppSSO package is installed and configured. In particular, its `domain_name` must match the ingress domain of your cluster. The presented YAML resources assume `my-tap.example.com` as the ingress domain. Therefore, the AppSSO configuration values look as follows:

```
#! AppSSO values
domain_name: "my-tap.example.com"
```

The default `domain_template: "{{.Name}}.{{.Namespace}}.{{.Domain}}"` works for most scenarios. If a scenario requires a bespoke `domain_template`, it contains the relevant instructions.

After applying each scenario, wait for your `AuthServer` to become ready and then test it by running:

```
kubectl wait --namespace login authserver/sso --for condition=Ready=True --timeout 500
s
curl --location "$(kubectl get --namespace login authserver sso --output=jsonpath='{.s
tatus.issuerURI}')/.well-known/openid-configuration"
```

Alternatively, visit the `AuthServer` with your browser. You can obtain its issuer URI by running:

```
kubectl get --namespace login authserver sso --output=jsonpath='{.status.issuerURI}'
```

Caution

Before applying each scenario, you must configure your AppSSO correctly, and make sure that all certificates and DNS names comply with your setup.

Using a default issuer

VMware recommend using a default issuer,
because this approach separates the responsibilities of platform operators and service operators. In this case, the `Authserver.spec.tls` field is not required.

To verify whether `AppSSO` was installed with a default issuer, run:

```
kctrl package installed get --namespace tap-install --package-install tap --values-fil
e-output tap-values.yaml
```

If a `shared.ingress_issuer` appears in your `tap-values.yaml` file, you have a default issuer.

Important

Ensure kctrl is installed.

```
apiVersion: v1
kind: Namespace
metadata:
  name: login

---
```

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: sso
  namespace: login
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
    sso.apps.tanzu.vmware.com/documentation: Uses the default issuer for TLS
spec:
  identityProviders:
    - name: test-users
      internalUnsafe:
        users:
          - username: user
            password: password
  tokenSignature:
    signAndVerifyKeyRef:
      name: signing-key

---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: signing-key
  namespace: login
spec:
  secretTemplate:
    type: Opaque
    stringData:
      key.pem: $(privateKey)
      pub.pem: $(publicKey)
```

Using a `ClusterIssuer`

A `ClusterIssuer` is a cluster-scoped API provided by cert-manager from which certificates can be obtained programmatically.

This scenario puts all resources into a single YAML file and uses Let's Encrypt's production API. You might get the `ClusterIssuer` from your platform operators.

For more information, see cert-manager documentation.

Caution

Let's Encrypt's production API rate limits apply.

```
---
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-production
spec:
  acme:
    privateKeySecretRef:
      name: letsencrypt-production
    server: https://acme-v02.api.letsencrypt.org/directory
    solvers:
      - http01:
          ingress:
            class: contour

---
```

```
apiVersion: v1
kind: Namespace
metadata:
  name: login

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: sso
  namespace: login
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
spec:
  #! --- TLS ---
  tls:
    issuerRef:
      name: letsencrypt-production
      kind: ClusterIssuer
  #! -----------
  identityProviders:
    - name: test-users
      internalUnsafe:
        users:
          - username: user
            password: password
  tokenSignature:
    signAndVerifyKeyRef:
      name: signing-key

---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: signing-key
  namespace: login
spec:
  secretTemplate:
    type: Opaque
    stringData:
      key.pem: $(privateKey)
      pub.pem: $(publicKey)
```

Using an `Issuer`

This scenario is identical to Using a ClusterIssuer, except that the Issuer is scoped to a namespace and must be located in the same namespace as the `AuthServer`.

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: login

---
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: letsencrypt-production
  namespace: login
spec:
  acme:
    privateKeySecretRef:
      name: letsencrypt-production
```

```
    server: https://acme-v02.api.letsencrypt.org/directory
    solvers:
      - http01:
          ingress:
            class: contour

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: sso
  namespace: login
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
spec:
  #! --- TLS ---
  tls:
    issuerRef:
      name: letsencrypt-production
  #! -----------
  identityProviders:
    - name: test-users
      internalUnsafe:
        users:
          - username: user
            password: password
  tokenSignature:
    signAndVerifyKeyRef:
      name: signing-key

---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: signing-key
  namespace: login
spec:
  secretTemplate:
    type: Opaque
    stringData:
      key.pem: $(privateKey)
      pub.pem: $(publicKey)
```

Using an existing `Certificate`

A `Certificate` is an API provided by cert-manager that is scoped to a namespace and represents a TLS certificate obtained from a `(Cluster)Issuer`. To create a `Certificate`, you must know the name and kind of your issuer.

These scenarios use Let's Encrypt's production API and require that a `ClusterIssuer` by the name `letsencrypt-production` exists. See Using a `ClusterIssuer` for how to set up the issuer.

When using `Certificate`, its `.spec.dnsNames` must contain the FQDN of the templated issuer URI. The `domain_name` and `domain_template` of your AppSSO package installation must comply with your DNS name.

If you have an existing `Certificate` in the same `Namespace` where the `AuthServer` is installed, use the following AppSSO configuration values:

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: login
```

```
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: sso
  namespace: login
spec:
  dnsNames:
    - "sso.login.my-tap.example.com"
  issuerRef:
    name: letsencrypt-production
    kind: ClusterIssuer
  secretName: sso-cert

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: sso
  namespace: login
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
spec:
  #! --- TLS ---
  tls:
    certificateRef:
      name: sso
  #! -----------
  identityProviders:
    - name: test-users
      internalUnsafe:
        users:
          - username: user
            password: password
  tokenSignature:
    signAndVerifyKeyRef:
      name: signing-key

---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: signing-key
  namespace: login
spec:
  secretTemplate:
    type: Opaque
    stringData:
      key.pem: $(privateKey)
      pub.pem: $(publicKey)
```

secretgen-controller allows you to export and import `Secrets` across namespaces. When your `Certificate` is located in another namespace, for example, it's controlled by another team, you can import its `Secret` to other namespaces. If you have an existing `Certificate` in another `Namespace`, use the following AppSSO configuration values:

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: tls

---
apiVersion: cert-manager.io/v1
```

```
kind: Certificate
metadata:
  name: sso
  namespace: tls
spec:
  dnsNames:
    - "sso.login.my-tap.example.com"
  issuerRef:
    name: letsencrypt-production
    kind: ClusterIssuer
  secretName: sso-cert

---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretExport
metadata:
  name: sso-cert
  namespace: tls
spec:
  toNamespace: login

---
apiVersion: v1
kind: Namespace
metadata:
  name: login

---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretImport
metadata:
  name: sso-cert
  namespace: login
spec:
  fromNamespace: tls

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: sso
  namespace: login
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
spec:
  #! --- TLS ---
  tls:
    secretRef:
      name: sso-cert
  #! -----------
  identityProviders:
    - name: test-users
      internalUnsafe:
        users:
          - username: user
            password: password
  tokenSignature:
    signAndVerifyKeyRef:
      name: signing-key

---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: signing-key
  namespace: login
```

```
spec:
  secretTemplate:
    type: Opaque
    stringData:
      key.pem: $(privateKey)
      pub.pem: $(publicKey)
```


Caution

Be cautious when using `SecretExport` and `SecretImport` to facilitate the transfer across namespaces.

Using an existing TLS certificate

If you have an existing TLS certificate and private key, for example, if your TLS certificate was created outside the cluster, you can apply it directly.

If you don't have a TLS certificate, there are numerous ways to obtain TLS certificates. One of the simplest methods is to use a tool such as mkcert, step or openssl in GitHub.

If you have an existing TLS certificate in the same `Namespace` where the AuthServer is installed, use the following AppSSO configuration values:

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: login

---
apiVersion: v1
kind: Secret
type: kubernetes.io/tls
metadata:
  name: my-cert
  namespace: login
stringData:
  #! --- ReplaceMe - certificate and private key for "sso.login.my-tap.example.com ---
  tls.crt: |
    -----BEGIN CERTIFICATE-----
    # redacted
    -----END CERTIFICATE-----
  tls.key: |
    -----BEGIN PRIVATE KEY-----
    # redacted
    -----END PRIVATE KEY-----
  #! ---------------------------------------------------------------------------

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: sso
  namespace: login
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
spec:
  #! --- TLS ---
  tls:
    secretRef:
      name: my-cert
  #! -----------
```

```
  identityProviders:
    - name: test-users
      internalUnsafe:
        users:
          - username: user
            password: password
  tokenSignature:
    signAndVerifyKeyRef:
      name: signing-key

---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: signing-key
  namespace: login
spec:
  secretTemplate:
    type: Opaque
    stringData:
      key.pem: $(privateKey)
      pub.pem: $(publicKey)
```

Important

The TLS certificate `tls.crt` and its corresponding private key `tls.key` must be stored in a secret with these keys.

If you have an existing TLS certificate in another `Namespace`, use the following AppSSO configuration values:

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: tls

---
apiVersion: v1
kind: Secret
type: kubernetes.io/tls
metadata:
  name: my-cert
  namespace: tls
stringData:
  #! --- ReplaceMe - certificate and private key for "sso.login.my-tap.example.com ---
  tls.crt: |
    -----BEGIN CERTIFICATE-----
    # redacted
    -----END CERTIFICATE-----
  tls.key: |
    -----BEGIN PRIVATE KEY-----
    # redacted
    -----END PRIVATE KEY-----
  #! ---------------------------------------------------------------------------

---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretExport
metadata:
  name: my-cert
  namespace: tls
```

```
spec:
  toNamespace: login

---
apiVersion: v1
kind: Namespace
metadata:
  name: login

---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretImport
metadata:
  name: my-cert
  namespace: login
spec:
  fromNamespace: tls

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: sso
  namespace: login
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
spec:
  #! --- TLS ---
  tls:
    secretRef:
      name: my-cert
  #! -----------
  identityProviders:
    - name: test-users
      internalUnsafe:
        users:
          - username: user
            password: password
  tokenSignature:
    signAndVerifyKeyRef:
      name: signing-key

---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: signing-key
  namespace: login
spec:
  secretTemplate:
    type: Opaque
    stringData:
      key.pem: $(privateKey)
      pub.pem: $(publicKey)
```

Important

The TLS certificate `tls.crt` and its corresponding private key `tls.key` must be stored in a secret with these keys.

Be cautious when using `SecretExport` and `SecretImport` to facilitate the transfer across namespaces.

Using an existing wildcard TLS certificate

To use wildcard certificates for DNS names such as `*.my-tap.example.com`, you must edit the AppSSO's `domain_template` so that the templated issuer URIs for `AuthServer` match the wildcard. For example:

- `sso.login.my-tap.example.com` does not match the wildcard.
- `sso-login.my-tap.example.com` matches the wildcard.

The following AppSSO configuration values accommodates a wildcard certificate for `*.my-tap.example.com`:

```
#! AppSSO values
domain_name: "my-tap.example.com"
domain_template: "{{.Name}}-{{.Namespace}}.{{.Domain}}"
#!                          ^ note the dash
```

The following scenarios require TLS Secrets, but the same concept applies to `Certificate`.

Important

When using a `(Cluster)Issuer` for Let's Encrypt, you cannot request wildcard certificates when it uses the http01 challenge solver.

If you have an existing wildcard TLS certificate in the same `Namespace` where the AuthServer is installed, use the following AppSSO configuration values:

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: login

---
apiVersion: v1
kind: Secret
type: kubernetes.io/tls
metadata:
  name: my-wildcard-cert
  namespace: login
stringData:
  #! --- ReplaceMe - certificate and private key for "*.my-tap.example.com ---
  tls.crt: |
    -----BEGIN CERTIFICATE-----
    # redacted
    -----END CERTIFICATE-----
  tls.key: |
    -----BEGIN PRIVATE KEY-----
    # redacted
    -----END PRIVATE KEY-----
  #! ---------------------------------------------------------------------------

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: sso
  namespace: login
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
spec:
```

```
  #! --- TLS ---
  tls:
    secretRef:
      name: my-wildcard-cert
  #! -----------
  identityProviders:
    - name: test-users
      internalUnsafe:
        users:
          - username: user
            password: password
  tokenSignature:
    signAndVerifyKeyRef:
      name: signing-key

---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: signing-key
  namespace: login
spec:
  secretTemplate:
    type: Opaque
    stringData:
      key.pem: $(privateKey)
      pub.pem: $(publicKey)
```

Note

This scenario is similar to using an existing TLS certificate in the same namespace, except that the certificate is a wildcard.

If you have an existing wildcard TLS certificate in another `Namespace`, use the following AppSSO configuration values:

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: tls

---
apiVersion: v1
kind: Secret
type: kubernetes.io/tls
metadata:
  name: my-wildcard-cert
  namespace: login
stringData:
  #! --- Certificate and private key for "*.my-tap.example.com ---
  tls.crt: |
    -----BEGIN CERTIFICATE-----
    # redacted
    -----END CERTIFICATE-----
  tls.key: |
    -----BEGIN PRIVATE KEY-----
    # redacted
    -----END PRIVATE KEY-----
  #! ------------------------------------------------------------

---
```

```
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretExport
metadata:
  name: my-cert
  namespace: tls
spec:
  toNamespace: login

---
apiVersion: v1
kind: Namespace
metadata:
  name: login

---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretImport
metadata:
  name: my-cert
  namespace: login
spec:
  fromNamespace: tls

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: sso
  namespace: login
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
spec:
  #! --- TLS ---
  tls:
    secretRef:
      name: my-wildcard-cert
  #! -----------
  identityProviders:
    - name: test-users
      internalUnsafe:
        users:
          - username: user
            password: password
  tokenSignature:
    signAndVerifyKeyRef:
      name: signing-key

---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: signing-key
  namespace: login
spec:
  secretTemplate:
    type: Opaque
    stringData:
      key.pem: $(privateKey)
      pub.pem: $(publicKey)
```

Note

This scenario is similar to using an existing TLS certificate in another namespace, except that the certificate is a wildcard.

CA certificates for Application Single Sign-On

This topic tells you how to configure CA certificates for `AuthServer` in Application Single Sign-On (commonly called AppSSO).

An `AuthServer` can trust custom CAs. You can establish either for all `AuthServer`s or for a single `AuthServer`. This is useful when either your identity provider or storage serves certificates from a custom CA.

In most cases, CA certificates are PEM-encoded and located in a `Secret` referred from `.spec.caCerts[].secretRef.name`. The controller considers all `Secret` entries matching `*.(crt|ca-bundle)`. That means you can include multiple CA certificates in a single `Secret` or spread them across multiple `Secret`s.

After being created, an `AuthServer` reports the trusted, total custom CA certificates in its `.status.caCerts` together with the location where it sources them from. This includes the CA certificates that are trusted by all `AuthServer`s.

For example:

```
---
apiVersion: v1
kind: Secret
metadata:
  name: my-ca
  namespace: services
stringData:
  my.ca-bundle: |
    This is My Company's custom CA. It's common name is "My CA".
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: login
  namespace: services
  # ...
spec:
  caCerts:
    - secretRef:
        name: my-ca
  # ...
status:
  caCerts:
    - cert:
        subject: CN=My CA,O=My Company,C=Happyland
      source:
        secretEntry: service/my-ca[my.ca-bundle]
    - cert:
        subject: CN=My other CA,O=My Company,C=Happyland
      source:
        secretEntry: appsso/appsso-controller[controller.yaml]
  # ...
```

CA certificates configured for all `AuthServer`s by using the package installation's `ca_cert_data` are sourced from `secretEntry: appsso/appsso-controller[controller.yaml]`. This denotes the

AppSSO controller's configuration `Secret`.

Configure workloads to trust a custom CA

This topic tells you how to configure workloads to trust a custom Certificate Authority (commonly called CA) for Application Single Sign-On (commonly called AppSSO).

Overview

If your `ClientRegistration` selects an `AuthServer` that serves a certificate from a custom CA, your `Workload` does not trust it by default. This is because the certificate is not issued by a trusted certificate authority from the `Workload`'s perspective.

To establish trust between a `Workload` and an `AuthServer`:

Step	Task	Link
1.	Service Operator exports the custom CA certificate `Secret` resource from the namespace in which it is issued.	Exporting custom CA certificate Secret
2.	Service Operator imports the custom CA certificate `Secret` to the namespace in which the `Workload` is created.	Importing custom CA certificate Secret
3.	Append the deployed `Workload` as a service resource claim, denoting the custom CA certificate `Secret` in the workload namespace.	Appending custom CA certificate Secret reference to Workload

Important

These steps are mandatory if Tanzu Application Platform is installed with the default self-signed `ClusterIssuer` resource, in which the CA is custom.

Exporting custom CA certificate Secret

A `ca-certificates` service binding `Secret` allows to configure trust for custom CAs.

For more information about exporting CA certificate Secrets, see Allow Workloads to trust a custom CA AuthServer.

Example: Create a `ca-certificates`-type ServiceBinding Secret from template and offer Tanzu Application Platform's default self-signed CA certificate Secret to workloads namespace.

```
---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretTemplate
metadata:
  name: tap-ca-cert
  namespace: cert-manager                    # The namespace in which your custom CA
Secret resides.
spec:
  inputResources:
    - name: tap-ingress-selfsigned-root-ca
      ref:
        apiVersion: v1                       # The custom CA certificate Secret.
        kind: Secret                         # ^^
        name: tap-ingress-selfsigned-root-ca  # ^^
  template:
    data:
      ca.crt: $(.tap-ingress-selfsigned-root-ca.data.tls\.crt)
    stringData:
      type: ca-certificates
```

```
---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretExport
metadata:
  name: tap-ca-cert           # The name of the SecretTemplate that created the "ca-ce
rtificates" Secret.
  namespace: cert-manager     # The namespace in which Tanzu Application Platform's se
lf-signed ClusterIssuer stores its CA cert Secret.
spec:
  toNamespace: my-apps        # The namespace in which Workloads are deployed.
```

Importing custom CA certificate Secret

After the custom CA certificate Secret is exported from its original namespace, you can import it into the workloads' namespace.

Example: Accept Tanzu Application Platform's default self-signed CA certificate Secret offer.

```
---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretImport
metadata:
  name: tap-ca-cert
  namespace: my-apps            # The namespace in which Workloads are deployed.
spec:
  fromNamespace: cert-manager   # The namespace in which your custom CA certificate Se
cret resides.
```

Appending custom CA certificate Secret reference to Workload

With custom CA certificate available in the workloads' namespace, you can append it to the `Workload` as a service resource claim:

Example: Appending custom CA certificate Secret as a resource claim.

```
---
apiVersion: carto.run/v1alpha1
kind: Workload
# ...
spec:
  serviceClaims:
    - name: ca-cert
      ref:
        apiVersion: v1    # The custom CA Secret template that is imported into the wo
rkloads' namespace.
        kind: Secret      # ^^
        name: tap-ca-cert # ^^
    # ...
```

Alternatively, you can provide the workload with a `--service-ref` parameter for the same effect:

```
--service-ref "ca-cert=v1:Secret:tap-ca-cert"
```

For more information about secretgen-controller and its APIs, see secretgen-controller documentation in GitHub.

Identity providers for AppSSO

This topic tells you how to configure Application Single Sign-On (commonly called AppSSO) to use external identity providers (commonly called IdPs).

Users can log in by using external identity providers. OpenID Connect and LDAP providers are supported. SAML providers have limited experimental support. An `AuthServer` does not manage users internally. Developers can get started quickly without needing to connect to an IdP by using static hard-coded users, which is for development purposes only.

Identity providers are configured under `spec.identityProviders`, learn more from the API reference.

> **Caution**
>
> Changes to `spec.identityProviders` do not take effect immediately because the operator will roll out a new deployment of the authorization server.

End-users will be able to log in with these providers when they go to `{spec.issuerURI}` in their browser.

Use the following sections to learn how to configure identity providers for an `AuthServer`:

- OpenID Connect
- LDAP
- SAML (experimental)
- InternalUnsafe

Use the following sections for a deeper understanding of advanced identity provider configurations:

- Identity token claims mapping
- Roles claim filtering
- Roles claim mapping and filtering explained
- Configure authorization
- Restrictions

OpenID Connect providers

To set up an OpenID Connect provider, provide the following information for your `AuthServer`:

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
# ...
spec:
  identityProviders:
    - name: my-oidc-provider
      openID:
        displayName: "MyOIDC Provider"
        clientID: my-client-abcdef
        clientSecretRef:
          name: my-openid-client-secret
        configurationURI: https://openid.example.com/.well-known/openid-configuration
        issuerURI: https://openid.example.com # This field is deprecated, optional and
relies on `configurationURI`. It must be unset if `configurationURI` is set.
        scopes:
          - "openid"
          - "other-scope"
        authorizationUri: https://example.com/oauth2/authorize
```

```
        tokenUri: https://example.com/oauth2/token
        jwksUri: https://example.com/oauth2/jwks
        roles:
          fromUpstream:
            claim: "MY-OIDC-PROVIDER-GROUPS"
        idToken:
          claims:
            - fromUpstream: ""
              toClaim: ""
  # ...
---
apiVersion: v1
kind: Secret
metadata:
  name: my-openid-client-secret
  # ...
stringData:
  clientSecret: very-secr3t
```

Where:

- `.openID`: The issuer identifier. You can define as many OpenID providers as you like. If the provider supports OpenID Connect Discovery, the value of `openID` auto-configures the provider by using the information from `https://openid.example.com/.well-known/openid-configuration`.
- `.openID.configurationURI`: The OpenID Connect provider configuration endpoint, which must be suffixed with `/.well-known/openid-configuration`. This endpoint must have the HTTPS scheme. For more information, see OpenID Connect provider configuration discovery.
- `.openID.issuerURI` (optional): The issuer URI. This field is deprecated and it relies on `.openID.configurationURI`. The value of `issuerURI` must not contain `.well-known/openid-configuration` and must match the value of the `issuer` field. For more information, see the OpenID Connect documentation for your issuer, for example, at `https://openid.example.com/.well-known/openid-configuration`.

  Note

  You can retrieve the values of `issuerURI` (`https://openid.example.com`) and `clientID` (`my-client-abcdef`) when registering a client with the provider, which in most cases, is by using a web UI.

  You can also run the following to retrieve the correct `issuerURI` value from the upstream identity provider:

  ```
  curl -s "https://openid.example.com/.well-known/openid-configuration" | jq -r
  ".issuer"
  ```

- `.openID.displayName` (optional): A user-friendly display name for the provider that is rendered on the login page.
- `.openID.scopes`: The scopes requested to the issuer in the authorization request. Its value must contain `"openid"`. Other common `openID.scopes` values include `"profile"` and `"email"`.
- `.openID.clientSecretRef`: The issuer's client secret. The value of `clientSecretRef` must be a `Secret` with the entry `clientSecret`.

- `.openID.authorizationUri` (optional): The URI for performing an authorization request and obtaining an `authorization_code`. If the `configurationURI` and `issuerURI` fields are not set, this field must not be empty.
- `.openID.tokenUri` (optional): The URI for performing a token request and obtaining a token. If the `configurationURI` and `issuerURI` fields are not set, this field must not be empty.
- `.openID.jwksUri` (optional): The JSON Web Key Set (JWKS) endpoint for obtaining the JSON Web Keys to verify token signatures. If the `configurationURI` and `issuerURI` fields are not set, this field must not be empty.
- `.openID.userinfoUri` (optional): The URI for obtaining the additional identity claims from the upstream IdP (identity provider).
- `.openID.roles.fromUpstream.claim` (optional): Selects which claim in the `id_token` contains the `roles` of the user. `roles` is not a standard OpenID Connect claim. If `ClientRegistrations` or `ClassClaims` have a `roles` scope, it populates the `roles` claim in the `id_token` issued by the `AuthServer`. For more information, see OpenID external groups mapping.
    - `MY-OIDC-PROVIDER-GROUPS`: Claim from the ID token issued by `my-oidc-provider` is mapped into the `roles` claim in the id tokens issued by AppSSO.
- `.openID.idToken.claims`: Allows mapping a claim from an upstream identity provider to the current authorization server. See Identity token claims mapping for more details.

Verify the configuration by visiting the `AuthServer`'s issuer URI in your browser and select `my-oidc-provider`.

You can find the `authorizationUri`, `tokenUri`, `jwksUri`, and `userinfoUri` properties in `<issuerUri>/.well-known/openid-configuration`.

OpenID external groups mapping

Service operators may map the identity provider's "groups" (or equivalent) claim to the `roles` claim within an `AuthServer`'s identity token.

Note

Read more about roles claim mapping and filtering here

App Operators may configure their ClientRegistration to have the `roles` claim included in the `id_token`.

Configure `AuthServer` with OpenID Connect groups mapping:

```
spec:
  identityProviders:
    - name: "openid-idp"
      openid:
        scopes:
          - upstream-identity-providers-groups-claim # Optional based on the identity
provider.
        roles:
          fromUpstream:
            claim: "upstream-identity-providers-groups-claim"
```


Caution

> Some OpenID providers, such as Okta OpenID, might require requesting the roles or groups scope from the identity provider, as a result, you must include it in the `.openid.scopes` list.

For every ClientRegistration that has the `roles` scope listed, the identity token is populated with the `roles` claim:

```
kind: ClientRegistration
metadata:
  name: my-client-registration
spec:
  scopes:
    - name: openid
    - name: roles
  # ...
```

This rule also applies to ClassClaim:

```
kind: ClassClaim
metadata:
  name: my-class-claim
spec:
  parameters:
    scopes:
      - name: openid
      - name: roles
  # ...
```

When groups are mapped (as described above), all the groups provided by the identity provider are retrieved, and the relevant groups that the logged-in user is part of are appended to the `roles` claim of an `id_token`. To filter the available roles within an `id_token`, see Roles claim filtering section.

Note for registering a client with the identity provider

The `AuthServer` will set up redirect URIs based on the provider name in the configuration. For example, for a provider with `name: my-provider`, the redirect URI will be `{spec.issuerURI}/login/oauth2/code/my-provider`. The externally accessible user URI for the `AuthServer`, including scheme and port is `spec.issuerURI`. If the `AuthServer` is accessible on `https://appsso.company.example.com:1234/`, the redirect URI registered with the identity provider should be `https://appsso.company.example.com:1234/login/oauth2/code/my-provider`.

OpenID Connect provider configuration discovery

OpenID Connect provider configuration discovery is enabled when you set the `AuthServer.spec.identityProviders[*].openID.configurationURI` field.

This field has the following requirements:

- It must be a valid URI.
- It must have the HTTPS scheme. For more information about testing with the HTTP scheme, see Non-HTTPS configuration.
- It must be suffixed with `/.well-known/openid-configuration`.

Non-HTTPS configuration

For testing purposes, you can add an annotation of `sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""` to the `AuthServer` metadata for communicating with the insecure upstream OpenID Connect providers by using HTTP scheme.

Caution

Use this option only in the testing or prototyping environments.

Configure custom CA

For more information about using an upstream OpenID provider with a custom or private certificate authority (CA), see CA certificates for Application Single Sign-On.

Discovery mechanism

Application Single Sign-On attempts to resolve the configuration document located at the designated configuration URI. The Application Single Sign-On client attempts to wait for at most 10 seconds to receive a response. If there is no response or the response status is not HTTP `200 OK`, the `AuthServer` resource enters an authorization server configuration error state and provides a corresponding error message. The client re-attempts discovery at least 1 minute after the last failed discovery attempt.

The Application Single Sign-On client supports connecting to OpenID Connect providers configured with the TLS protocol v1.2 and later.

The Application Single Sign-On client attempts to discover the following OpenID Connect fields:

- `authorization_endpoint`
- `token_endpoint`
- `jwks_uri`
- `userinfo_endpoint`

Supported token signing algorithms

AppSSO only supports the `RS256` algorithm for token signature. For more information, see OpenID Connect documentation.

You can find out the signing algorithms your OpenID provider supports by referring to the `id_token_signing_alg_values_supported` response parameter in the OpenID Connect documentation at `.well-known/openid-configuration`.

For example, you can run:

```
curl -s "ISSUER-URI/.well-known/openid-configuration" | jq ".id_token_signing_alg_valu
es_supported"
```

LDAP

Important

You can not configure more than one `ldap` identity provider.

For example:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
# ...
spec:
  identityProviders:
    - name: ldap
      ldap:
        url: "ldaps://example.com:636"
        bind:
          dn: uid=binduser,ou=Users,dc=example,dc=com
          passwordRef:
            name: ldap-password
        user:
          searchBase: ou=Users,dc=example,dc=com
          searchFilter: cn={0}
        roles:
          fromUpstream:
            attribute: cn
            search:
              filter: member={0}
              base: ou=Users,dc=example,dc=com
              searchSubTree: true
              depth: 10
          filterBy:
            - exactMatch: "users"
            - regex: "^admin"
        idToken:
          claims:
            - fromUpstream: ""
              toClaim: ""
        group: # DEPRECATED, use 'roles' instead
          search:
            filter: member={0}
            base: ou=Users,dc=example,dc=com
            searchSubTree: true
            depth: 10
          roleAttribute: cn
  # ...
---
apiVersion: v1
kind: Secret
metadata:
  name: ldap-password
  namespace: default
stringData:
  password: confidential-password-value
```

Where:

- `.ldap.url`: The URL of the LDAP server. It must be `ldaps` and must contain a port.
- `.ldap.bind.dn`: The DN to perform the bind.
- `.ldap.bind.passwordRef`: Must be a secret with the entry `password`, which is the password to perform the bind.
- `.ldap.user.searchBase`: The branch of tree where the users are located at. Search is performed in the nested entries.
- `.ldap.user.seachFilter`: The filter for LDAP search. It must contain the string `{0}`, which is replaced by the `dn` of the user when performing a search. For example, when logging in with the user name `marie`, the filter for LDAP search is `cn=marie`.

- `.ldap.roles` (optional): Configures how LDAP groups are mapped to user roles in the `id_token` claims. Defaults to unset, meaning the user has no roles.
- `.ldap.roles.fromUpstream.attribute`: Selects which attribute of the group entry are mapped to a user role. If an attribute has multiple values, the first value is selected.
- `.ldap.roles.fromUpstream.search` (optional): Toggles OpenLDAP-style group search, optionally uses recursive search to find groups for a user.
- `.ldap.roles.filterBy` (optional): Applies roles claim filters. See Roles claim filtering section for more details.
- `.ldap.idToken.claims`: Allows for mapping a claim from an upstream identity provider to the current authorization server. See Identity token claims mapping for more details.

  LDAP providers have the following claims mapped by default:

LDAP attribute	Id token claim
givenname	given_name
sn	family_name
cn	name
mail	email
telephonenumber	phone_number

The following fields are deprecated:

- `.ldap.group` (optional): Configures how LDAP groups are mapped to user roles in the `id_token` claims. If not set, the user has no roles. Use `.ldap.roles` instead.
- `.ldap.group.roleAttribute`: Selects which attributes of the group entry are mapped to a user role. If an attribute has multiple values, the first value is selected.
- `.ldap.group.search` (optional): Toggles Active Directory search and uses recursive search to find groups for a given user.

Verify the configuration by visiting the `AuthServer`'s issuer URI in your browser and log in with the user name and password from LDAP.

LDAP external groups mapping

Service operators can map the identity provider's "groups" or equivalent attribute to the `roles` claim within an `AuthServer`'s identity token.

Note

Read more about roles claim mapping and filtering here

Configure `AuthServer` with LDAP groups attribute mapping:

```
spec:
  identityProviders:
    - name: "ldap-idp"
      ldap:
        roles:
          fromUpstream:
            attribute: "upstream-identity-providers-groups-attribute" # e.g. "cn" or
"dn"
```

For every ClientRegistration that has the `roles` scope listed, the identity token is populated with the `roles` claim:

```
kind: ClientRegistration
metadata:
  name: my-client-registration
spec:
  scopes:
    - name: openid
    - name: roles
  # ...
```

This rule also applies to ClassClaim:

```
kind: ClassClaim
metadata:
  name: my-class-claim
spec:
  parameters:
    scopes:
      - name: openid
      - name: roles
  # ...
```

When groups are mapped (as described above), all the groups provided by the identity provider are retrieved, and the relevant groups that the logged-in user is part of are appended to the `roles` claim of an `id_token`. To filter the available roles within an `id_token`, see Roles claim filtering section.

ActiveDirectory group search

In ActiveDirectory groups, user entries have a multi-value `memberOf` attribute, which contains the DNs pointing to the groups of a particular user. To enable this search mode, make sure `roles.fromUpstream.attribute` is set and `roles.fromUpstream.search` is not set.

For example:

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
# ...
spec:
  identityProviders:
    - name: ldap
      ldap:
        # ...
        user:
          searchBase: OU=Cloud,DC=ad,DC=example,DC=com
          searchFilter: cn={0}
        roles:
          fromUpstream:
            attribute: sAMAccountName
```

The LDIF definition is as follows:

```
dn: CN=appsso-user,OU=Cloud,DC=ad,DC=example,DC=com
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: user
cn: appsso user
sn: user
```

```
givenName: appsso
distinguishedName: CN=appsso user,OU=Cloud,DC=ad,DC=example,DC=com
displayName: appsso user
memberOf: CN=ssogroup,OU=Cloud,DC=ad,DC=example,DC=com
memberOf: CN=developers,OU=Cloud,DC=ad,DC=example,DC=com
sAMAccountName: appssouser
userPrincipalName: appssouser@ad.example.com
objectCategory: CN=Person,CN=Schema,CN=Configuration,DC=ad,DC=example,DC=com
# ...

# Groups
dn: CN=ssogroup,OU=Cloud,DC=ad,DC=example,DC=com
objectClass: top
objectClass: group
cn: ssogroup
member: CN=appsso-user,OU=Cloud,DC=ad,DC=example,DC=com
sAMAccountName: SSO Group
# ...

dn: CN=developers,OU=Cloud,DC=ad,DC=example,DC=com
objectClass: top
objectClass: group
cn: developers
sAMAccountName: Developers
# ...
```

The user `appsso-user` has two values for `memberOf`, pointing to two groups. Given the configuration earlier, `sAMAccountName` is used for the role, so the user has `SSO Group` and `Developers` as roles. The group is not required to have `member` attribute point to the user for the role to be mapped.

“Classic” group search

In non-ActiveDirectory LDAP, users generally do not have a `memberOf` attribute. Group search is performed by looking up groups in a base branch and filtering based on the groups `member` attribute.

An AuthServer can optionally perform:

- group search in sub-branches.
- nested group search, that is, find a hierarchy of groups, in which a group is a member of another group.

The complete configuration is as follows:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
# ...
spec:
  identityProviders:
    - name: ldap
      ldap:
        # ...
        roles:
          fromUpstream:
            attribute: description
            search:
              base: ou=Users,dc=example,dc=com
              filter: member={0}
              depth: 10
              searchSubTree: true
  # ...
```

Where:

- `search.base` is the base for running an LDAP search for groups.
- `search.filter` is the filter for running an LDAP search for groups. It must contain the string `{0}`, which is replaced by the `dn` of the user when performing group search. For example, `member=cn=marie,ou=Users,dc=example,dc=org`.
- `search.depth` (optional) is the depth at which to perform nested group search. It defaults to unset if left empty.
- `search.searchSubTree` (optional) controls whether to look for groups in sub-trees of the `search.base`. It defaults to unset if left empty.

Direct group search only

Given the following minimal configuration:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
# ...
spec:
  identityProviders:
    - name: ldap
      ldap:
        # ...
        user:
          searchBase: ou=Users,dc=example,dc=com
          searchFilter: uid={0}
        roles:
          fromUpstream:
            attribute: description
            search:
              base: ou=Users,dc=example,dc=com
              filter: member={0}
  # ...
```

The LDIF definition is as follows:

```
######################
## Users
######################
## User Marie Curie
## Marie Salomea Skłodowska Curie ; https://en.wikipedia.org/wiki/Marie_Curie
dn: cn=marie,ou=Users,dc=example,dc=org
cn: Marie
sn: Skłodowska Curie
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: shadowAccount
uid: marie

######################
## Groups
######################
dn: cn=nobels,ou=Users,dc=example,dc=org
objectClass: groupOfNames
description: Nobel Prizes
member: cn=marie,ou=Users,dc=example,dc=org
```

User `marie` has roles `Nobel Prizes`.

Groups in sub-trees

AppSSO can perform group search in sub-trees of the base for group search. This is enabled when `roles.fromUpstream.search.searchSubTree` is explicitly set to `true`. For example:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
# ...
spec:
  identityProviders:
    - name: ldap
      ldap:
        # ...
        roles:
          fromUpstream:
            attribute: description
            search:
              base: ou=Users,dc=example,dc=com
              filter: member={0}
              searchSubTree: true
  # ...
```

The LDIF definition is as follows:

```
######################
## Users
######################
## User Corazon
## Maria Corazon Sumulong Cojuangco Aquino; https://en.wikipedia.org/wiki/Corazon_Aqui
no
dn: cn=corazon,ou=Users,dc=example,dc=com
cn: Maria Corazon
sn: Sumulong Cojuangco Aquino
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: shadowAccount
uid: corazon


######################
## Groups
######################
dn: cn=presidents,ou=Users,dc=example,dc=com
objectClass: groupOfNames
description: Presidents
member: cn=corazon,ou=Users,dc=example,dc=com

dn: cn=chief-commanders,ou=LegionHonor,ou=Users,dc=example,dc=com
objectClass: groupOfNames
description: Chief Commanders
member: cn=corazon,ou=Users,dc=example,dc=com
```

User `corazon` has roles `Presidents` and `Chief Commanders`, which are retrieved from `ou=LegionHonor,ou=Users,dc=example,dc=com`, a subtree of the search base.

Nested group search

AppSSO can perform nested group search by going up a chain where a user is a member of a group, which is itself a member of a group, and so on. This is enabled by setting

`roles.fromUpstream.search.depth` to greater than `1`. `roles.fromUpstream.search.depth` controls the number of “levels” that AppSSO fetches to get the groups of a user.

For example:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
# ...
spec:
  identityProviders:
    - name: ldap
      ldap:
        # ...
        user:
          searchBase: ou=Users,dc=example,dc=com
          searchFilter: uid={0}
        roles:
          fromUpstream:
            attribute: description
            search:
              base: ou=Users,dc=example,dc=com
              filter: member={0}
              depth: 2
  # ...
```

The LDIF definition is as follows:

```
######################
## Users
######################
## User Corazon
## Maria Corazon Sumulong Cojuangco Aquino; https://en.wikipedia.org/wiki/Corazon_Aqui
no
dn: cn=corazon,ou=Users,dc=example,dc=com
cn: Maria Corazon
sn: Sumulong Cojuangco Aquino
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: shadowAccount
uid: corazon


######################
## Groups
######################
# Citizen > Politicians > Presidents > corazon (depth = 3)
dn: cn=citizens,ou=Users,dc=example,dc=com
objectClass: groupOfNames
description: Citizens
member: cn=politicians,ou=Users,dc=example,dc=com

# Politicians > Presidents > corazon (depth = 2)
dn: cn=politicians,ou=Users,dc=example,dc=com
objectClass: groupOfNames
description: Politicians
member: cn=presidents,ou=Users,dc=example,dc=com

# Presidents > corazon (depth = 1, direct group)
dn: cn=presidents,ou=Users,dc=example,dc=com
objectClass: groupOfNames
description: Presidents
member: cn=corazon,ou=Users,dc=example,dc=com
```

User `corazon` has roles `Presidents` and `Politicians`. However, the search stops at depth 2, so they do not have the role `Citizens`, which requires a depth greater or equal to 3.

SAML (experimental)

Caution

Support for SAML providers is experimental.

For SAML providers only autoconfiguration through `metadataURI` is supported.

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
# ...
spec:
  identityProviders:
  - name: my-saml-provider
    saml:
      metadataURI: https://saml.example.com/sso/saml/metadata # required
      displayName: "My SAML provider" # optional
      idToken: # optional
        claims:
          - fromUpstream: "" # SAML attribute
            toClaim: ""      # claim that is part of an AppSSO id_token
```

- `.saml.displayName` (optional): A user-friendly display name for the provider that is rendered on the login page.
- `.saml.idToken.claims`: Allows for mapping a SAML attribute from an upstream SAML identity provider to the current authorization server. For more information, see Identity token claims mapping.

SAML external groups mapping

Service operators may map the identity provider's "groups" (or equivalent) attribute to the `roles` claim within an `AuthServer`'s identity token.

Note

Read more about roles claim mapping and filtering here

Configure `AuthServer` with SAML role attribute:

```
spec:
  identityProviders:
    - name: "saml-idp"
      saml:
        roles:
          fromUpstream:
            attribute: "saml-group-attribute"
```

For every ClientRegistration that has the `roles` scope listed, the identity token will be populated with the `roles` claim:

```
kind: ClientRegistration
metadata:
```

```
  name: my-client-registration
spec:
  scopes:
    - name: openid
    - name: roles
  # ...
```

When groups are mapped (as described above), all the groups provided by the identity provider are retrieved, and the relevant groups that the logged-in user is part of are appended to the `roles` claim of an `id_token`. To filter the available roles within an `id_token`, see Roles claim filtering section.

Note for registering a client with the identity provider

The `AuthServer` will set up SSO and metadata URLs based on the provider name in the configuration. For example, for a SAML provider with `name: my-provider`, the SSO URL will be `{spec.issuerURI}/login/saml2/sso/my-provider`. The metadata URL will be `{spec.issuerURI}/saml2/service-provider-metadata/my-provider`. `spec.issuerURI` is the externally accessible issuer URI for an `AuthServer`, including scheme and port. If the `AuthServer` is accessible on `https://appsso.company.example.com:1234/`, the SSO URL registered with the identity provider should be `https://appsso.company.example.com:1234/login/saml2/sso/my-provider`.

InternalUnsafe

> **Caution**
>
> The internal provider is **unsafe** and **must not** be used in production environments.

`InternalUnsafe` must be explicitly allowed by setting the annotation `sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""`.

During development, static users can be useful for testing purposes.

You cannot configure more than one `internalUnsafe` identity provider.

For example:

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: "" # required annotation
when using the internal unsafe provider
  # ...
spec:
  identityProviders:
    - name: test-users
      internalUnsafe:
        users:
          - username: ernie
            password: "password" # plain text
            roles:
              - "silly"
            claims: # custom identity token claims
              given_name: "Bert"
              family_name: "Muppet"
              middle_initial: "H"
              email: "bert@muppets.example.com"
              alt_address: "123 Sesame Street"
```

```
          - username: bert
            password: "{bcrypt}$2a$10$201z9o/tHlocFsHFTo0plukh03ApBYe4dRiXcqeyRQH6CNNt
S8jWK" # bcrypt-hashed "password"
            roles:
              - "grumpy"
            claims: # custom identity token claims
              alt_address: "456 Fake Street"
              middle_initial: "T"
  # ...
```

The passwords can be plain text or bcrypt-hashed. Bcrypt-hashing passwords must be prefixed with `{bcrypt}`. For more information about the bcrypt-hash string, see Generating a bcrypt hash from a plain-text password.

Verify the configuration by visiting the `AuthServer`'s issuer URI in your browser and logging in as `ernie/password` or `bert/password`.

To use custom claims defined within `.internalUnsafe.users[*].claims`, you must include the `profile` scope within your `ClientRegistration` resource.

Generating a bcrypt hash from a plain-text password

There are multiple options for generating bcrypt hashes:

1. Use an online bcrypt generator
2. On Unix platforms, use `htpasswd`. Note, you may need to install it, for example on Ubuntu by running `apt install apache2-utils`

   ```
   htpasswd -bnBC 12 "" your-password-here | tr -d ':\n'
   ```

Identity token claims mapping

Note

This section is applicable to OpenID, LDAP, and SAML (experimental) identity provider configurations.

Service Operators can control which claims appear in an `AuthServer`-issued `id_token`, and how to obtain this value from an upstream identity provider.

For example, consider the following configuration:

```
spec:
  identityProviders:
  - name: my-identity-provider
    openid:
      idToken:
        claims:
          - fromUpstream: "title"
            toClaim: "job_title"
```

With this, if the upstream OpenID identity provider makes a `"title"` claim available with the value `"developer"`, the corresponding AuthServer-issued `id_token` contains a claim `"job_title": "developer"`.

This field also allows for mapping the standard OpenID claims, such as `given_name`, `family_name`, and non-standard claim, such as the `job_title` claim in the earlier example. When mapping the standard claims, the field types are preserved according to the original OpenID specification. For example, the `given_name` claim is defined as a string in the OpenID specification, so if a custom

claim is mapped into this field, the value of the custom claim is coerced into a string type. If the custom claim is an array type, the first value of the array is used.

Important

For the custom claims defined `.spec.identityProviders[*].{openid,ldap,saml}.idToken.claims` to be available in `AuthServer`-issued `id_token`s, App Operators must include the `profile` scope within their `ClientRegistration` resource.

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
# ...
spec:
  identityProviders:
  - name: my-identity-provider
    {openid,ldap,saml}:
      idToken: # optional
        claims:
          - fromUpstream: "" # identity provider claim name
            toClaim: ""      # claim that is part of an AppSSO id_token
```

Constraints

You must adhere to the following identity token mapping constraints:

- Multiple upstream identity provider claims cannot be mapped to a single `AuthServer` identity token claim.
- The value of `claims.fromUpstream` is case-insensitive for LDAP identity providers, and is case-sensitive for OpenID and SAML identity providers.
- `idToken.claims.toClaim` is case-sensitive.
- Application Single Sign-On collects claims only from the upstream `id_token` and, for OpenID identity providers, from the `userinfo` endpoint defined in `.openID.userinfoUri`.
- Reserved claims can not be mapped to. Reserved claims are listed as follows:

Reserved Claim Names
`roles`
`acr`
`amr`
`at_hash`
`auth_time`
`azp`
`c_hash`
`nonce`
`aud`
`exp`
`iat`

Reserved Claim Names
`iss`
`jti`
`nbf`
`sub`

Roles claim filtering

Note

This section is applicable to OpenID, LDAP, and SAML (experimental) identity provider configurations.

Once an external groups mapping has been applied (as described per identity provider above), AppSSO is able to retrieve all the groups from an identity provider. An identity provider may have hundreds of groups, and any particular user may be part of a large subset of those groups. When a user logs in, those groups will be appended to their `id_token`'s `roles` claim. This section describes how to filter the `roles` claim.

To filter groups, for a supported identity provider, apply:

```
spec:
  identityProviders:
    - name: my-provider
      <idp>:
        roles:
          filterBy:
            - exactMatch: ""
            - regex: ""
```

where `<idp>` may be `openid`, `ldap`, or `saml`.

Filters are disjunctive ("OR" operator), so each filter is applied to the entire set of groups, and merged into a set of unique filtered groups values. See filter examples for more information.

Roles claim filters

Available filters are:

- `exactMatch` - match the groups exactly. This filter is **case-sensitive**. e.g. `exactMatch: "developer"` will match only the group named "developer" and no other.
- `regex` - match groups according to the defined regular expression pattern. , and This filter is **case-insensitive**. e.g. `regex: ^admin` will match groups starting with the word "admin".
    - The regular expression pattern syntax used is RE2
    - Expressions should not be surrounded by forward slashes (`/`) and should only contain the pattern (e.g. `.*`, `^dev`, `\w+`).

Roles claim filter examples

Given an example set of groups retrieved from a hypothetical identity provider:

```
it-admin
it-developer
devops-user
```

```
devops-admin
devops-developer
product-user
product-developer
org-user
hr-user
hr-admin
```

Basic exact match filters

```
- exactMatch: "product-user"
- exactMatch: "org-user"
```

returns:

```
product-user
org-user
```

Basic regular expression (RegEx) filters

```
- regex: ".*-developer"
```

returns:

```
it-developer
devops-developer
product-developer
```

Multiple regular expression (RegEx) filters

```
- regex: ".*-developer"
- regex: "^it"    # starts with "it"
- regex: "admin$" # ends with "admin"
- regex: "\w+"    # at least one word or more
```

returns:

```
it-admin
it-developer
devops-admin
devops-developer
product-developer
hr-admin
```

Note

filters are disjunctive and so multiple filters can filter down the same values, but the resulting set will always have unique values.

Exact match and regular expression (RegEx) filters

```
- exact-match: "hr-admin"
- exact-match: "org-user"
- regex: "developer$"    # ends with "developer"
```

returns:

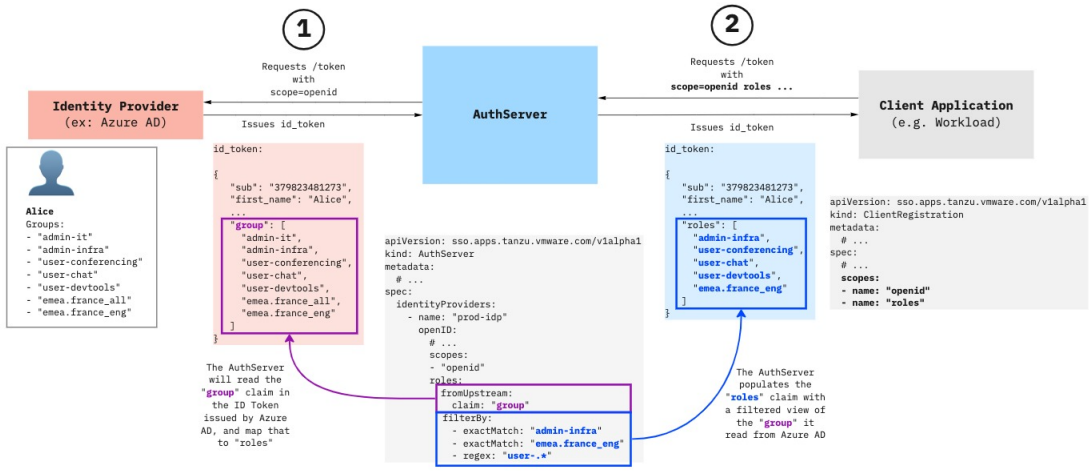
```
it-developer
devops-developer
```

```
product-developer
org-user
hr-admin
```

Roles claim mapping and filtering explained

When issuing an `id_token`, OpenID providers may include a (non-standard) claim describing the "groups" the user belongs to, the "roles" of the user, or something similar. This claim is identity provider specific. For example, Azure AD uses the "group" claim by default, and allows administrators to select the name of the claim.

Service Operators may choose to make these "groups", "roles", or equivalent, available in the id_token issued by an AppSSO `AuthServer`, in the `roles` claim, with filtering rules applied.



Restrictions

Each identity provider has a declared `name`. The following conditions apply:

- the names must be unique
- the names must not be blank
- the names must follow Kubernetes' DNS Subdomain Names guidelines
  - contain no more than 253 characters
  - contain only lowercase alphanumeric characters, '-' or '.'
  - start with an alphanumeric character
  - end with an alphanumeric character
- the names may not start with `client` or `unknown`

There can be at most one of each `internalUnsafe` and `ldap`.

Configure authorization for AppSSO

This topic tells you how to configure authorization for Application Single Sign-On (commonly called AppSSO).

Note

This topic is applicable to Internal, OpenID, LDAP, and SAML (experimental) identity provider `AuthServer` configurations. For more information, see AuthServer.

Overview

An application or `Workload` can protect certain resources based on user's level of authorization. Within OAuth 2, the application with protected resources, the Resource Server, verifies if the access token provided contains the scopes to perform an action on a protected resource.

The following excerpt is from a Spring Boot application, OAuth2 Resource Server, protecting its message API endpoints `/message/**`:

```
http.authorizeExchange(exchanges -> exchanges
                        .pathMatchers("/message/**").hasAuthority("SCOPE_message.rea
d")
                        .anyExchange().authenticated()
        )
```

The access token to access any endpoint under `/message/` path must have `message.read` scope within its access token.

For full example, see Spring Security documentation.

The following sections describe how to configure the mapping of user roles to authorization scopes at `AuthServer` identity provider and `ClientRegistration` levels.

Retrieving external groups or roles

Important

Skip this section if you work with an internal unsafe provider. External groups mapping is not required because roles are defined in the specifications.

To configure authorization for an identity provider, you must define from which claim or attribute the upstream identity provider supplies the groups or roles that a user is part of:

- OpenID external groups mapping
- LDAP external groups mapping
- SAML (experimental) external groups mapping

After external groups mapping is complete, and groups or roles are retrievable, you can optionally filter the roles that are appended to an identity token. For more information about how to filter roles, see Roles claim filtering.

Mapping individual roles into authorization scopes

After external groups or roles are mapped to AppSSO's `roles` claim, and optionally filtered for the desired set of retrieved roles, you can map each role to the desired authorization scopes.

For example, given a retrieved role "hr", any client authorizing by using `my-openid-provider` can request scopes `hr.read` or `hr.write`, provided that the client registered the scopes in `ClientRegistration.spec.scopes`:

```
kind: AuthServer
# ...
spec:
```

```
  identityProviders:
    - name: my-openid-provider
      openid:
        accessToken:
          scope:
            rolesToScopes:
              - fromRole: "hr" # -> Role "hr" is mapped to the "hr.read" and "hr.writ
e" scopes.
                toScopes:      #    Only users with the "hr" role can be issued access
token with these scopes.
                  - "hr.read"  # ^^
                  - "hr.write" # ^^
```

For example, given that a `ClientRegistration` is applied to include `hr.read` or `hr.write`:

```
kind: ClientRegistration
# ...
spec:
    scopes:
    - name: "roles" # Must request special 'roles' scope.
    - name: "hr.read"
    - name: "hr.write"
```

Any client can request an access token with the scopes, but an access token can be issued with those scopes only if the user that is being authorized has the role `hr` in the upstream identity provider.

If the user has the role `hr`, he or she must consent to allow the application to request the scopes. After the consent is provided, the user can access resources limited to the `hr.read` and `hr.write` scopes within the application by using their access token.

Default authorization scopes

You can define authorization scopes that are automatically granted to all users within an identity provider, regardless of user role.

For example, given an `AuthServer` with an OpenID provider, with defined authorization scope defaults:

```
kind: AuthServer
# ...
spec:
    identityProviders:
    - name: my-openid-provider
      openid:
        accessToken:
            scope:
                defaults:
                - "developer.read"
                - "developer.write"
                - "developer.delete"
```

For example, given that a `ClientRegistration` is applied to include any of the default scopes:

```
kind: ClientRegistration
# ...
spec:
    scopes:
    - name: "roles" # Must request special 'roles' scope.
    - name: "developer.read"
```

When an application or `Workload` is registered by using the `ClientRegistration`, that application, on behalf of the user, can request and be granted with the scope `developer.read` automatically within the issued access token. The user must consent to allow the application to request the scope. After the consent is provided, the user can access resources limited to the `developer.read` scope within the application by using their access token.

The following is a full sample of authorization configurations and the accompanying `ClientRegistration` configurations to allow clients to request the scopes:

```
kind: AuthServer
# ...
spec:
  identityProviders:
    - name: my-openid-provider
      openid:
        roles:
          fromUpstream:
            claim: "groups"            # -> Map the upstream identity provider's exter
nal groups or roles claim.
          filterBy:                    # -> Optionally filter the groups or roles retr
ieved from identity provider.
            - exactMatch: "finance"    # ^^
            - exactMatch: "hr"         # ^^
            - exactMatch: "marketing"  # ^^
        accessToken:
          scope:
            defaults:                  # -> Optional default scopes granted to any use
r within the identity provider.
              - "developer.read"       # ^^
              - "developer.write"      # ^^
              - "developer.delete"     # ^^
            rolesToScopes:
            - fromRole: "hr"           # -> Role "hr" is mapped to "hr.read", "hr.writ
e" scopes.
              toScopes:                #    Only users with "hr" role can be issued ac
cess token with these scopes.
                - "hr.read"            # ^^
                - "hr.write"           # ^^
            - fromRole: "finance"      # -> Role "finance" is mapped to "finance" scop
e.
              toScopes:                #    Only users with "finance" role can be issu
ed an access token with this scope.
                - "finance"            # ^^
            - fromRole: "marketing"    # -> Role "marketing" is mapped to "marketing-r
eader", "marketing-writer" scopes.
              toScopes:                #    Only users with "marketing" role can be is
sued an access token with these scopes.
                - "marketing-reader"   # ^^
                - "marketing-writer"   # ^^
```

```
kind: ClientRegistration
# ...
spec:
  scopes:
    - name: "roles" # Must request special 'roles' scope.
    - name: "developer.read"
    - name: "developer.write"
    - name: "developer.delete"
    - name: "hr.read"
    - name: "hr.write"
    - name: "finance"
    - name: "marketing-reader"
    - name: "marketing-writer"
```

Public clients and CORS for AppSSO

This topic tells you how to configure Application Single Sign-On (commonly called AppSSO) to use public clients.

Overview

A public client is a client application that does not require credentials to obtain tokens, such as single-page apps (SPAs) or mobile devices. Public clients rely on Proof Key for Code Exchange (PKCE) Authorization Code flow extension.

Follow these steps to configure an `AuthServer` and `ClientRegistration`s for use with public clients:

1. Specify allowed HTTP origin (one or more) by using the `AuthServer.spec.cors` API.

   The authorization server relaxes the same-origin policy for the specified domain (one or more), enabling browser-based, single-page applications to interact with the designated authorization server. For more information, see CORS configuration.

2. Set `clientAuthenticationMethod` to `none` within `ClientRegistration` resource.

   Native applications and browser-based applications are considered public clients and must not rely on client credentials. Instead, PKCE must be used. Setting `clientAuthenticationMethod: none` ensures client credentials are not used, and makes PKCE mandatory for those clients. For more information, see Client authentication.

CORS configuration

A browser-based public client can interact with an `AuthServer` if the `AuthServer` has the public clients' origin (one or more) specified in `AuthServer.spec.cors`.

VMware recommends designating specific origins as follows:

```
kind: AuthServer
# ...
spec:
  cors:
    allowOrigins:
    - "https://example.com"         # Specific domain.
    - "https://mydept.example.com" # Specific subdomain.
    - "https://*.example.com"       # Subdomain wildcard notation.
    - "https://*.apps.example.com" # Subdomain wildcard notation.
```

You can also designate that all origins are allowed as follows:

```
kind: AuthServer
metadata:
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-cors: ""
spec:
  cors:
    allowAllOrigins: true
```


Caution

Do not allow all origins in production environments.

You must use the `allow-unsafe-cors` annotation when allowing all origin allowance. The `AuthServer` sends the `Access-Control-Allow-Origin: *` HTTP response header.

Requirements for allowed origin designations include:

- Only `allowOrigins` or `allowAllOrigins` is allowed to be set.
- When using `allowAllOrigins`, you must explicitly set the annotation `sso.apps.tanzu.vmware.com/allow-unsafe-cors: ""`. This is an acknowledgement that using `allowAllOrigins` is inherently unsafe.

Client authentication

When configuring a `ClientRegistration` for a public client, you must set your client authentication method to `none` and ensure that your public client supports Authorization Code with PKCE. With PKCE, the client does not authenticate, but presents a code challenge and subsequent code verifier to establish trust with the authorization server.

To set Client Authentication Method to `none`, ensure your `ClientRegistration` resource defines the following:

```
kind: ClientRegistration
# ...
spec:
  clientAuthenticationMethod: none
```

Public clients supporting Authorization Code with PKCE flow ensure that:

- On every OAuth `authorize` request, parameters `code_challenge` and `code_challenge_method` are provided. Only `code_challenge_method=S256` is supported.
- On every OAuth `token` request, parameter `code_verifier` is provided. Public clients do not provide a Client Secret because they are not tailored to retain any secrets in public view.

For public clients, the `AuthServer` only supports the Authorization Code Flow: `response_type=code`, because public clients such as single-page apps cannot safely store sensitive information such as client secrets.

Client credentials code grant

Some single-page applications require obtaining a token by using the `client_credentials`. This is not a recommended practice because a browser-based app cannot protect its `client_secret`.

In this scenario, the `ClientRegistration` cannot use `none` as its `clientAuthenticationMethod`. It must use either `client_secret_basic` or `client_secret_post`. For example:

```
kind: ClientRegistration
# ...
spec:
  authorizationGrantTypes:
  - client_credentials
  clientAuthenticationMethod: client_secret_basic
```

Additional CORS configuration

When you define either `allowOrigins` or `allowAllOrigins`, default values are set for other CORS-related headers. VMware provides configuration options to customize some of these headers. The following is the full CORS configuration using `allowOrigins`:

```
kind: AuthServer
# ...
spec:
  cors:
```

```
    allowOrigins:
    - "https://example.com"
    allowMethods:
    - GET
    - POST
    - OPTIONS
    allowHeaders:
    - Authorization
    exposeHeaders: []
    allowCredentials: false
```

- `allowMethods` defines the list of HTTP methods allowed. The values are sent back in the response to pre-flight requests, in the `Access-Control-Allow-Methods` header. When a pre-flight request is issued with the `Access-Control-Request-Method` header set, the value is checked against `allowMethods`. If it matches, the request completes. If it does not match, the server answers with an HTTP 403 Unauthorized status. The valid values are either `["*"]` or any combination of `[GET, HEAD, POST, PUT, PATCH, DELETE, OPTIONS, TRACE]`. The values are case-sensitive and default to `["GET", "POST", "OPTIONS"]`.
- `allowHeaders` defines the list of headers allowed. When a client issues a pre-flight request with values in the `Access-Control-Request-Headers` header, only the values that are also present in `AllowHeaders` are sent back in the response's `Access-Control-Allow-Headers` header. The values are case-insensitive and default to `["Authorization"]`.
- `exposeHeaders` defines the values of the `Access-Control-Expose-Headers` header in the response to a CORS request. The default value is `[]`.
- `allowCredentials` defines the value of the `Access-Control-Allow-Credentials` header in the response to a CORS request. The default value is `false`.

References

- Proof Key for Code Exchange (PKCE) specification.
- PKCE code challenge/verifier example.
- Client types - OAuth 2.1 Draft 7 specification.

Token settings for Application Single Sign-On

This topic tells you how to configure token expiry settings for Application Single Sign-On (commonly called AppSSO).

Token expiry

AppSSO allows you to optionally configure the token expiry settings in your `AuthServer` resource.

The default token expiry settings are as follows:

Token type	Lifetime
Access token	12 hours
Identity token	12 hours
Refresh token	720 hours or 30 days

VMware recommends setting a shorter lifetime for access tokens, typically measured in hours, and a longer lifetime for refresh tokens, typically measured in days. Refresh tokens acquire new access tokens, so they have a longer lifespan.

To override the token expiry settings, configure the following in your `AuthServer` resource:

```
kind: AuthServer
# ...
spec:
  token:
    accessToken:
      expiry: "12h"
    idToken:
      expiry: "12h"
    refreshToken:
      expiry: "720h"
```

`expiry` field examples:

Type	Example	Definition
Seconds	`10s`	10 seconds
Minutes	`10m`	10 minutes
Hours	`10h`	10 hours

Note

The `expiry` field adheres to the duration constraints of the Go standard time library and does not support durations in units beyond hours, such as days or weeks. For more information, see the Go documentation.

Constraints

The token expiry constraints are as follows:

- The duration of the `expiry` field cannot be negative or zero.
- The refresh token's expiration time cannot be the same as or shorter than that of the access token.

Verify token settings

After you set up an Application Single Sign-On `AuthServer`, you can verify that the token received by applications looks as expected. For this purpose, you can create a simple application consuming your `AuthServer`. The following YAML file creates such an application. When you access its URL, it enables you to log in by using your `AuthServer` and displays the token it receives.

Caution

- The simple application is not intended for production use. It only serves a tool to help you verify your setup.
- The following YAML file pulls an unvetted public image `bitnami/oauth2-proxy:7.3.0`
- This section does not apply to an air-gapped environment.

If you stored the following YAML in a file named `token-viewer.yaml`, you can apply it to your cluster by running the following command:

```
  ytt -f token-viewer.yaml --data-value ingress_domain=YOUR-INGRESS-DOMAIN --data-valu
e-yaml 'authserver_selector=YOUR-AUTHSERVER-SELECTOR' | kubectl apply -f-
```

Where `YOUR-AUTHSERVER-SELECTOR` is the label name and its value. For example: `{"name": "ci"}`.

A full example is as follows:

```
#!
#! Token viewer
#!
#! usage:
#!
#! ytt -f token-viewer.yaml --data-value ingress_domain=example.com --data-value-yaml
'authserver_selector={"name": "ci"}'
#!
#! Then navigate to http://token-viewer.<INGRESS_DOMAIN>
#!

#@ load("@ytt:data", "data")
#@ fqdn = "token-viewer." + data.values.ingress_domain
#@ redirect_uri = "http://" + fqdn + "/oauth2/callback"
#@ namespace = data.values.namespace if "namespace" in data.values else "default"

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClientRegistration
metadata:
  name: my-client-registration
  namespace: #@ namespace
spec:
  authServerSelector:
    matchLabels: #@ data.values.authserver_selector
  redirectURIs:
    - #@ redirect_uri
  requireUserConsent: false
  clientAuthenticationMethod: client_secret_basic
  authorizationGrantTypes:
    - "authorization_code"
  scopes:
    - name: "openid"
    - name: "email"
    - name: "profile"
    - name: "roles"

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: token-viewer
  namespace: #@ namespace
spec:
  replicas: 1
  selector:
    matchLabels:
      name: token-viewer
  template:
    metadata:
      labels:
        name: token-viewer
    spec:
      securityContext:
        runAsNonRoot: true
        seccompProfile:
          type: RuntimeDefault
      containers:
        - image: bitnami/oauth2-proxy:7.3.0
          name: proxy
          securityContext:
            runAsNonRoot: true
```

```
      seccompProfile:
        type: RuntimeDefault
      allowPrivilegeEscalation: false
      capabilities:
        drop:
          - ALL
    ports:
      - containerPort: 4180
        name: proxy-port
        protocol: TCP
    env:
      - name: ISSUER_URI
        valueFrom:
          secretKeyRef:
            name: my-client-registration
            key: issuer-uri
      - name: CLIENT_ID
        valueFrom:
          secretKeyRef:
            name: my-client-registration
            key: client-id
      - name: CLIENT_SECRET
        valueFrom:
          secretKeyRef:
            name: my-client-registration
            key: client-secret
    command: [ "oauth2-proxy" ]
    args:
      - --oidc-issuer-url=$(ISSUER_URI)
      - --client-id=$(CLIENT_ID)
      - --insecure-oidc-skip-issuer-verification=true
      - --client-secret=$(CLIENT_SECRET)
      - --cookie-secret=0000000000000000
      - --cookie-secure=false
      - --http-address=http://:4180
      - --provider=oidc
      - --scope=openid email profile roles
      - --email-domain=*
      - --insecure-oidc-allow-unverified-email=true
      - --oidc-groups-claim=roles
      - --upstream=http://127.0.0.1:8000
      - #@ "--redirect-url=" + redirect_uri
      - --ssl-upstream-insecure-skip-verify=true
      - --ssl-insecure-skip-verify=true
      - --skip-provider-button=true
      - --pass-authorization-header=true
      - --prefer-email-to-user=true
  - image: python:3.9
    name: application
    securityContext:
      runAsNonRoot: true
      runAsUser: 1001
      seccompProfile:
        type: RuntimeDefault
      allowPrivilegeEscalation: false
      capabilities:
        drop:
          - ALL
    resources:
      limits:
        cpu: 100m
        memory: 100Mi
    command: [ "python" ]
    args:
      - -c
      - |
```

```
              from http.server import HTTPServer, BaseHTTPRequestHandler
              import base64
              import json

              class Handler(BaseHTTPRequestHandler):
                  def do_GET(self):
                      if self.path == "/token":
                          self.token()
                      elif self.path == "/jwt":
                          self.jwt()
                      else:
                          self.greet()

                  def greet(self):
                      username = self.headers.get("x-forwarded-user")
                      self.send_response(200)
                      self.send_header("Content-type", "text/html")
                      self.end_headers()
                      page = f"""
                      <h1>It Works!</h1>
                      <p>You are logged in as <b>{username}</b></p>
                      <p><a href="/jwt">Show me my id_token (JWT format)</a></p>
                      <p><a href="/token">Show me my id_token (JSON format)</a></p>
                      """
                      self.wfile.write(page.encode("utf-8"))

                  def token(self):
                      token = self.headers.get("Authorization").split("Bearer ")[-1]
                      payload = token.split(".")[1].replace("-","+").replace("_","/")
                      decoded = base64.b64decode(bytes(payload, "utf-8") + b'==').deco
de("utf-8")
                      self.send_response(200)
                      self.send_header("Content-type", "application/json")
                      self.end_headers()
                      self.wfile.write(decoded.encode("utf-8"))

                  def jwt(self):
                      token = self.headers.get("Authorization").split("Bearer ")[-1]
                      self.send_response(200)
                      self.send_header("Content-type", "text/plain")
                      self.end_headers()
                      self.wfile.write(token.encode("utf-8"))

              server_address = ('', 8000)
              httpd = HTTPServer(server_address, Handler)
              httpd.serve_forever()

---
apiVersion: v1
kind: Service
metadata:
  name: token-viewer
  namespace: #@ namespace
spec:
  ports:
    - port: 80
      targetPort: proxy-port
      name: proxy-svc-port
  selector:
    name: token-viewer

---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: token-viewer
```

```
  namespace: #@ namespace
spec:
  rules:
    - host: #@ fqdn
      http:
        paths:
          - path: "/"
            pathType: Prefix
            backend:
              service:
                name: token-viewer
                port:
                  name: proxy-svc-port
```

Manage token signature keys for Application Single Sign-On

This topic tells you how to manage token signature keys for Application Single Sign-On (commonly called AppSSO).

Overview

An `AuthServer` must have token signature keys configured to be able to mint tokens.

For more information about token signatures, see About token signatures.

To learn how to manage keys of an `AuthServer`:

- Creating keys
- Rotating keys
- Revoking keys

Important

"Token signature key" or just "key" is AppSSO's wording for a public/private key pair that is tasked with signing and verifying JSON Web Tokens (JWTs). For more information, please refer to the following resources:

- JSON Web Signature (JWS) spec
- JSON Web Algorithms (JWA) spec
- JSON Web Token (JWT) spec

Creating keys

You can deploy an `AuthServer` without `spec.tokenSignature` but it won't be able to mint tokens. Therefore, keys must be configured to make it fully operational. The following describe how to create and apply a keys for an `AuthServer`.

An RSA key can be created multiple ways. Below are two recommended approaches – choose one.

Using secretgen-controller

Important

> This section assumes you have Tanzu Application Platform running on your cluster with `secretgen-controller` installed.

An RSAKey CR allows for expedited creation of a Secret resource containing PEM-encoded public and private keys required by an `AuthServer`.

1. Create an `AuthServer` with `RSAKeys` as follows:

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
 name: authserver-sample
 namespace: default
spec:
 tokenSignature:
   signAndVerifyKeyRef:
     name: my-token-signing-key
   extraVerifyKeyRefs:
     - name: my-token-verification-key
 # ...
---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
 name: my-token-signing-key
 namespace: default
spec:
 secretTemplate:
   type: Opaque
   stringData:
     key.pem: $(privateKey)
     pub.pem: $(publicKey)
---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
 name: my-token-verification-key
 namespace: default
spec:
 secretTemplate:
   type: Opaque
   stringData:
     key.pem: $(privateKey)
     pub.pem: $(publicKey)
```

2. Observe the creation of an underlying `Secrets`. The name of the each `Secret` is the same as the `RSAKey` names:

```
# Verify Secret exists
kubectl get secret my-token-signing-key

# View the base64-encoded keys
kubectl get secret my-token-signing-key -o jsonpath='{.data}'
```

   You should be able to see two fields within the Secret resource: `key.pem` (private key) and `pub.pem` (public key).

3. Verify that the `AuthServer` serves its keys

```
curl -s authserver-sample.default/oauth2/jwks | jq
```

If you encounter any issues with this approach, see Carvel Secretgen Controller documentation.

Using OpenSSL

You can generate an RSA key yourself using OpenSSL. Here are the steps:

1. Generate a PEM-encoded RSA key pair

   This guide references the freely published OpenSSL Cookbook and the approaches mentioned therein around generating a public and private key pair.

   ```
   # Generate an 4096-bit RSA key
   openssl genpkey -out privatekey.pem -algorithm RSA -pkeyopt rsa_keygen_bits:409
   6
   # -> privatekey.pem
   # The resulting private key output is in the PKCS#8 format

   # Next, extract the public key
   openssl pkey -in privatekey.pem -pubout -out publickey.pem
   # -> publickey.pem
   # The resulting public key output is in the PKCS#8 format

   # To view details of the private key
   openssl pkey -in privatekey.pem -text -noout
   ```

   For OpenSSL key generation examples, see the OpenSSL documentation.

   Important

   The minimum key size for an `Authserver` is 2048.

2. Create a secret resource by using the key generated earlier in this procedure:

   ```
   kubectl create secret generic my-key \
   --from-file=key.pem=privatekey.pem \
   --from-file=pub.pem=publickey.pem \
   --namespace default
   ```

3. Apply your `AuthServer`:

   ```
   apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
   kind: AuthServer
   metadata:
     name: authserver-sample
     namespace: default
   spec:
     tokenSignature:
       signAndVerifyKeyRef:
         name: my-key
     # ...
   ```

4. Verify that the `AuthServer` serves its keys

   ```
   curl -s authserver-sample.default/oauth2/jwks | jq
   ```

Rotating keys

This section describes how to “rotate” token signature keys for an `AuthServer`.

The action of “rotating” means moving the active token signing key into the set of token verification keys, generating a new cryptographic key, and assigning it to be the designated token signing key.

Assuming that you have an `AuthServer` with token signature keys configured, rotate keys as follows:

1. Generate a new token signing key first. See creating keys. Verify that the new `Secret` exists before proceeding to the next step.
2. Edit `AuthServer.spec.tokenSignature`, append the existing `spec.tokenSignature.signAndVerifyKeyRef` to `spec.tokenSignature.extraVerifyKeys` and set your new key as `spec.tokenSignature.signAndVerifyKeyRef`.

   For example:

   ```
   # Before
   ---
   apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
   kind: AuthServer
   metadata:
    name: authserver-sample
    namespace: default
   spec:
    tokenSignature:
      signAndVerifyKeyRef:
        name: old-key
      extraVerifyKeys: []
    # ...
   ```

   ```
   # After
   ---
   apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
   kind: AuthServer
   metadata:
    name: authserver-sample
    namespace: default
   spec:
    tokenSignature:
      signAndVerifyKeyRef:
        name: new-key
      extraVerifyKeys:
        - name: old-key
    # ...
   ```

   Once you apply your changes, key rotation is effective immediately.

Moving the active token signing key to be a token verification key is an *optional* step – check out the Revoking keys section for more.

Revoking keys

This section describes how to “revoke” token signature keys for an `AuthServer`.

The action of “revoking” a key means to entirely remove the key from circulation by an `AuthServer`, whether it be a token signing key or a token verification key. This action might be needed if your organization requires a complete key refresh where older keys are never retained. Another scenario might be in the case of an emergency in which a key or a session has been compromised and a complete revocation is warranted.

To revoke an existing key or keys, you may remove any references to the keys in the `spec.tokenSignature` resource. By removing the reference to the key, the system shall no longer acknowledge that the key is used for signing or verifying JWTs.

For example, if you have a token signing key and a few verification keys:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: authserver-sample
  namespace: default
spec:
  tokenSignature:
    signAndVerifyKeyRef:
      name: key-3
    extraVerifyKeys:
      - name: key-2
      - name: key-1
  # ...
```

To revoke an existing verification key, remove it from the `extraVerifyKeys` list. In the example above, you can remove “key-2” and “key-1” from the list; JWTs signed with those keys will no longer be verifiable.

To revoke an existing token signing key, remove it from `signAndVerifyKeyRef` field. However, if you remove an existing token signing key without a replacement key, the `AuthServer` will not be able to issue access tokens until a valid token signing key is provided. In the example above, “key-3” would be removed; the system will not be able to sign or verify JWTs.

References and further reading

- JSON Web Signature (JWS) - rfc7515 (ietf.org)
- JSON Web Algorithms (JWA) spec
- JSON Web Token (JWT) - rfc7519 (ietf.org)

Session settings for AppSSO

This topic tells you how to configure the session expiry settings for Application Single Sign-On (commonly called AppSSO).

Session expiry

AppSSO allows you to optionally configure the session expiry settings in your `AuthServer` resource.

The default expiry time for an AppSSO session is 15 minutes, according to the STIG guidelines.

To override the session expiry settings, configure the following in your `AuthServer` resource:

```
kind: AuthServer
# ...
spec:
  session:
    expiry: "30m"
```

`expiry` field examples:

Type	Example	Definition
Minutes	`10m`	10 minutes
Hours	`5h`	5 hours

Note

> The `expiry` field adheres to the duration constraints of the Go standard time library and does not support durations in units beyond hours, such as days or weeks. For more information, see the Go documentation.

Constraints

The session expiry constraints are as follows:

- The duration of the `expiry` field cannot be negative or zero.
- The duration must be at least 1 minute.

Storage for AppSSO

This topic tells you how to configure the storage for Application Single Sign-On (commonly called AppSSO).

Overview

AppSSOs `AuthServer` handles data pertaining to user's session, identity, access tokens and approved or rejected consents. For production environments, it is critical to provide your own storage source to enable enterprise functions such as data backup and recovery, auditing and long-term persistence according to your organization's data and security policies.

AppSSO currently only supports Redis `v6.0` or above as a storage solution. `v6.0` introduced TLS support to ensure encrypted client-server communication - AppSSO enforces TLS by default.

Storage provided by default refers to an `AuthServer` resource where `.spec.storage` is not set.

Although data in motion is encrypted by using TLS, data at rest is not encrypted by default through `AuthServer`. Each storage provider is responsible for encrypting their own data. See data types for more information about storage.

Securing Data at rest

To be compliant with HIPAA, FISMA, PCI and GDPR, you must encrypt data at rest. Securing the underlying infrastructure that Redis uses is crucial to protect against a potential attack. The National Institute for Standards and Technology – Federal Information Processing Standards (NIST-FIPS) sets the standard for best practice when it comes to data security in the US. Symmetric cryptography can be used to protect data at rest. This means that the same key encrypts and decrypts the data, so there is no need for a different private and public key. The Advanced Encryption Standard (AES) encryption algorithm is an industry standard for securing data at rest. For the highest level security, VMware recommends using a 256-bit key.

Configuring Redis

To configure Redis as authorization server storage, you must have the following information of your Redis server:

- **Server CA certificate** (optional): the Certificate Authority (CA) certificate to verify Redis TLS connections. It is not required if Redis Server certificate is signed by a public CA.
- **host** (required): the domain name, IP address, or host name of your Redis server.
- **port** (optional): the port number of your Redis server. It default to `6379` and must be a string.
- **username** (optional): the username to authenticate against your Redis server.

- **password** (optional): the password to authenticate against your Redis server.

AppSSO takes the secure-by-default approach and does not establish non-encrypted communication channels. The `AuthServer` resource enters an error state if a non-encrypted connection is attempted.

mTLS is not supported, however Vanilla Redis uses mTLS by default. It can be turned off by setting `tls-auth-clients no`. For more information, see Redis documentation.

The following steps introduce the path to configuring Redis with AppSSO:

1. Configuring Redis Server CA certificate
2. Configuring a Redis Secret
3. Attaching storage to an AuthServer

Configuring Redis Server CA certificate

If your Redis includes a custom or non-public Server CA certificate, you must instruct AppSSO to trust the CA certificate. This is required for the authorization server to communicate with your Redis over TLS. See CA certificates for more information about configuring a CA certificate with AppSSO.

Configuring a Redis Secret

To provide coordinates (the location details) of your Redis server, you must create a `Secret` resource that follows well-known Secret entries conventions. For more information, see Service Bindings 1.0.0 specification.

Example of a properly formatted `Secret` resource:

Important

The Secret must be created in the same namespace as your `AuthServer`.

```
apiVersion: v1
kind: Secret
metadata:
  name: redis-credentials
  namespace: my-authserver
type: servicebinding.io/redis        # optional, must equal 'servicebinding.io/redis'
if defined
stringData:
  type: "redis"                      # required, must equal 'redis'
  ssl: "true"                        # required, must equal 'true'
  host: "redis01.prod.example.com"   # required
  port: "6379"                       # optional, must be a string, defaults to "6379"
if left empty
  password: "!!veryStrongPassword!!" # optional
  username: "redis01-user"           # optional
```

Attaching storage to an AuthServer

After a Redis Secret resource is applied, you can reference the Secret in `.spec.storage`. An example of an AuthServer with a reference to a Redis Secret is as follows:

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: my-authserver-example
```

```
  namespace: my-authserver
spec:
  # ...
  storage:
    redis:
      serviceRef:
        apiVersion: "v1"
        kind: "Secret"
        name: redis-credentials
```

After `AuthServer` is applied, ensure its `Status` is `Ready`.

Inspecting storage of an AuthServer

You can inspect the status of an `AuthServer`'s storage as follows:

```
kubectl get authserver <authserver-name> \
  --namespace <authserver-namespace> \
  --output jsonpath="{.status.storage.redis}" | jq
```

Expect to see the following output with the actual Redis host and port:

```
{
  "redis": {
    "host": "ci-redis.authservers.svc.cluster.local",
    "port": "6379"
  }
}
```

Storage provided by default

If no storage is defined, an `AuthServer` provides its own short-lived ephemeral storage solution, Redis. The provided Redis is configured to never flush any data to any volume that might be attached to the pods that operate the authorization server.

Caution

The default storage configuration is desisged for prototyping or testing environments and must not be used in production environments.

To view details for Redis of an `AuthServer`:

```
# Get the Redis image
kubectl get authserver <authserver-name> \
  --namespace <authserver-namespace> \
  --output jsonpath="{.status.deployments.redis}" | jq

# Get the Redis host and port
kubectl get authserver <authserver-name> \
  --namespace <authserver-namespace> \
  --output jsonpath="{.status.storage.redis}" | jq
```

Data types

The following data is stored in Redis:

- Client information
    - Authorization grant type

    - Client id
- User session
    - Session token
    - Refresh token
- Identity and access tokens

Note

This is the data that carries the highest level risk.

    - Authentication token including the principal
        - Personally identifying information such as email and name
- Approved or rejected consents
    - A client identifier
    - A reference to the user
    - A list of the Authorities that the user has granted to this client

Known limitations of storage providers

Redis Cluster

When your storage is provided by Redis Cluster, additional settings might be required.

The nodes and the maximum number of redirects must be set in your Service Bindings' `Secret`. For example, in addition to the entries in Configuring a Redis Secret, you must provide `cluster` settings as follows:

```
apiVersion: v1
kind: Secret
metadata:
  name: redis-cluster-credentials
  namespace: authservers
type: servicebinding.io/redis
stringData:
  #...
  cluster.max-redirects: 5
  cluster.nodes: 100.90.1.10:6379,100.90.1.11:6379,100.90.1.12:6379
```

Important

`cluster.nodes` must be a comma-separated list of `<ip>:<port>`.

AuthServer readiness for AppSSO

This topic tells you how to use `AuthServer.status` as a reliable source to verify an `AuthServer`'s readiness for Application Single Sign-On (commonly called AppSSO).

You can verify your `AuthServer` by ensuring:

- there is at least one token signing key configured.

```
curl -X GET {spec.issuerURI}/oauth2/jwks
```

The response body should yield at least one key in the list. If there are no keys, please apply a token signing key

- OpenID discovery endpoint is available.

  ```
  curl -X GET {spec.issuerURI}/.well-known/openid-configuration
  ```

  The response body should yield a valid JSON body containing information about the `AuthServer`.

Client registration check

It is helpful to verify an `AuthServer` by running a test run with a test `ClientRegistration`. It ensures that app developers can register clients with the `AuthServer` successfully.

Follow the steps below to ensure that your installation can:

1. Add a test client.
2. Get an access token.
3. Invalidate/remove the test client.

Prerequisites

Ensure that you have successfully applied a token signing key to your `AuthServer` before proceeding.

Define and apply a test client

Apply a `ClientRegistration` to your cluster in a Namespace that the `AuthServer` should allow clients from:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClientRegistration
metadata:
  name: test-client
  namespace: default
spec:
  authServerSelector:
    matchLabels:
    # appropriate labels for your `AuthServer`
  authorizationGrantTypes:
    - client_credentials
  clientAuthenticationMethod: client_secret_basic
```

See the ClientRegistration API reference for more field definitions.

This defines a test `ClientRegistration` with the `client_credentials` OAuth grant type.

Apply the `ClientRegistration`:

```
kubectl apply -f appsso-test-client.yaml
```

Once the `ClientRegistration` is applied, inspects its status and verify it's ready.

Get an access token

You should be able to get a token with the client credentials grant for example:

```
# Get client id (`base64` command has to be available on the command line)
export APPSSO_TEST_CLIENT_ID=$(kubectl get secret test-client -n default -o jsonpath="
{.data['client-id']}" | base64 --decode)

# Get client secret (`base64` command has to be available on the command line)
export APPSSO_TEST_CLIENT_SECRET=$(kubectl get secret test-client -n default -o jsonpa
th="{.data['client-secret']}" | base64 --decode)

# Attempt to fetch access token
curl \
 --request POST \
 --location "{spec.issuerURI}/oauth2/token" \
 --header "Content-Type: application/x-www-form-urlencoded" \
 --header "Accept: application/json" \
 --data "grant_type=client_credentials" \
 --basic \
 --user $APPSSO_TEST_CLIENT_ID:$APPSSO_TEST_CLIENT_SECRET
```

You should see a response JSON containing populated field `access_token`. If so, the system is working as expected, and client registration check is successful.

Make sure to delete the test `ClientRegistration` once you are done.

Scale AuthServer for AppSSO

This topic tells you how to scale `AuthServer` for Application Single Sign-On (commonly called AppSSO).

The number of authorization server replicas for an `AuthServer` can be specified under `spec.replicas`.

Furthermore, `AuthServer` implements the `scale` subresource. That means you can scale an `AuthServer` with the existing tooling. For example:

```
kubectl scale authserver authserver-sample --replicas=3
```

The resource of the authorization server and Redis `Deployments` can be configured under `spec.resources` and `spec.redisResources` respectively. See the API reference for details.

Curate a service offering

This topic describes how you expose an `AuthServer` as a ready-to-claim service offering using a `ClusterWorkloadRegistrationClass`.

`ClusterWorkloadRegistrationClass` creates resources so that application operators can discover and claim credentials for an Application Single Sign-On service offering. A `ClusterWorkloadRegistrationClass` has a description that is shown when application operators discover services by running `tanzu service class list`. This allows you identify the offering as an Application Single Sign-On service.

Furthermore, `ClusterWorkloadRegistrationClass` carries a base `WorkloadRegistration`, which is the blueprint for claims against this service. This base selects the target `AuthServer`. It can optionally receive a custom domain template, labels, and annotations that all `WorkloadRegistration` inherit.

Prerequisites

Before you create a service offering, you must create and configure an `AuthServer`. For instructions, see the topics in the service operator guide.

Create a `ClusterWorkloadRegistrationClass`

For an `AuthServer` with the following labels:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  labels:
    sso.apps.tanzu.vmware.com/env: staging
    sso.apps.tanzu.vmware.com/ldap: ""
#! ...
```

You can expose it as a claimable service offering by configuring a `ClusterWorkloadRegistrationClass` as follows:

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClusterWorkloadRegistrationClass
metadata:
  name: demo
spec:
  base:
    spec:
      authServerSelector:
        matchLabels:
          sso.apps.tanzu.vmware.com/env: staging
          sso.apps.tanzu.vmware.com/ldap: ""
```

After you apply this resource, application operators can discover it by running `tanzu service class list`, for example:

```
$ tanzu service class list
  NAME  DESCRIPTION
  demo  Login by AppSSO
```

Application operators can claim credentials for this service either by running the command `tanzu service class-claim create` or with a `ClassClaim` resource.

When a claim is created, a `WorkloadRegistration` is created from the base and it targets the `AuthServer`.

Customize the `ClusterWorkloadRegistrationClass`

Each `WorkloadRegistration` has `https://` redirect URIs templated. The default template is configured with `default_workload_domain_template`. If `spec.workloadDomainTemplate` is omitted, the default template is used. For more information, see default_workload_domain_template. Otherwise, you can customize it by setting a template on the base, for example, `"{{.Name}}-{{.Namespace}}.demo.{{.Domain}}"`.

You can further customize each `WorkloadRegistration` by setting labels and annotations for them.

The default description of an Application Single Sign-On service offering is `"Login by AppSSO"`, but you can customize this. Consider using a good name and description. For more information, see Names and descriptions later in this topic.

For example, if you want the `WorkloadRegistration` to template redirect URIs from a custom template and with both `https://` and `http://`, and you want to say that in the service's description, edit the `ClusterWorkloadRegistrationClass` as follows:

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClusterWorkloadRegistrationClass
metadata:
  name: demo
spec:
  description:
    short: Login by AppSSO with LDAP for apps in the "demo" subdomain
  base:
    metadata:
      annotations:
        sso.apps.tanzu.vmware.com/template-unsafe-redirect-uris: ""
    spec:
      workloadDomainTemplate: "{{.Name}}-{{.Namespace}}.demo.{{.Domain}}"
      authServerSelector:
        matchLabels:
          sso.apps.tanzu.vmware.com/env: staging
          sso.apps.tanzu.vmware.com/ldap: ""
```

Names and descriptions

When choosing a name and a description for a `ClusterWorkloadRegistrationClass` consider the following:

- When the name of a service is stable across environments, for example, from dev to production, application operators can use the same `ClassClaim` in all environments.
- The description of a service must clearly communicate its flavor and provider. The default description of a `ClusterWorkloadRegistrationClass` is `"Login by AppSSO"`.

  If there is a single Application Single Sign-On service offering the default description is usually good enough.

  To customize the description for your `ClusterWorkloadRegistrationClass`, consider prefixing it with `"Login by AppSSO - "`, for example, `"Login by AppSSO - LDAP and GitHub`.

Application Single Sign-On for App Operators

The following topics explain how to consume an Application Single Sign-On service:

- Discover AppSSO services
- Claim credentials for an AppSSO service
- Secure a workload
- Secure a Spring Boot workload
- Secure a single-page app workload

Discover Application Single Sign-On service offerings

This topic describes the recommended method for you to discover Application Single Sign-On service offerings.

VMware recommends that you consume Application Single Sign-On by claiming credentials for an Application Single Sign-On service using Services Toolkit. Service operators are responsible for curating service offerings.

You can discover the available service offerings with the Tanzu Service CLI:

```
tanzu service class list
```

The output contains a list of services you can claim credentials for, for example:

```
$ tanzu service class list
  NAME                    DESCRIPTION
  mysql-unmanaged         MySQL by Bitnami
  postgresql-unmanaged    PostgreSQL by Bitnami
  rabbitmq-unmanaged      RabbitMQ by Bitnami
  redis-unmanaged         Redis by Bitnami
  sso                     Login by AppSSO - LDAP
  sso-demo                Login by AppSSO - user:password - UNSAFE FOR PRODUCTION!
```

By looking at the `DESCRIPTION` you can identify two of the services as being for Application Single Sign-On: `sso` and `sso-demo`.

Claim credentials for an Application Single Sign-On service offering

This topic describes the recommended method for you to consume Application Single Sign-On service offerings, which is by using a class claim.

When you create a claim for an Application Single Sign-On service, you receive your service credentials through service bindings. This makes it easier to load the credentials into a workload running on Tanzu Application Platform.

To learn about the different levels of Application Single Sign-On service consumption, see The three levels of Application Single Sign-On consumption.

Discover available parameters

To create a claim for an Application Single Sign-On service, target the specific service and provide the required and optional parameters. These parameters allow you to configure the OAuth2 client according to your needs.

To discover the parameter schema for a service, run:

```
tanzu service class get NAME
```

For example:

```
$ tanzu service class get sso

NAME:           app-sso
DESCRIPTION:    Login by AppSSO - OAuth2
READY:          true

PARAMETERS:
  KEY                          DESCRIPTION  TYPE     DEFAULT                REQUIRED
  authorizationGrantTypes      [...]        array    [authorization_code]   false
  clientAuthenticationMethod   [...]        string   client_secret_basic    false
  displayName                  [...]        string   <nil>                  false
  redirectPaths                [...]        array    <nil>                  false
  requireUserConsent           [...]        boolean  true                   false
  scopes                       [...]        array    [map[...]]             false
  workloadRef.name             [...]        string   <nil>                  true
```

Here you can see all the parameters with a brief description, their types, default values, and whether they are required or not. The only required parameter is `workloadRef.name`.

Claim credentials

To claim credentials you can either use the `tanzu service class-claim create` command or create a `ClassClaim` directly.

- **If using the Tanzu CLI,** claim credentials by running:

  ```
  tanzu service class-claim create CLAIM-NAME \
    --class SERVICE-NAME \
    --namespace NAMESPACE \
    --parameter workloadRef.name=WORKLOAD-NAME \
    --parameter PARAMETER
  ```

  Where:

  - `CLAIM-NAME` is a name you choose for your claim.
  - `SERVICE-NAME` is the name of the service that you want to claim.
  - `NAMESPACE` is the namespace that your workload is in.
  - `WORKLOAD-NAME` is the name of your workload.
  - (Optional) `PARAMETER` is a parameter that you choose in the format `KEY=VALUE`. You can add more than one optional parameter. For how to discover parameters you can add, see Discover available parameters.

  For example:

  ```
  $ tanzu service class-claim create my-class-claim \
    --class app-sso \
    --namespace my-namespace \
    --parameter workloadRef.name=my-workload \
    --parameter displayName='My sample app' \
    --parameter redirectPaths='["/login/oauth2/code/sso"]' \
    --parameter authorizationGrantTypes='["client_credentials", "authorization_co
  de"]' \
    --parameter requireUserConsent=false
  ```

- **If using a `ClassClaim`,** create a YAML file similar to the following example:

  ```
  ---
  apiVersion: services.apps.tanzu.vmware.com/v1alpha1
  kind: ClassClaim
  metadata:
    name: my-class-claim
    namespace: my-namespace
  spec:
    classRef:
      name: app-sso
    parameters:
      workloadRef:
        name: my-workload
      displayName: "My sample app"
      redirectPaths:                        # Optional
        - /login/oauth2/code/sso
      authorizationGrantTypes:              # Optional
        - client_credentials
        - authorization_code
      requireUserConsent: false             # Optional
  ```

Important

When iterating on your `ClassClaim`, you must recreate it when you make changes. Updates to an existing `ClassClaim` have no effect. For more information, see Class claims compared to resource claims.

Inspect the progress of your claim

You can inspect the progress of your claim creation by running:

```
tanzu service class-claim get MY-CLAIM-NAME --namespace MY-NAMESPACE
```

or

```
kubectl get classclaim MY-CLAIM-NAME --namespace MY-NAMESPACE --output yaml
```

Caution

It can take approximately 60 to 120 seconds for your Application Single Sign-On credentials to propagate into your service bindings secret.

Next steps

You now have service credentials that you can use to secure your workload with SSO. To learn about the specific client settings and how you can use a claim to secure a workload with Application Single Sign-On, see Secure a workload. For tutorials that show how to secure specific types of workloads with Application Single Sign-On, see Secure a single-page app workload and Secure a Spring Boot workload.

If you have problems claiming credentials for an Application Single Sign-On service, learn how to troubleshoot. For more information about the `tanzu service` command, classes, and claims, see the Tanzu CLI Command Reference documentation.

Secure a workload with Application Single Sign-On

This tutorial tells you how to secure a `Workload` running on Tanzu Application Platform with Application Single Sign-On (commonly called AppSSO).

For specific stack implementations, see Secure a single-page app workload and Secure a Spring Boot workload.

The Getting Started topic explains how to obtain OAuth2 client credentials for an authorization server by claiming them from an Application Single Sign-On service offering. You can enable this by running the `tanzu service class-claims create` command or by using a `ClassClaim` resource. In either case, you can customize your OAuth2 client settings by providing parameters within your claim. To secure your `Workload`, you must provide the appropriate parameters relevant to the given situation.

The following sections elaborate on each parameter in detail and guide the process of loading credentials into a `Workload`.

When editing your `ClassClaim`, you must recreate it in order for the changes to take effect. Updating an existing `ClassClaim` does not produce any impact. For more information, see Using a `ClassClaim`.

OAuth2 client parameters

A `ClassClaim` for an Application Single Sign-On service is the request for OAuth2 client credentials for an authorization server.

The following is an example of a claim for a service called `sso`, which customizes all available client parameters:

```
---
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ClassClaim
metadata:
  name: my-sso-client
  namespace: my-apps
spec:
  classRef:
    name: sso
  parameters:
    workloadRef:
      name: my-workload
    redirectPaths:
      - /login/oauth2/code/my-sso-client
    displayName: "My SSO client"
    scopes:
      - name: openid
      - name: email
      - name: profile
      - name: roles
      - name: coffee.make
        description: bestows the ultimate power
    authorizationGrantTypes:
      - client_credentials
      - authorization_code
      - refresh_token
    clientAuthenticationMethod: client_secret_basic
    requireUserConsent: true
```

To verify the status of this `ClassClaim`, run:

```
kubectl get classclaim my-sso-client --namespace my-apps
```

After a `ClassClaim` is applied and its status shows `Ready`, it can be consumed by a `Workload`. When the credentials are loaded into a `Workload`, your application code can use them to initiate OAuth2 flows.

`redirectPaths`

As a critical part of your client parameters, `redirectPaths` define the locations to which your application's end-users are redirected to after the authentication. Incorrect redirect URIs often cause errors for clients and are the most common source of such issues. When redirect URIs are not configured accurately, your application encounters errors and can not perform OAuth2 flows.

For servlet-based Spring Boot workloads using Spring Security OAuth 2 Client library, the default redirect path template looks as follows:

```
/login/oauth2/code/{ClassClaim.metadata.name}`
```

For more information about the format, see the Spring documentation.

For example, configure a single redirect path in `ClassClaim`:

```
spec:
  parameters:
```

```
    redirectPaths:
      - /login/oauth2/code/my-sso-client
```

Behind the scenes, your redirect paths are templated into the full redirect URIs including a scheme and fully-qualified domain name. For example, your actual redirect URI might look as follows:

```
https://my-workload.my-apps.example.com/login/oauth2/code/my-sso-client
```

The choice of scheme (`https` or `http`), the template for the subdomain and the ingress domain are under the control of your service operators and platform operators. If your redirect URIs do not template according to your needs, reach out to your service operators and platform operators and request adjustments to the templates.

Redirect paths are one of the values for templating redirect URIs. However, there is another parameter that needs to be considered, which leads us to the next parameter.

`workloadRef`

`workloadRef.name` is the name of the workload which acts as the OAuth2 client. You can use this value when templating the fully-qualified domain name of your redirect URIs.

```
spec:
  parameters:
    workloadRef:
      name: my-workload
```

Therefore, if you rename your workload from `my-workload` to a different name, you must update this parameter to align with the new name.

Note

`workloadRef` is not resolved to an actual `Workload` existing on the cluster. This means that the claim does not depend on the existence of a `Workload`.

Display name

An optional field to designate a user-friendly display name for your registered client. The display name is rendered on the authorization scope consent page if the field `.spec.parameters.requireUserConsent` is toggled on.

The display name must be between 2 and 32 characters in length.

`authorizationGrantTypes`

In OAuth2, a grant type is the way your application obtains tokens from the authorization server. There are different grant types. Some of them allow your application to act on behalf of an end user, but others do not.

The `authorizationGrantTypes` parameter denotes all the grant types your application can use:

```
spec:
  parameters:
    authorizationGrantTypes:
      - client_credentials
      - authorization_code
      - refresh_token
```

Application Single Sign-On supports the following OAuth2 grant types:

- Authorization Code: `authorization_code`

  Applications use this grant type to authenticate and authorize end users. An `AuthServer` issues identity and access tokens to applications to identify end users' identities and their level of access to the protected resources.

- Client Credentials: `client_credentials`

  Applications use this grant type to communicate directly to other protected applications through a client identifier and a client secret. For example, in service-to-service communication, an `AuthServer` issues access tokens that define the level of access that the requesting service has to the protected service they seek to communicate with.

  Note

  Use cases for the grant type `authorization_code` and `client_credentials` are typically different, so VMware recommends creating separate client registrations for each grant type.

- Refresh Token: `refresh_token`

  Applications use this grant type to obtain access tokens. If the `refresh_token` grant type is included, a refresh token is attached to every access token issued by an `AuthServer`. You can use the refresh token to fetch the new access tokens before the older ones expire to continue accessing the protected resources.

Client authentication method

A client authentication method defines how your application presents its credentials to the authorization server.

```
spec:
  parameters:
    clientAuthenticationMethod: client_secret_basic
```

There are three client authentication methods:

- `client_secret_basic`(default): Send the client credentials by using the HTTP "Basic" authentication scheme. This is the recommended method for authenticating server-based applications such as Spring Boot or .NET Core apps (confidential clients).
- `client_secret_post`: Send the client credentials in the body of an HTTP POST request.
- `none`: Do not send the client credentials. This is for browser-based single-page apps.

`scopes`

The `scopes` field defines the OAuth2 scopes requested by your application, including the standard OpenID claims.

```
spec:
  parameters:
    scopes:
      - name: openid # Standard OpenID scope, containing claims "sub" (subject), "aud"
(audience) and so on.
      - name: email # Standard OpenID scope, containing claims "email" and "email_veri
fied".
      - name: profile # Standard OpenID scope, containing claims "name", "given_name",
"family_name" and so on.
      - name: roles # AppSSO special scope, requesting the user roles or groups be pop
```

```
ulated in the "roles" claim.
      - name: coffee.make # Custom the authorization scope.
        description: bestows the ultimate power # With a custom description.
```

To activate the issuance of the users' identity tokens and authentication, you must include the `openid` scope.

To activate fetching of the user roles or groups, you must include the `roles` scope.

`requireUserConsent`

The `requireUserConsent` field defines the toggling scopes approved by the end users.

```
spec:
  parameters:
    requireUserConsent: true
```

If activated, the end users are prompted to consent to or reject the scopes that the client requests on behalf of them. If deactivated, all scopes that the client requests are auto-approved or consented to without prompting.

Behind the scenes

When you create a `ClassClaim` for an Application Single Sign-On service, several resources are created behind the scenes. These resources can be helpful for debugging purposes.

- You create a `ClassClaim` for an Application Single Sign-On service with your parameters.
- An `XWorkloadRegistration` with your parameter is created in the same namespace.
- A `WorkloadRegistration` with your parameter is created in the same namespace. The `WorkloadRegistration` templates your redirect URIs.
- A `ClientRegistration` with your parameters and the templated redirect URIs are created in the same namespace.
- The `ClientRegistration` receives the client credentials and passes them all the way to your `ClassClaim`.

Loading client credentials into a `Workload`

Now that you have a client credentials for your application, you can reference the `ClassClaim` from your `Workload`.

An example `Workload` in `my-apps` namespace is as follows:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  labels:
    apps.tanzu.vmware.com/workload-type: web
  name: my-workload
  namespace: my-apps
spec:
  serviceClaims:
    - name: my-sso-client # Must match the name of the referenced `ClassClaim`.
      ref:
        apiVersion: services.apps.tanzu.vmware.com/v1alpha1
        kind: ClassClaim
        name: my-sso-claim
  #! ...
```

Alternatively, you can refer to your `ClassClaim` when deploying your workload with the Tanzu CLI:

```
tanzu apps workload create my-workload \
  --service-ref "my-sso-client=services.apps.tanzu.vmware.com/v1alpha1:ClassClaim:my-sso-claim" \
  # ...
```

Important

The service ref name must match the name of the referenced `ClassClaim`.

Thanks to service bindings, your credentials are provided to your `Workload`'s `Pod` (one or more) by mounting a volume containing your client credentials.

The credentials provided by the claim are:

- `client-id`: The identifier of your `Workload` that Application Single Sign-On is registered with. This is a unique identifier.
- `client-secret`: The secret string value used by Application Single Sign-On to verify your client. Keep this value secret.
- `issuer-uri`: The web address of the Application Single Sign-On `AuthServer` and the primary location that your `Workload` goes to when interacting with Application Single Sign-On.
- `authorization-grant-types`: A list of the desired OAuth 2 grant types.
- `client-authentication-method`: The method in which the client is authenticated when requesting an identity or access token.
- `scope`: A list of the desired scopes that your application's users have access to.

These credentials are mounted onto your `Workload`'s `Pod`(one or more) as individual files at the following locations:

```
/bindings
  /<name-of-service-claim>
    /client-id
    /client-secret
    /issuer-uri
    /authorization-grant-types
    /client-authentication-method
    /scope
```

Following the earlier example, you can find the location of mounted credentials on every `Pod` at:

```
/bindings/my-sso-client
```

Given these auto-generated values, your `Workload` can now load them at runtime and bind to an Application Single Sign-On authorization server at startup time. Reading the values from the file system is left to the implementer.

Trusting an authorization server

Your application makes a request to the authorization server. The authorization server serves traffic using TLS. If your company uses non-public certificate authority (CA), you must explicitly trust the authorization server or rather the certificate authority. For more information, see Configure Workloads to trust a custom certificate authority.

Summary

This concludes the tutorial explaining how to claim the client credentials for an Application Single Sign-On authorization server and how to secure a workload.

Secure a Spring Boot workload

This tutorial tells you how to secure a sample Spring Boot `Workload` with Application Single Sign-On (commonly called AppSSO), which runs on Tanzu Application Platform (commonly called TAP).

Follow these steps to deploy a sample Spring Boot `Workload`:

1. Get the sample application.
2. Create a namespace for workloads.
3. Claim client credentials.
4. (Optional) Ensure `Workload` trusts `AuthServer`.
5. Deploy the workload.

Get the sample application

Follow these steps to fetch the Application Single Sign-On Spring Boot application source code:

1. Download the Application Single Sign-On Starter Java accelerator from the Tanzu Developer Portal accelerators located on your Tanzu Application Platform cluster:
    - Option 1: Use the Tanzu Developer Portal dashboard through the browser.

      Navigate to Application Accelerators and download the “AppSSO Starter Java” accelerator.
    - Option 2: Use the Tanzu Accelerator CLI.

      Download the zip file of the accelerator source code by running:

      ```
      tanzu accelerator generate appsso-starter-java --server-url TAP_GUI_SERVE
      R_URL
      ```
2. Unzip the resulting `.zip` file into the `appsso-starter-java` directory in your workspace.

    ```
    unzip appsso-starter-java
    ```
3. With the resulting project, create an accessible remote Git repository and push your accelerator to the Git remote repository.

Create a namespace for workloads

You must create a namespace for your workloads for the `Workload` resources to function properly. If you have a workloads namespace already, you can skip this step.

```
kubectl create namespace my-apps
kubectl label namespaces my-apps apps.tanzu.vmware.com/tap-ns=""
```

For more information about provisioning namespaces for workloads, see Set up developer namespaces.

Claim client credentials

Follow these steps to claim credentials for an Application Single Sign-On service so that you can secure your workload:

1. Discover the available Application Single Sign-On services with the Tanzu Service CLI:

   ```
   $ tanzu service class list
   NAME      DESCRIPTION
   sso       Login by AppSSO
   ```

   The actual names of your AppSSO services might be different. VMware assumes that there's one AppSSO service with the name `sso`.

2. Claim credentials for that service by creating a `ClassClaim` named `appsso-starter-java` in the `my-apps` namespace.

   ```
   ---
   apiVersion: services.apps.tanzu.vmware.com/v1alpha1
   kind: ClassClaim
   metadata:
     name: appsso-starter-java
     namespace: my-apps
   spec:
     classRef:
       name: sso
     parameters:
       workloadRef:
         name: appsso-starter-java
       redirectPaths:
         - /login/oauth2/code/appsso-starter-java
       scopes:
         - name: openid
       authorizationGrantTypes:
         - authorization_code
       clientAuthenticationMethod: client_secret_basic
   ```

3. Apply the `ClassClaim` and verify its status by running:

   ```
   kubectl get classclaim appsso-starter-java --namespace my-apps
   ```

Ensure `Workload` trusts `AuthServer`

For Tanzu Application Platform cluster with a custom or self-signed CA certificate, see Configure workloads to trust a custom Certificate Authority (CA).

Deploy the `Workload`

Follow these steps to deploy the `Workload`:

1. Create the Spring Boot accelerator `Workload` by running:

   ```
   tanzu apps workload create appsso-starter-java \
       --namespace my-apps \
       --type web \
       --label app.kubernetes.io/part-of=appsso-starter-java \
       --build-env "BP_JVM_VERSION=17" \
       --service-ref "appsso-starter-java=services.apps.tanzu.vmware.com/v1alpha1:
   ClassClaim:appsso-starter-java" \
       --service-ref "ca-cert=v1:Secret:tap-ca-cert" \
       --git-repo "<GIT_LOCATION_OF_YOUR_ACCELERATOR>" \
       --git-branch main \
       --live-update
   ```

Important

Although you can assign any name to the `ClassClaim`, the `Workload`'s service reference name must match the `ClassClaim`'s name.

```
--service-ref "**appsso-starter-java**=services.apps.tanzu.vmware.
com/v1alpha1:ClassClaim:appsso-starter-java"
```

If the service reference name does not match the `ClassClaim` name, the `Workload` generates a redirect URI that the authorization server will reject.

It might take a few minutes for the workload to become available through a browser-accessible URL.

2. Query the latest status of the workload by running:

```
tanzu apps workload get appsso-starter-java --namespace my-apps
```

3. Monitor the `Workload` logs:

```
tanzu apps workload tail appsso-starter-java --namespace my-apps
```

After the status of the workload reaches the `Ready` state, you can navigate to the provided URL, which looks similar to:

```
https://appsso-starter-java.my-apps.<TAP_CLUSTER_DOMAIN_NAME>
```

4. Open your preferred web browser and navigate to the URL.

Expect to see a large log-in button tailored for authenticating with AppSSO.

Cleaning up

Delete the running application by running the following commands:

1. Delete the sample application `Workload`:

```
tanzu apps workload delete appsso-starter-java --namespace my-apps
```

2. Delete the claim:

```
tanzu service class-claims delete appsso-starter-java --namespace my-apps
```

Secure a single-page app workload

This tutorial tells you how to secure a sample single-page Angular app `Workload` with Application Single Sign-On (commonly called AppSSO), which runs on Tanzu Application Platform (commonly known as TAP).

Follow these steps to deploy a sample single-page app `Workload`:

1. Get the sample application.
2. Create a namespace for workloads.
3. Claim client credentials.
4. Verify application authentication settings.

5. Start a sample back end.
6. Deploy the Workload.

Get the sample application

Follow these steps to fetch the single-page Angular app source code:

1. Download the Angular Frontend accelerator from the Tanzu Developer Portal accelerators located on your Tanzu Application Platform cluster:
    - Option 1: Use the Tanzu Developer Portal dashboard by using a browser.

      Navigate to Application Accelerators and choose the **Angular Frontend** accelerator and then select the **Single Sign-on** option.
    - Option 2: Use the Tanzu Accelerator CLI.

      Download the zip file of the accelerator source code and identify your `AuthServer` Issuer URI by running:

      ```
      kubectl get authserver -A
      ```

      Generate the accelerator by using the `tanzu accelerator` CLI:

      ```
      tanzu accelerator generate angular-frontend \
        --server-url TAP-GUI-SERVER-URL \
        --options '{
          "useSingleSignOn": true,
          "authority": "AUTHSERVER-ISSUER-URI",
          "namespace": "my-apps",
          "authorityLabelKey": "my-sso",
          "authorityLabelValue": "true"
        }'
      ```
2. Unzip the resulting `.zip` file into the `angular-frontend` directory in your workspace:

    ```
    unzip angular-frontend
    cd angular-frontend
    git init
    git branch -M main
    git remote add origin YOUR-ACCELERATOR-GIT-REPOSITORY
    git push origin main -u
    ```

    For public clients, the `AuthServer` only supports the Authorization Code Flow: `response_type=code`, because public clients such as single-page apps cannot safely store sensitive information such as client secrets.
3. Push the resulting directory to the remote Git repository.

Create a namespace for workloads

You must create a namespace for your workloads for the `Workload` resources to function properly. If you have a workloads namespace already, you can skip this step.

```
kubectl create namespace my-apps
kubectl label namespaces my-apps apps.tanzu.vmware.com/tap-ns=""
```

For more information about provisioning namespaces for running `Workloads`, see Set up developer namespaces.

Claim the client credentials

Follow these steps to claim the credentials for an Application Single Sign-On service so that you can secure your workload:

1. Discover the available Application Single Sign-On services with the Tanzu Service CLI:

   ```
   $ tanzu service class list
     NAME      DESCRIPTION
     sso       Login by AppSSO
   ```

   The actual names of your Application Single Sign-On services might be different. VMWare assumes that there's one Application Single Sign-On service with the name `sso`.

2. Claim the credentials for that service by creating a `ClassClaim` named `angular-frontend` in the `my-apps` namespace.

   ```
   ---
   apiVersion: services.apps.tanzu.vmware.com/v1alpha1
   kind: ClassClaim
   metadata:
     name: angular-frontend
     namespace: my-apps
   spec:
     classRef:
       name: sso
     parameters:
       workloadRef:
         name: angular-frontend
       redirectPaths:
         - /user-profile
         - /customer-profiles/list
         - /
       scopes:
         - name: openid
         - name: email
         - name: profile
         - name: message.read
         - name: message.write
       authorizationGrantTypes:
         - authorization_code
       clientAuthenticationMethod: none
   ```

3. Apply the `ClassClaim` and verify the status is `Ready` by running:

   ```
   kubectl get classclaim angular-frontend --namespace my-apps
   ```

Verify application authentication settings

Within the single-page Angular app accelerator, authentication configuration settings are located in `src/assets/auth.conf.json`. After generating the accelerator, expect to observe the populated settings.

Open the file and verify that it adheres to the following structure:

```
{
  "authority": "ISSUER-URI",
  "clientId": "CLIENT-ID",
  "scope": [ "openid", "profile", "email", "message.read", "message.write" ]
}
```

Retrieve your client id by running:

```
kubectl get secret \
  $(kubectl get classclaim -n my-apps angular-frontend -o jsonpath='{.status.binding.name}') \
  --namespace my-apps \
  --template='{{ index .data "client-id" | base64decode}}'
```

Retrieve your issuer URI by running:

```
kubectl get secret \
  $(kubectl get classclaim -n my-apps angular-frontend -o jsonpath='{.status.binding.name}') \
  --namespace my-apps \
  --template='{{ index .data "issuer-uri" | base64decode}}'
```

Start a sample back end

Important

You can skip this step if you have a `java-rest-service` back end running already.

The `angular-frontend` sample application requires a back end application to start properly:

1. Start a sample simulated back end by running:

   ```
   kubectl run sample-backend --image nginx:NGINX-VERSION -n my-apps
   kubectl expose pod sample-backend --port 80 -n my-apps
   ```

2. In the `angular-frontend` source code, edit the `.server.location[/api/].proxy_pass` field in the `nginx.conf` file at the root of the source directory.

3. After updating the value, commit the changes to the Git remote repository:

   ```
   server {
    # ...
    location /api/ {
      proxy_pass http://sample-backend.my-apps/api/;
    }
    # ...
   }
   ```

Deploy the `Workload`

Follow these steps to deploy the `Workload`:

1. Create the `angular-frontend` accelerator `Workload` by running:

   ```
   tanzu apps workload create angular-frontend \
       --namespace my-apps \
       --type web \
       --param "clusterBuilder=base" \
       --param "annotations=autoscaling.knative.dev/minScale: \"1\"" \
       --label app.kubernetes.io/part-of=angular-frontend \
       --git-repo "GIT-LOCATION-OF-YOUR-ACCELERATOR" \
       --git-branch main
   ```

   Note

> As an alternative approach to creating the workload, you can declaratively define a `Workload` resource by using `config/workload.yaml` within the source repository.

It might take a few minutes for the workload to become available through a browser-accessible URL.

2. Query the latest status of the workload by running:

```
tanzu apps workload get angular-frontend --namespace my-apps
```

3. Monitor the `Workload` logs:

```
tanzu apps workload tail angular-frontend --namespace my-apps
```

After the status of the workload reaches the `Ready` state, you can navigate to the provided URL, which looks similar to:

```
https://angular-frontend.my-apps.TAP-CLUSTER-DOMAIN-NAME/user-profile
```

4. Open your preferred web browser and navigate to the URL.

Expect to be prompted to sign in by using AppSSO. After successfully signing in, the profile page displays your identifying information.

Clean up

Delete the running application by running these commands:

1. Delete the sample application `Workload`:

```
tanzu apps workload delete angular-frontend --namespace my-apps
```

2. Delete the claim:

```
tanzu service class-claims delete angular-frontend --namespace my-apps
```

3. Delete the sample back end if was previously applied:

```
kubectl delete svc sample-backend --namespace my-apps
kubectl delete pod sample-backend --namespace my-apps
```

Troubleshoot Application Single Sign-on

This topic tells you how to troubleshoot Application Single Sign-On (commonly called AppSSO).

Why is my AuthServer not working?

Generally, `AuthServer.status` is designed to provide you with helpful feedback to debug a faulty `AuthServer`.

Find all `AuthServer` related Kubernetes resources

Identify all `AuthServer` components with Kubernetes common labels. For more information, see Kubernetes documentation.

Query all related `AuthServer` sub-resources by using `app.kubernetes.io/part-of` label. For example:

```
kubectl get all,ingress,service -A -l app.kubernetes.io/part-of=<authserver-name>
```

See logs of all AuthServers

With stern, you can tail the logs of all AppSSO managed `Pods` inside your cluster with:

```
stern --all-namespaces --selector=app.kubernetes.io/managed-by=sso.apps.tanzu.vmware.com
```

Wait for change propagation

When applying changes to an `AuthServer`, changes to issuer URI, IdP (identity provider), server and logging configuration take a moment to be effective as the operator rolls out the authorization server `Deployment`.

Set up debug logs

Caution

Debug logs display all identity information from the upstream identity providers, for example, all LDAP attributes for a user. VMware recommends not using debug logs in production.

An `AuthServer` can display more information related to configuration or claim mappings. To enable debug logs, update your configuration with:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  # ...
spec:
  # ...
  # Do not forget the pipe character and preserve the indentation.
  logging: |
    level:
      com.vmware.tanzu.apps.sso: DEBUG
      appsso: DEBUG
```

Misconfigured `clientSecret`

Problem:

When attempting to sign in, you see `This commonly happens due to an incorrect [client_secret].` It might be because the client secret of an identity provider is misconfigured.

Solution:

Validate the `AuthServer.spec.openid.clientSecretRef`.

Misconfigured redirect URI

Problem:

You see `Error: [invalid_request] OAuth 2.0 Parameter: redirect_uri` when signing in.

Solution:

The `redirectURIs` of a `ClientRegistration` must refer to the URI (one or more) of the registered `Workload`. It does not refer to the URI of the AuthServer. For more information, see Redirect URIs.

Unsupported `id_token_signed_response_alg` with openid `identityProviders`

Problem:

When trying to log in with an OpenID Connect `identityProvider`, you are unable to sign in and observe the following error in the logs:

```
[invalid_id_token] An error occurred while attempting to decode the Jwt: Signed JWT re
jected: Another algorithm expected, or no matching key(s) found.
```

Solution:

Verify the `identityProvider`'s discovery endpoint at `ISSUER-URI/.well-known/openid-configuration` where `ISSUER-URI` is the value set at `spec.identityProviders.openid.issuerURI`.

The value of `id_token_signing_alg_values_supported` must include `RS256`. If it is not in the list, your identity configuration might not support AppSSO.

If `RS256` is present, expect to see a `jwks_uri` key in the discovery endpoint. If you visit the URL stored in this key, it must return at least one RSA key. Otherwise, your identity provider might be misconfigured.

Refer to your identity provider's documentation to enable `RS256` token signing.

Misconfigured identity provider clientSecret

Problem:

- When attempting to sign in, you see `<WORKLOAD_URL> redirected you too many times.` It might be because the client secret of an identity provider is misconfigured.
- If you have access to the authserver logs, verify if there is an entry with the text `"error":"[invalid_client] Client authentication failed: client_secret"`.

Solution:

Validate the secret referenced by the `clientSecretRef` for this particular identity provider in your `authserver.spec`.

Missing scopes

Problem:

When attempting to fetch data after signing in to your application by using AppSSO, you see `[invalid_scope] OAuth 2.0 Parameter: scope`.

Solution:

Add the required scopes into your `ClientRegistration` yaml under `spec.scopes`.

Changes to the secret do not propagate to the `ClientRegistration`. If you recreated the `Secret` that contains the `clientSecret`, you must re-deploy the `ClientRegistration`.

Misconfigured `sub` claim

Problem:

The `sub` claims in `id_token`s and `access_token`s follow the `<providerId>_<userId>` pattern. The previous `<providerId>/<userId>` pattern might cause bugs in URLs without proper URL-encoding in client applications.

Solution:

If your client application stores `sub` claims, you must update the `sub` claims to match the new pattern `<providerId>_<userId>`.

Troubleshoot a `ClassClaim` for an AppSSO service

Problem:

The service binding secret of a `ClassClaim` for an Application Single Sign-On service is empty.

Solution:

Be patient as it can take up to `~60-120s` for the client credentials to be propagated into the claim's service binding secret.

If you have waited for a considerable amount of time and the credentials still haven't appeared, see the Services Toolkit documentation for more troubleshooting information.

Application Single Sign-On Reference

This topic provides you with the following reference documentation of Application Single Sign-On (commonly called AppSSO):

- API
- Known issues
- OpenShift
- Package configuration
- RBAC
- AuthServer audit logs

AppSSO APIs

This topic provides you with detailed information about the following APIs of Application Single Sign-On (commonly called AppSSO):

- AuthServer
- ClientRegistration
- ClusterUnsafeTestLogin
- ClusterWorkloadRegistrationClass

- WorkloadRegistration
- XWorkloadRegistration

AuthServer API for AppSSO

In Application Single Sign-On (commonly called AppSSO), `AuthServer` represents the request for an OIDC authorization server. It causes the deployment of an authorization server backed by Redis over mutual TLS if no storage is defined.

An `AuthServer` should have labels which allow to uniquely match it amongst others. `ClientRegistration` selects an `AuthServer` by label selector and needs a unique match to be successful.

To allow `ClientRegistrations` only from a restricted set of `Namespaces`, you must set the annotation `sso.apps.tanzu.vmware.com/allow-client-namespaces`. Its value is a comma-separated list of allowed `Namespaces`, for example, `"app-team-red,app-team-green"`. If the annotation is missing, the default value is `*`, denoting that all client namespaces are allowed.

The issuer URI, which is the point of entry for clients and end-users, is constructed through the package's `domain_template`. You can view the issuer URI by running `kubectl get authserver -n authservers`.

See Issuer URI & TLS for more information.

Token signature keys are configured by using `spec.tokenSignature`. This is a required field. See Token signatures for more context.

You can configure identity providers under `spec.identityProviders`. If there is none, end-users can not log in. For more information about configuring identity providers, see Identity providers.

The deployment can be further customized by configuring replicas, resources, http server and logging properties.

Spec

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: ""
  namespace: ""
  labels: { } # required, must uniquely identify this AuthServer
  annotations:
    sso.apps.tanzu.vmware.com/allow-client-namespaces: "" # optional, a comma-separate
d list of allowed client namespaces
    sso.apps.tanzu.vmware.com/allow-unsafe-issuer-uri: "" # optional
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: "" # optional
    sso.apps.tanzu.vmware.com/allow-unsafe-cors: "" # optional
spec:
  # .tls is optional if a default issuer is set
  tls:
    # must be one and only one of issuerRef, certificateRef or secretRef, unless deact
ivated
    issuerRef:
      name: ""
      kind: ""
      group: cert-manager.io
    certificateRef:
      name: ""
    secretRef:
      name: ""
    deactivated: false # If true, requires annotation `sso.apps.tanzu.vmware.com/allow
-unsafe-issuer-uri: ""`.
```

```
  cors:
    allowOrigins: # optional, cannot be combined with 'allowAllOrigins'.
      - ""
    allowAllOrigins: false # optional
                           # If true, requires annotation `sso.apps.tanzu.vmware.com/a
llow-unsafe-cors: ""`.
                           # Cannot be combined with 'allowOrigins'.

    allowMethods: # optional, defaults to ["GET", "POST", "OPTIONS"]
      - ""
    allowHeaders: # optional, defaults to ["Authorization"]
      - ""
    exposeHeaders: # optional, defaults to "[]"
      - ""
    allowCredentials: false # optional, defaults to false
  token:             # optional
    accessToken:     # optional
      expiry: "12h"  # optional, default expiry is 12 hours
    refreshToken:    # optional
      expiry: "720h" # optional, default expiry is 720 hours (30 days)
    idToken:         # optional
      expiry: "12h"  # optional, default expiry is 12 hours
  tokenSignature: # required
    signAndVerifyKeyRef:
      name: "" # Must be a secret that contains an RSA private key with a minimum leng
th of 2048 bits.
    extraVerifyKeyRefs:
      - name: "" # Must be a secret that contains an RSA private key with a minimum le
ngth of 2048 bits.
  storage: # optional
    redis: # required if 'storage' is defined
      serviceRef: # Reference to a provisioned service within the same namespace as th
is AuthServer. Only supports Secret reference.
        apiVersion: "v1"
        kind: "Secret"
        name: ""
  caCerts: # optional
    - secretRef: # Reference to Secret resource within the same namespace as this Auth
Server.
        name: ""
  identityProviders: # optional
    # each must be one and only one of internalUnsafe, ldap, openID or saml
    - name: "" # must be unique
               # must follow the DNS Subdomain formatting (RFC 1123):
               # https://kubernetes.io/docs/concepts/overview/working-with-objects/nam
es/#dns-subdomain-names
               # must not start with 'client' or 'unknown'
      internalUnsafe: # requires annotation `sso.apps.tanzu.vmware.com/allow-unsafe-id
entity-provider: ""`
        users:
          - username: ""
            password: ""
            claims: # optional
              custom_claim: ""
              another_custom_claim: ""
            roles:
              - ""
        accessToken: # optional
          scope:
            defaults: # optional
            - ""
            rolesToScopes: # optional
              - fromRole: ""
                toScopes:
                  - ""
    - name: "" # must be unique
```

```
              # must follow the DNS Subdomain formatting (RFC 1123):
              # https://kubernetes.io/docs/concepts/overview/working-with-objects/nam
es/#dns-subdomain-names
              # > must not start with 'client' or 'unknown'
      ldap:
        server:
          scheme: ""
          host: ""
          port: 0
          base: ""
        bind:
          dn: ""
          passwordRef:
            name: ldap-password
        user:
          searchFilter: ""
          searchBase: ""
        roles: # optional
          fromUpstream:
            attribute: "" # required
            search:
              filter: ""
              base: ""
              subTree: false
              depth: 0
          filterBy: # optional
            - exactMatch: ""
            - regex: "" # must be valid regular expression
        idToken: # optional
          claims:
            - fromUpstream: ""
              toClaim: ""
        accessToken: # optional
          scope:
            defaults: # optional
            - ""
            rolesToScopes: # optional
              - fromRole: ""
                toScopes:
                - ""
        group:    # deprecated, use 'ldap.roles.fromUpstream' instead.
          search: # deprecated, use 'ldap.roles.fromUpstream.search' instead.
            filter: ""
            base: ""
            subTree: false
            depth: 0
          roleAttribute: "" # deprecated, use 'ldap.roles.fromUpstream.attribute' inst
ead.
    - name: "" # must be unique
              # must follow the DNS Subdomain formatting (RFC 1123):
              # https://kubernetes.io/docs/concepts/overview/working-with-objects/nam
es/#dns-subdomain-names
              # must not start with 'client' or 'unknown'
      openID:
        configurationURI: "" # optional, if not specified, must define the alternative
fields: `issuerURI` or `authorizationUri`, `tokenUri`, and `jwksUri`. This field must
be suffixed with '/.well-known/openid-configuration'
        issuerURI: "" # deprecated, optional, use 'openid.configurationURI' instead. T
his field must not be set if 'configurationURI' is set.
        authorizationUri: "" # optional. Must be set if 'configurationURI' and 'issuer
URI' are not set.
        tokenUri: "" # optional, must be set if 'configurationURI' and 'issuerURI' are
not set.
        jwksUri: "" # optional, must be set if 'configurationURI' and 'issuerURI' are
not set.
        userinfoUri: "" # optional
```

```
        displayName: "" # optional, must be between 2 and 32 characters in length
        clientID: ""
        clientSecretRef:
          name: ""
        scopes:
          - ""
        roles: # optional
          fromUpstream:
            claim: "" # required
          filterBy: # optional
            - exactMatch: ""
            - regex: "" # must be valid regular expression
        idToken: # optional
          claims:
            - fromUpstream: ""
              toClaim: ""
        accessToken: # optional
          scope:
            defaults: # optional
            - ""
            rolesToScopes: # optional
              - fromRole: ""
                toScopes:
                - ""
    - name: "" # must be unique
                # must follow the DNS Subdomain formatting (RFC 1123):
                # https://kubernetes.io/docs/concepts/overview/working-with-objects/nam
es/#dns-subdomain-names
                # must not start with 'client' or 'unknown'
      saml:
        metadataURI: ""
        displayName: "" # optional, must be between 2 and 32 characters in length
        roles: # optional
          fromUpstream:
            attribute: "" # required
          filterBy: # optional
            - exactMatch: ""
            - regex: "" # must be valid regular expressions
        idToken: # optional
          claims:
            - fromUpstream: ""
              toClaim: ""
        accessToken: # optional
          scope:
            defaults: # optional
              - ""
            rolesToScopes: # optional
              - fromRole: ""
                toScopes:
                  - ""
  replicas: 1 # optional, default 2
  logging: "" # optional, must be valid YAML
  server: "" # optional, must be valid YAML
  resources: # optional, default {requests: {cpu: "256m", memory: "300Mi"}, limits: {c
pu: "2", memory: "768Mi"}}
    requests:
      cpu: ""
      mem: ""
    limits:
      cpu: ""
      mem: ""
  redisResources: # optional, default {requests: {cpu: "50m", memory: "100Mi"}, limit
s: {cpu: "100m", memory: "256Mi"}}
    requests:
      cpu: ""
      mem: ""
```

```
    limits:
      cpu: ""
      mem: ""
status:
  observedGeneration: 0
  issuerURI: ""
  clientRegistrationCount: 1
  tokenSignatureKeyCount: 0
  deployments:
    authServer:
      configHash: ""
      image: ""
      replicas: 0
    redis: # left empty if storage is configured by the service operator
      image: ""
  storage:
    redis:
      host: "" # the hostname of the configured Redis
      port: "" # the port of the configured Redis
  tls:
    deactivated: false
    issuerRef:
      name: ""
      kind: ""
      group: cert-manager.io
  conditions:
    - lastTransitionTime:
      message: ""
      reason: ""
      status: "True" # or "False"
      type: ""
```

Alternatively, you can interactively discover the spec with:

```
kubectl explain authservers.sso.apps.tanzu.vmware.com
```

Status & conditions

The `.status` subresource helps you to learn the `AuthServer`'s readiness, resulting deployments, attached clients and to troubleshoot issues.

`.status.issuerURI` is the templated issuer URI. This is the entry point for any traffic.

`.status.tls` is the actual TLS configuration.

`.status.tokenSignatureKeyCount` is the number of currently configured token signature keys.

`.status.clientRegistrationCount` is the number of currently registered clients.

`.status.deployments.authServer` describes the current authorization server deployment by listing the running image, its replicas, the hash of the current configuration and the generation which has last resulted in a restart.

`.status.deployments.redis` describes the current provided Redis deployment by listing its running image. This field is nil if storage is defined explicitly by using `.spec.storage`.

`.status.storage.redis` describes the configured Redis storage such as host name and port number.

`.status.conditions` documents each step in the reconciliation:

- `Valid`: Is the spec valid?
- `ImagePullSecretApplied`: Has the image pull secret been applied?
- `SignAndVerifyKeyResolved`: Has the single sign-and-verify key been resolved?

- `ExtraVerifyKeysResolved`: Have the single extra verify keys been resolved?
- `IdentityProvidersResolved`: Has all identity provider configuration been resolved?
- `ConfigResolved`: Has the complete configuration for the authorization server been resolved?
- `AuthServerConfigured`: Has the complete configuration for the authorization server been applied?
- `IssuerURIReady`: Is the authorization server yet responding to `{.status.issuerURI}/.well-known/openid-configuration`?
- `Ready`: whether all the previous conditions are “True”

The super condition `Ready` denotes a fully successful reconciliation of a given `ClientRegistration`.

If everything goes well you will see something like this:

```
issuerURI: "https://..."
observedGeneration: 1
tokenSignatureKeyCount: 0
clientRegistrationCount: 0
caCerts:
  - cert:
      subject: ""
    source:
      secretEntry: ""
deployments:
  authServer:
    LastParentGenerationWithRestart: 1
    configHash: "11216479096262796218"
    image: "..."
    replicas: 1
  redis: # leave empty if external storage is defined
    image: "..."
storage:
  redis:
   host: "" # the host name of the configured Redis
   port: "" # the port of the configured Redis
tls:
  deactivated: false
  # One of issuerRef, certificateRef or secretRef is set if TLS is enabled
  issuerRef:
    name: ""
    kind: ""
    group: ""
  certificateRef:
    name: ""
  secretRef:
    name: ""
conditions:
  - lastTransitionTime: "2022-08-24T09:58:10Z"
    message: ""
    reason: KeysConfigSecretUpdated
    status: "True"
    type: AuthServerConfigured
  - lastTransitionTime: "2022-08-24T09:58:10Z"
    message: ""
    reason: Resolved
    status: "True"
    type: ConfigResolved
  - lastTransitionTime: "2022-08-24T09:58:10Z"
    message: ""
    reason: ExtraVerifyKeysResolved
    status: "True"
    type: ExtraVerifyKeysResolved
  - lastTransitionTime: "2022-08-24T09:58:10Z"
```

```
    message: ""
    reason: Resolved
    status: "True"
    type: IdentityProvidersResolved
  - lastTransitionTime: "2022-08-24T09:58:10Z"
    message: ""
    reason: ImagePullSecretApplied
    status: "True"
    type: ImagePullSecretApplied
  - lastTransitionTime: "2022-08-24T09:58:28Z"
    message: ""
    reason: Ready
    status: "True"
    type: IssuerURIReady
  - lastTransitionTime: "2022-08-24T09:58:28Z"
    message: ""
    reason: Ready
    status: "True"
    type: Ready
  - lastTransitionTime: "2022-08-24T09:58:10Z"
    message: ""
    reason: SignAndVerifyKeyResolved
    status: "True"
    type: SignAndVerifyKeyResolved
  - lastTransitionTime: "2022-08-24T09:58:10Z"
    message: ""
    reason: Valid
    status: "True"
    type: Valid
```

RBAC

The `ServiceAccount` of the authorization server has a `Role` with the following permissions:

```
- apiGroups:
    - ""
  resources:
    - secrets
  verbs:
    - get
    - list
    - watch
  resourceNames:
    - { name }-auth-server-keys
    - { name }-auth-server-clients
```

Example

This example requests an authorization server with two token signature keys and two identity providers.

Note

The label used for matching to ClientRegistrations must be unique across namespaces.

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
```

```
  name: authserver-sample
  namespace: default
  labels:
    identifier: authserver-identifier
    sample: "true"
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
    sso.apps.tanzu.vmware.com/allow-unsafe-cors: ""
spec:
  replicas: 1
  tls:
    issuerRef:
      name: my-cluster-issuer
      kind: ClusterIssuer
  tokenSignature:
    signAndVerifyKeyRef:
      name: sample-token-signing-key
    extraVerifyKeyRefs:
      - name: sample-token-verification-key
  cors:
    allowAllOrigins: true
  identityProviders:
    - name: internal
      internalUnsafe:
        users:
          - username: user
            password: password
            claims:
              alt_address: "123 Alternate Street"
            roles:
              - message.write
    - name: okta
      openID:
        configurationURI: https://dev-xxxxxx.okta.com/.well-known/openid-configuration
        displayName: "Okta"
        clientID: xxxxxxxxxxxxx
        clientSecretRef:
          name: okta-client-secret
        scopes:
          - openid
        roles:
          fromUpstream:
            claim: my_custom_okta_roles_claim
        idToken:
          claims:
            - fromUpstream: "alternate_address"
              toClaim: "alt_address"
        accessToken:
          scope:
            defaults:
              - "developer.read"
              - "developer.write"
            rolesToScopes:
              - fromRole: "finance"
                toScopes:
                  - "finance.read"
                  - "finance.write"

---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: sample-token-signing-key
  namespace: default
spec:
  secretTemplate:
```

```
    type: Opaque
    stringData:
      key.pem: $(privateKey)
      pub.pem: $(publicKey)

---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: sample-token-verification-key
  namespace: default
spec:
  secretTemplate:
    type: Opaque
    stringData:
      key.pem: $(privateKey)
      pub.pem: $(publicKey)

---
apiVersion: v1
kind: Secret
metadata:
  name: okta-client-secret
  namespace: default
stringData:
  clientSecret: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

ClientRegistration API for AppSSO

In Application Single Sign-On (commonly called AppSSO), `ClientRegistration` is the request for client credentials for an AuthServer.

`ClientRegistration` is created automatically during the process of claiming credentials. However, there is also the option of creating it manually.

It implements the Service Bindings `ProvisionedService`. The credentials are returned as a Service Bindings `Secret`.

A `ClientRegistration` must uniquely identify an `AuthServer` by using `spec.authServerSelector`. If it matches none, too many or a disallowed `AuthServer`, it does not get credentials. The other fields are for the configuration of the client on the `AuthServer`.

Spec

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClientRegistration
metadata:
  name: ""
  namespace: ""
spec:
  authServerSelector: # required
    matchLabels: { }
  redirectURIs: # optional
    - ""
  scopes: # optional
    - name: ""
      description: ""
  displayName: "" # optional, must be between 2 and 32 chars in length
  authorizationGrantTypes: # optional
    - client_credentials
    - authorization_code
    - refresh_token
  clientAuthenticationMethod: "" # optional, values accepted are described in Client a
```

```
uthentication methods section
  requireUserConsent: false # optional
status:
  authServerRef:
    apiVersion: ""
    issuerURI: ""
    kind: ""
    name: ""
    namespace: ""
  binding:
    name: ""
  clientID: ""
  clientSecretHelp: ""
  conditions:
    - lastTransitionTime: ""
      message: ""
      reason: ""
      status: "True" # or "False"
      type: ""
  observedGeneration: 0
```

Alternatively, you can interactively discover the spec with:

```
kubectl explain clientregistrations.sso.apps.tanzu.vmware.com
```

Scopes

The following scopes must be included for the issuance of identity tokens:

- `openid` must be included for the identity tokens to be issued.
- `profile` must be included so the custom-mapped claims are included in an issued identity token, for example, `AuthServer.identityProviders[*].{openID,ldap,saml}.idToken.claims`. For more information, see Identity token claims mapping.
- `email` must be included to retain the `email` and `email_verified` claims.
- `address` must be included to retain the `address` claim.
- `phone` must be included to retain the `phone_number` and `phone_number_verified` claims.
- `roles` must be included to retrieve the user role information from an upstream identity provider. For more information, see Configure authorization.

Client authentication methods

Client authentication methods supported by `ClientRegistration` resource are:

- `client_secret_basic`: HTTP header based client authentication (default).
- `client_secret_post`: HTTP POST body based client authentication.
- `none`: No client authentication. Required for public clients. For more information, see Public clients and CORS.

Status & conditions

The `.status` subresource helps you to learn about your client credentials, the matched `AuthServer` and to troubleshoot issues.

`.status.authServerRef` identifies the successfully matched `AuthServer` and its issuer URI.

`.status.binding.name` is the name of the Service Bindings `Secret` which contains the client credentials.

`.status.conditions` documents each step in the reconciliation:

- `Valid`: Is the spec valid?
- `AuthServerResolved`: Has the targeted `AuthServer` been resolved?
- `ClientSecretResolved`: Has the client secret been resolved?
- `ServiceBindingSecretApplied`: Has the Service Bindings Secret with the client credentials been applied?
- `AuthServerConfigured`: Has the resolved `AuthServer` been configured with the client?
- `Ready`: whether all the previous conditions are “True”

The super condition `Ready` denotes a fully successful reconciliation of a given `ClientRegistration`.

If everything goes well you will see something like this:

```
status:
  authServerRef:
    apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
    issuerURI: http://authserver-sample.default
    kind: AuthServer
    name: authserver-sample
    namespace: default
  binding:
    name: clientregistration-sample
  clientID: default_clientregistration-sample
  clientSecretHelp: 'Find your clientSecret: ''kubectl get secret clientregistration-s
ample --namespace default'''
  conditions:
    - lastTransitionTime: "2022-05-13T07:56:41Z"
      message: ""
      reason: Updated
      status: "True"
      type: AuthServerConfigured
    - lastTransitionTime: "2022-05-13T07:56:40Z"
      message: ""
      reason: Resolved
      status: "True"
      type: AuthServerResolved
    - lastTransitionTime: "2022-05-13T07:56:40Z"
      message: ""
      reason: ResolvedFromBindingSecret
      status: "True"
      type: ClientSecretResolved
    - lastTransitionTime: "2022-05-13T07:56:41Z"
      message: ""
      reason: Ready
      status: "True"
      type: Ready
    - lastTransitionTime: "2022-05-13T07:56:40Z"
      message: ""
      reason: Applied
      status: "True"
      type: ServiceBindingSecretApplied
    - lastTransitionTime: "2022-05-13T07:56:40Z"
      message: ""
      reason: Valid
      status: "True"
      type: Valid
  observedGeneration: 1
```

Example

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClientRegistration
metadata:
  name: my-client-registration
  namespace: app-team
spec:
  displayName: "My sample app"
  authServerSelector:
    matchLabels:
      for: app-team
      ldap: "true"
  redirectURIs:
    - "https://127.0.0.1:8080/authorized"
    - "https://my-application.com/authorized"
  requireUserConsent: false
  clientAuthenticationMethod: client_secret_basic
  authorizationGrantTypes:
    - "authorization_code"
    - "refresh_token"
  scopes:
    - name: "openid"
      description: "To indicate that the application intends to use OIDC to verify the
user's identity"
    - name: "email"
      description: "The user's email"
    - name: "profile"
      description: "The user's profile information"
```

The client is registered with the authorization server with the given `spec`. The resulting client credentials are available in a `Secret` that the `ClientRegistration` owns.

```
apiVersion: v1
kind: Secret
type: servicebinding.io/oauth2
metadata:
  name: my-client-registration
  namespace: app-team
data: # fields below are base64-decoded for display purposes only
  type: oauth2
  provider: appsso
  client-id: default_my-client-registration
  client-secret: c2VjcmV0 # auto-generated
  issuer-uri: https://appsso.example.com
  client-authentication-method: client_secret_basic
  scope: openid,email,profile
  authorization-grant-types: client_credentials,refresh_token
```

ClusterUnsafeTestLogin API for Application Single Sign-On

`ClusterUnsafeTestLogin` is the recommended way to get started with Application Single Sign-On (commonly called AppSSO) in non-production environments, and it is not safe for production.

`ClusterUnsafeTestLogin` represents the request for an unsafe, ready-to-claim AppSSO service offering. It reconciles into an unsafe `AuthServer`, a token signing key `Secret`, and a `ClusterWorkloadRegistrationClass`. It is cluster-scoped and has no specifications.

Its `AuthServer` is `http` only, which allows all CORS origins and runs with a single replica. Its name is prefixed with `unsafe-`. Its issuer URI resembles `http://unsafe-demo.appsso.example.com`.

There is a single `user:password` login. That user has roles `user` and `test` and claims `first_name=First-Name`, `last_name=Last-Name` and `email=user@example.com`.

Its `ClusterWorkloadRegistrationClass` templates `WorkloadRegistration` with safe and unsafe redirect URIs. That means a redirect path is templated with both `https` and `http` as the scheme.

Its namespace-scoped resources are placed in the cluster resource namespace. The cluster resource namespace is configurable. The default cluster resource namespace is `appsso`.

Once created, you can discover it with the Tanzu Service CLI:

```
$ tanzu service class list
  NAME      DESCRIPTION
  <name>    Login by AppSSO - user:password - UNSAFE FOR PRODUCTION!
```

Specification

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClusterUnsafeTestLogin
metadata:
  name: "" #! required
status:
  issuerURI: ""
  clusterResourceNamespace: ""
  conditions:
  - lastTransitionTime: ""
    message: ""
    reason: ""
    status: ""
    type: AuthServerReady
  - lastTransitionTime: ""
    message: ""
    reason: ""
    status: ""
    type: ClusterWorkloadRegistrationClassReady
  - lastTransitionTime: ""
    message: ""
    reason: ""
    status: ""
    type: Ready
  - lastTransitionTime: ""
    message: ""
    reason: ""
    status: ""
    type: SignAndVerifyKeyConditionReady
  observedGeneration: 0
  tokenSignature:
    signAndverifyKey:
      name: ""
      pem: ""
```

Example

Due to the zero-configuration nature of the `ClusterUnsafeTestLogin` API, there exists only a single feasible example:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClusterUnsafeTestLogin
```

```
metadata:
  name: demo
```

ClusterWorkloadRegistrationClass API for Application Single Sign-On

In Application Single Sign-On (commonly called AppSSO), `ClusterWorkloadRegistrationClass` represents the request to expose an `AuthServer` as a claimable service offering. It is cluster-scoped and is identified by its short name `cwrc`.

`ClusterWorkloadRegistrationClass` optionally receives a free-form description which explains the offering to those which discover it with the Tanzu CLI. It also receives a base `WorkloadRegistration` section.

`ClusterWorkloadRegistrationClass` reconciles into a Crossplane `Composition` and a Services Toolkit `ClusterInstanceClass`.

The `Composition` defines how to create a `WorkloadRegistration` for an `AuthServer`. The `Composition` is for the composite resource `XWorkloadRegistration`. It relies on Crossplane's provider-kubernetes.

The `ClusterInstanceClass` is applied with its description and a selector for the `Composition`. Instances of this class are claimed by using Services Toolkit's `ClassClaim`. For more information, see claims.

The available parameters of the `ClusterInstanceClass` are those of `XWorkloadRegistration`. For more information about the fields, see XWorkloadRegistration.

The base `WorkloadRegistration` section is the blueprint for claims to this class. It is the base `Object` for the `Composition`. This base is patched with the parameters from claims. Setting fields on the base sets the fields on all claimed `WorkloadRegistration`. This is how they can match a certain `AuthServer`, or have labels and annotations.

The base is a partial projection of `WorkloadRegistration`'s specification. It is limited to the fields `metadata.labels`, `metadata.annotations`, `spec.workloadDomainTemplate`, and `spec.authServerSelector`. For more information about the fields, see WorkloadRegistration.

After the offering is created, you can discover it with the Tanzu Service CLI:

```
$ tanzu service class list
  NAME     DESCRIPTION
  <name>   <description>
```

You can also discover the service's parameters with the Tanzu Service CLI:

```
$ tanzu service class get <name>
NAME:           <name>
DESCRIPTION:    <description>
READY:          true

PARAMETERS:
  KEY                          DESCRIPTION  TYPE     DEFAULT               REQUIRED
  authorizationGrantTypes      [...]        array    [authorization_code]  false
  clientAuthenticationMethod   [...]        string   client_secret_basic   false
  displayName                  [...]        string   <nil>                 false
  redirectPaths                [...]        array    <nil>                 false
  requireUserConsent           [...]        boolean  true                  false
  scopes                       [...]        array    [map[...]]            false
  workloadRef.name             [...]        string   <nil>                 true
```

`ClusterWorkloadRegistrationClass` aggregates the readiness of its children. It is not ready until its base's `authServerSelector` uniquely selects an `AuthServer`. When an `AuthServer` is matched, its reference is written to the status.

The `Composition` label selector is written to the status.

Specification

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClusterWorkloadRegistrationClass
metadata:
  name: "" #! required
spec:
  description:
    short: "" #! required, default to "Login by AppSSO"
  base:
    metadata:
      labels: {} #! optional
      annotations: {} #! optional
    spec:
      workloadDomainTemplate: "" #! optional
      authServerSelector:
        matchLabels: {} #! required
status:
  authServerRef:
    apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
    issuerURI: ""
    kind: AuthServer
    name: ""
    namespace: ""
  compositionSelector:
    matchLabels: {}
  conditions:
  - lastTransitionTime: ""
    message: ""
    reason: ""
    status: ""
    type: AuthServerResolved
  - lastTransitionTime: ""
    message: ""
    reason: ""
    status: ""
    type: ClusterInstanceClassReady
  - lastTransitionTime: ""
    message: ""
    reason: ""
    status: ""
    type: CompositionApplied
  - lastTransitionTime: ""
    message: ""
    reason: ""
    status: ""
    type: Ready
  observedGeneration: 1
```

Alternatively, you can interactively discover the API with:

```
kubectl explain cwrc
```

Examples

This is a minimal example that selects an `AuthServer` that is uniquely identified by the label `sso.apps.tanzu.vmware.com/env=dev`:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClusterWorkloadRegistrationClass
metadata:
  name: sample-minimal
spec:
  base:
    spec:
      authServerSelector:
        matchLabels:
          sso.apps.tanzu.vmware.com/env: dev
```

This is a full example that selects an `AuthServer` that is uniquely identified by the label `sso.apps.tanzu.vmware.com/env=dev`. It sets a custom `workloadDomainTemplate` on the base, a label and an annotation. The annotation triggers safe and unsafe redirect URI templating.

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClusterWorkloadRegistrationClass
metadata:
  name: sample-full
spec:
  description:
    short: Login by AppSSO with a custom domain template
  base:
    metadata:
      labels:
        app.kubernetes.io/part-of: service-claims
      annotations:
        sso.apps.tanzu.vmware.com/template-unsafe-redirect-uris: ""
    spec:
      workloadDomainTemplate: "{{.Namespace}}-{{.Name}}.apps.{{.Domain}}"
      authServerSelector:
        matchLabels:
          sso.apps.tanzu.vmware.com/env: staging
```

Claims

Instances of this class are claimed by using Services Toolkit's `ClassClaim`.

The class is identified by its name and the client registration's configuration is set as its parameters. The `spec` of parameters is the specification of XWorkloadRegistration.

For more information about the `ClassClaim` API, see ClusterInstanceClass and ClassClaim.

You can set the propagation time of client credentials to the `ClassClaim` up to `120s`.

Claims specification

This is the specification of a `ClassClaim` for a `ClusterWorkloadRegistrationClass`, not the specification of the `ClassClaim` API.

```
---
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ClassClaim
metadata:
  name: "" #! required
  namespace: #! required
spec:
```

```
  classRef:
    name: "" #! required
  parameters:
    workloadRef:
      name: "" #! required
    redirectPaths: #! optional
      - "" #! must be an absolute path
    scopes: #! optional
      - name: "" #! required
        description: "" #! optional
    authorizationGrantTypes: #! optional
      - "" #! must be one of "authorization_code", "client_credentials" or "refresh_to
ken"
    clientAuthenticationMethod: "" #! optional, must be one of "client_secret_post",
"client_secret_basic" or "none"
    requireUserConsent: false #! optional
```

Claims examples

If a claimable AppSSO service offering exists as follows:

```
$ tanzu service class list
  NAME               DESCRIPTION
  demo               Single Sign-On Demo
```

This is a minimal example claim for the AppSSO service offering `demo`:

```
---
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ClassClaim
metadata:
  name: sample-minimal
spec:
  classRef:
    name: demo
  parameters:
    workloadRef:
      name: sample-workload
    redirectPaths:
      - /redirect/uri/1
      - /redirect/uri/2
```

This is a fully configured example claim for the AppSSO service offering `demo`:

```
---
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ClassClaim
metadata:
  name: sample-full
spec:
  classRef:
    name: demo
  parameters:
    workloadRef:
      name: sample-workload
    redirectPaths:
      - /redirect/uri/1
      - /redirect/uri/2
    scopes:
      - name: openid
      - name: email
      - name: profile
      - name: roles
      - name: coffee.make
```

```
        description: bestows the ultimate power
    authorizationGrantTypes:
      - client_credentials
      - authorization_code
      - refresh_token
    clientAuthenticationMethod: client_secret_basic
    requireUserConsent: false
```

Claims updates

`ClassClaim` performs a point-in-time look-up, so updates to an existing `ClassClaim.spec.parameters` have no effect. For more information, see Class claims compared to resource claims.

When creating your `ClassClaim` with kapp, you can configure it to replace the resource instead of updating it when it changes:

```
---
apiVersion: kapp.k14s.io/v1alpha1
kind: Config
rebaseRules:
  - resourceMatchers:
      - apiVersionKindMatcher:
          { apiVersion: services.apps.tanzu.vmware.com/v1alpha1, kind: ClassClaim }
    path: [metadata, annotations, "classclaims.services.apps.tanzu.vmware.com/xrd-nam
e"]
    type: copy
    sources: [new, existing]

---
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ClassClaim
metadata:
  annotations:
    kapp.k14s.io/update-strategy: always-replace
#! ...
```

WorkloadRegistration API for AppSSO

In Application Single Sign-On (commonly called AppSSO), `WorkloadRegistration` represents the request for client credentials for an AuthServer. It is a portable, higher-level abstraction over ClientRegistration. It is namespaced and is identified by its short name `workloadreg`.

`WorkloadRegistration` templates redirect URIs and reconciles into a ClientRegistration. For more information, see Redirect URI templating.

`WorkloadRegistration` exposes most of the fields of ClientRegistration. The exceptions are `spec.redirectPaths` (instead of `spec.redirectURIs`), `spec.workloadDomainTemplate` and `spec.workloadRef`.

By templating redirect URIs, a `WorkloadRegistration` is decoupled from a client workload's specific FQDN. As a result, it is portable across environments.

`spec.workloadDomainTemplate` is a golang text/template. It is optional with default value `{{.Name}}.{{.Namespace}}.{{.Domain}}`. You can change the default value to the package configuration value `default_workload_domain_template`.

The values for `{{.Name}}` and `{{.Namespace}}` are inherited from `spec.workloadRef`. The field `spec.workloadRef` is not resolved to an actual workload running on the cluster. It is a holder for template values which identify a workload.

The package configuration value `workload_domain_name` defines the value for `{{.Domain}}`.

Similar to `ClientRegistration`:

- `WorkloadRegistration` is bindable. It implements the Service Bindings `ProvisionedService`. The credentials are returned as a Service Bindings `Secret`.
- A `WorkloadRegistration` must uniquely identify an `AuthServer` by using `spec.authServerSelector`. If it matches none, too many or a disallowed `AuthServer`, it does not get credentials.

`WorkloadRegistration` aggregates the readiness of its child `ClientRegistration`.

Note

The term “workload” is meant to express the general notion of a workload. The overlap with Cartographer’s `Workload` API is therefore both incidental and accidental, because that API too is not opinionated as to what specific resources consolidat a “workload”.

`WorkloadRegistration` represents the client registration of any “workload” whose redirect URIs can be templated. It must not be a `Workload` or even be running on a Kubernetes cluster.

Specification

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: WorkloadRegistration
metadata:
  name: "" #! required
  namespace: "" #! required
  annotations:
    sso.apps.tanzu.vmware.com/template-unsafe-redirect-uris: "" #! optional
spec:
  workloadDomainTemplate: "" #! optional
  workloadRef:
    name: "" #! required
    namespace: "" #! required
  authServerSelector:
    matchLabels: {} #! required
  displayName: "" #! optional, must be between 2 and 32 chars in length
  redirectPaths: #! optional
    - "" #! must be an absolute path
  scopes: #! optional
    - name: "" #! required
      description: "" #! optional
  authorizationGrantTypes: #! optional
    - "" #! must be one of "authorization_code", "client_credentials" or "refresh_toke
n"
  clientAuthenticationMethod: "" #! optional, must be one of "client_secret_post", "cl
ient_secret_basic" or "none"
  requireUserConsent: false #! optional
status:
  authServerRef:
    apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
    issuerURI: ""
    kind: AuthServer
    name: ""
    namespace: ""
  binding:
    name: ""
  conditions:
  - lastTransitionTime: ""
```

```
    message: ""
    reason: ""
    status: ""
    type: ClientRegistrationReady
  - lastTransitionTime: ""
    message: ""
    reason: ""
    status: ""
    type: Ready
  observedGeneration: 0
  redirectURIs:
  - ""
  workloadDomainTemplate: ""
```

Alternatively, you can interactively discover the API with:

```
kubectl explain workloadreg
```

Examples

This is a minimal example which selects an `AuthServer` that is uniquely identified by the label `sso.apps.tanzu.vmware.com/env=dev`:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: WorkloadRegistration
metadata:
  name: sample-minimal
spec:
  workloadRef:
    name: test-workload-name
    namespace: test-workload-namespace
  authServerSelector:
    matchLabels:
      sso.apps.tanzu.vmware.com/env: dev
```

This is a full example which selects an `AuthServer` that is uniquely identified by label `sso.apps.tanzu.vmware.com/env=dev`. It uses all possible client configurations and sets a custom `workloadDomainTemplate` and an annotation. The annotation causes safe and unsafe redirect URI templating.

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: WorkloadRegistration
metadata:
  name: sample-full
  annotations:
    sso.apps.tanzu.vmware.com/template-unsafe-redirect-uris: ""
spec:
  workloadDomainTemplate: "hi-i-live-in-{{.Namespace}}-and-my-name-is-{{.Name}}.sampl
e.{{.Domain}}"
  workloadRef:
    name: test-workload-name
    namespace: test-workload-namespace
  authServerSelector:
    matchLabels:
      name: authserver-sample
      sample: "true"
  displayName: "Full sample app"
  redirectPaths:
    - /redirect/uri/1
    - /redirect/uri/2
```

```
  scopes:
    - name: openid
    - name: email
    - name: profile
    - name: roles
    - name: coffee.make
      description: bestows the ultimate power
  authorizationGrantTypes:
    - client_credentials
    - authorization_code
    - refresh_token
  clientAuthenticationMethod: client_secret_basic
  requireUserConsent: true
```

Redirect URI templating

Redirect URIs are templated as follows:

1. If `spec.redirectPaths` is empty, no further action is required.
2. Resolve the workload domain template by reading `spec.workloadDomainTemplate` or default it to `default_workload_domain_template` if omitted.
3. Resolve the values for `{{.Name}}` and `{{.Namespace}}` from `spec.workloadRef`.
4. Resolve the value for `{{.Domain}}` from `workload_domain_name`.
5. For each entry in `spec.redirectPaths`, template a full redirect URI by joining the path with the rendered workload domain template. Use `https` as the scheme.
6. If the annotation `sso.apps.tanzu.vmware.com/template-unsafe-redirect-uris` is present, template an additional unsafe redirect URI for each entry in `spec.redirectPaths` by using `http` as the scheme.

For example, if you set `workload_domain_name` to `tap.example.com`, a hypothetical `WorkloadRegistration` might look as follows:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: WorkloadRegistration
metadata:
  name: demo
  annotations:
    sso.apps.tanzu.vmware.com/template-unsafe-redirect-uris: ""
spec:
  workloadRef:
    name: my-workload
    namespace: my-ns
  redirectPaths:
    - /login/success
    - /login/error
  #! ...
status:
  redirectURIs:
    - https://my-workload.my-ns.tap.example.com/login/success
    - http://my-workload.my-ns.tap.example.com/login/success
    - https://my-workload.my-ns.tap.example.com/login/error
    - http://my-workload.my-ns.tap.example.com/login/error
  workloadDomainTemplate: '{{.Name}}.{{.Namespace}}.{{.Domain}}'
  #! ...
```

XWorkloadRegistration API for AppSSO

In Application Single Sign-On (commonly called AppSSO), `XWorkloadRegistration` is a cluster-scoped Crossplane XRD. It serves as an integration API between Services Toolkit, Crossplane and AppSSO.

Note

This API is not intended for direct usage. Although it is supported, VMware recommend using `ClassClaim`, `WorkloadRegistration`, or `ClientRegistration` instead when you need direct access to this API.

In most cases, when creating a `ClassClaim` for an AppSSO service offering, for example, ClusterWorkloadRegistrationClass, Services Toolkit creates an `XWorkloadRegistration`. By using a `Composition`, the `XWorkloadRegistration` is reconciled into a `WorkloadRegistration` with Crossplane's `provider-kubernete`'s `Object` as an intermediary.

The specification of `XWorkloadRegistration` is identical to WorkloadRegistration but without `spec.workloadRef.namespace` and `spec.authServerSelector`.

Specification

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: XWorkloadRegistration
metadata:
  name: "" #! required
spec:
  workloadRef:
    name: "" #! required
  redirectPaths: #! optional
    - "" #! must be an absolute path
  scopes: #! optional
    - name: "" #! required
      description: "" #! optional
  displayName: "" #! optional
  authorizationGrantTypes: #! optional
    - "" #! must be one of "authorization_code", "client_credentials" or "refresh_toke
n"
  clientAuthenticationMethod: "" #! optional, must be one of "client_secret_post", "cl
ient_secret_basic" or "none"
  requireUserConsent: false #! optional
status:
  authServerRef:
    apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
    issuerURI: ""
    kind: AuthServer
    name: ""
    namespace: ""
  binding:
    name: ""
```

Note

Crossplane's standard Crossplane Resource Model (commonly called XRM) fields are omitted.

Examples

If a `Composition` for `XWorkloadRegistration` exists, for example, by using a `ClusterWorkloadRegistrationClass`, this is a minimal example:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: XWorkloadRegistration
metadata:
  name: sample-minimal
spec:
  workloadRef:
    name: test-workload-name
```

This is a fully configured example:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: XWorkloadRegistration
metadata:
  name: sample-full
spec:
  workloadRef:
    name: test-workload-name
  redirectPaths:
    - /redirect/uri/1
    - /redirect/uri/2
  displayName: "Full sample app"
  scopes:
    - name: openid
    - name: email
    - name: profile
    - name: roles
    - name: coffee.make
      description: bestows the ultimate power
  authorizationGrantTypes:
    - client_credentials
    - authorization_code
    - refresh_token
  clientAuthenticationMethod: client_secret_basic
  requireUserConsent: true
```

Known issues for Application Single Sign-On

This topic describes known limitations and workarounds related to working with Application Single Sign-On (commonly called AppSSO). For further troubleshooting guidance, see Troubleshoot Application Single Sign-on.

Unregistration by deletion

You can only deregister an existing, ready `ClientRegistration` from its selected `AuthServer` by deleting it. Breaking the match between the two resources by updating either the labels of the `AuthServer` or the label selector on the `ClientRegistration` does not deregister the client from the authorization server.

Limited number of `ClientRegistrations` per `AuthServer`

The number of `ClientRegistration` for an `AuthServer` is limited to around 2,000. This is a soft limitation. If you attempt to apply more `ClientRegistration` resources than the limit, those clients applied past the limit will work. This is subject to change in future product versions.

LetsEncrypt: domain name for Issuer URI limited to 64 characters maximum

If you use LetsEncrypt to issue TLS certificates for an `AuthServer`, the domain name for the Issuer URI (excluding the `http{s}` prefix) cannot exceed 64 characters in length. If exceeded, you might receive a LetsEncrypt specific error during the certificate issuance process. You might observe this limitation when your base domain and subdomain joined together exceed the maximum limit.

If your default Issuer URI is too long, use the `domain_template` field in Application Single Sign-On values YAML to shorten the domain.

For example, you can forgo the namespace in the Issuer URI as follows:

```
domain_template: "{{.Name}}.{{.Domain}}"
```


Caution

By leaving out the namespace in your domain template, application routes might conflict if there are multiple `AuthServer`s with the same name but in different namespaces.

`ClassClaim` credential propagation time

It can take up to 60 to 120 seconds for the client credentials to propagate up into a `ClassClaim`'s service binding secret.

Application Single Sign-On for OpenShift clusters

On OpenShift clusters, AppSSO must run with a custom SecurityContextConstraint (SCC) to enable compliance with restricted Kubernetes Pod Security Standards. Tanzu Application Platform configures the following SCC for AppSSO controller and its `AuthServer` managed resources when you configure the `kubernetes_distribution: openshift` key in the `tap-values.yaml` file.

Specification follows:

```
---
kind: SecurityContextConstraints
apiVersion: security.openshift.io/v1
metadata:
  name: appsso-scc
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: false
allowPrivilegedContainer: false
allowedCapabilities: null
defaultAddCapabilities: null
fsGroup:
  type: MustRunAs
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities:
  - KILL
  - MKNOD
  - SETUID
  - SETGID
```

```
runAsUser:
  type: MustRunAsNonRoot
seLinuxContext:
  type: MustRunAs
volumes:
  - configMap
  - downwardAPI
  - emptyDir
  - persistentVolumeClaim
  - projected
  - secret
seccompProfiles:
  - 'runtime/default'
```

AppSSO controller's `ServiceAccount` is given the following additional permissions, including a `use` permission for AppSSO SCC, so `AuthServer` can use the custom SCC:

```
- apiGroups:
    - security.openshift.io
  resources:
    - securitycontextconstraints
  verbs:
    - "get"
    - "list"
    - "watch"
```

```
- apiGroups:
    - security.openshift.io
  resourceNames:
    - appsso-scc
  resources:
    - securitycontextconstraints
  verbs:
    - "use"
```

Package configuration for Application Single Sign-On

In most cases, the Application Single Sign-on (commonly called AppSSO) package installation is configured by using the meta package installation of Tanzu Application Platform (commonly called TAP). The Tanzu Application Platform package has a `shared` top-level configuration key for sharing common configuration between the packages it installs.

Application Single Sign-on inherits the `shared.{ingress_domain, ingress_issuer, ca_cert_data, kubernetes_distribution}` configuration values from Tanzu Application Platform. You can override these values with Application Single Sign-On specific values under `appsso`. Application Single Sign-On specific configuration has precedence over the shared values of Tanzu Application Platform.

For example:

```
#! my-tap-values.yaml

shared:
# Shared configuration goes here.

appsso:
# AppSSO-specific configuration goes here.
```

Configuration values

The package installation of `sso.apps.tanzu.vmware.com` has the following configuration values:

`ca_cert_data`

You can configure trust for custom CAs by providing their certificates as a PEM bundle to `ca_cert_data`. As a result, all `AuthServer`s trust your custom CAs.

This is useful if you have Identity providers serving certificates from a custom CA and configuring AuthServer storage.

Alternatively, you can configure trust for a single AuthServer.

Note

Application Single Sign-on specific `ca_cert_data` is concatenated with `shared.ca_cert_data`. The resulting PEM bundle contains both.

`cluster_resource_namespace`

This setting designates the target namespace when the cluster-scoped APIs reconcile into namespace-scoped resources.

In particular, the cluster-scoped ClusterUnsafeTestLogin reconciles into namespace-scoped resources such as an `AuthServer`. These resources are placed in the `cluster_resource_namespace`.

The default value of `cluster_resource_namespace` is `appsso`.

Note

Updating the `cluster_resource_namespace` of an existing package installation delete or create namespace-scoped resources in the new namespace. This causes downtime for the impacted resources momentarily.

`default_authserver_clusterissuer`

You can denote a `cert-manager.io/v1/ClusterIssuer` as a default issuer for `AuthServer.spec.tls.issuerRef` and omit `AuthServer.spec.tls`. When the value of `AuthServer.spec.tls.issuerRef` is the empty string `""`, no default issuer is assumed and no `AuthServer.spec.tls` is required.

If you configured `shared.ingress_issuer` and omitted `default_authserver_clusterissuer` while installing Tanzu Application Platform, Application Single Sign-on uses the ingress issuer of Tanzu Application Platform and sets `default_authserver_clusterissuer` to `shared.ingress_issuer`.

`default_workload_domain_template`

This is the default template from which WorkloadRegistration renders redirect URIs. It is used when `WorkloadRegistration.spec.workloadDomainTemplate` is omitted.

This is a golang text/template. The default is `"{{.Name}}.{{.Namespace}}.{{.Domain}}"`.

The domain template is applied with the configured `workload_domain_name` and the name and namespace specified in `WorkloadRegistration.spec.workloadRef`. For more information, see Redirect URI templating.

VMware recommends maintaining consistency between the values of `default_workload_domain_template` and Cloud Native Runtimes' `domain_template`. Both values

share the same default configuration. However, for customization purposes, using the same value for both settings ensures that `Workloads` get the expected redirect URIs templated.

domain_name

The Application Single Sign-on package has one required configuration value, its `domain_name`. It templates the issuer URI for `AuthServer`. `domain_name` must be the shared ingress domain of your Tanzu Application Platform package installation. If your Tanzu Application Platform installation is configured with `shared.ingress_domain`, Application Single Sign-on inherits the correct configuration.

If omitted, `domain_name` is set to `shared.ingress_domain`.

domain_template

You can customize how Application Single Sign-on template's `issuerURIs` with the `domain_template` configuration. This is a golang text/template. The default value is `"{{.Name}}.{{.Namespace}}.{{.Domain}}"`.

The domain template is applied with the given `domain_name` and the `AuthServer`'s name and namespace:

- `{{.Domain}}` evaluates to the configured `domain_name`
- `{{.Name}}` evaluates to `AuthServer.metadata.name`
- `{{.Namespace}}` evaluates to `AuthServer.metadata.namespace`

To use a wild-card certificate, consider `"{{.Name}}-{{.Namespace}}.{{.Domain}}"`.

VMware recommends keeping the name and namespace sections of the template to avoid domain name collisions.

kubernetes_distribution

This setting toggles behavior for specific Kubernetes distributions. Currently, the only supported values are `""` and `openshift`.

It is processed in combination with `kubernetes_version`.

Application Single Sign-On installs OpenShift specific RBAC and resources. For more information, see Application Single Sign-On for OpenShift clusters.

If omitted, `kubernetes_distribution` is set to `shared.kubernetes_distribution`.

kubernetes_version

This setting toggles behavior for specific Kubernetes distributions. Currently, the only supported values are `""` or semantic versions in the form of `\d*\.\d*\.\d*`.

It is processed in combination with `kubernetes_distribution`.

Application Single Sign-On installs OpenShift specific RBAC and resources. For more information, see Application Single Sign-On for OpenShift clusters.

If omitted, `kubernetes_version` is set to `shared.kubernetes_version`.

workload_domain_name

This is used as the value for `{{.Domain}}` in WorkloadRegistration's domain template.

`workload_domain_name` defaults to the value of `domain_name`.

In most cases, the value of `workload_domain_name` matches the value of Cloud Native Runtimes' `domain_name`.

`replicas`

The controller is run with this many replicas. The default value is `1`.

The controller uses leader election so that there is only a single active replica at a time. Increasing this value does not improve the controller's performance.

`resources`

This is the value for the controller's `Deployment.spec.template.spec.containers[0].resources`.

See Configuration schema for more information about its structure and default values.

`resync_period`

This is the duration after which the controller re-synchronizes all resources. That means that every instance of a Application Single Sign-On API is reconciled at this point.

The default value is `2h`.

VMware does not recommend customizing this value in all practical scenarios.

Configuration schema

The package installation of `sso.apps.tanzu.vmware.com` has the following configuration schema:

```
---

#@schema/desc "Optional: Kubernetes platform distribution that this package is being i
nstalled on. Accepted values: ['','openshift']"
kubernetes_distribution: ""

#@schema/desc "Optional: Kubernetes platform version that this package is being instal
led on. Accepted format: ['x.x.x']"
kubernetes_version: ""

#@schema/desc "Domain name for AuthServers."
domain_name: "example.com"

#@schema/desc "Optional: Golang template/text string for constructing AuthServer FQDN
s."
domain_template: "{{.Name}}.{{.Namespace}}.{{.Domain}}"

#@schema/desc "Optional: A cert-manager.io/v1/ClusterIssuer for defaulting AuthServer
TLS."
default_authserver_clusterissuer: ""

#@schema/desc "Optional: Domain name for Workloads. Defaults to the value of domain_na
me."
workload_domain_name: ""

#@schema/desc "Optional: Golang template/text string for defaulting Workload FQDNs tem
plating."
default_workload_domain_template: "{{.Name}}.{{.Namespace}}.{{.Domain}}"

#@schema/desc "The namespace which children of cluster-scoped resources are located i
n."
cluster_resource_namespace: "appsso"

#@schema/desc "Optional: PEM-encoded certificate data for AuthServers to trust TLS con
```

```
nections with a custom CA."
ca_cert_data: ""

#@schema/desc "Optional: Interval at which the controller will re-synchronize applied
resources."
resync_period: "2h"

#@schema/desc "Optional: Number of controller replicas to deploy."
replicas: 1

#@schema/desc "Optional: Resource requirements of the controller deployment."
resources:
  requests:
    #@schema/desc "CPU request of the controller."
    cpu: "20m"
    #@schema/desc "Memory request of the controller."
    memory: "100Mi"
  limits:
    #@schema/desc "CPU limit of the controller."
    cpu: "500m"
    #@schema/desc "Memory limit of the controller."
    memory: "500Mi"
```

RBAC for AppSSO

The Application Single Sign-On (commonly called AppSSO) package aggregates the following permissions into Tanzu Application Platform's well-known roles. For more information, see Role descriptions for Tanzu Application Platform.

User aggregated rules

app-operator

```
apiGroups:
  - sso.apps.tanzu.vmware.com
resources:
  - clientregistrations
  - workloadregistrations
verbs:
  - '*'
```

app-editor

```
apiGroups:
  - sso.apps.tanzu.vmware.com
resources:
  - clientregistrations
  - workloadregistrations
verbs:
  - get
  - list
  - watch
```

app-viewer

```
apiGroups:
  - sso.apps.tanzu.vmware.com
resources:
  - clientregistrations
  - workloadregistrations
```

```
verbs:
  - get
  - list
  - watch
```

service-operator

```
apiGroups:
  - sso.apps.tanzu.vmware.com
resources:
  - authservers
  - clusterunsafetestlogins
  - clusterworkloadregistrationclasses
verbs:
  - '*'
```

Controller

To manage the life cycle of AppSSO's APIs, the AppSSO controller's `ServiceAccount` has a `ClusterRole` with the following permissions:

```
- apiGroups:
    - sso.apps.tanzu.vmware.com
  resources:
    - authservers
    - clientregistrations
    - clusterunsafetestlogins
    - clusterworkloadregistrationclasses
    - workloadregistrations
  verbs:
    - '*'
- apiGroups:
    - sso.apps.tanzu.vmware.com
  resources:
    - authservers/status
    - clientregistrations/status
    - clusterunsafetestlogins/status
    - clusterworkloadregistrationclasses/status
    - workloadregistrations/status
  verbs:
    - patch
    - update
- apiGroups:
    - sso.apps.tanzu.vmware.com
  resources:
    - authservers/finalizers
    - clientregistrations/finalizers
    - clusterunsafetestlogins/finalizers
    - clusterworkloadregistrationclasses/finalizers
    - workloadregistrations/finalizers
  verbs:
    - '*'
- apiGroups:
    - ""
  resources:
    - events
  verbs:
    - create
    - update
    - patch
- apiGroups:
    - coordination.k8s.io
  resources:
```

```
    - leases
  verbs:
    - create
    - get
    - update
- apiGroups:
    - ""
  resources:
    - secrets
    - configmaps
    - services
    - serviceaccounts
  verbs:
    - '*'
- apiGroups:
    - apps
  resources:
    - deployments
  verbs:
    - '*'
- apiGroups:
    - rbac.authorization.k8s.io
  resources:
    - roles
    - rolebindings
  verbs:
    - '*'
- apiGroups:
    - cert-manager.io
  resources:
    - certificates
    - issuers
  verbs:
    - '*'
- apiGroups:
    - cert-manager.io
  resources:
    - clusterissuers
  verbs:
    - get
    - list
    - watch
- apiGroups:
    - networking.k8s.io
  resources:
    - ingresses
  verbs:
    - '*'
- apiGroups:
    - servicebinding.io
  resources:
    - servicebindings
  verbs:
    - '*'
- apiGroups:
    - services.apps.tanzu.vmware.com
  resources:
    - clusterinstanceclasses
  verbs:
    - '*'
- apiGroups:
    - services.apps.tanzu.vmware.com
  resources:
    - clusterinstanceclasses
  verbs:
    - '*'
```

```
- apiGroups:
    - apiextensions.crossplane.io
  resources:
    - compositions
  verbs:
    - '*'
```

AppSSO also installs OpenShift specific RBAC and resources. For more information, see Application Single Sign-On for OpenShift clusters.

AuthServer audit logs for AppSSO

This topic tells you how to use `AuthServer` audit logs in Application Single Sign-On (commonly called AppSSO).

Overview

`AuthServer`s perform the following tasks:

- Handle user authentication
- Issue `id_token` and `access_token`

Each audit event contains the following information:

- `ts` - date/time of the event
- `remoteIpAddress` - the IP of the user-authentication or if not attainable, the IP of the last proxy

Authentication

`AuthServer` produce the following authentication events:

- `AUTHENTICATION_SUCCESS`
  - **Trigger** successful authentication
  - **Data recorded** Username, Provider ID, Provider Type (INTERNAL, OPENID, ...)
- `AUTHENTICATION_LOGOUT`
  - **Trigger** successful logout
  - **Data recorded** Username, Provider ID, Provider Type (INTERNAL, OPENID, ...)
- `AUTHENTICATION_FAILURE`
  - **Trigger** failed authentication using either `internalUnsafe` or `ldap` identity provider
  - **Data recorded** Username, Provider ID, Provider Type (INTERNAL or LDAP)
- `INVALID_IDENTITY_PROVIDER_CONFIGURATION`
  - **Trigger** some cases of failed authentication with an `openId` or `saml` identity provider
  - **Data recorded** Provider ID, Provider Type, error
  - **Note** usually followed by a human-readable help message, with `"logger": "appsso.help"`

Token flows

`AuthServer` produce the following authorization_code and token events:

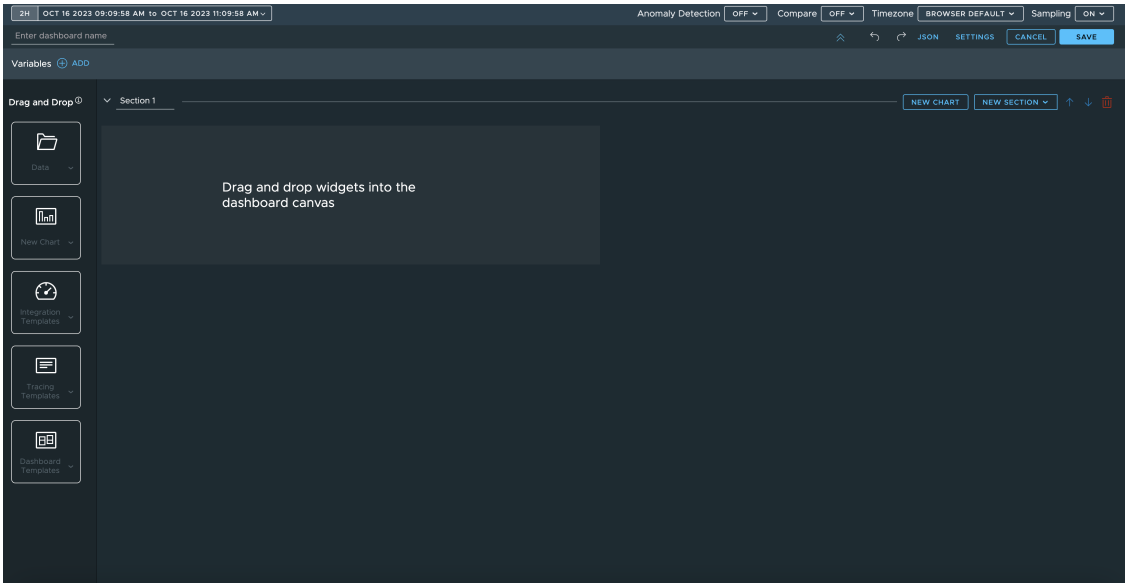
- `AUTHORIZATION_CODE_ISSUED`

    - **Trigger** `authorization_code` grant type, successful call to `/oauth2/authorize`
    - **Data recorded** Username, Provider ID, Provider Type, Client ID, Scopes requested, Redirect URI
- `AUTHORIZATION_CODE_REQUEST_REJECTED`
    - **Trigger** `authorization_code` grant type, unsuccessful call to `/oauth2/authorize`, for example invalid Client ID, invalid Redirect URI, ...
    - **Data recorded** Error, Error Code (ex: `invalid_scope`), Client ID, Scopes requested Redirect URI, Username (may be `anonymousUser`), Provider ID and Provider Type if available
- `TOKEN_ISSUED`
    - **Trigger** successful call to `/oauth2/token`
    - **Data recorded** Scopes, Client ID, Grant Type (`authorization_code` or `client_credentials`), Username
- `TOKEN_REQUEST_REJECTED`
    - **Trigger** unsuccessful call to `/oauth2/token`, for example invalid Client Secret
    - **Data recorded** Client ID, Scopes requested, Error

Overview of the Aria Operations for Applications dashboard for Tanzu Application Platform (beta)

The Aria Operations for Applications (AOA) dashboard helps you monitor the health of a cluster. The AOA dashboard shows whether the deployed Tanzu Application Platform (commonly known as TAP) components are behaving as expected.

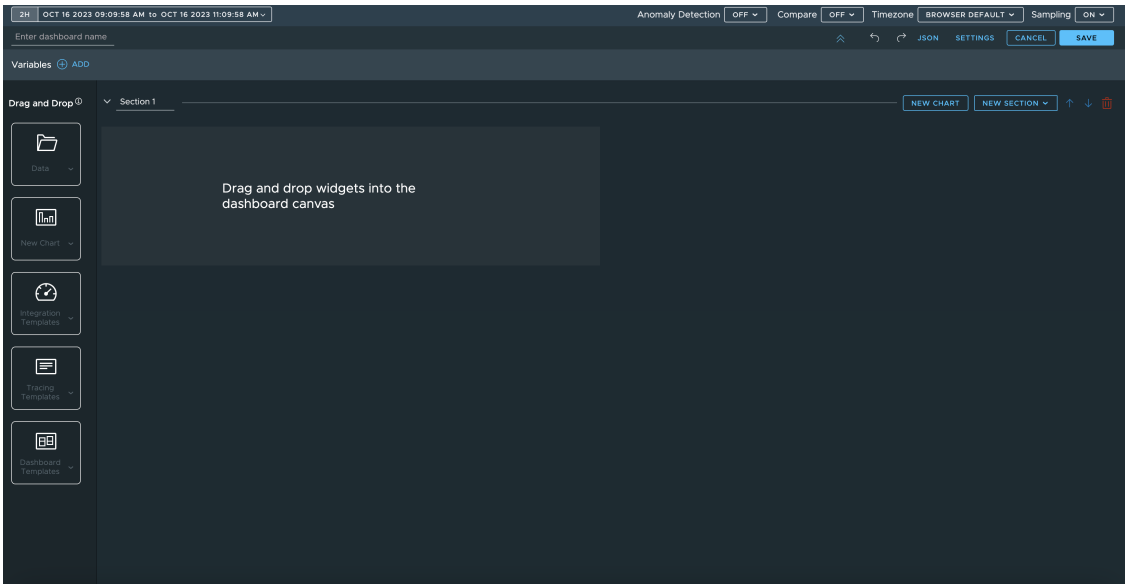
Aria Operations for Applications (formerly Tanzu Observability) powers AOA dashboard.



Overview of the Aria Operations for Applications dashboard for Tanzu Application Platform (beta)

The Aria Operations for Applications (AOA) dashboard helps you monitor the health of a cluster. The AOA dashboard shows whether the deployed Tanzu Application Platform (commonly known as TAP) components are behaving as expected.

Aria Operations for Applications (formerly Tanzu Observability) powers AOA dashboard.



Install the Aria Operations for Applications dashboard for Tanzu Application Platform (beta)

This topic tells you how to integrate a Kubernetes cluster of any distribution with Aria Operations for Applications (AOA) so that you can use the AOA dashboard. This topic also tells you how to set up and configure AOA dashboard for use with Tanzu Application Platform (commonly known as TAP).

Prerequisites

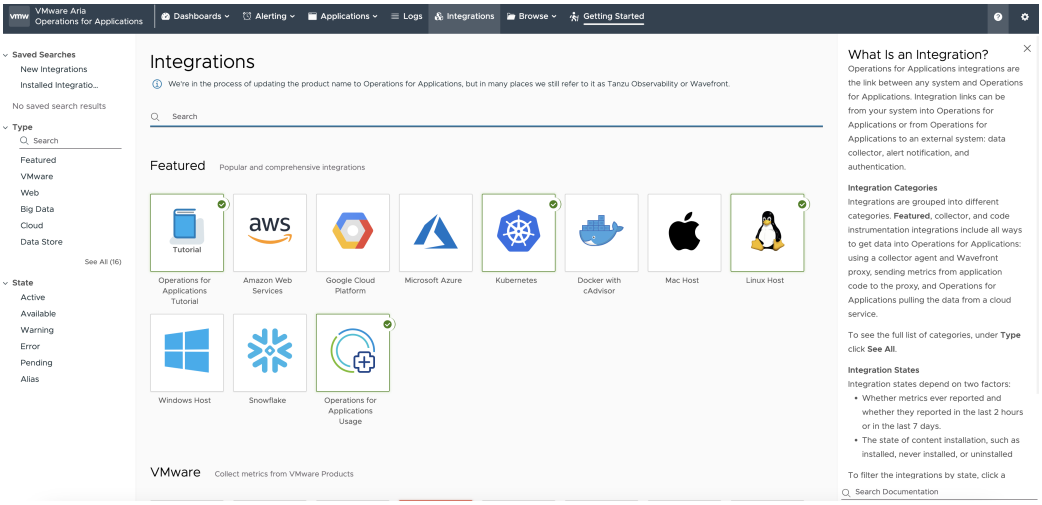
Before integrating a Kubernetes cluster with AOA you must have:

- AOA Wavefront access with permissions to integrate Kubernetes clusters and view dashboards
- A Kubernetes cluster with Tanzu Application Platform installed

Integrate with AOA Wavefront

To integrate with AOA Wavefront:

1. Log in to your AOA instance, click the **INTEGRATIONS** tab, and then click **Kubernetes**.



2. Click **ADD AN INTEGRATION INSTANCE** on the next page.

3. Fill out the **Kubernetes Integration Setup** page:
    1. Select **Kubernetes Cluster** as the distribution type.
    2. Type the cluster name in the **Cluster Name** text box.
    3. Enable **Logs**.
    4. Enable **Metrics**.
    5. Configure authentication. If the token is not listed, verify that you have AOA Wavefront access.
4. Run the kubectl commands in the **Deployment Script** text box on the cluster that you want to onboard.

Set up metrics collection in a cluster

Perform the following procedures to set up metrics collection in a cluster.

Download the dashboard and set up the cluster

To download the dashboard and set up the cluster:

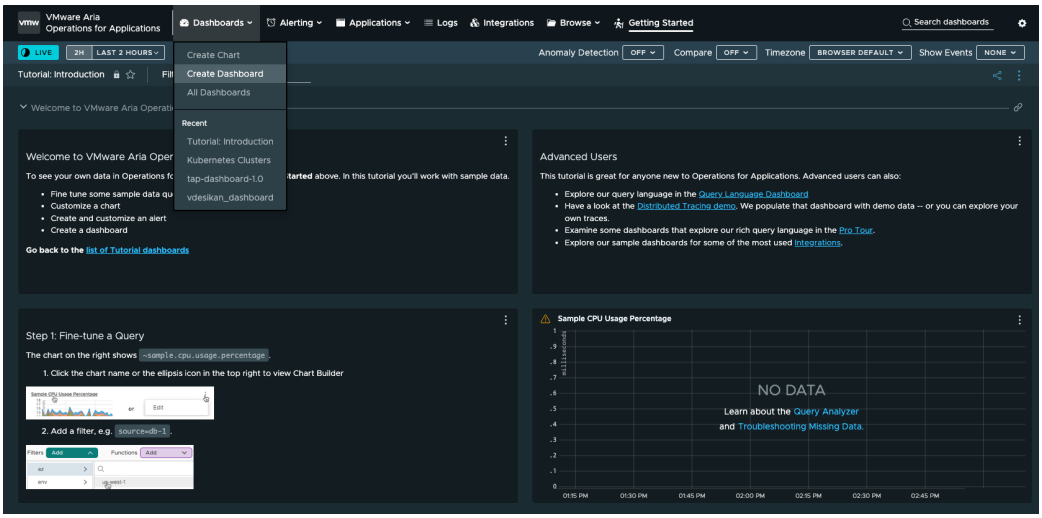
1. Download and install the Tanzu CLI binary file from the Tanzu Application Platform tile in VMware Tanzu Network if you have not already installed it.
2. Download the dashboard zip file from the tile in VMware Tanzu Network for Tanzu Application Platform v1.9.1 or later.
    - Click **Aria Operations for Applications Dashboard for Tanzu Application Platform (Beta)**.
    - Download the `aoa-dashboard-for-tap` zip file for your operating system.
3. Use an extraction tool to unpack the binary file.
4. Apply the included `tap-metrics.yaml` file to the onboarded cluster, which enables the collection of Tanzu Application Platform `CustomResource` metrics, by running:

    ```
    kubectl apply -f tap-metrics.yaml
    ```

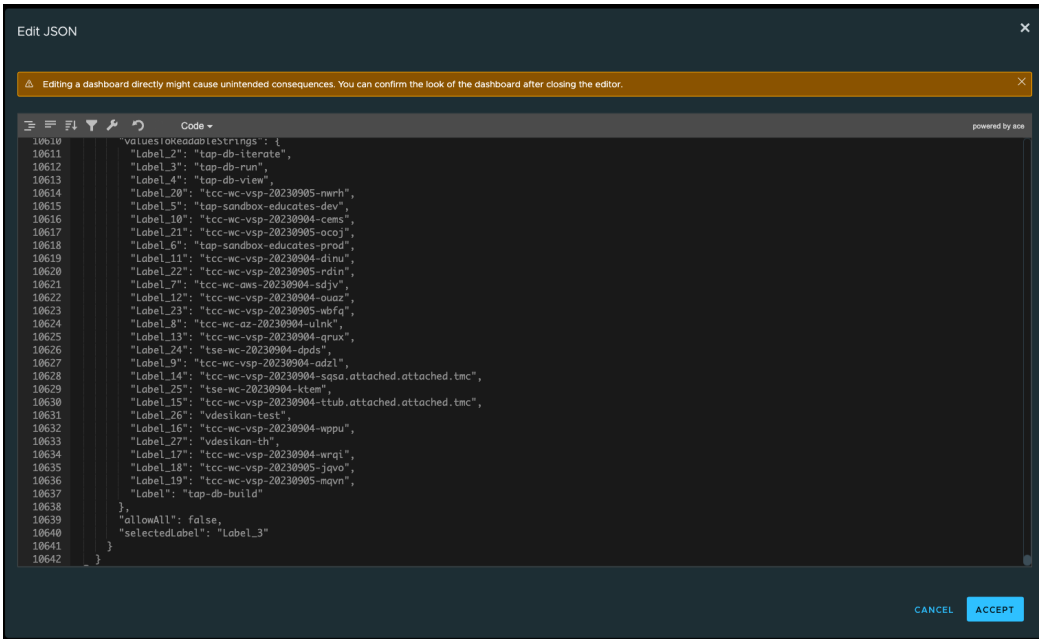
Create the dashboard in AOA Wavefront

To create a dashboard in AOA Wavefront:

1. Go to the AOA Wavefront home page and then click **Dashboards** > **Create Dashboard**.



2. Click **JSON** in the upper right corner.
3. Click **Tree** > **Code** and extract the downloaded content from `tap-health-dashboard.json`.
4. Copy the content from `tap-health-dashboard.json` into the code block and then click **ACCEPT**.



5. Click **SAVE** in the top right corner to save the dashboard.
6. In the search text box, type the name of the cluster that you onboarded and select it from the list of available clusters.

(Optional) Onboard additional clusters

The dashboard currently only supports monitoring one cluster at a time. But you can follow the steps in Set up metrics collection in a cluster to onboard additional clusters to AOA. The additional clusters then appear in the **Cluster** drop-down menu.

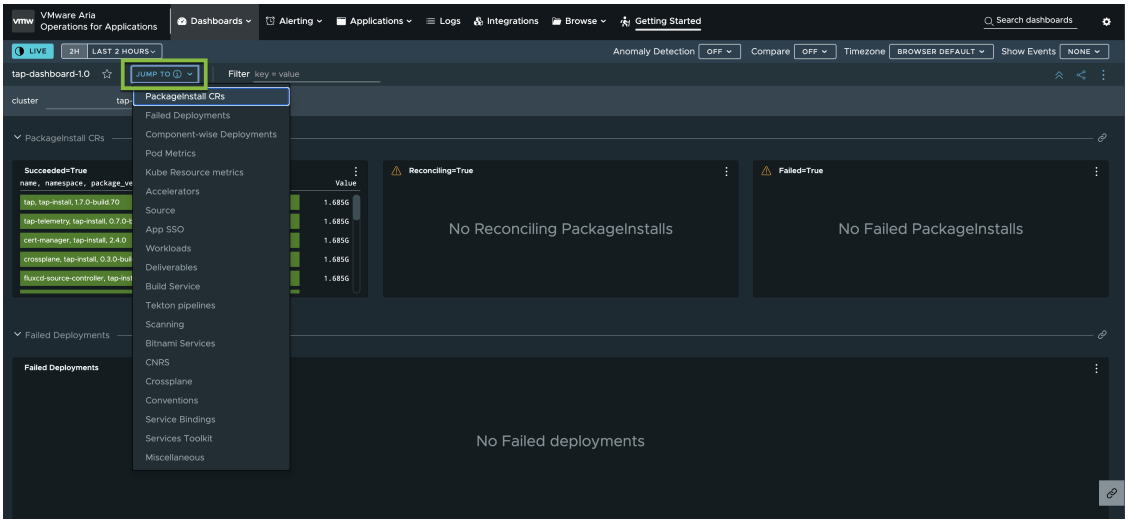
Use the Aria Operations for Applications dashboard for Tanzu Application Platform (beta)

This topic tells you about using the Aria Operations for Applications (AOA) dashboard for use with Tanzu Application Platform (commonly known as TAP).

Use this dashboard to monitor:

- Statuses of the various Tanzu Application Platform-related `PackageInstall` resources that are associated with each Tanzu Application Platform component
- Failed deployments and the statuses of deployments related to Tanzu Application Platform components
- Pod metrics, which include metrics such as memory and CPU use
- The health of individual Tanzu Application Platform components

Use the **JUMP TO** drop-down menu to quickly go to a view.



Overview of AWS Services

AWS Services provides an integration with Amazon Web Services (AWS) for Tanzu Application Platform (commonly known as TAP).

Through integration with Crossplane and Services Toolkit, you can offer these AWS services for apps teams to consume with only minimal setup and configuration required from ops teams. This makes it quick and easy to get started working with AWS services on Tanzu Application Platform.

Getting started

If this is your first time working with AWS Services on Tanzu Application Platform, you can start by reading About the AWS Services package to help you to understand the goals of the package and some of its limitations.

After you understand the concepts, if you are a service operator, you can Install AWS Services. If you are an app developer, see Working with AWS Services.

You can also see the How-to guides and Reference material.

Overview of AWS Services

AWS Services provides an integration with Amazon Web Services (AWS) for Tanzu Application Platform (commonly known as TAP).

Through integration with Crossplane and Services Toolkit, you can offer these AWS services for apps teams to consume with only minimal setup and configuration required from ops teams. This

makes it quick and easy to get started working with AWS services on Tanzu Application Platform.

Getting started

If this is your first time working with AWS Services on Tanzu Application Platform, you can start by reading About the AWS Services package to help you to understand the goals of the package and some of its limitations.

After you understand the concepts, if you are a service operator, you can Install AWS Services. If you are an app developer, see Working with AWS Services.

You can also see the How-to guides and Reference material.

Install AWS Services

This topic tells you how to install the AWS Services package from the Tanzu Application Platform (commonly known as TAP) package repository.

Note

The AWS Services package is not in any of the Tanzu Application Platform profiles. To use this package, you must follow the instructions in this topic.

Prerequisites

Before you install the AWS Services package:

- Install Tanzu Application Platform
- VMware recommends that you read About the AWS Services package. The topic provides context about the features and goals of the package, and some of the considerations and compromises that were made as part of its development.

Step 1: Plan and configure your infrastructure

There are a wide range of infrastructure and networking setups available when integrating services on AWS into Tanzu Application Platform. Therefore, the first step is to decide which of these setups you want and to configure the AWS Services package for this topology.

Note

This section provides setup guidance for the most simple setup, which is a Tanzu Application Platform cluster running on AWS EKS in a virtual private cloud (VPC) connecting to RDS PostgreSQL service instances running in the same VPC.

To plan and configure your infrastructure:

1. Decide which topology you want to use. For more information about the topologies supported by the AWS Services package, see Supported Topologies.
2. Create a DBSubnetGroup and SecurityGroups:
    1. To learn how to create a DBSubnetGroup, see the AWS documentation.
    2. To learn how to create SecurityGroups, see the AWS documentation.

> **Note**
>
> The current version of the AWS Services package does not create these resources for you. You must create them manually using the AWS console. This is a one-time manual setup step that you must complete before installing the package.

3. If you are configuring RabbitMQ, choose an existing subnet or create one. For how to create a subnet, see the AWS documentation.
4. Record the name of the DBSubnetGroup, which includes IDs of the SecurityGroups and Subnets. These are required when installing the package.
5. Complete any remaining configuration tasks listed in Supported Topologies.

Step 2: Install the AWS Services package

The AWS Services package is not installed as part of any profile so you must explicitly install it. To install the AWS Services package:

1. Confirm that you have the AWS Services package available by running:

```
tanzu package available get aws.services.tanzu.vmware.com -n tap-install
```

2. Prepare an `aws-services-values.yaml` file to configure the installation:

```
# aws-services-values.yaml
---
# Optional, add any custom CA certificate data required by your Tanzu Applicati
on Platform installation
ca_cert_data: |
    -----BEGIN CERTIFICATE-----
    MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...
    -----END CERTIFICATE-----

# Optional, the ARN for the role to be associated with the service account runn
ing the providers.
role_arn: "ROLE-ARN"

# Configuration specific to the RDS PostgreSQL service
postgresql:
  # Enable the RDS PostgreSQL service. The default is set to false.
  enabled: true
  region: "REGION"
  provider_config_ref:
    name: "PROVIDER-CONFIG-NAME"
  # Infrastructure configuration for the RDS PostgreSQL service
  infrastructure:
    subnet_group:
      name: "SUBNET-GROUP-NAME"
    security_groups:
      - id: "SECURITY-GROUP-ID"
  # Instance-level configuration for the RDS PostgreSQL service applied to all
service instances
  # All instance_configuration is optional. See below for default values.
  instance_configuration:
    instance_class: "INSTANCE-CLASS"
    engine_version: "ENGINE-VERSION"
    skip_final_snapshot: SKIP-FINAL-SNAPSHOT
    publicly_accessible: PUBLICLY-ACCESSIBLE
    maintenance_window: "MAINTENANCE-WINDOW"
```

```
# Configuration specific to the RDS MySQL service
mysql:
  # Enable the RDS MySQL service. The default is set to false.
  enabled: true
  region: "REGION"
  provider_config_ref:
    name: "PROVIDER-CONFIG-NAME"
  # Infrastructure configuration for the RDS MySQL service
  infrastructure:
    subnet_group:
      name: "SUBNET-GROUP-NAME"
    security_groups:
      - id: "SECURITY-GROUP-ID"
  # Instance-level configuration for the RDS MySQL service applied to all servi
ce instances
  # All instance_configuration is optional. See below for default values.
  instance_configuration:
    instance_class: "INSTANCE-CLASS"
    engine_version: "ENGINE-VERSION"
    skip_final_snapshot: SKIP-FINAL-SNAPSHOT
    publicly_accessible: PUBLICLY-ACCESSIBLE
    maintenance_window: "MAINTENANCE-WINDOW"

# Configuration specific to the Amazon MQ (RabbitMQ) service
rabbitmq:
  # Enable the Amazon MQ (RabbitMQ) service. The default is set to false.
  enabled: true
  region: "REGION"
  provider_config_ref:
    name: "PROVIDER-CONFIG-NAME"
  # Infrastructure configuration for the Amazon MQ (RabbitMQ) service
  infrastructure:
    subnet_id: "SUBNET-ID"
    security_groups:
      - id: "SECURITY-GROUP-ID"
  # Instance-level configuration for the Amazon MQ (RabbitMQ) service applied t
o all service instances
  # All instance_configuration is optional. See below for default values.
  instance_configuration:
    publicly_accessible: PUBLICLY-ACCESSIBLE
    engine_version: "ENGINE-VERSION"
    instance_class: "INSTANCE-CLASS"
    maintenance_window_start_time:
      day_of_week: "DAY-OF-WEEK"
      time_of_day: "TIME-OF-DAY"
      time_zone: "TIMEZONE"
```

Where:

- (Optional) `ROLE_ARN` is the ARN for the role to be associated with the service account running the providers.
- `REGION` is the AWS region you want, for example, `us-east-1`.
- `PROVIDER-CONFIG-NAME` is the name of the ProviderConfig for this service. Choose a name, or enter `default`. Choosing a name allows you to use a different ProviderConfig per service type offered by the AWS Services package.
- `SUBNET-GROUP-NAME` is the name of the DBSubnetGroup you created in Plan and configure your infrastructure earlier.
- `SECURITY-GROUP-ID` are the IDs of any security groups you created in Plan and configure your infrastructure earlier.
- `SUBNET-ID` is the ID of the subnet used for the resource that you chose or created in Plan and configure your infrastructure earlier.

- `INSTANCE-CLASS` is the instance type of the RDS instance. The default is `db.t3.micro`.
- `ENGINE-VERSION` is the engine version. For PostgreSQL the default is `13.7`. For MySQL the default is `8.0`. You can find the list of available versions in the AWS documentation.
- `SKIP-FINAL-SNAPSHOT` is whether a final snapshot is created before the instance is deleted. If you specify `true`, no snapshot is created. If you specify `false`, a snapshot called `final-snapshot-INSTANCE-NAME` is created before the instance is deleted. The default is `false`.
- `PUBLICLY-ACCESSIBLE` is whether or not PostgreSQL service instances are publicly accessible over the Internet. The value can be `true` or `false` depending on the topology you chose. See Supported Topologies. The default is `false`.
- `MAINTENANCE-WINDOW` is the window to perform maintenance in. The syntax is `ddd:hh24:mi-ddd:hh24:mi`. The default is `Mon:00:00-Mon:03:00`.
- `DAY-OF-WEEK` is the day of the week to perform maintenance in. The syntax is `MONDAY`. The default is `MONDAY`.
- `TIME-OF-DAY` is time of day to perform maintenance in. The syntax is `00:00` (24 hour). The default is `00:00`.
- `TIMEZONE` is the timezone for the maintenance window. The syntax is `UTC`. The default is `UTC`.

For the full list of values you can configure, see Package values for AWS Services.

3. Review which versions of AWS Services are available to install by running:

```
tanzu package available list -n tap-install aws.services.tanzu.vmware.com
```

For example:

```
$ tanzu package available list -n tap-install aws.services.tanzu.vmware.com
  NAME                               VERSION           RELEASED-AT
  aws.services.tanzu.vmware.com  0.1.0             2023-11-07 14:35:15 +0000 UT
C
```

4. Install the AWS Services package by running:

```
tanzu package install aws-services \
  --package aws.services.tanzu.vmware.com \
  --version VERSION-NUMBER \
  --namespace tap-install \
  --values-file aws-services-values.yaml
```

Where `VERSION-NUMBER` is the AWS Services version you want to install. For example, `0.1.0`.

5. Verify that the package installed by running:

```
tanzu package installed get aws-services -n tap-install
```

In the output, confirm that the `STATUS` value is `Reconcile succeeded`.

For example:

```
$ tanzu package installed get aws-services -n tap-install
NAMESPACE:          tap-install
NAME:               aws-services
PACKAGE-NAME:       aws.services.tanzu.vmware.com
PACKAGE-VERSION:    0.1.0
```

```
STATUS:             Reconcile succeeded
CONDITIONS:         - type: ReconcileSucceeded
  status: "True"
  reason: ""
  message: ""
```

Step 3: Configure credentials for access to your AWS account

You configure credentials and access information for your AWS account through the `ProviderConfig` resource.

This section shows you how to create a `ProviderConfig` using the `Secret` source in which your AWS account credentials are stored in a `Secret` on the cluster. However, there are alternative methods, for example, an option to assume an IAM Role. To learn about the full range of configuration options available, see the Upbound documentation.

To create a `ProviderConfig` using the `Secret` source:

1. Create the `Secret` to hold the AWS credentials by running:

   ```
   export AWS_ACCESS_KEY_ID="foo"
   export AWS_SECRET_ACCESS_KEY="bar"

   echo -e "[default]\naws_access_key_id = $AWS_ACCESS_KEY_ID\naws_secret_access_k
   ey = $AWS_SECRET_ACCESS_KEY" > creds.conf

   # (optional) if you are required to use a session token to access your AWS acco
   unt, you must also set AWS_SESSION_TOKEN
   # export AWS_SESSION_TOKEN=""
   # echo -e "aws_session_token = $AWS_SESSION_TOKEN" >> creds.conf

   kubectl create secret generic aws-creds -n crossplane-system --from-file=creds
   =./creds.conf
   rm -f creds.conf
   ```

2. Create a `ProviderConfig` and configure it with the `Secret` source by running:

   ```
   kubectl apply -f -<<EOF
   ---
   apiVersion: aws.upbound.io/v1beta1
   kind: ProviderConfig
   metadata:
     name: PROVIDER-CONFIG-NAME
   spec:
     credentials:
       source: Secret
       secretRef:
         namespace: crossplane-system
         name: aws-creds
         key: creds
   EOF
   ```

   Where `PROVIDER-CONFIG-NAME` is the `postgresql.provider_config_ref.name` value you configured in your `aws-services-values.yaml` file. The default is `default`.

3. Verify your setup by inspecting the `SubnetGroup` and `SecurityGroups` resources created as part of the installation of the package by running:

   ```
   kubectl get securitygroup
   kubectl get subnetgroup
   ```

When both resources report `SYNCED: True`, the AWS providers have connected to your AWS account and pulled down the information about each of the resources.

AWS Services concepts

This section introduces you to AWS Services concepts.

In this section:

- About the AWS Services package

About the AWS Services package

This topic tells you about the goal of the AWS Services package and some of its limitations.

Goals

The AWS Services package aims to provide simple and seamless integration into Tanzu Application Platform for a variety of services from AWS. The goal is to minimize the time to value so that users can both offer and consume the services that they want on Tanzu Application Platform with minimal effort.

Limitations

The AWS Services package has to accommodate a wide range of infrastructure and networking setups. Each setup has different infrastructure and configuration requirements.

Example setups include:

- A Tanzu Application Platform cluster running on AWS EKS in “VPC A” connecting to RDS PostgreSQL service instances running in “VPC A”
- A Tanzu Application Platform cluster running on AWS EKS in “VPC A” connecting to RDS PostgreSQL service instances running in “VPC B”
- A Tanzu Application Platform cluster running on AWS EKS in “VPC A” connecting to RDS MySQL service instances running in “VPC B”
- A Tanzu Application Platform cluster running on Azure AKS connecting to RDS MySQL service instances running in “VPC A”

If not handled carefully, accommodating these differences could lead to the package configuration becoming too complex.

To control the potential complexity, the AWS Services package makes the following compromises:

- You might have to complete one-time infrastructure setup tasks, for example, creating an RDS DBSubnetGroup.
- All service instances for each service type, such as RDS PostgreSQL, are created in the same VPC.
- There is no multi-region support.
- Instance-level configuration, such as instance class or engine version, is applied globally to all service instances of a service type.

As the AWS Services package matures and VMware learns more about how it is being used, VMware might remove some of these compromises if it can be achieved while keeping the experience as simple as possible. This is not guaranteed.

Other Considerations

The AWS Services package is not installed as part of any of the Tanzu Application Platform profiles. This is because not every user needs to integrate services from AWS into their Tanzu Application Platform cluster. However, the package is in the Tanzu Application Platform packageRepository, which means that you can install it using the same tooling, credentials, and image repositories that you use to install Tanzu Application Platform. See Install AWS Services.

You might not want to integrate all of the services supported by the package. Therefore, none of the services are enabled by default. It is up to service operators to enable the services they want using the Package values. For the list of services you can enable, see Supported AWS Services.

AWS Services tutorials

This section contains tutorials for how to use AWS Services.

In this section:

- Working with AWS Services

Working with AWS Services

In this tutorial you learn how application operators and application developers can use AWS Services on Tanzu Application Platform. You will learn how to discover, claim, and bind services to application workloads.

Prerequisites

Before starting this tutorial:

- VMware recommends that you read About the AWS Services package. The topic provides context about the goals of the package, and some of the considerations and compromises that were made as part of its development.
- Install AWS Services.

About this tutorial

Target user roles: Application Operator, Application Developer
Complexity: Basic
Estimated time: 15 minutes
Topics covered: AWS, RDS, PostgreSQL, MySQL, Classes, Claims
Learning outcomes: An understanding of how to work with RDS PostgreSQL and MySQL instances on Tanzu Application Platform using the AWS Services package

Step 1: Discover the list of available AWS services

To find out about available AWS services:

1. To discover the list of available services, run:

   ```
   tanzu service class list
   ```

   The expected output has two services: `postgresql-managed-aws` and `mysql-managed-aws`.

2. To learn more about the service, including which claim parameters are provided, run:

```
tanzu service class get CLASS-NAME
```

Where `CLASS-NAME` is one of the services listed in the previous step.

Step 2: Claim an instance of an AWS RDS service

To claim an instance of an AWS RDS service:

```
tanzu service class-claim create CLAIM-NAME --class CLASS-NAME
```

Where:

- `CLAIM-NAME` is the name you choose for your claim, for example, `rds-1`.
- `CLASS-NAME` is one of the services listed earlier.

It takes about 5 to 10 minutes for the service instance to be provisioned. You can watch the claim and wait for it to transition into a `READY: True` state to know when it is ready to use.

Step 3: Bind the service to a workload

After creating the claim, you can bind it to one or more of your application workloads.

Important

If you want to bind to more than one application workload, all application workloads must be in the same namespace. This is a known limitation. For more information, see Cannot claim and bind to the same service instance from across multiple namespaces.

1. Find the reference for the claim by running:

   ```
   tanzu service class-claim get CLAIM-NAME
   ```

   Where `CLAIM-NAME` is the name of the claim you want to bind your workload to, for example, `rds-1`.

   The reference is in the output under the heading Claim Reference.

2. Bind the claim to a workload of your choice by passing a reference to the claim to the `--service-ref` flag of the `tanzu apps workload create` command. For example:

   ```
   tanzu apps workload create my-workload --image my-registry/my-app-image --servi
   ce-ref db=services.apps.tanzu.vmware.com/v1alpha1:ClassClaim:rds-1
   ```

   You must pass the claim reference with a corresponding name that follows the format `--service-ref db=services.apps.tanzu.vmware.com/v1alpha1:ClassClaim:rds-1`. The `db=` prefix to this example reference is an arbitrary name for the reference.

AWS Services how-to guides

This section contains how-to guides for AWS Services.

In this section:

- Configure AWS Service Endpoint
- Troubleshoot AWS Services

Configure the AWS service endpoint

This topic describes how to configure the service endpoint that the providers included with the AWS Services package use.

Overview

By default, AWS providers make calls to the default service endpoint for each service in an AWS region. However, you can configure the providers to make calls to an endpoint of your choice.

Configure the endpoint by using a ProviderConfig resource

To configure the service endpoint through a `ProviderConfig` resource:

1. Update the `ProviderConfig` you created when installing the package. For example:

   ```
   ---
   apiVersion: aws.upbound.io/v1beta1
   kind: ProviderConfig
   metadata:
     name: custom-endpoint
   spec:
     endpoint:
       url:
         static: https://my-custom-aws-endpoint.com
         type: Static
   # ...
   ```

   For the full list of configuration options, see the Upbound documentation.

2. In your `aws-services-values.yaml` file, configure the AWS Services package to use the `ProviderConfig` you just configured. For example:

   ```
   # aws-services-values.yaml
   ---
   postgresql:
     provider_config_ref:
       name: custom-endpoint
   # ...
   ```

3. Update the AWS Services package:

   ```
   tanzu package installed update aws-services -n tap-install --values-file aws-se
   rvices-values.yaml
   ```

Troubleshoot AWS Services

This topic explains how you troubleshoot issues related to AWS Services on Tanzu Application Platform (commonly known as TAP).

Secret key name for Amazon MQ claims do not match the name used in the Spring Cloud Bindings library

Symptom:

When you create a claim for Amazon MQ (RabbitMQ), the resulting binding secret contains a key named `endpoint`. This does not match the key name that the Spring Cloud Bindings library expects,

which is `addresses`. As a result, when you bind Spring-based workloads to the Amazon MQ service, the connection is not established automatically.

Cause:

This issue was caused by a misconfiguration in the `CompositeResourceDefinition` and `Composition` for the Amazon MQ service. These resources have been updated to resolve the issue. However, Crossplane does not support changes to `connectionSecretKeys` in `CompositeResourceDefinition` resources.

Solution:

The following workaround is only required if you have upgraded Tanzu Application Platform from v1.8.0 or v1.8.1.

To workaround this issue:

1. Find the name of the Crossplane pod, for example:

   ```
   kubectl get pod -lapp=crossplane -n crossplane-system
   ```

2. Delete the pod, for example:

   ```
   kubectl delete pod CROSSPLANE-POD-NAME -n crossplane-system
   ```

   Where `CROSSPLANE-POD-NAME` is the name of the Crossplane pod you retrieved.

After you delete the pod, it is automatically recreated. This causes Crossplane to re-read the updated list of `connectionSecretKeys` from the `CompositeResourceDefinition`. New claims for Amazon MQ (RabbitMQ) will now contain the correct set of key names.

AWS Services reference

This section provides reference documentation for AWS Services.

In this section:

- Package values
- Supported Services
- Supported Topologies
- Version matrix

Package values for AWS Services

This topic lists the keys and values that you can use to configure the behavior of the AWS Services package.

Globals

The following table lists global configuration that applies across all services.

KEY	DEFAULT	TYPE	DESCRIPTION
ca_cert_data	`""`	string	PEM-encoded certificate data for the AWS Providers to trust TLS connections with a custom CA
globals.crossplane_system_namespace	`crossplane-system`	string	The name of the namespace in which Crossplane and the providers are deployed to

KEY	DEFAULT	TYPE	DESCRIPTION
globals.create_clusterroles	`true`	boolean	Specifies whether to create default ClusterRoles that grant `claim` permissions to all Tanzu Application Platform application operators
role_arn	""	string	The ARN for the role to be associated with the service account running the providers. Necessary when setting `spec.credentials.source` to `IRSA` in your `ProviderConfig`. For more information, see the Upbound documentation.

PostgreSQL

The following table lists configuration that applies to the `postgresql` service.

KEY	DEFAULT	TYPE	DESCRIPTION
postgresql.enabled	`false`	boolean	Enables the PostgreSQL service class
postgresql.infrastructure.security_groups	*n/a*	array	Security groups for your PostgreSQL databases to belong to
postgresql.infrastructure.subnet_group.name	`""`	string	Subnet group for your PostgreSQL databases
postgresql.instance_configuration.engine_version	`"13.7"`	string	The PostgreSQL version. For more information, see the AWS documentation.
postgresql.instance_configuration.instance_class	`"db.t3.micro"`	string	The instance type of the RDS instance. For more information, see the AWS documentation.
postgresql.instance_configuration.maintenance_window	`"Mon:00:00-Mon:03:00"`	string	The window to perform maintenance in. Syntax: `ddd:hh24:mi-ddd:hh24:mi`, for example, `Mon:00:00-Mon:03:00`. For more information, see the AWS documentation.
postgresql.instance_configuration.publicly_accessible	`false`	boolean	Controls whether your instances are publicly accessible
postgresql.instance_configuration.skip_final_snapshot	`false`	boolean	Determines whether a final snapshot is created before the instance is deleted. If you specify `true`, no snapshot is created. If you specify `false`, a snapshot called `final-snapshot-INSTANCE-NAME` is created before the instance is deleted.
postgresql.provider_config_ref.name	`"default"`	string	Name of your ProviderConfig for the `postgresql` service
postgresql.region	`"us-east-1"`	string	The AWS region to create databases in

MySQL

The following table lists configuration that applies to the `mysql` service.

KEY	DEFAULT	TYPE	DESCRIPTION
mysql.enabled	`false`	boolean	Enables the MySQL service class

KEY	DEFAULT	TYPE	DESCRIPTION
mysql.infrastructure.security_groups	*n/a*	array	Security groups for your MySQL databases to belong to
mysql.infrastructure.subnet_group.name	`""`	string	Subnet group for your MySQL databases
mysql.instance_configuration.engine_version	`"8.0"`	string	The MySQL version. For more information, see the AWS documentation.
mysql.instance_configuration.instance_class	`"db.t3.micro"`	string	The instance type of the RDS instance. For more information, see the AWS documentation.
mysql.instance_configuration.maintenance_window	`"Mon:00:00-Mon:03:00"`	string	The window to perform maintenance in. Syntax: `ddd:hh24:mi-ddd:hh24:mi`, for example, `Mon:00:00-Mon:03:00`. For more information, see the AWS documentation.
mysql.instance_configuration.publicly_accessible	`false`	boolean	Controls whether your instances are publicly accessible
mysql.instance_configuration.skip_final_snapshot	`false`	boolean	Determines whether a final snapshot is created before the instance is deleted. If you specify `true`, no snapshot is created. If you specify `false`, a snapshot called `final-snapshot-INSTANCE-NAME` is created before the instance is deleted.
mysql.provider_config_ref.name	`"default"`	string	Name of your ProviderConfig for the `mysql` service
mysql.region	`"us-east-1"`	string	The AWS region to create databases in

RabbitMQ

The following table lists configuration that applies to the `rabbitmq` service.

KEY	DEFAULT	TYPE	DESCRIPTION
rabbitmq.enabled	`false`	boolean	Enables the RabbitMQ service class
rabbitmq.infrastructure.security_groups	*n/a*	array	The security groups for your RabbitMQ brokers to belong to. Do not provide any security groups if you're setting `rabbitmq.instance_configuration.publicly_accessible` to true.
rabbitmq.infrastructure.subnet_id	`""`	string	The ID of the subnet for your RabbitMQ brokers to belong to
rabbitmq.instance_configuration.publicly_accessible	`false`	boolean	Controls if your instances are publicly accessible
rabbitmq.instance_configuration.engine_version	`3.11.20`	string	The RabbitMQ version. For more information, see the AWS documentation.
rabbitmq.instance_configuration.instance_class	`mq.t3.micro`	string	The instance type of the MQ broker. For more information, see AWS documentation.
rabbitmq.instance_configuration.maintenance_window_start_time.day_of_week	`MONDAY`	string	The day of the week to perform maintenance in. Possible values: `MONDAY`, `TUESDAY`, `WEDNESDAY`, `THURSDAY`, `FRIDAY`, `SATURDAY`, `SUNDAY`.

KEY	DEFAULT	TYPE	DESCRIPTION
rabbitmq.instance_configuration.maintenance_window_start_time.time_of_day	`"00:00"`	string	The time to perform maintenance in in 24-hour format
rabbitmq.instance_configuration.maintenance_window_start_time.time_zone	`UTC`	string	The time zone for the maintenance window
rabbitmq.provider_config_ref.name	`default`	string	The name of the ProviderConfig to use to create your RabbitMQ brokers
rabbitmq.region	`us-east-1`	string	The AWS region to create brokers in

Supported AWS services

This topic lists the services that the AWS Services package supports.

In Tanzu Application Platform v1.9.1, the AWS Services package supports the following AWS services:

- RDS PostgreSQL
- RDS MySQL
- Amazon MQ (RabbitMQ)

VMware plans to add more services to the package in future releases.

Supported topologies for AWS Services

This topic gives you the list of Tanzu Application Platform (commonly known as TAP) cluster to service instance Virtual Private Cloud (VPC) topologies that the AWS Services package supports. Each supported topology lists relevant package values configurations and one-time manual setup steps.

The topologies described in this topic are available for the PostgreSQL, MySQL, and RabbitMQ services.

Topology 1: service instance accessed by a workload in the same VPC

Topology 1 is a service instance in a VPC accessed by a workload in a Tanzu Application Platform cluster in the same VPC.

This topology is very similar to a database instance in a VPC accessed by an EC2 instance in the same VPC as described in the AWS documentation.

Key properties for topology 1

The key properties of this topology are:

- Uses subnets to separate where application workloads run and where service instances run
- Uses security groups to manage access between the subnets
- Service instances are not publicly accessible

This topology is recommended if your Tanzu Application Platform cluster is running in AWS.

Configuration tasks for topology 1

To configure the service from the AWS Services package for this type of topology you must:

- Manually create all required subnets and security groups in AWS.
- Add a rule to permit the TCP port from the subnet where the Tanzu Application Platform application workloads run to the subnet where the service instances run. Create the rule for TCP port 5432 for PostgreSQL and MySQL, or 5671 for RabbitMQ.
- For PostgreSQL and MySQL, create a database subnet group consisting of the subnets.

For instructions for these tasks, see the AWS documentation.

After completing configuration in AWS, you must configure your `aws-services-values.yaml` file using the following values when installing the package:

PostgreSQL

```
postgresql:
  enabled: true
  region: "REGION"
  infrastructure:
    subnet_group:
      name: "SUBNET-GROUP-NAME"
    security_groups:
      - id: "SECURITY-GROUP-ID"
```

MySQL

```
mysql:
  enabled: true
  region: "REGION"
  infrastructure:
    subnet_group:
      name: "SUBNET-GROUP-NAME"
    security_groups:
      - id: "SECURITY-GROUP-ID"
```

RabbitMQ

```
rabbitmq:
  enabled: true
  region: "REGION"
  infrastructure:
    subnet_id: "SUBNET-ID"
    security_groups:
      - id: "SECURITY-GROUP-ID"
```

Topology 2: service instance accessed by a workload external to AWS

Topology 2 is a service instance in a VPC accessed by a workload in a Tanzu Application Platform cluster running external to AWS.

This topology is very similar to a database instance in a VPC accessed by a client application through the Internet as described in the AWS documentation.

Key properties for topology 2

The key properties of this topology are:

- Uses a public subnet in which to run service instances

- Uses an Internet gateway to provide connectivity over the Internet
- Uses security groups to manage access to the subnet
- Service instances are publicly accessible over the Internet

This topology is recommended if your Tanzu Application Platform cluster is running external to AWS, for example, on-prem or in another cloud such as Azure.

Configuration tasks for topology 2

To configure the service from the AWS Services package for this type of topology you must:

- Manually create all required subnets and security groups in AWS.
- Create an Internet gateway.
- Add a rule to permit TCP port 5432 (PostgreSQL, MySQL) or 5671 (RabbitMQ) from the subnet where the Tanzu Application Platform application workloads run to the subnet where the service instances run.
- Create a database subnet group (PostgreSQL, MySQL) consisting of the subnets.

For instructions for these tasks, see the AWS documentation.

After completing configuration in AWS, you must configure your `aws-services-values.yaml` file using the following values when installing the package:

PostgreSQL

```
postgresql:
  enabled: true
  region: "REGION"
  infrastructure:
    subnet_group:
      name: "SUBNET-GROUP-NAME"
    security_groups:
      - id: "SECURITY-GROUP-ID"
  instance_configuration:
    publicly_accessible: true
```

MySQL

```
mysql:
  enabled: true
  region: "REGION"
  infrastructure:
    subnet_group:
      name: "SUBNET-GROUP-NAME"
    security_groups:
      - id: "SECURITY-GROUP-ID"
  instance_configuration:
    publicly_accessible: true
```

RabbitMQ

```
rabbitmq:
  enabled: true
  region: "REGION"
  infrastructure:
    subnet_id: "SUBNET-ID"
  instance_configuration:
    publicly_accessible: true
```

Version matrix for AWS Services

This topic provides you with a version matrix for the AWS Services package and its open source components in Tanzu Application Platform v1.9 (commonly known as TAP).

To view this information for another Tanzu Application Platform version, select the version from the drop-down menu at the top of this page.

The following table has the component versions for the AWS Services package.

Component	Version
AWS Services package	0.3.0
provider-aws	1.2.0

Note

Tanzu Application Platform patch releases are only added to the table when there is a change to one or more of the other versions in the table. Otherwise, the corresponding versions remain the same for each Tanzu Application Platform patch release.

Uninstall AWS Services

This topic tells you how to uninstall the AWS Services package from Tanzu Application Platform (commonly known as TAP).

Step 1: Prepare your infrastructure

Before uninstalling the package, you must decide whether to retain or delete the existing resources that have been claimed.

Failing to follow one of the following procedures for all of your resources leaves your Tanzu Application Platform in a state that might prevent you from reinstalling the AWS Services package in the future.

Retain the resources

If you want to keep a resource:

1. Find the Composite Resource associated with your claim by running:

   ```
   kubectl get classclaim CLASS-CLAIM-NAME -n CLASS-CLAIM-NAMESPACE -ojsonpath="{.status.provisionedResourceRef}"
   ```

   Where:

   - `CLASS-CLAIM-NAME` is the name of the claim.
   - `CLASS-CLAIM-NAMESPACE` is the namespace the claim is in.

   Example output for a PostgreSQL claim:

   ```
   {"apiVersion":"aws.database.tanzu.vmware.com/v1alpha1","kind":"XPostgreSQLInstance","name":"rds-2-r4mgc"}
   ```

2. Find the `Instance` associated with the Composite Resource by running:

```
kubectl get XR-API XR-NAME -ojsonpath="{.spec.resourceRefs}"
```

Where:

- `XR-API` is in the format `KIND.APIVERSION` using the `kind` and `apiVersion` from the output of the previous step. `APIVERSION` is the part of `apiVersion` before the `/`, for example, `aws.database.tanzu.vmware.com`.
- `XR-NAME` is the value of `name` from the output of the previous step.

For example:

```
$ kubectl get xpostgresqlinstance.aws.database.tanzu.vmware.com rds-2-r4mgc -oj
sonpath="{.spec.resourceRefs}"

[{"apiVersion":"rds.aws.upbound.io/v1beta1","kind":"Instance","name":"rds-2-r4m
gc-zc69h"}]
```

3. Open the `Instance` resource for editing by running:

   ```
   kubectl edit instance.rds.aws.upbound.io INSTANCE-NAME
   ```

   Where `INSTANCE-NAME` is the value of `name` from the output of the previous step.

   For example:

   ```
   $ kubectl edit instance.rds.aws.upbound.io rds-2-r4mgc-zc69h
   ```

4. Change the `Instance` resource `spec.deletionPolicy` field from `Delete` to `Orphan`:

   ```
   apiVersion: rds.aws.upbound.io/v1beta1
   kind: Instance
   metadata:
     # ...
   spec:
     deletionPolicy: Orphan
     # ...
   ```

5. Save, and close your editor.
6. Delete the claim by running:

   ```
   tanzu service class-claim delete CLAIM-NAME
   ```

Delete the resources

If you want to delete a resource:

1. Delete the corresponding claim by running:

   ```
   tanzu service class-claim delete CLAIM-NAME
   ```

2. Wait for the resource to disappear in the AWS console.

Step 2: Uninstall the AWS Services package

To uninstall the AWS Services package:

1. Confirm that you have the AWS Services package installed by running:

   ```
   tanzu package installed list -A
   ```

2. Search the `PACKAGE-NAME` column of the output for `aws.services.tanzu.vmware.com`, and record the name and namespace.
3. Uninstall the package:

   ```
   tanzu package installed delete PACKAGE-NAME -n PACKAGE-NAMESPACE
   ```

   Where `PACKAGE-NAME` and `PACKAGE-NAMESPACE` are the values you recorded in the previous step.

   For example:

   ```
   $ tanzu package installed delete aws-services -n tap-install
   ```

Step 3: Delete the `ProviderConfig`

You must delete the `ProviderConfig` you created after installing the AWS Services package to complete the uninstall.

Crossplane installs a finalizer when the `ProviderConfig` is created. To delete the `ProviderConfig` you must remove the finalizer. If you don't do this step, the `ProviderConfig` and its related Custom Resource Definition (CRD) remain in the cluster and prevent you from reinstalling the package in the future.

1. Open the `ProviderConfig` resource for editing by running:

   ```
   kubectl edit providerconfig.aws.upbound.io PROVIDER-CONFIG-NAME
   ```

   Where `PROVIDER-CONFIG-NAME` is the name you chose for your `ProviderConfig`.
2. In the `ProviderConfig`:
   1. Record the name and namespace of the `Secret` referenced in `spec.credentials`.
   2. Delete the `metadata.finalizers` entry. This allows the `ProviderConfig` to be deleted.
3. Save, and close your editor. The `ProviderConfig` is automatically deleted.
4. Verify the `ProviderConfig` has been removed by running:

   ```
   kubectl get providerconfig.aws.upbound.io
   ```

   The expected output is an error message like this:

   ```
   Error from server (NotFound): Unable to list "aws.upbound.io/v1beta1, Resource=
   providerconfigs": the server could not find the requested resource (get provide
   rconfigs.aws.upbound.io)
   ```
5. Delete the `Secret` by running:

   ```
   kubectl delete secret SECRET-NAME -n SECRET-NAMESPACE
   ```

   Where `SECRET-NAME` and `SECRET-NAMESPACE` are the values you recorded earlier.

Overview of Bitnami Services

Bitnami Services provides a set of backing services for Tanzu Application Platform (commonly known as TAP). The services are MySQL, PostgreSQL, RabbitMQ, Redis, MongoDB, and Kafka all of which are backed by the corresponding Bitnami Helm Chart.

Through integration with Crossplane and Services Toolkit, these six services are immediately ready for apps teams to consume, with no additional setup or configuration required from ops teams. This makes it incredibly quick and easy to get started working with services on Tanzu Application Platform.

Note

The Bitnami Services package provides unmanaged services that are not designed to support long lived instances. Therefore, there is no supported path to upgrade individual instances. Bitnami Services are instantiated using the configuration and version of the package at creation time and so upgrading the Bitnami Services package has no effect on existing instances. VMware discourages changing the `compositionUpdatePolicy` and `compositionRevisionRef` on the individual composite resources (XRs) because this might cause unintended side effects.

Getting started

If this is your first time working with Bitnami Services on Tanzu Application Platform, you can start with the tutorial Working with Bitnami Services. Otherwise, see the how-to guides and reference material.

Overview of Bitnami Services

Bitnami Services provides a set of backing services for Tanzu Application Platform (commonly known as TAP). The services are MySQL, PostgreSQL, RabbitMQ, Redis, MongoDB, and Kafka all of which are backed by the corresponding Bitnami Helm Chart.

Through integration with Crossplane and Services Toolkit, these six services are immediately ready for apps teams to consume, with no additional setup or configuration required from ops teams. This makes it incredibly quick and easy to get started working with services on Tanzu Application Platform.

Note

The Bitnami Services package provides unmanaged services that are not designed to support long lived instances. Therefore, there is no supported path to upgrade individual instances. Bitnami Services are instantiated using the configuration and version of the package at creation time and so upgrading the Bitnami Services package has no effect on existing instances. VMware discourages changing the `compositionUpdatePolicy` and `compositionRevisionRef` on the individual composite resources (XRs) because this might cause unintended side effects.

Getting started

If this is your first time working with Bitnami Services on Tanzu Application Platform, you can start with the tutorial Working with Bitnami Services. Otherwise, see the how-to guides and reference material.

Install Bitnami Services

This topic tells you how to install Bitnami Services from the Tanzu Application Platform (commonly known as TAP) package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Bitnami Services. For more information about profiles, see Components and installation profiles.

Prerequisites

Before installing Bitnami Services, you must:

- Fulfill all prerequisites for installing Tanzu Application Platform
- Install Crossplane
- Install Services Toolkit

Install Bitnami Services

To install Bitnami Services:

1. See what versions of Bitnami Services are available to install by running:

   ```
   tanzu package available list -n tap-install bitnami.services.tanzu.vmware.com
   ```

   For example:

   ```
   $ tanzu package available list -n tap-install bitnami.services.tanzu.vmware.com
     NAME                               VERSION           RELEASED-AT
     bitnami.services.tanzu.vmware.com  0.1.0             2023-03-10 14:35:15 +000
   0 UTC
   ```

2. Install Bitnami Services by running:

   ```
   tanzu package install bitnami-services \
     --package bitnami.services.tanzu.vmware.com \
     --version VERSION-NUMBER \
     --namespace tap-install
   ```

   Where `VERSION-NUMBER` is the Bitnami Services version you want to install. For example, `0.1.0`.

3. Verify that the package installed by running:

   ```
   tanzu package installed get bitnami-services -n tap-install
   ```

   In the output, confirm that the `STATUS` value is `Reconcile succeeded`.

   For example:

   ```
   $ tanzu package installed get bitnami-services -n tap-install
   NAMESPACE:          tap-install
   NAME:               bitnami-services
   PACKAGE-NAME:       bitnami.services.tanzu.vmware.com
   PACKAGE-VERSION:    0.1.0
   STATUS:             Reconcile succeeded
   CONDITIONS:         - type: ReconcileSucceeded
     status: "True"
     reason: ""
     message: ""
   ```

Bitnami Services tutorials

This section contains tutorials for how to use Bitnami Services.

In this section:

- Working with Bitnami Services

Working with Bitnami Services

In this tutorial you learn how application operators can discover, claim, and bind services to application workloads.

Tanzu Application Platform has six services that are available in the Bitnami Services package. These are MySQL, PostgreSQL, RabbitMQ, Redis, MongoDB, and Kafka. The corresponding Bitnami Helm Chart backs each of these services.

About this tutorial

Target user role: Application Operator
Complexity: Basic
Estimated time: 15 minutes
Topics covered: Classes, Claims, Bitnami
Learning outcomes: An understanding of how work with the standard Bitnami services

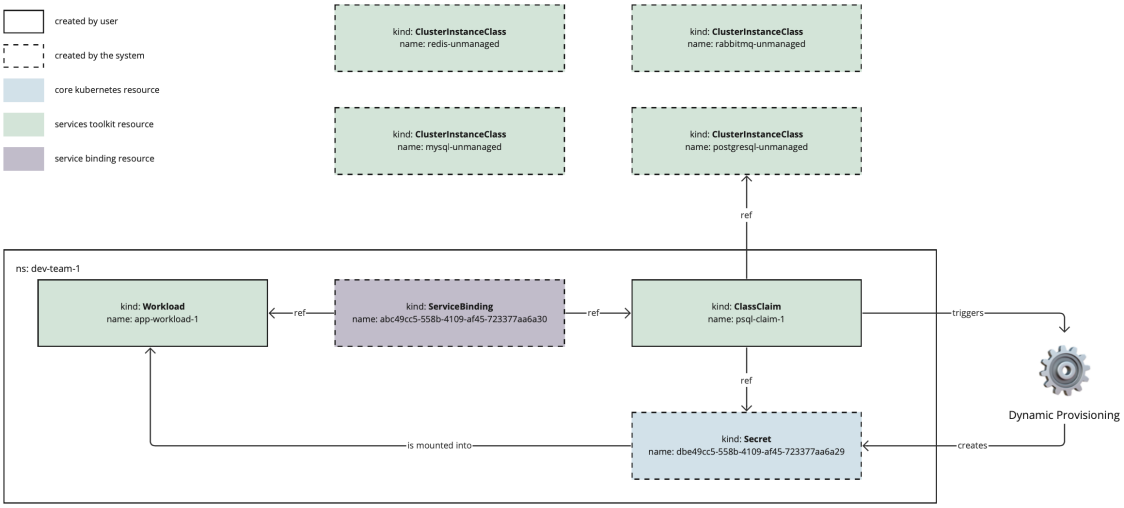
Prerequisites

To follow this tutorial, you must have:

- Access to a Tanzu Application Platform cluster v1.5.0 and later
- The Tanzu services CLI plug-in v0.6.0 and later

Concepts

The following diagram provides an overview of the elements you will use during this tutorial and how they all fit together.



In this diagram:

- There are only two elements that require user input, which are creating a `ClassClaim` and creating a `Workload`. The workload is configured to refer to the class claim`.

- The life cycles of the `ClassClaim` and the `Workload` are separate. This allows you to update one without affecting the other.
- The dynamic provisioning process is simplified. This is intentional because Application Operators and Developers do not need to know about the inner workings and configurations of service instances.

Procedure

The following steps explain how to work with Bitnami Services.

Step 1: Discover services

Application teams can discover the range of services on offer to them by running:

```
tanzu service class list
```

The expected output is similar to the following:

```
  NAME                  DESCRIPTION
  kafka-unmanaged       Kafka by Bitnami
  mongodb-unmanaged     MongoDB by Bitnami
  mysql-unmanaged       MySQL by Bitnami
  postgresql-unmanaged  PostgreSQL by Bitnami
  rabbitmq-unmanaged    RabbitMQ by Bitnami
  redis-unmanaged       Redis by Bitnami
```

Here the output shows six classes. These are the six services available in the Bitnami Services package. You can see from the names and descriptions that they are all *unmanaged* services. This implies that the resulting service instances run on cluster, that is, they are not a managed service running in the cloud. Other classes might be listed here as well.

As an Application Operator, you review the classes on offer and choose one that meets your requirements.

You can learn and discover more about a class by running:

```
tanzu service class get postgresql-unmanaged
```

Example output:

```
NAME:           postgresql-unmanaged
DESCRIPTION:    PostgreSQL by Bitnami
READY:          true

PARAMETERS:
  KEY        DESCRIPTION                                                TYPE     DEF
AULT  REQUIRED
  storageGB  The desired storage capacity of the database, in Gigabytes.  integer  1
false
```

The output shows the name and a short description for the class, its current status, and the parameters. The parameters represent the set of configuration options that are available to application teams.

The `postgresql-unmanaged` class here has one parameter, which is `storageGB`. You can also see that it is not required to pass this parameter when creating a claim for the class, in which case the default value of `1` is used.

Step 2: Claim services

In this example, you have an application workload that requires a PostgreSQL database to function correctly. You can claim the PostgreSQL Bitnami Service to obtain such a database.

To create the claim in a namespace named `dev-team-1`, you must first create the namespace by running:

```
kubectl create namespace dev-team-1
```

You can use the `tanzu service class-claim create` command to create a claim for the `postgresql-unmanaged` class, then bind your application workload to the resulting claim. In this example, you are also choosing to override the default value of `1` for the `storageGB` parameter, setting it instead to `3`. You can override any of the options as you see fit.

```
tanzu service class-claim create psql-1 --class postgresql-unmanaged --parameter stora
geGB=3 -n dev-team-1
```

Example output:

```
Creating claim 'psql-1' in namespace 'dev-team-1'.
Please run `tanzu service class-claim get psql-1 --namespace dev-team-1` to see the pr
ogress of create.
```

As the output states, you can then confirm the status of the claim by using the `tanzu service class-claim get` command as follows:

```
tanzu service class-claim get psql-1 --namespace dev-team-1
```

Example output:

```
Name: psql-1
Namespace: dev-team-1
Claim Reference: services.apps.tanzu.vmware.com/v1alpha1:ClassClaim:psql-1
Class Reference:
  Name: postgresql-unmanaged
Parameters:
  storageGB: 3
Status:
  Ready: True
  Claimed Resource:
    Name: 7974379c-7b4d-41c3-af57-f4f1ae08c65d
    Namespace: dev-team-1
    Group:
    Version: v1
    Kind: Secret
```

It might take a moment or two before the claim reports `Ready: True`. After the claim is ready, you then have a successful claim for a PostgreSQL database configured to your needs with 3 GB of storage.

Step 3: Bind the claim to a workload

After creating the claim, you can bind it to one or more of your application workloads.

Important

If binding to more than one application workload then all application workloads must exist in the same namespace. This is a known limitation. For more information, see

Cannot claim and bind to the same service instance from across multiple namespaces.

1. Find the reference for the claim by running the following command.

   ```
   tanzu service class-claim get psql-1
   ```

   The reference is in the output under the heading Claim Reference.

2. Bind the claim to a workload of your choice by pass a reference to the claim to the `--service-ref` flag of the `tanzu apps workload create` command. For example:

   ```
   tanzu apps workload create my-workload --image my-registry/my-app-image --servi
   ce-ref db=services.apps.tanzu.vmware.com/v1alpha1:ClassClaim:psql-1
   ```

   You must pass the claim reference with a corresponding name that follows the format `--service-ref db=services.apps.tanzu.vmware.com/v1alpha1:ClassClaim:psql-1`. The `db=` prefix to this example reference is an arbitrary name for the reference.

Bitnami Services how-to guides

This section contains how-to guides for Bitnami Services.

In this section:

- Configure private registry and VMware Tanzu Application Catalog integration for Bitnami Services
- Obtain credentials for VMware Tanzu Application Catalog integration
- Upgrading to Tanzu Kubernetes releases v1.26 or later
- Troubleshoot Bitnami Services

Configure private registry and VMware Tanzu Application Catalog integration for Bitnami Services

This topic tells you how to integrate Bitnami Services with private registries or with VMware Tanzu Application Catalog. You can configure this globally for all services, or on a per-service basis.

Prerequisites

Before you integrate Bitnami Services with a private registry or VMware Tanzu Application Catalog, you must:

- Have your Helm Chart repository URL in the format `oci://REGISTRY-HOSTNAME/REPOSITORY-PATH/charts`. Some VMware Tanzu Application Catalog instances append the operating system to the repository URL, in which case, use the URL format `oci://REGISTRY-HOSTNAME/REPOSITORY-PATH/charts/centos-7`.
- Have the credentials to access the private registry.

For how to obtain both of these prerequisites for VMware Tanzu Application Catalog integration, see Obtain credentials for VMware Tanzu Application Catalog integration.

Procedure

1. Create two Kubernetes `Secrets`, one with credentials to pull Helm charts and the other with credentials to pull images. The following examples put these in the `default` namespace, but you can choose to place them in any namespace you prefer.

```
$ kubectl create secret generic tac-chart-pull \
  -n default \
  --from-literal=username='USERNAME' \
  --from-literal=password='TOKEN'
```

```
$ kubectl create secret docker-registry tac-container-pull \
  -n default \
  --docker-server='REGISTRY-HOSTNAME' \
  --docker-username='USERNAME' \
  --docker-password='TOKEN'
```

2. Apply the configuration either to all Bitnami services or to one specific service.
    - **Apply configuration to all Bitnami services:**
        1. Add the following to your `tap-values.yaml` file:

```
bitnami_services:
  globals:
    helm_chart:
      repo: oci://REGISTRY-HOSTNAME/REPOSITORY-PATH/charts # Updat
e this value.
      chart_pull_secret_ref:
        name: tac-chart-pull
        namespace: default
      container_pull_secret_ref:
        name: tac-container-pull
        namespace: default
```

        2. Update Tanzu Application Platform by running:

```
tanzu package installed update tap -p tap.tanzu.vmware.com --value
s-file tap-values.yaml -n tap-install
```

    - **Apply configuration to one specific Bitnami service:**
        1. Add the following to your `tap-values.yaml` file:

```
bitnami_services:
  mysql: # Choose from mysql, postgresql, rabbitmq, redis, mongod
b, and kafka.
    helm_chart:
      repo: oci://REGISTRY-HOSTNAME/REPOSITORY-PATH/charts # Updat
e this value.
      chart_pull_secret_ref:
        name: tac-chart-pull
        namespace: default
      container_pull_secret_ref:
        name: tac-container-pull
        namespace: default
```

        2. Update Tanzu Application Platform by running:

```
tanzu package installed update tap -p tap.tanzu.vmware.com --value
s-file tap-values.yaml -n tap-install
```

3. If your VMware Tanzu Application Catalog instance does not have the default version of a given chart or you want to use a different version, configure the version for the chart by

updating the `helm_chart.version` value for the service. For example:

```
bitnami_services:
  mysql: # Choose from mysql, postgresql, rabbitmq, redis, mongodb, and kafka.
    helm_chart:
      version: VERSION
```

Known issue

As of Tanzu Application Platform v1.5.0 there is a known issue that occurs if you try to configure private registry integration for the Bitnami services after having already created a claim for one or more of the Bitnami services using the default configuration. The issue is that the updated private registry configuration does not appear to take effect. This is due to caching behavior in the system which is not currently accounted for during configuration updates.

Workaround

There is a temporary workaround to this issue, which is to delete the `provider-helm-*` pods in the `crossplane-system` namespace and wait for new pods to come back online after having applied updated registry configuration.

Obtain credentials for VMware Tanzu Application Catalog integration with Bitnami Services

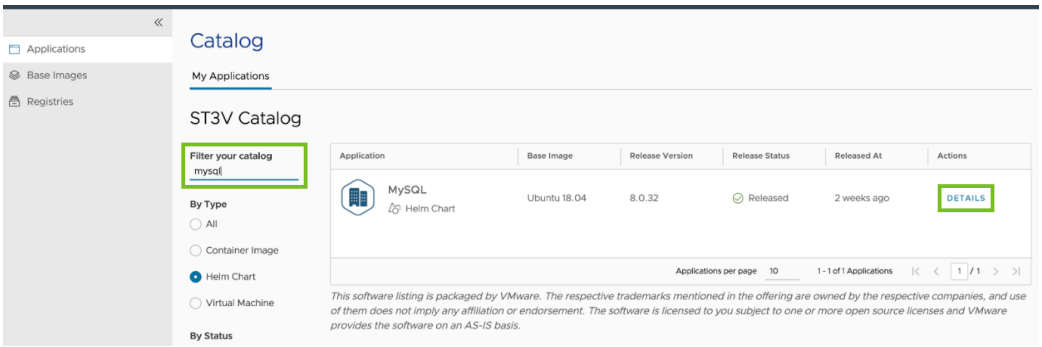
This topic tells you how to obtain credentials for VMware Tanzu Application Catalog to use when following the procedure in Configure private registry and VMware Tanzu Application Catalog integration for Bitnami Services.

Prerequisites

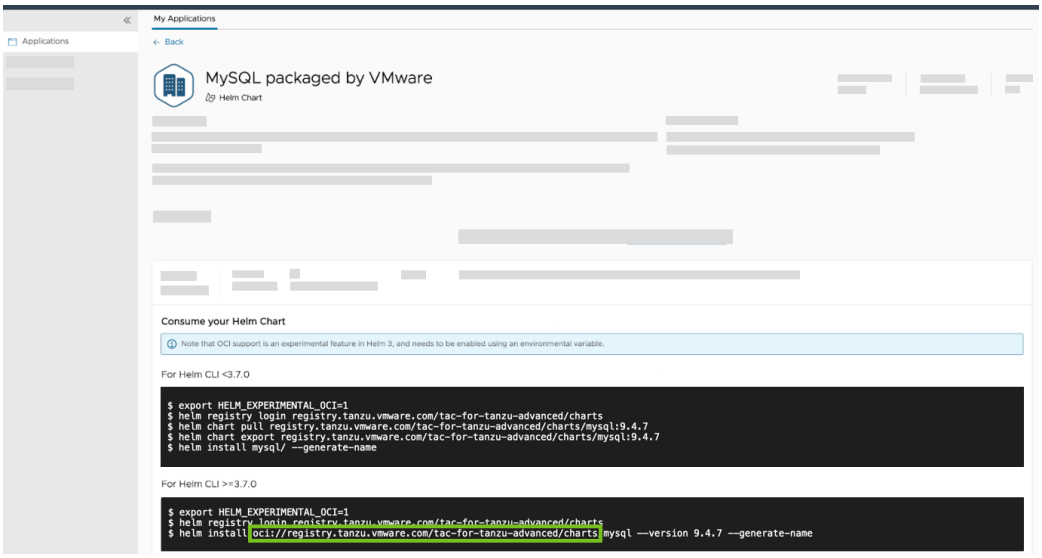
Before obtaining credentials, you must have a VMware Tanzu Application Catalog instance that can create access tokens from within the VMware Tanzu Application Catalog UI.

Obtain the Helm chart repository for VMware Tanzu Application Catalog

1. In VMware Tanzu Application Catalog, navigate to the **Applications** side tab.
2. Under **Filter your catalog**, search for Helm Charts in your catalog, for example, `MySQL`, and click **Details** for one of the charts you found:

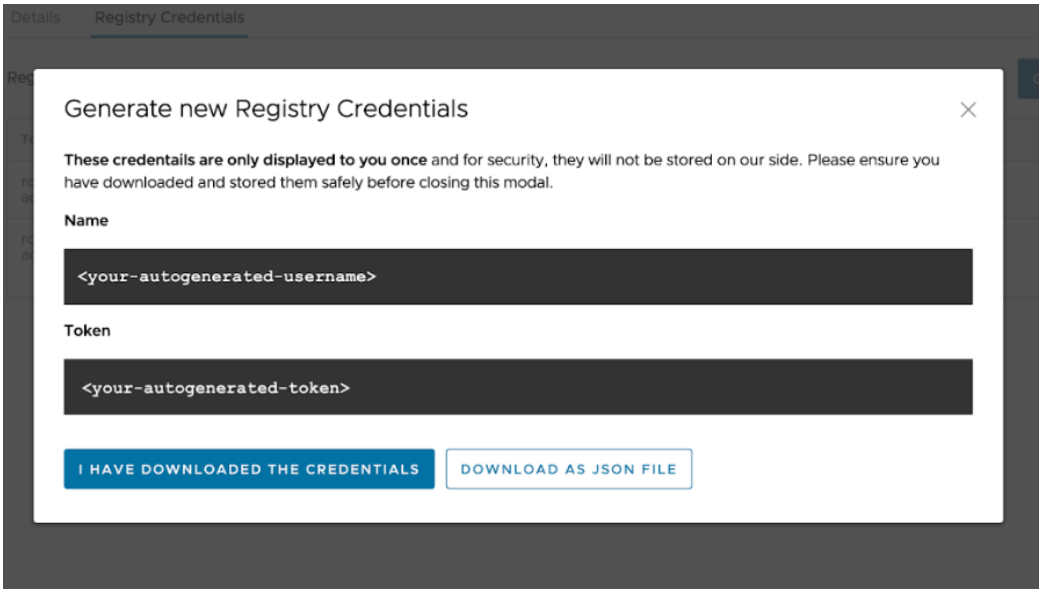


3. Take note of the repository shown under **For Helm CLI >= 3.7.0**. You must include the `oci://` prefix as shown on the page:



Obtain pull credentials for VMware Tanzu Application Catalog

1. In VMware Tanzu Application Catalog, navigate to the **Registries** side tab:
2. Click on the registry that contains your Helm Charts and container images and record the **Registry URL**.
3. Click the **Registry Credentials** tab.
4. Click **Generate New Credentials**.
5. Record the user name and token you are presented with.



You can now take the repository, user name, and token and use it to configure VMware Tanzu Application Catalog integration with the Bitnami services by following the steps in Configure Private Registry and VMware Tanzu Application Catalog Integration for Bitnami Services.

Upgrading to Tanzu Kubernetes releases v1.26 or later

This topic describes how to update your existing Bitnami Services service instances if you upgraded to Tanzu Kubernetes releases v1.26 and later.

Tanzu Kubernetes releases v1.26 and later enforces a restricted Pod Security Standard (PSS) for all pods running on the cluster. This change affects your running services.

New services claimed on Tanzu Application Platform v1.7 and later run with no issues in a restricted PSS. Existing services claimed on Tanzu Application Platform v1.6 will fail to start. To resolve the issue for existing instances, you must update CompositionRevision references for any existing Bitnami Services service instances.

Update existing services

To repair your existing services, upgrade their corresponding managed resources to the latest composition revision:

1. Find the managed resource associated with your claim by running:

   ```
   kubectl get classclaim CLASS-CLAIM-NAME -n CLASS-CLAIM-NAMESPACE -ojsonpath="{.
   status.provisionedResourceRef}"
   ```

   Where:

   - `CLASS-CLAIM-NAME` is the name of your claim.
   - `CLASS-CLAIM-NAMESPACE` is the namespace your claim is in.

   Example output for a MongoDB claim:

   ```
   {"apiVersion":"bitnami.database.tanzu.vmware.com/v1alpha1","kind":"XMongoDBInst
   ance","name":"mongodb-zfjr5"}
   ```

2. Find the newest composition revision for your resource type by running:

   ```
   kubectl get compositionrevisions
   ```

   Example output:

   ```
   NAME                                                          REVISION   XR-
   KIND               XR-APIVERSION                              AGE
   ...
   xmongodbinstances.bitnami.database.tanzu.vmware.com-734d138   4          XMo
   ngoDBInstance      bitnami.database.tanzu.vmware.com/v1alpha1 3h4m
   xmongodbinstances.bitnami.database.tanzu.vmware.com-889eaeb   1          XMo
   ngoDBInstance      bitnami.database.tanzu.vmware.com/v1alpha1 3h29m
   xmongodbinstances.bitnami.database.tanzu.vmware.com-d869e8c   2          XMo
   ngoDBInstance      bitnami.database.tanzu.vmware.com/v1alpha1 3h29m
   xmongodbinstances.bitnami.database.tanzu.vmware.com-f1f3fe9   3          XMo
   ngoDBInstance      bitnami.database.tanzu.vmware.com/v1alpha1 3h5m
   ...
   ```

   Record the name of the highest revision. In the above output, this is revision 4 (`xmongodbinstances.bitnami.database.tanzu.vmware.com-734d138`).

3. Open your managed resource for editing by running:

   ```
   kubectl edit RESOURCE-API RESOURCE-NAME
   ```

   Where:

   - `RESOURCE-API` is in the format `KIND.APIVERSION` using the `kind` and `apiVersion` from the output of the `kubectl get classclaim` command earlier. `APIVERSION` is the part of `apiVersion` before the `/`, for example, `bitnami.database.tanzu.vmware.com`.

- `RESOURCE-NAME` is the value of `name` from the output of the `kubectl get classclaim` command earlier.

For example:

```
$ kubectl edit xmongodbinstance.bitnami.database.tanzu.vmware.com mongodb-zfjr5
```

4. Change the resource `compositionRevisionRef` to point to the new composition revision. For example:

```
apiVersion: bitnami.database.tanzu.vmware.com/v1alpha1
kind: XMongoDBInstance
metadata:
  # ...
spec:
  compositionRef:
    name: xmongodbinstances.bitnami.database.tanzu.vmware.com
  compositionRevisionRef:
    name: xmongodbinstances.bitnami.database.tanzu.vmware.com-734d138
```

5. Save, and close your editor.
6. Verify that the resource is ready by running:

```
kubectl get RESOURCE-API RESOURCE-NAME
```

For example:

```
$ kubectl get xmongodbinstance.bitnami.database.tanzu.vmware.com mongodb-zfjr5

NAME            SYNCED   READY   COMPOSITION
AGE
mongodb-zfjr5   True     True    xmongodbinstance.bitnami.database.tanzu.vmwar
e.com   3h24m
```

Troubleshoot Bitnami Services

This topic explains how you troubleshoot issues related to Bitnami Services on Tanzu Application Platform (commonly known as TAP).

Private registry or VMware Tanzu Application Catalog configuration does not take effect

Symptom:

If you configure private registry integration for the Bitnami services after creating a claim for a Bitnami service using the default configuration, the updated private registry configuration does not appear to take effect.

Cause:

This is due to caching behavior in the system that is not accounted for during configuration updates.

Solution:

Delete the `provider-helm-*` pods in the `crossplane-system` namespace and wait for new pods to come back online after having applied the updated registry configuration.

Services aren't starting after upgrading to Tanzu Kubernetes releases v1.26

Symptom:

After upgrading to Tanzu Kubernetes releases v1.26, Bitnami services are not starting. When you inspect a `Statefulset` or `Replicaset` associated with the service, you see an error message about an issue with the Pod Security Standard for the cluster:

```
Error creating: pods "xxx" is forbidden: violates PodSecurity "restricted:latest": ...
```

Cause:

Tanzu Kubernetes releases v1.26 and later enforces a `restricted` Pod Security Standard. This prevents services created with Tanzu Application Platform v1.6 or earlier from starting.

Solution:

Follow the steps in Upgrading to Tanzu Kubernetes releases v1.26 or later.

Bitnami Services reference

This section provides reference documentation for Bitnami Services.

In this section:

- Dependencies
- Package values
- Version matrix

Dependencies for Bitnami Services

Bitnami Services is an integration package, which means that it provides configuration for other Tanzu Application Platform (commonly known as TAP) components. As such, it has a number of dependencies that must be met before you can use it.

The dependencies for Bitnami Services are:

- **Crossplane and the two providers** `provider-helm` **and** `provider-kubernetes`: See Install Crossplane
- **Services Toolkit**: See Install Services Toolkit

These dependencies are met if you install Tanzu Application Platform using the full, iterate, or run profiles.

Package values for Bitnami Services

This topic lists the keys and values that you can use to configure the behavior of the Bitnami Services package. You can apply configuration globally to all services using the `globals` key, or on a per-service basis using the `mysql`, `postgresql`, `rabbitmq`, `redis`, `mongodb`, and `kafka` keys.

If you are applying configuration to Tanzu Application Platform through the use of profiles and the `tap-values.yaml`, all configuration must exist under the `bitnami_services` top-level key.

For example:

```
bitnami_services:
  globals:
    helm_chart:
```

```
      # If you choose to use a custom Helm Chart repo, it's possible you'll also need
to configure specific versions
      # for each Chart as well, see example configuration below for postgresql.
      repo: https://charts.mycompany.example.com
  mysql:
    enabled: false
  postgresql:
    helm_chart:
      version: 12.2.6
    instance_class:
      name: company-redis
      description: My company postgres
  rabbitmq:
    instance_class:
      name: company-redis
      description: My company rabbit
  redis:
    instance_class:
      name: company-redis
      description: My company redis
```

Globals

The following table lists configuration that applies to all services.

KEY	DEFAULT	TYPE	DESCRIPTION
globals.create_clusterroles	`true`	boolean	Optional: Specifies whether to create default ClusterRoles that grant `claim` permissions to all Tanzu Application Platform Application operators.
globals.helm_chart.chart_pull_secret_ref.name	`""`	string	Name of the pull secret. Can be overridden by individual services.
globals.helm_chart.chart_pull_secret_ref.namespace	`""`	string	Namespace of the pull secret. Can be overridden by individual services.
globals.helm_chart.container_pull_secret_ref.name	`""`	string	Name of the secret. Can be overridden by individual services.
globals.helm_chart.container_pull_secret_ref.namespace	`""`	string	Namespace of the secret. Can be overridden by individual services.
globals.helm_chart.repo	`https://charts.bitnami.com/bitnami`	string	Optional: Repository hosting the Helm charts used to provision the instances of all services. Can be overridden by individual services.
globals.shared_namespace	`""`	string	Optional: Name of the namespace that is shared by all provisioned instances of all services. By default, each instance is provisioned in its own dedicated namespace. Cannot be set if `claim_namespace` is configured. Can be overridden by individual services.
globals.claim_namespace	`false`	boolean	Optional: Specifies whether to create the Service resources inside the same namespace as the claim. By default, each instance is provisioned in its own dedicated namespace. Cannot be set to `true` if `shared_namespace` is configured. Can be overridden by individual services.

MySQL

The following table lists configuration that applies to the `mysql` service.

KEY	DEFAULT	TYPE	DESCRIPTION
mysql.defaults.storage_size_gb	`1`	integer	Optional: The amount of storage to give each MySQL instance by default, in Gigabytes.
mysql.enabled	`true`	boolean	Optional: Provide developers an offering for unmanaged MySQL instances.
mysql.helm_chart.repo	`""`	string	Optional: Repository hosting the Helm chart used to provision MySQL instances.
mysql.helm_chart.version	`9.5.0`	string	Optional: Version of the Helm chart used to provision MySQL instances.
mysql.helm_chart.chart_pull_secret_ref.name	`""`	string	Name of the pull secret.
mysql.helm_chart.chart_pull_secret_ref.namespace	`""`	string	Namespace of the pull secret.
mysql.helm_chart.container_pull_secret_ref.name	`""`	string	Name of the secret. Can be overridden by individual services.
mysql.helm_chart.container_pull_secret_ref.namespace	`""`	string	Namespace of the secret. Can be overridden by individual services.
mysql.instance_class.description	`MySQL by Bitnami`	string	Optional: Description of the ClusterInstanceClass that developers use to provision and claim MySQL instances.
mysql.instance_class.name	`mysql-unmanaged`	string	Optional: Name of the ClusterInstanceClass that developers use to provision and claim MySQL instances.
mysql.shared_namespace	`""`	string	Optional: Name of the namespace that is shared by all provisioned MySQL instances. By default, each instance is provisioned in its own dedicated namespace. Cannot be set if `claim_namespace` is configured.
mysql.claim_namespace	`false`	boolean	Optional: Specifies whether to create the MySQL resources inside the same namespace as the claim. By default, each instance is provisioned in its own dedicated namespace. Cannot be set to `true` if `shared_namespace` is configured.

PostgreSQL

The following table lists configuration that applies to the `postgresql` service.

KEY	DEFAULT	TYPE	DESCRIPTION
postgresql.enabled	`true`	boolean	Optional: Provide developers an offering for unmanaged PostgreSQL instances.
postgresql.helm_chart.chart_pull_secret_ref.name	`""`	string	Name of the pull secret.
postgresql.helm_chart.chart_pull_secret_ref.namespace	`""`	string	Namespace of the pull secret.

KEY	DEFAULT	TYPE	DESCRIPTION
postgresql.helm_chart.container_pull_secret_ref.name	""	string	Name of the secret. Can be overridden by individual services.
postgresql.helm_chart.container_pull_secret_ref.namespace	""	string	Namespace of the secret. Can be overridden by individual services.
postgresql.helm_chart.repo	""	string	Optional: Repository hosting the Helm chart used to provision PostgreSQL instances.
postgresql.helm_chart.version	12.2.0	string	Optional: Version of the Helm chart used to provision PostgreSQL instances.
postgresql.instance_class.description	PostgreSQL by Bitnami	string	Optional: Description of the ClusterInstanceClass that developers use to provision and claim PostgreSQL instances.
postgresql.instance_class.name	postgresql-unmanaged	string	Optional: Name of the ClusterInstanceClass that developers use to provision and claim PostgreSQL instances.
postgresql.shared_namespace	""	string	Optional: Name of the namespace that is shared by all provisioned PostgreSQL instances. By default, each instance is provisioned in its own dedicated namespace.
postgresql.claim_namespace	false	boolean	Optional: Specifies whether to create the PostgreSQL resources inside the same namespace as the claim. By default, each instance is provisioned in its own dedicated namespace. Cannot be set to `true` if `shared_namespace` is configured.
postgresql.defaults.storage_size_gb	1	integer	Optional: The amount of storage to give each PostgreSQL instance by default, in Gigabytes.

RabbitMQ

The following table lists configuration that applies to the `rabbitmq` service.

KEY	DEFAULT	TYPE	DESCRIPTION
rabbitmq.enabled	true	boolean	Optional: Provide developers an offering for unmanaged RabbitMQ instances
rabbitmq.helm_chart.container_pull_secret_ref.name	""	string	Name of the secret. Can be overridden by individual services.
rabbitmq.helm_chart.container_pull_secret_ref.namespace	""	string	Namespace of the secret. Can be overridden by individual services.
rabbitmq.helm_chart.repo	""	string	Optional: Repository hosting the Helm chart used to provision RabbitMQ instances.
rabbitmq.helm_chart.version	11.10.0	string	Optional: Version of the Helm chart used to provision RabbitMQ instances.
rabbitmq.helm_chart.chart_pull_secret_ref.name	""	string	Name of the pull secret.
rabbitmq.helm_chart.chart_pull_secret_ref.namespace	""	string	Namespace of the pull secret.

KEY	DEFAULT	TYPE	DESCRIPTION
rabbitmq.instance_class.description	`RabbitMQ by Bitnami`	string	Optional: Description of the ClusterInstanceClass that developers use to provision and claim RabbitMQ instances.
rabbitmq.instance_class.name	`rabbitmq-unmanaged`	string	Optional: Name of the ClusterInstanceClass that developers use to provision and claim RabbitMQ instances.
rabbitmq.shared_namespace	`""`	string	Optional: Name of the namespace that is shared by all provisioned RabbitMQ instances. By default, each instance is provisioned in its own dedicated namespace.
rabbitmq.claim_namespace	`false`	boolean	Optional: Specifies whether to create the RabbitMQ resources inside the same namespace as the claim. By default, each instance is provisioned in its own dedicated namespace. Cannot be set to `true` if `shared_namespace` is configured.
rabbitmq.defaults.replica_count	`1`	integer	Optional: The number of replicas to create for each RabbitMQ instance by default.
rabbitmq.defaults.storage_size_gb	`1`	integer	Optional: The amount of storage to give each RabbitMQ instance by default, in Gigabytes.

Redis

The following table lists configuration that applies to the `redis` service.

KEY	DEFAULT	TYPE	DESCRIPTION
redis.instance_class.description	`Redis by Bitnami`	string	Optional: Description of the ClusterInstanceClass that is used by developers to provision and claim Redis instances.
redis.instance_class.name	`redis-unmanaged`	string	Optional: Name of the ClusterInstanceClass that is used by developers to provision and claim Redis instances.
redis.shared_namespace	`""`	string	Optional: Name of the namespace that is shared by all provisioned Redis instances. By default, each instance is provisioned in its own dedicated namespace.
redis.claim_namespace	`false`	boolean	Optional: Specifies whether to create the Redis resources inside the same namespace as the claim. By default, each instance is provisioned in its own dedicated namespace. Cannot be set to `true` if `shared_namespace` is configured.
redis.defaults.storage_size_gb	`1`	integer	Optional: The amount of storage to give each Redis instance by default, in Gigabytes.
redis.enabled	`true`	boolean	Optional: Provide developers an offering for unmanaged Redis instances.
redis.helm_chart.chart_pull_secret_ref.name	`""`	string	Name of the pull secret.
redis.helm_chart.chart_pull_secret_ref.namespace	`""`	string	Namespace of the pull secret.
redis.helm_chart.container_pull_secret_ref.name	`""`	string	Name of the secret. Can be overridden by individual services.
redis.helm_chart.container_pull_secret_ref.namespace	`""`	string	Namespace of the secret. Can be overridden by individual services.

KEY	DEFAULT	TYPE	DESCRIPTION
redis.helm_chart.repo	""	string	Optional: Repository hosting the Helm chart used to provision Redis instances.
redis.helm_chart.version	17.8.0	string	Optional: Version of the Helm chart used to provision Redis instances.

MongoDB

The following table lists configuration that applies to the `mongodb` service.

KEY	DEFAULT	TYPE	DESCRIPTION
mongodb.instance_class.description	MongoDB by Bitnami	string	Optional: Description of the ClusterInstanceClass that is used by developers to provision and claim MongoDB instances.
mongodb.instance_class.name	mongodb-unmanaged	string	Optional: Name of the ClusterInstanceClass that is used by developers to provision and claim MongoDB instances.
mongodb.shared_namespace	""	string	Optional: Name of the namespace that is shared by all provisioned MongoDB instances. By default, each instance is provisioned in its own dedicated namespace. Cannot be set if `claim_namespace` is configured.
mongodb.claim_namespace	false	boolean	Optional: Specifies whether to create the MongoDB resources inside the same namespace as the claim. By default, each instance is provisioned in its own dedicated namespace. Cannot be set to `true` if `shared_namespace` is configured.
mongodb.defaults.storage_size_gb	1	integer	Optional: The amount of storage in Gigabytes to give each MongoDB instance by default.
mongodb.enabled	true	boolean	Optional: Provide developers an offering for unmanaged MongoDB instances.
mongodb.helm_chart.chart_pull_secret_ref.name	""	string	Name of the pull secret.
mongodb.helm_chart.chart_pull_secret_ref.namespace	""	string	Namespace of the pull secret.
mongodb.helm_chart.container_pull_secret_ref.name	""	string	Name of the secret. Can be overridden by individual services.
mongodb.helm_chart.container_pull_secret_ref.namespace	""	string	Namespace of the secret. Can be overridden by individual services.
mongodb.helm_chart.repo	""	string	Optional: Repository hosting the Helm chart used to provision MongoDB instances.
mongodb.helm_chart.version	13.13.1	string	Optional: Version of the Helm chart used to provision MongoDB instances.

Kafka

The following table lists configuration that applies to the `kafka` service.

KEY	DEFAULT	TYPE	DESCRIPTION
kafka.instance_class.description	`Kafka by Bitnami`	string	Optional: Description of the ClusterInstanceClass that is used by developers to provision and claim Kafka instances.
kafka.instance_class.name	`kafka-unmanaged`	string	Optional: Name of the ClusterInstanceClass that is used by developers to provision and claim Kafka instances.
kafka.shared_namespace	`""`	string	Optional: Name of the namespace that is shared by all provisioned Kafka instances. By default, each instance is provisioned in its own dedicated namespace. Cannot be set if `claim_namespace` is configured.
kafka.claim_namespace	`false`	boolean	Optional: Specifies whether to create the Kafka resources inside the same namespace as the claim. By default, each instance is provisioned in its own dedicated namespace. Cannot be set to `true` if `shared_namespace` is configured.
kafka.defaults.storage_size_gb	`1`	integer	Optional: The amount of storage in Gigabytes to give each Kafka instance by default.
kafka.enabled	`true`	boolean	Optional: Provide developers an offering for unmanaged Kafka instances.
kafka.helm_chart.chart_pull_secret_ref.name	`""`	string	Name of the pull secret.
kafka.helm_chart.chart_pull_secret_ref.namespace	`""`	string	Namespace of the pull secret.
kafka.helm_chart.container_pull_secret_ref.name	`""`	string	Name of the secret. Can be overridden by individual services.
kafka.helm_chart.container_pull_secret_ref.namespace	`""`	string	Namespace of the secret. Can be overridden by individual services.
kafka.helm_chart.repo	`""`	string	Optional: Repository hosting the Helm chart used to provision Kafka instances.
kafka.helm_chart.version	`22.0.0`	string	Optional: Version of the Helm chart used to provision Kafka instances.

Version matrix for Bitnami Services

This topic provides you with a version matrix for the Bitnami Services package and its open source components in Tanzu Application Platform v1.9 (commonly known as TAP).

To view this information for another Tanzu Application Platform version, select the version from the drop-down menu at the top of this page.

The following table has the component versions for the Bitnami Services package.

Component	Version
Bitnami Services package	0.5.0
MySQL Chart	9.5.0
PostgreSQL Chart	12.2.0
RabbitMQ Chart	11.10.0
Redis Chart	17.8.0

Component	Version
MongoDB Chart	13.13.1
Kafka Chart	22.0.0

Note

Tanzu Application Platform patch releases are only added to the table when there is a change to one or more of the other versions in the table. Otherwise, the corresponding versions remain the same for each Tanzu Application Platform patch release.

Overview of Cartographer Conventions

This topic describes an overview of Cartographer Conventions and how you can use it with Tanzu Application Platform.

Overview

Cartographer Conventions provides a means for operators to express their knowledge about how applications can run on Kubernetes as a convention. Cartographer Conventions supports defining and applying conventions to pods. It applies these opinions to fleets of developer workloads as they are deployed to the platform, saving operator and developer time.

The service is composed of two components

- **convention controller:** The convention service's convention controller provides the metadata to the convention server and executes the updates to a PodTemplateSpec in accordance with convention server's requests.
- **convention server:** The convention server receives and evaluates metadata associated with a workload and requests updates to the PodTemplateSpec associated with that workload. You can have one or more convention servers for a single controller instance.

About applying conventions

The convention server uses criteria defined in the convention to discover whether the configuration of a workload must change. The server receives the OCI metadata from the convention controller. If the metadata meets the criteria defined by the convention server, the conventions are applied. A convention can apply to all workloads regardless of metadata.

Applying conventions by using image metadata

You can define conventions to target workloads by using properties of their OCI metadata.

Conventions can use this information to only apply changes to the configuration of workloads when they match specific criteria. Such as, Spring Boot or .Net apps, or Spring Boot v2.3+. Targeted conventions can ensure that uniformity across specific workload types deployed on the cluster.

You can use all the metadata details of an image when evaluating workloads. To see the metadata details, use the Docker CLI command:

```
docker image inspect IMAGE`.
```

Note

Depending on how the image was built, metadata might not be available to reliably identify the image type and match the criteria for a convention server. Images built with Cloud Native Buildpacks reliably include rich descriptive metadata. Images built by some other process might not include the same metadata.

Applying conventions without using image metadata

Conventions can apply to workloads without targeting build service metadata. Examples of possible uses of this type of convention include:

- appending a logging or metrics sidecar,
- adding environment variables, or
- adding cached volumes.

These kinds of conventions ensure that infrastructure uniformity exists across workloads deployed on the cluster while reducing developer toil.

Important

Adding a sidecar alone does not make the log or metrics collection work. This requires having collector agents deployed and accessible from the Kubernetes cluster, and configuring required access by using role-based access control (RBAC) policy.

Overview of Cartographer Conventions

This topic describes an overview of Cartographer Conventions and how you can use it with Tanzu Application Platform.

Overview

Cartographer Conventions provides a means for operators to express their knowledge about how applications can run on Kubernetes as a convention. Cartographer Conventions supports defining and applying conventions to pods. It applies these opinions to fleets of developer workloads as they are deployed to the platform, saving operator and developer time.

The service is composed of two components

- **convention controller:** The convention service's convention controller provides the metadata to the convention server and executes the updates to a PodTemplateSpec in accordance with convention server's requests.
- **convention server:** The convention server receives and evaluates metadata associated with a workload and requests updates to the PodTemplateSpec associated with that workload. You can have one or more convention servers for a single controller instance.

About applying conventions

The convention server uses criteria defined in the convention to discover whether the configuration of a workload must change. The server receives the OCI metadata from the convention controller. If the metadata meets the criteria defined by the convention server, the conventions are applied. A convention can apply to all workloads regardless of metadata.

Applying conventions by using image metadata

You can define conventions to target workloads by using properties of their OCI metadata.

Conventions can use this information to only apply changes to the configuration of workloads when they match specific criteria. Such as, Spring Boot or .Net apps, or Spring Boot v2.3+. Targeted conventions can ensure that uniformity across specific workload types deployed on the cluster.

You can use all the metadata details of an image when evaluating workloads. To see the metadata details, use the Docker CLI command:

```
docker image inspect IMAGE`.
```

Note

Depending on how the image was built, metadata might not be available to reliably identify the image type and match the criteria for a convention server. Images built with Cloud Native Buildpacks reliably include rich descriptive metadata. Images built by some other process might not include the same metadata.

Applying conventions without using image metadata

Conventions can apply to workloads without targeting build service metadata. Examples of possible uses of this type of convention include:

- appending a logging or metrics sidecar,
- adding environment variables, or
- adding cached volumes.

These kinds of conventions ensure that infrastructure uniformity exists across workloads deployed on the cluster while reducing developer toil.

Important

Adding a sidecar alone does not make the log or metrics collection work. This requires having collector agents deployed and accessible from the Kubernetes cluster, and configuring required access by using role-based access control (RBAC) policy.

Install Cartographer Conventions

This topic tells you how to install Cartographer Conventions from the Tanzu Application Platform (commonly known as TAP) package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Source Controller. For more information about profiles, see Components and installation profiles.

Prerequisites

Before installing Source Controller:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- Install cert-manager on the cluster. For more information, see Install cert-manager.

Install

To install Source Controller:

1. List version information for the package by running:

```
tanzu package available list cartographer.conventions.apps.tanzu.vmware.com --n
amespace tap-install
```

   For example:

```
$ tanzu package available list cartographer.conventions.apps.tanzu.vmware.com -
-namespace tap-install
- Retrieving package versions for cartographer.conventions.apps.tanzu.vmware.co
m...
  NAME                                            VERSION        RELEASED-AT
  cartographer.conventions.apps.tanzu.vmware.com  0.8.0-build.2  2024-02-27 10:
22:43 -0500 EST
```

2. (Optional) Gather the values schema:

```
tanzu package available get cartographer.conventions.apps.tanzu.vmware.com/VERS
ION-NUMBER --values-schema --namespace tap-install
```

   Where `VERSION-NUMBER` is the version of the package listed in step 1 above.

   For example:

```
tanzu package available get cartographer.conventions.apps.tanzu.vmware.com/0.8.
0-build.2 --values-schema --namespace tap-install

  KEY               DEFAULT  TYPE    DESCRIPTION
  aws_iam_role_arn  ""       string  Optional: Arn role that has access to pull
images from ECR container registry
  ca_cert_data      ""       string  Optional: PEM Encoded certificate data for
image registries with private CA.
  resources                          Optional: Cartographer Conventions control
ler resource limit configuration
```

3. (Optional) Create a file named `cartographer-conventions-values.yaml` to override the default installation settings. You can configure the following fields:

   - `ca_cert_data`: Enables Cartographer Conventions controller to connect to image registries that use self-signed or private certificate authorities. If a certificate error `x509: certificate signed by unknown authority` occurs, use this option to trust additional certificate authorities.

     To provide a custom certificate, add the PEM-encoded CA certificate data to `source-controller-values.yaml`. For example:

```
ca_cert_data: |
    -----BEGIN CERTIFICATE-----
    MIICpTCCAYUCBgkqhkiG9w0BBQ0wMzAbBgkqhkiG9w0BBQwwDgQIYg9x6gkCAggA
    ...
    9TlA7A4FFpQqbhAuAVH6KQ8WMZIrVxJSQ03c9lKVkI62wQ==
    -----END CERTIFICATE-----
```

- `aws_iam_role_arn`: Annotates the Cartographer Conventions controller service with an AWS Identity and Access Management (IAM) role. This allows Cartographer Conventions controllerto pull images from Amazon Elastic Container Registry (ECR).

  To add the AWS IAM role Amazon Resource Name (ARN) to the Cartographer Conventions controller service, add the ARN to `cartographer-conventions-values.yaml`. For example:

  ```
  aws_iam_role_arn: "eks.amazonaws.com/role-arn: arn:aws:iam::112233445566:
  role/cartographer-conventions-controller-manager"
  ```

- `resources`: Allows updating the default resource configuration for the Cartographer Conventions controller. By default Cartographer Convention controller resource configuration is set as following:

  ```
  resources:
   limits:
     cpu: 100m
     memory: 512Mi
   requests:
     cpu: 100m
     memory: 20Mi
  ```

  To update the controller's default resource configuration, add the desired configuration to `cartographer-conventions-values.yaml`. For example:

  ```
  resources:
    limits:
       cpu: 100m
       memory: 1Gi
  ```

4. Install the package by running:

   ```
   tanzu package install cartographer-conventions \
     --package cartographer.conventions.apps.tanzu.vmware.com \
     --version VERSION-NUMBER \
     --namespace tap-install \
     --values-file VALUES-FILE
   ```

   Where:

   - `VERSION-NUMBER` is the version of the package listed in step 1 above.
   - `VALUES-FILE` is the path to the file created in step 3.

   For example:

   ```
   $ tanzu package install cartographer-conventions
       --package cartographer.conventions.apps.tanzu.vmware.com
       --version 0.8.0-build.2
       --namespace tap-install
       --values-file cartogrpaher-conventions-values.yaml

   4:16:45PM: Creating service account 'cartographer-conventions-tap-install-sa'
   4:16:45PM: Creating cluster admin role 'cartographer-conventions-tap-install-cl
   uster-role'
   4:16:45PM: Creating cluster role binding 'cartographer-conventions-tap-install-
   cluster-rolebinding'
   4:16:45PM: Creating secret 'cartographer-conventions-tap-install-values'
   4:16:45PM: Creating overlay secrets
   4:16:45PM: Creating package install resource
   4:16:45PM: Waiting for PackageInstall reconciliation for 'cartographer-conventi
   ons'
   ```

```
4:16:45PM: Fetch started (3s ago)
4:16:48PM: Fetching
        | apiVersion: vendir.k14s.io/v1alpha1
        | directories:
        | - contents:
        |   - imgpkgBundle:
        |       image: registry.tanzu.vmware.com/tanzu-application-platform/con
stellation/cartographer.conventions.apps.tanzu.vmware.com@sha256:c4bc900b883537
f168f64ac6b725751752bcbe5cb522cd75abde96a1aa9cbfd5
        |     path: .
        |   path: "0"
        | kind: LockConfig
        |
4:16:48PM: Fetch succeeded
4:16:48PM: Template succeeded
4:16:48PM: Deploy started (2s ago)
4:16:50PM: Deploying
        | Target cluster 'https://10.96.0.1:443' (nodes: tap-local-control-plan
e)
        | Changes
        | Namespace           Name
Kind                          Age  Op      Op st.  Wait to    Rs  Ri
        | (cluster)           cartographer-conventions-manager-role
ClusterRole                     -    create  -       reconcile  -   -
        | ^                   cartographer-conventions-manager-rolebinding
ClusterRoleBinding              -    create  -       reconcile  -   -
        | ^                   cartographer-conventions-mutating-webhook-configu
ration      MutatingWebhookConfiguration    -    create  -       reconcile  -
-
        | ^                   cartographer-conventions-proxy-role
ClusterRole                     -    create  -       reconcile  -   -
        | ^                   cartographer-conventions-proxy-rolebinding
ClusterRoleBinding              -    create  -       reconcile  -   -
        | ^                   cartographer-conventions-validating-webhook-confi
guration    ValidatingWebhookConfiguration  -    create  -       reconcile  -
-
        | ^                   clusterpodconventions.conventions.carto.run
CustomResourceDefinition        -    create  -       reconcile  -   -
        | ^                   conventions-system
Namespace                       -    create  -       reconcile  -   -
        | ^                   podintents.conventions.carto.run
CustomResourceDefinition        -    create  -       reconcile  -   -
        | conventions-system  cartographer-conventions-ca-certificates
Secret                          -    create  -       reconcile  -   -
        | ^                   cartographer-conventions-controller-manager
Deployment                      -    create  -       reconcile  -   -
        | ^                   cartographer-conventions-controller-manager
ServiceAccount                  -    create  -       reconcile  -   -
        | ^                   cartographer-conventions-controller-manager-metri
cs-service  Service                         -    create  -       reconcile  -
-
        | ^                   cartographer-conventions-leader-election-role
Role                            -    create  -       reconcile  -   -
        | ^                   cartographer-conventions-leader-election-rolebind
ing         RoleBinding                     -    create  -       reconcile  -
-
        | ^                   cartographer-conventions-reg-creds
Secret                          -    create  -       reconcile  -   -
        | ^                   cartographer-conventions-selfsigned-issuer
Issuer                          -    create  -       reconcile  -   -
        | ^                   cartographer-conventions-serving-cert
Certificate                     -    create  -       reconcile  -   -
        | ^                   cartographer-conventions-webhook-service
Service                         -    create  -       reconcile  -   -
        | Op:      19 create, 0 delete, 0 update, 0 noop, 0 exists
        | Wait to: 19 reconcile, 0 delete, 0 noop
```

```
        | 8:16:48PM: ---- applying 7 changes [0/19 done] ----
        | ...
        | 8:17:07PM: ok: reconcile deployment/cartographer-conventions-controll
er-manager (apps/v1) namespace: conventions-system
        | 8:17:07PM: ---- applying complete [19/19 done] ----
        | 8:17:07PM: ---- waiting complete [19/19 done] ----
        | Succeeded
4:17:07PM: Deploy succeeded
```

5. Verify the package installation by running:

   ```
   tanzu package installed get cartographer-conventions -n tap-install
   ```

   For example:

   ```
   tanzu package installed get cartographer-conventions -n tap-install

   NAMESPACE:          tap-install
   NAME:               cartographer-conventions
   PACKAGE-NAME:       cartographer.conventions.apps.tanzu.vmware.com
   PACKAGE-VERSION:    0.8.0-build.2
   STATUS:             Reconcile succeeded
   CONDITIONS:         - status: "True"
     type: ReconcileSucceeded
   ```

   Verify that `STATUS` is `Reconcile succeeded`:

   ```
   kubectl get pods -n conventions-system
   ```

   For example:

   ```
   $ kubectl get pods -n source-system
   NAME                                                        READY   STATUS
   RESTARTS   AGE
   cartographer-conventions-controller-manager-6cc74788-2z2v9   1/1     Running
   0          5m6s
   ```

   Verify that `STATUS` is `Running`.

Create conventions with Cartographer Conventions

This topic describes how you can create and deploy custom conventions to the Tanzu Application Platform by using Cartographer Conventions.

Introduction

Tanzu Application Platform helps developers transform their code into containerized workloads with a URL. The Supply Chain Choreographer for Tanzu manages this transformation. For more information, see Supply Chain Choreographer.

Cartographer Conventions is a key component of the supply chain compositions the choreographer calls into action. Cartographer Conventions enables people in operational roles to efficiently apply their expertise. They can specify the runtime best practices, policies, and conventions of their organization to workloads as they are created on the platform. The power of this component becomes evident when the conventions of an organization are applied consistently, at scale, and without hindering the velocity of application developers.

Opinions and policies vary from organization to organization. Cartographer Convention supports the creation of custom conventions to meet the unique operational needs and requirements of an organization.

Before jumping into the details of creating a custom convention, you can view two distinct components of Cartographer Conventions:

- Convention controller
- Convention server

Convention server

The convention server is the component that applies a convention already defined on the server. For a golang example of creating a convention server to add Spring Boot conventions, see spring-convention-server in GitHub. The resource that structures the request body of the request and response from the server is the PodConventionContext.

The `PodConventionContext` is a webhooks.conventions.carto.run/v1alpha1 type that defines the structure used to communicate internally by the webhook convention server. It ***does not exist*** on the Kubernetes API Server.

`PodConventionContext` is a wrapper for two types:

- `PodConventionContextSpec` which acts as a wrapper for a `PodTemplateSpec` and a list of `ImageConfigs` provided in the request body of the server.
- `PodConventionContextStatus` which is a status type used to represent the current status of the context retrieved by the request.

For information about an example `PodConventionContext`, see PodConventionContext in GitHub. For information about a Convention server and the structure of these types, see OpenAPI Spec in GitHub.

How the convention server works

Each convention server can host one or more conventions. The application of each convention by a convention server are controlled conditionally. The conditional criteria governing the application of a convention is customizable and are based on the evaluation of a custom Kubernetes resource called PodIntent. PodIntent is the vehicle by which Cartographer Conventions as a whole delivers its value.

A PodIntent is created, or updated if already existing, when a workload is run by using a Tanzu Application Platform supply chain. The custom resource includes both the PodTemplateSpec and the OCI image metadata associated with a workload. See the Kubernetes documentation. The conditional criteria for a convention are based on any property or value found in the PodTemplateSpec or the Open Containers Initiative (OCI) image metadata available in the PodIntent.

If a convention's criteria are met, the convention server enriches the PodTemplateSpec in the PodIntent. The convention server also updates the `status` section of the PodIntent with the name of the convention that's applied. You can figure out after the fact which conventions were applied to the workload.

To provide flexibility in how conventions are organized, you can deploy multiple convention servers. Each server can contain a convention or set of conventions focused on a specific class of runtime modifications, on a specific language framework, and so on. How the conventions are organized, grouped, and deployed is up to you and the needs of your organization.

Convention servers deployed to the cluster does not take action unless triggered to do so by the second component of Cartographer Conventions, the Convention service's controller.

Convention controller

The convention controller is the orchestrator of one or many convention servers deployed to the cluster. There are resources available on the `conventions.carto.run/v1aplha1` API that allow the controller to carry out its functions. These resources include:

- ClusterPodConvention `ClusterPodConvention` is a resource type that allows the conventions author to register a webhook server with the controller using its `spec.webhook` field.

  ```
  ...
  spec:
    selectorTarget: PodTemplateSpec # optional field with options, defaults to Po
  dTemplateSpec
    selectors: # optional, defaults to match all workloads
    - <metav1.LabelSelector>
    webhook:
      certificate:
        name: sample-cert
        namespace: sample-conventions
      clientConfig:
        <admissionregistrationv1.WebhookClientConfig>
  ```

- PodIntent

  `PodIntent` is a `conventions.carto.run/v1alpha1` resource type that is continuously reconciled and applies decorations to a workload `PodTemplateSpec` exposing the enriched `PodTemplateSpec` on its status. Whenever the status of the `PodIntent` is updated, no side effects are caused on the cluster.

As key types defined on the `conventions.carto.run` API, the ClusterPodConvention`and`PodIntent`resources are both present on the Kubernetes API Server and are queried using`clusterpodconventions.conventions.carto.run`for the former and`podintents.conventions.carto.run` for the later.

How the convention services's controller works

When the Supply Chain Choreographer creates or updates a PodIntent for a workload, the convention controller retrieves the OCI image metadata from the repository containing the workload's images and sets it in the PodIntent.

The convention controller then uses a webhook architecture to pass the PodIntent to each convention server deployed to the cluster. The controller orchestrates the processing of the PodIntent by the convention servers sequentially, based on the `priority` value that's set on the convention server. For more information, see ClusterPodConvention.

After all convention servers are finished processing a PodIntent for a workload, the convention controller updates the PodIntent with the latest version of the PodTemplateSpec and sets `PodIntent.status.conditions[].status=True` where `PodIntent.status.conditions[].type=Ready`. This status change signals the Supply Chain Choreographer that Cartographer Conventions is finished with its work. The status change also executes whatever steps are waiting in the supply chain.

Getting started

With this high-level understanding of Cartographer Conventions components, you can create and deploy a custom convention.

Note

This topic covers developing conventions using GOLANG, but this is done using other languages by following the specifications.

Prerequisites

The following prerequisites must be met before a convention is developed and deployed:

- The Kubernetes CLI tool (kubectl) CLI is installed. For more information, see the Kubernetes documentation.
- Tanzu Application Platform prerequisites are installed. For more information, see Prerequisites
- Tanzu Application Platform components are installed. For more information, see the Installing the Tanzu CLI.
- The default supply chain is installed. Download Supply Chain Security Tools for VMware Tanzu from Tanzu Network.
- Your kubeconfig context is set to the Tanzu Application Platform-enabled cluster:

  ```
  kubectl config use-context CONTEXT_NAME
  ```

- You use GitHub to install the ko CLI. See the google/ko GitHub repository. These instructions use `ko` to build an image. If there is an existing image or build process, `ko` is optional.)

Define convention criteria

The `server.go` file contains the configuration for the server and the logic the server applies when a workload matches the defined criteria. For example, adding a Prometheus sidecar to web applications, or adding a `workload-type=spring-boot` label to any workload that has metadata, indicating it is a Spring Boot app.

Important

For this example, the package `model` defines resource types.

1. The example `server.go` configures the `ConventionHandler` to ingest the webhook requests from the convention controller. See PodConventionContext. Here the handler must only deal with the existing PodTemplateSpec and ImageConfig.

   ```
   ...
   import (
     corev1 "k8s.io/api/core/v1"
   )
   ...
   func ConventionHandler(template *corev1.PodTemplateSpec, images []model.ImageCo
   nfig) ([]string, error) {
      // Create custom conventions
   }
   ...
   ```

   Where:

   - `template` is the predefined `PodTemplateSpec` that the convention edits. For more information about `PodTemplateSpec`, see the Kubernetes documentation.

- `images` are the ImageConfig used as reference to make decisions in the conventions. In this example, the type was created within the `model` package.

2. The example `server.go` also configures the convention server to listen for requests:

```
...
import (
    "context"
    "fmt"
    "log"
    "net/http"
    "os"
    ...
)
...
func main() {
    ctx := context.Background()
    port := os.Getenv("PORT")
    if port == "" {
        port = "9000"
    }
    http.HandleFunc("/", webhook.ServerHandler(convention.ConventionHandler))
    log.Fatal(webhook.NewConventionServer(ctx, fmt.Sprintf(":%s", port)))
}
...
```

Where:

- `PORT` is a possible environment variable, for this example, defined in the Deployment.
- `ServerHandler` is the *handler* function called when any request comes to the server.
- `NewConventionServer` is the function in charge of configuring and creating the *http webhook* server.
- `port` is the calculated port of the server to listen for requests. It must match the Deployment if the `PORT` variable is not defined in it.
- The `path` or pattern (default to `/`) is the convention server's default path. If it is changed, it must be changed in the ClusterPodConvention.

Note

The *Server Handler*, `func ConventionHandler(...)`, and the configure or start web server, `func NewConventionServer(...)`, is defined in the convention controller in the `webhook` package, but you can use a custom one.

3. Creating the *Server Handler*, which handles the request from the convention controller with the PodConventionContext serialized to JSON.

```
package webhook
...
func ServerHandler(conventionHandler func(template *corev1.PodTemplateSpec, ima
ges []model.ImageConfig) ([]string, error)) http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        ...
        // Check request method
        ...
        // Decode the PodConventionContext
        podConventionContext := &model.PodConventionContext{}
```

```
        err = json.Unmarshal(body, &podConventionContext)
        if err != nil {
            w.WriteHeader(http.StatusBadRequest)
            return
        }
        // Validate the PodTemplateSpec and ImageConfig
        ...
        // Apply the conventions
        pts := podConventionContext.Spec.Template.DeepCopy()
        appliedConventions, err := conventionHandler(pts, podConventionContext.
Spec.Images)
        if err != nil {
            w.WriteHeader(http.StatusInternalServerError)
            return
        }
        // Update the applied conventions and status with the new PodTemplateSp
ec
        podConventionContext.Status.AppliedConventions = appliedConventions
        podConventionContext.Status.Template = *pts
        // Return the updated PodConventionContext
        w.Header().Set("Content-Type", "application/json")
        w.WriteHeader(http.StatusOK)
        json.NewEncoder(w).Encode(podConventionContext)
    }
}
...
```

4. Configure and start the web server by defining the `NewConventionServer` function, which starts the server with the defined port and current context. The server uses the `.crt` and `.key` files to handle *TLS* traffic.

```
package webhook
...
// Watch handles the security by certificates.
type certWatcher struct {
    CrtFile string
    KeyFile string

    m       sync.Mutex
    keyPair *tls.Certificate
}
func (w *certWatcher) Load() error {
    // Creates a X509KeyPair from PEM encoded client certificate and private ke
y.
    ...
}
func (w *certWatcher) GetCertificate() *tls.Certificate {
    w.m.Lock()
    defer w.m.Unlock()

    return w.keyPair
}
...
func NewConventionServer(ctx context.Context, addr string) error {
    // Define a health check endpoint to readiness and liveness probes.
    http.HandleFunc("/healthz", func(w http.ResponseWriter, r *http.Request) {
        w.WriteHeader(http.StatusOK)
    })

    if err := watcher.Load(); err != nil {
        return err
    }
    // Defines the server with the TLS configuration.
    server := &http.Server{
        Addr: addr,
```

```
        TLSConfig: &tls.Config{
            GetCertificate: func(_ *tls.ClientHelloInfo) (*tls.Certificate, err
or) {
                cert := watcher.GetCertificate()
                return cert, nil
            },
            PreferServerCipherSuites: true,
            MinVersion:               tls.VersionTLS13,
        },
        BaseContext: func(_ net.Listener) context.Context {
            return ctx
        },
    }
    go func() {
        <-ctx.Done()
        server.Close()
    }()

    return server.ListenAndServeTLS("", "")
}
```

Define the convention behavior

Any property or value within the PodTemplateSpec or OCI image metadata associated with a workload defines the criteria for applying conventions. See PodTemplateSpec in the Kubernetes documentation. The following are a few examples.

Matching criteria by labels or annotations

The `conventions.carto.run/v1alpha1` API allows convention authors to use the `selectorTarget` field which complements the `ClusterPodConvention` matchers to specify whether to consider labels on either one of the following available options:

- PodTemplateSpec

  ```
  ...
  template:
    metadata:
      labels:
        awesome-label: awesome-value
      annotations:
        awesome-annotation: awesome-value
  ...
  ```

- PodIntent

  ```
    ...
    kind: PodIntent
    metadata:
      name: test-pod
      labels:
        environment: production
        ...
  ```

The `selectorTarget` field is configured on the ClusterPodConvention as follows:

```
...
spec:
  selectorTarget: PodIntent # optional, defaults to PodTemplateSpec
  selectors: # optional, defaults to match all workloads
  - <metav1.LabelSelector>
  webhook:
```

```
  certificate:
    name: sample-cert
    namespace: sample-conventions
  clientConfig:
    <admissionregistrationv1.WebhookClientConfig>
```

If you do not provide a value for this optional field while using the `conventions.carto.run/v1alpha1` API, the default value is set to `PodTemplateSpec` without the conventions author explicitly doing so.

Matching criteria by environment variables

When using environment variables to define whether the convention is applicable, it must be present in the PodTemplateSpec, spec, containers, and env to validate the value.

- PodTemplateSpec

  ```
  ...
  template:
    spec:
      containers:
        - name: awesome-container
          env:
  ...
  ```

- Handler

  ```
  package convention
  ...
  func conventionHandler(template *corev1.PodTemplateSpec, images []model.ImageCo
  nfig) ([]string, error) {
      if len(template.Spec.Containers[0].Env) == 0 {
          template.Spec.Containers[0].Env = append(template.Spec.Containers[0].En
  v, corev1.EnvVar{
              Name: "MY_AWESOME_VAR",
              Value: "MY_AWESOME_VALUE",
          })
          return []string{"awesome-envs-convention"}, nil
      }
      return []string{}, nil
      ...
  }
  ```

Matching criteria by image metadata

For each image contained within the PodTemplateSpec, the convention controller fetches the OCI image metadata and known bill of materials (BOMs), providing it to the convention server as ImageConfig. This metadata is introspected to make decisions about how to configure the PodTemplateSpec.

Configure and install the convention server

The `server.yaml` defines the Kubernetes components that enable the convention server in the cluster. The next definitions are within the file.

1. A `namespace` is created for the convention server components and has the required objects to run the server. It's used in the ClusterPodConvention section to indicate to the controller where the server is.

   ```
   ...
   ---
   apiVersion: v1
   ```

```
kind: Namespace
metadata:
  name: awesome-convention
---
...
```

2. (Optional) A certificate manager `Issuer` is created to issue the> certificate needed for TLS communication.

```
...
---
# The following manifests contain a self-signed issuer CR and a certificate CR.
# More document can be found at https://docs.cert-manager.io
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: awesome-selfsigned-issuer
  namespace: awesome-convention
spec:
  selfSigned: {}
---
...
```

3. (Optional) A self-signed `Certificate` is created.

```
...
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: awesome-webhook-cert
  namespace: awesome-convention
spec:
  subject:
    organizations:
    - vmware
    organizationalUnits:
    - tanzu
  commonName: awesome-webhook.awesome-convention.svc
  dnsNames:
  - awesome-webhook.awesome-convention.svc
  - awesome-webhook.awesome-convention.svc.cluster.local
  issuerRef:
    kind: Issuer
    name: awesome-selfsigned-issuer
  secretName: awesome-webhook-cert
  revisionHistoryLimit: 10
---
...
```

4. A Kubernetes `Deployment` is created to run the webhook from. The Service uses the container port defined by the `Deployment` to expose the server.

```
...
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: awesome-webhook
  namespace: awesome-convention
spec:
  replicas: 1
  selector:
    matchLabels:
```

```
    app: awesome-webhook
  template:
    metadata:
      labels:
        app: awesome-webhook
    spec:
      containers:
      - name: webhook
        # Set the prebuilt image of the convention or use ko to build an image
from code.
        # see https://github.com/google/ko
        image: ko://awesome-repo/awesome-user/awesome-convention
      env:
      - name: PORT
        value: "8443"
      ports:
      - containerPort: 8443
        name: webhook
      livenessProbe:
        httpGet:
          scheme: HTTPS
          port: webhook
          path: /healthz
      readinessProbe:
        httpGet:
          scheme: HTTPS
          port: webhook
          path: /healthz
      volumeMounts:
      - name: certs
        mountPath: /config/certs
        readOnly: true
    volumes:
    - name: certs
      secret:
        defaultMode: 420
        secretName: awesome-webhook-cert
---
...
```

5. A Kubernetes `Service` to expose the convention deployment is created. For this example, the exposed port is the default `443`. If you change the port, the ClusterPodConvention must be updated.

```
...
---
apiVersion: v1
kind: Service
metadata:
  name: awesome-webhook
  namespace: awesome-convention
  labels:
    app: awesome-webhook
spec:
  selector:
    app: awesome-webhook
  ports:
    - protocol: TCP
      port: 443
      targetPort: webhook
---
...
```

6. The ClusterPodConvention adds the convention to the cluster to make it available for the convention controller:

Important

The `annotations` block is only needed if you use a self-signed certificate. See the cert-manager documentation.

```
...
---
apiVersion: conventions.carto.run/v1alpha1
kind: ClusterPodConvention
metadata:
  name: awesome-convention
  annotations:
    conventions.carto.run/inject-ca-from: "awesome-convention/awesome-webhook-c
ert"
spec:
  webhook:
    clientConfig:
      service:
        name: awesome-webhook
        namespace: awesome-convention
        # path: "/" # default
        # port: 443 # default
```

Deploy a convention server

To deploy a convention server:

1. Build and install the convention.
    - To build and deploy the convention, use the ko tool on GitHub. It compiles your Go code into a Docker image and pushes it to the registry `KO_DOCKER_REGISTRY`.

      ```
      ko apply -f dist/server.yaml
      ```

    - If a different tool builds the image, the configuration is also applied by using either kubectl or `kapp`, setting the correct image in the Deployment descriptor.

      kubectl

      ```
      kubectl apply -f server.yaml
      ```

      kapp

      ```
      kapp deploy -y -a awesome-convention -f server.yaml
      ```

2. Verify the convention server. To verify the status of the convention server, confirm the running convention pods:
    - If the server is running, `kubectl get all -n awesome-convention` returns output such as:

      ```
      NAME                                    READY   STATUS    RESTARTS   A
      GE
      pod/awesome-webhook-1234567890-12345    1/1     Running   0          8
      h

      NAME                          TYPE        CLUSTER-IP    EXTERNAL-IP   POR
      T(S)   AGE
      service/awesome-webhook       ClusterIP   10.56.12.49   <none>        44
      3/TCP   28h
      ```

```
NAME                                    READY   UP-TO-DATE   AVAILABLE   AG
E
deployment.apps/awesome-webhook         1/1     1            1           28
h

NAME                                              DESIRED   CURRENT   READ
Y   AGE
replicaset.apps/awesome-webhook-1234563213        0         0         0
23h
replicaset.apps/awesome-webhook-5b79d5cb59        0         0         0
28h
replicaset.apps/awesome-webhook-5bf557c9f8        1         1         1
20h
replicaset.apps/awesome-webhook-77c647c987        0         0         0
23h
replicaset.apps/awesome-webhook-79d9c6f74c        0         0         0
23h
replicaset.apps/awesome-webhook-7d9d667b8d        0         0         0
9h
replicaset.apps/awesome-webhook-8668664d75        0         0         0
23h
replicaset.apps/awesome-webhook-9b6957476         0         0         0
24h
```

- To verify that the conventions are applied, ensure that the `PodIntent` of a workload that matches the convention criteria:

```
kubectl -o yaml get podintents.conventions.apps.tanzu.vmware.co awesome-a
pp
```

```
apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
  creationTimestamp: "2021-10-07T13:30:00Z"
  generation: 1
  labels:
    app.kubernetes.io/component: intent
    carto.run/cluster-supply-chain-name: awesome-supply-chain
    carto.run/cluster-template-name: convention-template
    carto.run/component-name: config-provider
    carto.run/template-kind: ClusterConfigTemplate
    carto.run/workload-name: awesome-app
    carto.run/workload-namespace: default
  name: awesome-app
  namespace: default
ownerReferences:
- apiVersion: carto.run/v1alpha1
  blockOwnerDeletion: true
  controller: true
  kind: Workload
  name: awesome-app
  uid: "********"
resourceVersion: "********"
uid: "********"
spec:
imagePullSecrets:
  - name: registry-credentials
    serviceAccountName: default
    template:
      metadata:
        annotations:
          developer.conventions/target-containers: workload
        labels:
          app.kubernetes.io/component: run
```

```
          app.kubernetes.io/part-of: awesome-app
          carto.run/workload-name: awesome-app
      spec:
        containers:
        - image: awesome-repo.com/awesome-project/awesome-app@sha256:****
****
          name: workload
          resources: {}
          securityContext:
          runAsUser: 1000
status:
  conditions:
  - lastTransitionTime: "2021-10-07T13:30:00Z"
    status: "True"
    type: ConventionsApplied
  - lastTransitionTime: "2021-10-07T13:30:00Z"
    status: "True"
    type: Ready
observedGeneration: 1
template:
  metadata:
    annotations:
      awesome-annotation: awesome-value
      conventions.carto.run/applied-conventions: |-
        awesome-label-convention
        awesome-annotation-convention
        awesome-envs-convention
        awesome-image-convention
        developer.conventions/target-containers: workload
    labels:
      awesome-label: awesome-value
      app.kubernetes.io/component: run
      app.kubernetes.io/part-of: awesome-app
      carto.run/workload-name: awesome-app
      conventions.carto.run/framework: go
  spec:
    containers:
    - env:
      - name: MY_AWESOME_VAR
        value: "MY_AWESOME_VALUE"
      image: awesome-repo.com/awesome-project/awesome-app@sha256:********
      name: workload
      ports:
        - containerPort: 8080
          protocol: TCP
      resources: {}
      securityContext:
        runAsUser: 1000
```

Next Steps

Keep Exploring:

- Try to use different matching criteria for the conventions or enhance the supply chain with multiple conventions.

Troubleshoot Cartographer Conventions

This topic describes how you can troubleshoot Cartographer Conventions.

No server in the cluster

Symptoms

- When a `PodIntent` is submitted, no `convention` is applied.

Cause

When there are no `convention servers` (ClusterPodConvention) deployed in the cluster or none of the existing convention servers applied any conventions, the `PodIntent` is not being mutated.

Solution

Deploy a `convention server` (ClusterPodConvention) in the cluster.

Server with wrong certificates configured

Symptoms

- When a `PodIntent` is submitted, the `conventions` are not applied.
- The `convention-controller` logs report an error `failed to get CABundle` as follows:


```
{
"level": "error",
"ts": 1638222343.6839523,
"logger": "controllers.PodIntent.PodIntent.ResolveConventions",
"msg": "failed to get CABundle",
"ClusterPodConvention": "base-convention",
"error": "unable to find valid certificaterequests for certificate \"convention
-template/webhook-certificate\"",
"stacktrace": "reflect.Value.Call\n\treflect/value.go:339\ngithub.com/vmware-la
bs/reconciler-runtime/reconcilers.(*SyncReconciler).sync\n\tgithub.com/vmware-l
abs/reconciler-runtime@v0.3.0/reconcilers/reconcilers.go:287\ngithub.com/vmware
-labs/reconciler-runtime/reconcilers.(*SyncReconciler).Reconcile\n\tgithub.com/
vmware-labs/reconciler-runtime@v0.3.0/reconcilers/reconcilers.go:276\ngithub.co
m/vmware-labs/reconciler-runtime/reconcilers.Sequence.Reconcile\n\tgithub.com/v
mware-labs/reconciler-runtime@v0.3.0/reconcilers/reconcilers.go:815\ngithub.co
m/vmware-labs/reconciler-runtime/reconcilers.(*ParentReconciler).reconcile\n\tg
ithub.com/vmware-labs/reconciler-runtime@v0.3.0/reconcilers/reconcilers.go:146
\ngithub.com/vmware-labs/reconciler-runtime/reconcilers.(*ParentReconciler).Rec
oncile\n\tgithub.com/vmware-labs/reconciler-runtime@v0.3.0/reconcilers/reconcil
ers.go:120\nsigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controlle
r).Reconcile\n\tsigs.k8s.io/controller-runtime@v0.10.3/pkg/internal/controller/
controller.go:114\nsigs.k8s.io/controller-runtime/pkg/internal/controller.(*Con
troller).reconcileHandler\n\tsigs.k8s.io/controller-runtime@v0.10.3/pkg/interna
l/controller/controller.go:311\nsigs.k8s.io/controller-runtime/pkg/internal/con
troller.(*Controller).processNextWorkItem\n\tsigs.k8s.io/controller-runtime@v0.
10.3/pkg/internal/controller/controller.go:266\nsigs.k8s.io/controller-runtime/
pkg/internal/controller.(*Controller).Start.func2.2\n\tsigs.k8s.io/controller-r
untime@v0.10.3/pkg/internal/controller/controller.go:227"
```


Cause

`convention server` (ClusterPodConvention) is configured with the wrong certificates. The `convention-controller` cannot figure out the CA Bundle to perform the request to the server.

Solution

Ensure that the `convention server` (ClusterPodConvention) is configured with the correct certificates. To do so, verify the value of annotation `conventions.carto.run/inject-ca-from` which must be set to the used Certificate.

Important

Do not set annotation `conventions.carto.run/inject-ca-from` if no certificate is used.

Server fails when processing a request

Symptoms

- When a `PodIntent` is submitted, the `convention` is not applied.
- The `convention-controller` logs reports `failed to apply convention` error like this.

```
{"level":"error","ts":1638205387.8813763,"logger":"controllers.PodIntent.PodInt
ent.ApplyConventions","msg":"failed to apply convention","Convention":{"Nam
e":"base-convention","Selectors":null,"Priority":"Normal","ClientConfig":{"serv
ice":{"namespace":"convention-template","name":"webhook","port":443},"caBundl
e":"..."}},"error":"Post \"https://webhook.convention-template.svc:443/?timeout
=30s\": EOF","stacktrace":"reflect.Value.call\n\treflect/value.go:543\nreflect.
Value.Call\n\treflect/value.go:339\ngithub.com/vmware-labs/reconciler-runtime/r
econcilers.(*SyncReconciler).sync\n\tgithub.com/vmware-labs/reconciler-runtime@
v0.3.0/reconcilers/reconcilers.go:287\ngithub.com/vmware-labs/reconciler-runtim
e/reconcilers.(*SyncReconciler).Reconcile\n\tgithub.com/vmware-labs/reconciler-
runtime@v0.3.0/reconcilers/reconcilers.go:276\ngithub.com/vmware-labs/reconcile
r-runtime/reconcilers.Sequence.Reconcile\n\tgithub.com/vmware-labs/reconciler-r
untime@v0.3.0/reconcilers/reconcilers.go:815\ngithub.com/vmware-labs/reconciler
-runtime/reconcilers.(*ParentReconciler).reconcile\n\tgithub.com/vmware-labs/re
conciler-runtime@v0.3.0/reconcilers/reconcilers.go:146\ngithub.com/vmware-labs/
reconciler-runtime/reconcilers.(*ParentReconciler).Reconcile\n\tgithub.com/vmwa
re-labs/reconciler-runtime@v0.3.0/reconcilers/reconcilers.go:120\nsigs.k8s.io/c
ontroller-runtime/pkg/internal/controller.(*Controller).Reconcile\n\tsigs.k8s.i
o/controller-runtime@v0.10.0/pkg/internal/controller/controller.go:114\nsigs.k8
s.io/controller-runtime/pkg/internal/controller.(*Controller).reconcileHandler
\n\tsigs.k8s.io/controller-runtime@v0.10.0/pkg/internal/controller/controller.go
:311\nsigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).pro
cessNextWorkItem\n\tsigs.k8s.io/controller-runtime@v0.10.0/pkg/internal/control
ler/controller.go:266\nsigs.k8s.io/controller-runtime/pkg/internal/controller.
(*Controller).Start.func2.2\n\tsigs.k8s.io/controller-runtime@v0.10.0/pkg/inter
nal/controller/controller.go:227"}
```

- When a `PodIntent` status message is updated with `failed to apply convention from source base-convention: Post "https://webhook.convention-template.svc:443/?timeout=30s": EOF`.

Cause

An unmanaged error occurs in the `convention server` when processing a request.

Solution

1. Inspect the `convention server` logs to identify the cause of the error:
    1. To retrieve the `convention server` logs:

        ```
        kubectl -n convention-template logs deployment/webhook
        ```

        Where:

        - The convention server was deployed as a `Deployment`
        - `webhook` is the name of the convention server `Deployment`.

    - `convention-template` is the namespace where the convention server is deployed.

2. Identify the error and deploy a fixed version of `convention server`.
    - The new deployment is not applied to the existing `PodIntent`s. It is only applied to the new `PodIntent`s.
    - To apply new deployment to exiting `PodIntent`, you must update the `PodIntent`, so the reconciler applies if it matches the criteria.

Connection refused due to unsecured connection

Symptoms

- When a `PodIntent` is submitted, the `convention` is not applied.
- The `convention-controller` logs reports a connection refused error as follows:

```
{"level":"error","ts":1638202791.5734537,"logger":"controllers.PodIntent.PodInt
ent.ApplyConventions","msg":"failed to apply convention","Convention":{"Nam
e":"base-convention","Selectors":null,"Priority":"Normal","ClientConfig":{"serv
ice":{"namespace":"convention-template","name":"webhook","port":443},"caBundl
e":"..."}},"error":"Post \"https://webhook.convention-template.svc:443/?timeout
=30s\": dial tcp 10.56.13.206:443: connect: connection refused","stacktrace":"r
eflect.Value.call\n\treflect/value.go:543\nreflect.Value.Call\n\treflect/value.
go:339\ngithub.com/vmware-labs/reconciler-runtime/reconcilers.(*SyncReconcile
r).sync\n\tgithub.com/vmware-labs/reconciler-runtime@v0.3.0/reconcilers/reconci
lers.go:287\ngithub.com/vmware-labs/reconciler-runtime/reconcilers.(*SyncReconc
iler).Reconcile\n\tgithub.com/vmware-labs/reconciler-runtime@v0.3.0/reconciler
s/reconcilers.go:276\ngithub.com/vmware-labs/reconciler-runtime/reconcilers.Seq
uence.Reconcile\n\tgithub.com/vmware-labs/reconciler-runtime@v0.3.0/reconciler
s/reconcilers.go:815\ngithub.com/vmware-labs/reconciler-runtime/reconcilers.(*P
arentReconciler).reconcile\n\tgithub.com/vmware-labs/reconciler-runtime@v0.3.0/
reconcilers/reconcilers.go:146\ngithub.com/vmware-labs/reconciler-runtime/recon
cilers.(*ParentReconciler).Reconcile\n\tgithub.com/vmware-labs/reconciler-runti
me@v0.3.0/reconcilers/reconcilers.go:120\nsigs.k8s.io/controller-runtime/pkg/in
ternal/controller.(*Controller).Reconcile\n\tsigs.k8s.io/controller-runtime@v0.
10.0/pkg/internal/controller/controller.go:114\nsigs.k8s.io/controller-runtime/
pkg/internal/controller.(*Controller).reconcileHandler\n\tsigs.k8s.io/controlle
r-runtime@v0.10.0/pkg/internal/controller/controller.go:311\nsigs.k8s.io/contro
ller-runtime/pkg/internal/controller.(*Controller).processNextWorkItem\n\tsigs.
k8s.io/controller-runtime@v0.10.0/pkg/internal/controller/controller.go:266\nsi
gs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).Start.func2.
2\n\tsigs.k8s.io/controller-runtime@v0.10.0/pkg/internal/controller/controller.
go:227"}
```

- The `convention server` fails to start due to `server gave HTTP response to HTTPS client`:
    - When checking the `convention server` events by running:

```
kubectl -n convention-template describe pod webhook-594d75d69b-4w4s8
```

    Where:

    - The convention server was deployed as a `Deployment`
    - `webhook-594d75d69b-4w4s8` is the name of the `convention server` Pod.
    - `convention-template` is the namespace where the convention server is deployed.

    For example:

```
Name:         webhook-594d75d69b-4w4s8
Namespace:    convention-template
```

```
...
Containers:
  webhook:
...
Events:
Type     Reason     Age                    From               Message
----     ------     ----                   ----               -------
Normal   Scheduled  14m                    default-scheduler  Successfully assig
ned convention-template/webhook-594d75d69b-4w4s8 to pool
Normal   Pulling    14m                    kubelet            Pulling image "awe
some-repo/awesome-user/awesome-convention-..."
Normal   Pulled     14m                    kubelet            Successfully pulle
d image "awesome-repo/awesome-user/awesome-convention..." in 1.06032653s
Normal   Created    13m (x2 over 14m)      kubelet            Created container
webhook
Normal   Started    13m (x2 over 14m)      kubelet            Started container
webhook
Warning  Unhealthy  13m (x9 over 14m)      kubelet            Readiness probe fa
iled: Get "https://10.52.2.74:8443/healthz": http: server gave HTTP response to
HTTPS client
Warning  Unhealthy  13m (x6 over 14m)      kubelet            Liveness probe fai
led: Get "https://10.52.2.74:8443/healthz": http: server gave HTTP response to
HTTPS client
Normal   Pulled     9m13s (x6 over 13m)    kubelet            Container image "a
wesome-repo/awesome-user/awesome-convention" already present on machine
Warning  BackOff    4m22s (x32 over 11m)   kubelet            Back-off restartin
g failed container
```

Cause

When a `convention server` is provided without using Transport Layer Security (TLS) but the `Deployment` is configured to use TLS, Kubernetes fails to deploy the `Pod` because of the `liveness probe`.

Solution

1. Deploy a `convention server` with TLS enabled.
2. Create `ClusterPodConvention` resource for the convention server with annotation `conventions.carto.run/inject-ca-from` as a pointer to the deployed `Certificate` resource.

Self-signed certificate authority (CA) not propagated to the Convention Service

Symptoms

The self-signed certificate authority (CA) for a registry is not propagated to the Convention Service.

Cause

When you provide the self-signed certificate authority (CA) for a registry through `convention-controller.ca_cert_data`, it cannot be propagated to the Convention Service.

Solution

Define the CA by using the available `.shared.ca_cert_data` top-level key to supply the CA to the Convention Service.

No imagePullSecrets configured

Symptoms

When a PodIntent is submitted:

- No convention is applied.
- You see an `unauthorized to access repository` or `fetching metadata for Images failed` error when you inspect the workload.

Cause

The errors are seen when a `workload` is created in a developer namespace where `imagePullSecrets` are not defined on the `default` serviceAccount or on the preferred serviceAccount.

Solution

Add the `imagePullSecrets` name to the default serviceAccount or the preferred serviceAccount.

For example:

```
kind: ServiceAccount
metadata:
  name: default
  namespace: my-workload-namespace
imagePullSecrets:
  - name: registry-credentials # ensure this secret is defined
secrets:
- name: registry-credentials
```

`OOMKilled` convention controller

Symptoms

While processing workloads with large SBOM, the Cartographer Convention controller manager pod can fail with the status `CrashLoopBackOff` or `OOMKilled`.

To work around this problem you can increase the memory limit to `512Mi` to fix the pod crash.

For example:

```
NAME                                                    READY   STATUS
RESTARTS          AGE
cartographer-conventions-controller-manager-ff4cdf59d-5nzl5   0/1     CrashLoopBackOff
1292 (109s ago)   5d3h
```

The following is an example controller pod status:

```
containerStatuses:
  - containerID: containerd://b7b7159a9e00ef726944d642a1b649108bba610b34d8d10f9b5270ea
25d3db94
    image: sha256:9827e8e5b30d47c9373a1907dc5e7e15a76d2a4581e803eb6f2cb24e3a9ea62e
    imageID: my.image.registry.com/tanzu-application-platform/tap-packages@sha256:3cd1
ae92f534ff935fbaf992b8308aa3dac3d1b6cbc8cf8a856451c8c92540f66
    lastState:
      terminated:
        containerID: containerd://b7b7159a9e00ef726944d642a1b649108bba610b34d8d10f9b52
70ea25d3db94
        exitCode: 137
        finishedAt: "2023-11-06T21:02:56Z"
        reason: OOMKilled
```

```
        startedAt: "2023-11-06T21:02:10Z"
    name: manager
```

Cause

This error usually occurs when a `workload` image, built by the supply chain, contains a large SBOM. The default resource limit set during installation might not be large enough to process the pod conventions which can lead to the controller pod crashing.

Solution

To increase the Cartographer Convention controller manager memory limit via the TAP values.yaml. For example:

- To increase the memory limit for convention server, see Increase the memory limit for convention server.
- To increase the memory limit for convention webhook servers, such as app-live-view-conventions, spring-boot-webhook, and developer-conventions/webhook, see Increase the memory limit for convention webhook servers.

Increase the memory limit for convention server

To increase the memory limit, add the desired resource limit under key `cartographer_conventions` in the TAP value.yaml:

```
cartographer_conventions:
  resource:
    memory: 512Mi
```

1. Update Tanzu Application Platform by running:

   ```
   tanzu package installed update tap -p tap.tanzu.vmware.com -v 1.9.1  --values-f
   ile tap-values.yaml -n tap-install
   ```

For information about the package customization, see Customize your package installation.

Increase the memory limit for convention webhook servers

You might need to increase the memory limit for the following convention webhook servers:

- app-live-view-conventions
- spring-boot-webhook
- developer-conventions/webhook

Use this procedure to increase the memory limit:

1. Create a `Secret` with the following ytt overlay.

   ```
   apiVersion: v1
   kind: Secret
   metadata:
   name: patch-app-live-view-conventions
   namespace: tap-install
   stringData:
   patch-conventions-controller.yaml: |
     #@ load("@ytt:overlay", "overlay")

     #@overlay/match by=overlay.subset({"kind":"Deployment", "metadata":{"name":"a
   ppliveview-webhook", "namespace": "app-live-view-conventions"}})
   ```

```
  ---
  spec:
    template:
      spec:
        containers:
          #@overlay/match by=overlay.subset({"name": "webhook"})
          - name: webhook
            resources:
              limits:
                memory: 512Mi
---
apiVersion: v1
kind: Secret
metadata:
name: patch-spring-boot-conventions
namespace: tap-install
stringData:
patch-conventions-controller.yaml: |
  #@ load("@ytt:overlay", "overlay")

  #@overlay/match by=overlay.subset({"kind":"Deployment", "metadata":{"name":"s
pring-boot-webhook", "namespace": "spring-boot-convention"}})
  ---
  spec:
    template:
      spec:
        containers:
          #@overlay/match by=overlay.subset({"name": "webhook"})
          - name: webhook
            resources:
              limits:
                memory: 512Mi
---
apiVersion: v1
kind: Secret
metadata:
name: patch-developer-conventions
namespace: tap-install
stringData:
patch-conventions-controller.yaml: |
  #@ load("@ytt:overlay", "overlay")

  #@overlay/match by=overlay.subset({"kind":"Deployment", "metadata":{"name":"w
ebhook", "namespace": "developer-conventions"}})
  ---
  spec:
    template:
      spec:
        containers:
          #@overlay/match by=overlay.subset({"name": "webhook"})
          - name: webhook
            resources:
              limits:
                memory: 512Mi
```

2. Update the Tanzu Application Platform values YAML file to include a `package_overlays` field:

```
package_overlays:
- name: appliveview-conventions
secrets:
- name: patch-app-live-view-conventions
- name: spring-boot-conventions
secrets:
- name: patch-spring-boot-conventions
```

```
- name: developer-conventions
secrets:
- name: patch-developer-conventions
```

3. Update Tanzu Application Platform by running:

```
tanzu package installed update tap -p tap.tanzu.vmware.com -v 1.9.1  --values-f
ile tap-values.yaml -n tap-install
```

For information about the package customization, see Customize your package installation.

Convention Service Resources for Cartographer Conventions

This reference topic describes the convention service resources you can use with Cartographer Conventions.

Overview

There are several resources involved in the application of conventions to workloads and these are typically consumed by platform developers and operators rather than by application developers.

- ClusterPodConvention

  The following is an example `conventions.carto.run/v1alpha1` type:

  ```
  ---
  apiVersion: conventions.carto.run/v1alpha1
  kind: ClusterPodConvention
  metadata:
  name: sample
  spec:
  selectorTarget: PodTemplateSpec # optional field with options, defaults to PodT
  emplateSpec
  selectors: # optional, defaults to match all workloads
  - <metav1.LabelSelector>
  webhook:
    certificate:
      name: sample-cert
      namespace: sample-conventions
    clientConfig:
      <admissionregistrationv1.WebhookClientConfig>
  ```

  A `ClusterPodConvention` can target a one or more workloads of different types. You can apply multiple conventions to a single workload. It is at the discretion of the “Conventions Author” how a convention is applied.

  To list out available conventions in your cluster, run the following `kubectl`command

  ```
  $ kubectl get clusterpodconventions.conventions.carto.run

    NAME                     AGE
    appliveview-sample       23h
    developer-conventions    23h
    spring-boot-convention   23h
  ```

- PodIntent

  The following is an example `conventions.carto.run/v1alpha1` resource:

```
apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
name: sample
spec:
imagePullSecrets: <[]corev1.LocalObjectReference> # optional
serviceAccountName: <string> # optional, defaults to 'default'
template:
  <corev1.PodTemplateSpec>
status:
observedGeneration: 1 # reflected from .metadata.generation
conditions:
- <metav1.Condition>
template: # enriched PodTemplateSpec
  <corev1.PodTemplateSpec>
```

To list out available `PodIntent` resources in your cluster, run the following kubectl command

```
# specify relevant namespace
kubectl get podintents.conventions.carto.run -n my-apps

NAME            READY   REASON               AGE
spring-sample   True    ConventionsApplied   8m5s
```

When a`PodIntent` is created, the `PodIntent` reconciler lists all `ClusterPodConventions` resources and applies them serially. To ensure that the consistency of the enriched PodTemplateSpec, the list of `ClusterPodConventions`is sorted alphabetically by name before applying the conventions.

Tip : *You can use strategic naming to control the order in which the conventions are applied.*

After the conventions are applied, the `Ready` status condition on the `PodIntent` resource is used to indicate whether it is applied. A list of all applied conventions is stored under the annotation `conventions.carto.run/applied-conventions`.

There are also a few other resources available to the `Conventions Author` that are not persisted in your cluster, including:

- ImageConfig
- PodConventionContextSpec
- PodConventionContextStatus
- PodConventionContext
- BOM

Collecting Logs from the Controller

A successful deployment of the convention service creates it's resources on the following `conventions-system` namespace:

```
$ kubectl get all -n conventions-system
  NAME                                                              READY   STATUS
RESTARTS   AGE
  ...
  pod/cartographer-conventions-controller-manager-76fd86789f-lzh86   1/1     Running
0          20h

  NAME                                                                 TYPE        CL
USTER-IP     EXTERNAL-IP   PORT(S)    AGE
```

```
  ...
  service/cartographer-conventions-controller-manager-metrics-service   ClusterIP   1
0.0.250.231   <none>        8443/TCP   20h
  service/cartographer-conventions-webhook-service                      ClusterIP   1
0.0.6.254     <none>        443/TCP    20h
  ...

  NAME                                                          READY   UP-TO-DATE   A
VAILABLE   AGE
  ...
  deployment.apps/cartographer-conventions-controller-manager   1/1     1            1
20h

  NAME                                                                    DESIRED   C
URRENT   READY   AGE
  ...
  replicaset.apps/cartographer-conventions-controller-manager-76fd86789f   1         1
1       20h
```

In order to examine logs from the cartographer conventions controller to help identify issues, inspect the cartographer conventions controller manager pod as follows

```
 kubectl -n conventions-system logs -l control-plane=controller-manager
...
{"level":"info","ts":"2023-02-06T20:49:19.850086032Z","logger":"MetricsReconciler","ms
g":"reconciling builders configmap","controller":"configmap","controllerGroup":"","con
trollerKind":"ConfigMap","ConfigMap":{"name":"controller-manager-metrics-data","namesp
ace":"conventions-system"},"namespace":"conventions-system","name":"controller-manager
-metrics-data","reconcileID":"6f5e38c7-0ce0-4c74-aff3-f938fb742dab","diff":"  map[stri
ng]string{\n- \t\"clusterpodconventions_names\": \"appliveview-sample\",\n+ \t\"cluste
rpodconventions_names\": \"appliveview-sample\nspring-boot-convention\",\n  \t\"podint
ents_count\":            \"0\",\n  }\n"}
{"level":"info","ts":"2023-02-06T20:49:20.101742252Z","logger":"MetricsReconciler","ms
g":"reconciling builders configmap","controller":"configmap","controllerGroup":"","con
trollerKind":"ConfigMap","ConfigMap":{"name":"controller-manager-metrics-data","namesp
ace":"conventions-system"},"namespace":"conventions-system","name":"controller-manager
-metrics-data","reconcileID":"3a1950bc-4c55-47bb-8380-2de574bd5d5e","diff":"  map[stri
ng]string{\n  \t\"clusterpodconventions_names\": strings.Join({\n  \t\t\"appliveview-s
ample\n\",\n+ \t\t\"developer-conventions\n\",\n  \t\t\"spring-boot-convention\",\n
\t}, \"\"),\n  \t\"podintents_count\": \"0\",\n  }\n"}
...
```

Convention Service Resources for Cartographer Conventions

This reference topic describes the convention service resources you can use with Cartographer Conventions.

Overview

There are several resources involved in the application of conventions to workloads and these are typically consumed by platform developers and operators rather than by application developers.

- ClusterPodConvention

  The following is an example `conventions.carto.run/v1alpha1` type:

  ```
  ---
  apiVersion: conventions.carto.run/v1alpha1
  kind: ClusterPodConvention
  metadata:
  name: sample
  ```

```
spec:
selectorTarget: PodTemplateSpec # optional field with options, defaults to PodT
emplateSpec
selectors: # optional, defaults to match all workloads
- <metav1.LabelSelector>
webhook:
  certificate:
    name: sample-cert
    namespace: sample-conventions
  clientConfig:
    <admissionregistrationv1.WebhookClientConfig>
```

A `ClusterPodConvention` can target a one or more workloads of different types. You can apply multiple conventions to a single workload. It is at the discretion of the “Conventions Author” how a convention is applied.

To list out available conventions in your cluster, run the following `kubectl`command

```
$ kubectl get clusterpodconventions.conventions.carto.run

  NAME                     AGE
  appliveview-sample       23h
  developer-conventions    23h
  spring-boot-convention   23h
```

- PodIntent

  The following is an example `conventions.carto.run/v1alpha1` resource:

```
apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
name: sample
spec:
imagePullSecrets: <[]corev1.LocalObjectReference> # optional
serviceAccountName: <string> # optional, defaults to 'default'
template:
  <corev1.PodTemplateSpec>
status:
observedGeneration: 1 # reflected from .metadata.generation
conditions:
- <metav1.Condition>
template: # enriched PodTemplateSpec
  <corev1.PodTemplateSpec>
```

  To list out available `PodIntent` resources in your cluster, run the following kubectl command

```
# specify relevant namespace
kubectl get podintents.conventions.carto.run -n my-apps

NAME            READY   REASON               AGE
spring-sample   True    ConventionsApplied   8m5s
```

  When a`PodIntent` is created, the `PodIntent` reconciler lists all `ClusterPodConventions` resources and applies them serially. To ensure that the consistency of the enriched PodTemplateSpec, the list of `ClusterPodConventions`is sorted alphabetically by name before applying the conventions.

  Tip : *You can use strategic naming to control the order in which the conventions are applied.*

After the conventions are applied, the `Ready` status condition on the `PodIntent` resource is used to indicate whether it is applied. A list of all applied conventions is stored under the annotation `conventions.carto.run/applied-conventions`.

There are also a few other resources available to the `Conventions Author` that are not persisted in your cluster, including:

- ImageConfig
- PodConventionContextSpec
- PodConventionContextStatus
- PodConventionContext
- BOM

Collecting Logs from the Controller

A successful deployment of the convention service creates it's resources on the following `conventions-system` namespace:

```
$ kubectl get all -n conventions-system
  NAME                                                              READY   STATUS
RESTARTS   AGE
  ...
  pod/cartographer-conventions-controller-manager-76fd86789f-lzh86   1/1     Running
0          20h

  NAME                                                                 TYPE        CL
USTER-IP     EXTERNAL-IP   PORT(S)    AGE
  ...
  service/cartographer-conventions-controller-manager-metrics-service   ClusterIP   1
0.0.250.231   <none>        8443/TCP   20h
  service/cartographer-conventions-webhook-service                      ClusterIP   1
0.0.6.254     <none>        443/TCP    20h
  ...

  NAME                                                          READY   UP-TO-DATE   A
VAILABLE   AGE
  ...
  deployment.apps/cartographer-conventions-controller-manager   1/1     1            1
20h

  NAME                                                                    DESIRED   C
URRENT   READY   AGE
  ...
  replicaset.apps/cartographer-conventions-controller-manager-76fd86789f   1         1
1       20h
```

In order to examine logs from the cartographer conventions controller to help identify issues, inspect the cartographer conventions controller manager pod as follows

```
 kubectl -n conventions-system logs -l control-plane=controller-manager
...
{"level":"info","ts":"2023-02-06T20:49:19.855086032Z","logger":"MetricsReconciler","ms
g":"reconciling builders configmap","controller":"configmap","controllerGroup":"","con
trollerKind":"ConfigMap","ConfigMap":{"name":"controller-manager-metrics-data","namesp
ace":"conventions-system"},"namespace":"conventions-system","name":"controller-manager
-metrics-data","reconcileID":"6f5e38c7-0ce0-4c74-aff3-f938fb742dab","diff":"  map[stri
ng]string{\n- \t\"clusterpodconventions_names\": \"appliveview-sample\",\n+ \t\"cluste
rpodconventions_names\": \"appliveview-sample\nspring-boot-convention\",\n  \t\"podint
ents_count\":            \"0\",\n  }\n"}
{"level":"info","ts":"2023-02-06T20:49:20.101742252Z","logger":"MetricsReconciler","ms
```

```
g":"reconciling builders configmap","controller":"configmap","controllerGroup":"","con
trollerKind":"ConfigMap","ConfigMap":{"name":"controller-manager-metrics-data","namesp
ace":"conventions-system"},"namespace":"conventions-system","name":"controller-manager
-metrics-data","reconcileID":"3a1950bc-4c55-47bb-8380-2de574bd5d5e","diff":"  map[stri
ng]string{\n  \t\"clusterpodconventions_names\": strings.Join({\n  \t\t\"appliveview-s
ample\n\",\n+ \t\t\"developer-conventions\n\",\n  \t\t\"spring-boot-convention\",\n
\t}, \"\"),\n  \t\"podintents_count\": \"0\",\n  }\n"}
...
```


ImageConfig for Cartographer Conventions

This reference topic describes the ImageConfig object you can use with Cartographer Conventions.

Overview

The image configuration object holds the name of the image, the BOM, and the OCI image configuration with image metadata from the repository.

OCI image configuration contains the metadata from the image repository.

The BOM represents the content of the image and may be zero or more per image.

```
{
  "name": "oci-image-name",
  "boms": [{
      "name": "bom-name",
      "raw": "`a byte array`"
  }],
  "config": {
      {
        "created": "2015-10-31T22:22:56.015925234Z",
        "author": "Alyssa P. Hacker <alyspdev@example.com>",
        "architecture": "amd64",
        "os": "linux",
        "config": {
            "User": "alice",
            "ExposedPorts": {
                "8080/tcp": {}
            },
            "Env": [
                "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
                "FOO=oci_is_a",
                "BAR=well_written_spec"
            ],
            "Entrypoint": [
                "/bin/my-app-binary"
            ],
            "Cmd": [
                "--foreground",
                "--config",
                "/etc/my-app.d/default.cfg"
            ],
            "Volumes": {
                "/var/job-result-data": {},
                "/var/log/my-app-logs": {}
            },
            "WorkingDir": "/home/alice",
            "Labels": {
                "com.example.project.git.url": "https://example.com/project.git",
                "com.example.project.git.commit": "45a939b2999782a3f005621a8d0f29aa387
e1d6b"
            }
        },
        "rootfs": {
```

```
        "diff_ids": [
            "sha256:c6f988f4874bb0add23a778f753c65efe992244e148a1d2ec2a8b664fb66bbd1",
            "sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef"
        ],
        "type": "layers"
        },
        "history": [
        {
            "created": "2015-10-31T22:22:54.690851953Z",
            "created_by": "/bin/sh -c #(nop) ADD file:a3bc1e842b69636f9df5256c49c5374f
b4eef1e281fe3f282c65fb853ee171c5 in /"
        },
        {
            "created": "2015-10-31T22:22:55.613815829Z",
            "created_by": "/bin/sh -c #(nop) CMD [\"sh\"]",
            "empty_layer": true
        },
        {
            "created": "2015-10-31T22:22:56.329850019Z",
            "created_by": "/bin/sh -c apk add curl"
        }
        ]
    }
  }
}
```

PodConventionContextSpec for Cartographer Conventions

This reference topic describes the `PodConventionContextSpec` you can use with Cartographer Conventions.

Overview

The Pod convention context specification is a wrapper of the PodTemplateSpec and the ImageConfig provided in the request body of the server. It represents the original `PodTemplateSpec`. For more information on `PodTemplateSpec`, see the Kubernetes documentation.

```
{
"template": {
    "metadata": {
        ...
    },
    "spec": {
        ...
    }
},
"imageConfig": {
    ...
  "name": "oci-image-name",
  "config": {
        ...
    }
  }
}
```

PodConventionContextStatus for Cartographer Conventions

This reference topic describes the `PodConventionContextStatus` status type that you can use with Cartographer Conventions.

Overview

The Pod convention context status type is used to represent the current status of the context retrieved by the request. It holds the applied conventions by the server and the modified version of the `PodTemplateSpec`. For more information about `PodTemplateSpec`, see the Kubernetes documentation.

The field `.template` is populated with the enriched PodTemplateSpec. The field `.appliedConventions` is populated with the names of any applied conventions.

```
{
    "template": {
        "metadata": {
            ...
        },
        "spec": {
            ...
        }
    },
    "appliedConventions": [
        "convention-1",
        "convention-2",
        "convention-4"
    ]
}
```

yaml version:

```
---
apiVersion: webhooks.conventions.carto.run/v1alpha1
kind: PodConventionContext
metadata:
  name: sample # the name of the ClusterPodConvention
spec: # the request
  imageConfig:
  template:
    <corev1.PodTemplateSpec>
status: # the response
  appliedConventions: # list of names of conventions applied
  - my-convention
  template:
  spec:
      containers:
      - name : workload
        image: helloworld-go-mod
```

PodConventionContext for Cartographer Conventions

This reference topic describes the `PodConventionContext` that you can use with Cartographer Conventions.

Overview

The Pod convention context is the body of the webhook request and response. The specification is provided by the convention controller and the status is set by the convention server.

The context is a wrapper of the individual object description in an API (TypeMeta), the persistent metadata of a resource (ObjectMeta), the PodConventionContextSpec and the PodConventionContextStatus.

PodConventionContext Objects

In the `PodConventionContext` API resource:

- Object path `.spec.template` field defines the `PodTemplateSpec` to be enriched by conventions. For more information about `PodTemplateSpec`, see the Kubernetes documentation.
- Object path `.spec.imageConfig[]` field defines ImageConfig. Each entry of it is populated with the name of the image (`.spec.imageConfig[].image`) and its OCI metadata (`.spec.imageConfig[].config`). These entries are generated for each image referenced in PodTemplateSpec (`.spec.template`).

The following is an example of a `PodConventionContext` resource request received by the convention server. This resource is generated for a Go language-based application image in GitHub. It is built with Cloud Native Paketo Buildpacks that use Go mod for dependency management.

```
---
apiVersion: webhooks.conventions.carto.run/v1alpha1
kind: PodConventionContext
metadata:
  name: sample # the name of the ClusterPodConvention
spec: # the request
  imageConfig: # one entry per image referenced by the PodTemplateSpec
  - image: sample/go-based-image
    boms:
    - name: cnb-app:.../sbom.cdx.json
      raw: ...
    config:
      entrypoint:
      - "/cnb/process/web"
      domainname: ""
      architecture: "amd64"
      image: "sha256:05b698a4949db54fdb36ea431477867abf51054abd0cbfcfd1bb81cda1842288"
      labels:
        "io.buildpacks.stack.distro.version": "18.04"
        "io.buildpacks.stack.homepage": "https://github.com/paketo-buildpacks/stacks"
        "io.buildpacks.stack.id": "io.buildpacks.stacks.bionic"
        "io.buildpacks.stack.maintainer": "Paketo Buildpacks"
        "io.buildpacks.stack.distro.name": "Ubuntu"
        "io.buildpacks.stack.metadata": `{"app":[{"sha":"sha256:ea4ec23266a3af1204fd64
3de0f3572dd8dbb5697a5ef15bdae844777c19bf8f"}],
        "buildpacks":[{"key":"paketo-buildpac`...,
        "io.buildpacks.build.metadata": `{"bom":[{"name":"go","metadata":{"licenses":
[],"name":"Go","sha256":"7fef8ba6a0786143efcce66b0bbfbfbab02afeef522b4e09833c5b550d7
`...
  template:
    spec:
      containers:
      - name : workload
        image: helloworld-go-mod
```

PodConventionContext Structure

This section introduces more information about the image configuration in `PodConventionContext`. The convention-controller passes this information for each image in good faith. The controller is not the source of the metadata, and there is no guarantee that the information is correct.

The `config` field in the image configuration passes through the OCI Image metadata in GitHub loaded from the registry for the image.

The `boms` field in the image configuration passes through the BOMs of the image. Conventions might parse the BOMs they want to inspect. There is no guarantee that an image contains a BOM or that the BOM is in a certain format.

ClusterPodConvention for Cartographer Conventions

This reference topic describes the `ClusterPodConvention` that you can use with Cartographer Conventions.

Overview

A `ClusterPodConvention` defines how to connect to convention servers. It provides a way to apply a set of conventions to a PodTemplateSpec and artifact metadata. A convention typically focuses on a particular application framework, but might be cross cutting. Applied conventions must be pure functions.

Define conventions

Webhook servers are the only way to define conventions.

```
apiVersion: conventions.carto.run/v1alpha1
kind: ClusterPodConvention
metadata:
  name: base-convention
  annotations:
    conventions.carto.run/inject-ca-from: "convention-template/webhook-cert"
spec:
  selectorTarget: PodTemplateSpec # optional, defaults to PodTemplateSpec; field optio
ns include PodTemplateSpec|PodIntent
  webhook:
    clientConfig:
      service:
        name: webhook
        namespace: convention-template
```

PodIntent for Cartographer Conventions

This reference topic describes `PodIntent` that you can use with Cartographer Conventions.

Overview

The conditional criteria governing the application of a convention is customizable and is based on the evaluation of a custom Kubernetes resource called PodIntent.

`PodIntent` applies conventions to a workload. A PodIntent is created, or updated, when a workload is run by using a Tanzu Application Platform supply chain.

The `.spec.template`'s PodTemplateSpec is enriched by the conventions and exposed as the `.status.template`s PodTemplateSpec. A log of which sources and conventions are applied is captured with the `conventions.carto.run/applied-conventions` annotation on the PodTemplateSpec.

```
apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
  name: sample
spec:
  template:
```

```
    spec:
      containers:
      - name: workload
        image: ubuntu
```

BOM for Cartographer Conventions

This reference topic describes the `BOM` structure you can use with Cartographer Conventions.

Overview

The `BOM` is a type/structure wrapping a Software Bill of Materials (SBOM) describing the software components and their dependencies.

Structure

The structure of the `BOM` is defined as follows:

```
{
  "name": "BOM-NAME",
  "raw": "BYTE-ARRAY"
}
```

Where:

- `BOM-NAME` is the prefix `cnb-sbom:`, followed by the location of the BOM definition in the layer for a cloud native buildpack (CNB) SBOM. For example: `cnb-sbom:/layers/sbom/launch/paketo-buildpacks_executable-jar/sbom.cdx.json`. For a non-CNB SBOM, the value of `name` might change.
- `BYTE-ARRAY`: The content of the BOM. The content may be in any format or encoding. Consult the name to infer how the content is structured.

The convention controller forwards BOMs to the convention servers that it can discover from known sources, including:

- CNB-SBOM

Overview of cert-manager

cert-manager adds certificates and certificate issuers as resource types to Kubernetes clusters. It also helps you to obtain, renew, and use those certificates. For more information about cert-manager, see cert-manager documentation.

The cert-manager package allows you to, optionally, configure a number of `ClusterIssuer`. When you install Tanzu Application Platform by using profiles, a self-signed `ClusterIssuer` is included by default.

As of `cert-manager.tanzu.vmware.com/2.0.0`, versioning departs from the upstream, open-source project's version. The contained cert-manager version is reflected in `Package.spec.includedSoftware`. You can identify the version of cert-manager as follows:

```
kubectl get package -n tap-install cert-manager.tanzu.vmware.com.2.0.0 -ojsonpath='{.s
pec.includedSoftware}' | jq
[
 {
   "description": "X.509 certificate management for Kubernetes and OpenShift",
   "displayName": "cert-manager",
   "version": "1.9.1"
```

```
 }
]
```

Caution

ACME HTTP01 challenges can fail under certain conditions. For more information, see ACME challenges.

Overview of cert-manager

cert-manager adds certificates and certificate issuers as resource types to Kubernetes clusters. It also helps you to obtain, renew, and use those certificates. For more information about cert-manager, see cert-manager documentation.

The cert-manager package allows you to, optionally, configure a number of `ClusterIssuer`. When you install Tanzu Application Platform by using profiles, a self-signed `ClusterIssuer` is included by default.

As of `cert-manager.tanzu.vmware.com/2.0.0`, versioning departs from the upstream, open-source project's version. The contained cert-manager version is reflected in `Package.spec.includedSoftware`. You can identify the version of cert-manager as follows:

```
kubectl get package -n tap-install cert-manager.tanzu.vmware.com.2.0.0 -ojsonpath='{.s
pec.includedSoftware}' | jq
[
 {
   "description": "X.509 certificate management for Kubernetes and OpenShift",
   "displayName": "cert-manager",
   "version": "1.9.1"
 }
]
```

Caution

ACME HTTP01 challenges can fail under certain conditions. For more information, see ACME challenges.

Install cert-manager

This topic tells you how to install cert-manager from the Tanzu Application Platform (commonly known as TAP) package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install cert-manager. For more information about profiles, see Components and installation profiles.

The cert-manager package installs cert-manager and, optionally, a number of `ClusterIssuer`.

To install cert-manager with a self-signed `ClusterIssuer` from the Tanzu Application Platform package repository:

1. List version information for the package by running:

```
tanzu package available list cert-manager.tanzu.vmware.com -n tap-install
```

For example:

```
$ tanzu package available list cert-manager.tanzu.vmware.com -n tap-install
/ Retrieving package versions for cert-manager.tanzu.vmware.com...
  NAME                           VERSION           RELEASED-AT
  cert-manager.tanzu.vmware.com  2.0.0             ...
```

2. Discover available configuration for the package by running:

```
tanzu package available get cert-manager.tanzu.vmware.com/2.0.0 --namespace tap
-install --values-schema
```

For example:

```
$ tanzu package available get cert-manager.tanzu.vmware.com/2.0.0 --namespace t
ap-install --values-schema

KEY                    DEFAULT  TYPE    DESCRIPTION
certManager.pspNames   []       array   PodSecurityPolicy names which cert-manag
er is allowed to use
issuers                []       array   The ClusterIssuers to install - default:
[]
namespace                       string  Cert-manager's namespace - also used as
its cluster resource namespace
https://cert-manager.io/v1.9-docs/faq/cluster-resource/
```

3. Create a file named `cert-manager-rbac.yaml` by using the following sample:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: cert-manager-tap-install-cluster-admin-role
rules:
- apiGroups:
  - '*'
  resources:
  - '*'
  verbs:
  - '*'
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: cert-manager-tap-install-cluster-admin-role-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cert-manager-tap-install-cluster-admin-role
subjects:
- kind: ServiceAccount
  name: cert-manager-tap-install-sa
  namespace: tap-install
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: cert-manager-tap-install-sa
  namespace: tap-install
```

Apply the configuration:

```
kubectl apply -f cert-manager-rbac.yaml
```

4. Create a file named `cert-manager-install.yaml` by using the following sample:

```
---
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: cert-manager
  namespace: tap-install
spec:
  serviceAccountName: cert-manager-tap-install-sa
  packageRef:
    refName: cert-manager.tanzu.vmware.com
    versionSelection:
      constraints: "VERSION-NUMBER"
      prereleases: {}
  values:
    - secretRef:
        name: cert-manager-values

---
apiVersion: v1
kind: Secret
metadata:
  name: cert-manager-values
  namespace: tap-install
stringData:
  values.yaml: |
    issuers:
      - name: tap-ingress-selfsigned
        self_signed: {}
```

   Where:

   - `VERSION-NUMBER` is the version of the package listed earlier.
   - Secret `cert-manager-values` contains your configuration of the cert-manager package.

   Apply the configuration:

```
kubectl apply -f cert-manager-install.yaml
```

5. Verify the package installation:

```
tanzu package installed get cert-manager -n tap-install
```

   For example:

```
$ tanzu package installed get cert-manager -n tap-install
/ Retrieving installation details for cert-manager...
NAME:                    cert-manager
PACKAGE-NAME:            cert-manager.tanzu.vmware.com
PACKAGE-VERSION:         2.0.0
STATUS:                  Reconcile succeeded
CONDITIONS:              [{ReconcileSucceeded True}]
USEFUL-ERROR-MESSAGE:
```

   Verify that `STATUS` is `Reconcile succeeded`

6. Verify that cert-manager is up and running:

```
kubectl get deployment -n cert-manager
```

For example:

```
$ kubectl get deployment -n cert-manager
NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
cert-manager              1/1     1            1           5m
cert-manager-cainjector   1/1     1            1           5m
cert-manager-webhook      1/1     1            1           5m
```

7. Verify that the self-signed `ClusterIssuer` is present:

```
kubectl get clusterissuer
```

For example:

```
$ kubectl get clusterissuer
NAME                                READY   AGE
tap-ingress-selfsigned              True    5m
tap-ingress-selfsigned-bootstrap    True    5m
...
```

ACME challenges

cert-manager.io provides APIs for managing certificates on Kubernetes. It is one of the most popular extensions for Kubernetes and has found ubiquitous adoption. Automatic Certificate Management Environment (commonly called ACME) is a protocol for automatically obtaining certificates from certificate authorities. LetsEncrypt has designed and pioneered ACME and is one of the most-popular ACME-style, public CA.

You can use ACME with either an HTTP01 or a DNS01 challenge. In both cases the certificate requester must prove ownership of the domain either by answering a plain HTTP request or by setting a DNS record.

When using cert-manager's `(Cluster)Issuer` with an ACME HTTP01 challenge solver, a `Pod` is run and exposed to the network by using ingress. The `Pod` receives the challenge from the CA and responds. If the challenge is solved successfully, the certificate is issued.

When using cert-manager's `(Cluster)Issuer` with an ACME DNS01 challenge solver, the owner of the domain answers the challenge by setting a `TXT` record under the domain name. If the challenge is solved successfully, the certificate is issued.

Working with DNS01 challenges is harder than with HTTP01 challenges, but can work in situations where HTTP01 can't.

HTTP01 challenges can fail

All of components' images of Tanzu Application Platform are relocated to and pulled from a private registry. This also applies to `cert-manager.tanzu.vmware.com`, including the ACME HTTP01 solver `Pod`'s image.

Due to the design of cert-manager's `(Cluster)Issuer` resources, it is not easy to provide them with credentials to your private registry in a way that works consistently across all namespaces in your cluster.

You can deeply integrate a cluster with a private registry so that image pull secrets don't have to be provided explicitely. This is a common practice with popular cloud-based Kubernetes providers such as GKE, AKS and EKS.

As a result, ACME HTTP01 challenges can fail when your cluster is not deeply integrated with your private registry. In that case VMware recommends the following workarounds:

- (Recommended) Use DNS01 challenges
- Provide your `(Cluster)Issuer` with a `ServiceAccount` by using its pod template so that it can pull from your registry. For example:

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
 name: tap-acme-http01-solver
 namespace: #! ...
imagePullSecrets:
 - registry-credentials

---
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata: #! ...
spec:
 #! ...
 acme:
   solvers:
     - http01:
         ingress:
           podTemplate:
             spec:
               serviceAccountName: tap-acme-http01-solver
```

  The challenge lies in ensuring that the same `ServiceAccount` is available in every namespace that obtains `Certificates` from the `ClusterIssuer`.

Overview of Cloud Native Runtimes

This topic gives you an overview what Cloud Native Runtimes is and how you can use it with Tanzu Application Platform.

Overview

Cloud Native Runtimes is enterprise supported Knative, with Carvel tools for deployment and Contour for networking. Cloud Native Runtimes offers everything Knative does and some extras that make it ideal for cloud-native application development. Cloud Native Runtimes gives developers environmental simplicity and administrators deployment control. It works on any Kubernetes cluster running Kubernetes v1.26 and later.

The Cloud Native Runtimes component documentation consists of the following sections:

- How-to guides give you steps to solve a specific problem.
- Reference give you specific information, such as the Cloud Native Runtimes Compatibility Matrix.

Cloud Native Runtimes uses Knative Serving to provide:

- Automatic pod scaling
- Traffic splitting by code release version

Kubernetes Developers must know:	Cloud Native Runtimes Developers must know:
Pods	Pods

Kubernetes Developers must know:	Cloud Native Runtimes Developers must know:
Deployment and Rollout Progress	Knative Service
Service (networking model)	*n/a*
Ingress	*n/a*
Labels and selectors	*n/a*

Cloud Native Runtimes increases administrator control and support.

Administrators can:

- Manage infrastructure costs with request driven autoscaling
- Test deployments with traffic splitting by code version
- Use Carvel command tools to simplify deployment
- Receive Enterprise Support when they need it

Cloud Native Runtimes works well with these use cases:

- Batch Jobs Processing
- AI and ML
- Application or Network Monitoring
- IOT
- Serverless application architectures

For more information about the software related to Cloud Native Runtimes, see:

- Knative Documentation Home - Knative
- Carvel Tools Suite Documentation Carvel - Home
- Contour Networking Documentation Contour

Overview of Cloud Native Runtimes

This topic gives you an overview what Cloud Native Runtimes is and how you can use it with Tanzu Application Platform.

Overview

Cloud Native Runtimes is enterprise supported Knative, with Carvel tools for deployment and Contour for networking. Cloud Native Runtimes offers everything Knative does and some extras that make it ideal for cloud-native application development. Cloud Native Runtimes gives developers environmental simplicity and administrators deployment control. It works on any Kubernetes cluster running Kubernetes v1.26 and later.

The Cloud Native Runtimes component documentation consists of the following sections:

- How-to guides give you steps to solve a specific problem.
- Reference give you specific information, such as the Cloud Native Runtimes Compatibility Matrix.

Cloud Native Runtimes uses Knative Serving to provide:

- Automatic pod scaling
- Traffic splitting by code release version

Kubernetes Developers must know:	Cloud Native Runtimes Developers must know:
Pods	Pods
Deployment and Rollout Progress	Knative Service
Service (networking model)	*n/a*
Ingress	*n/a*
Labels and selectors	*n/a*

Cloud Native Runtimes increases administrator control and support.

Administrators can:

- Manage infrastructure costs with request driven autoscaling
- Test deployments with traffic splitting by code version
- Use Carvel command tools to simplify deployment
- Receive Enterprise Support when they need it

Cloud Native Runtimes works well with these use cases:

- Batch Jobs Processing
- AI and ML
- Application or Network Monitoring
- IOT
- Serverless application architectures

For more information about the software related to Cloud Native Runtimes, see:

- Knative Documentation Home - Knative
- Carvel Tools Suite Documentation Carvel - Home
- Contour Networking Documentation Contour

Cloud Native Runtimes how-to guides

This topic lists the how-to guides available for Cloud Native Runtimes, commonly known as CNRs.

- Configure Cloud Native Runtimes with VMware NSX Advanced Load Balancer
- Cloud Native Runtimes for App Operators
- Configuring observability for Cloud Native Runtimes
- Configure garbage collection for the Knative revisions
- Configure an external DNS for Cloud Native Runtimes
- Customizing Cloud Native Runtimes
- Installing Cloud Native Runtimes with your existing Contour installation
- Cloud Native Runtimes AutoTLS how-to guides
- Configuring Cloud Native Runtimes with Tanzu Service Mesh
- Troubleshooting Cloud Native Runtimes
- Use your existing TLS Certificate for Cloud Native Runtimes

Cloud Native Runtimes for app operators

This topic outlines topics related to installing, updating, and uninstalling Cloud Native Runtimes.

Overview

The following topics are related to app operator tasks:

- Install Cloud Native Runtimes on Tanzu Application Platform without a Profile
- Manage Knative Serving Resources
- Uninstall Cloud Native Runtimes
- Upgrade Cloud Native Runtimes
- Verify Cloud Native Runtimes Installation

Install Cloud Native Runtimes

This topic describes how you can install Cloud Native Runtimes, commonly known as CNRs, from the Tanzu Application Platform package repository.

Note

Use the instructions in this topic if you do not want to use a profile to install packages. The full profile includes Cloud Native Runtimes. For more information about profiles, see Installing the Tanzu Application Platform Package and Profiles.

Prerequisites

Before installing Cloud Native Runtimes:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- Contour is installed in the cluster. You can install Contour from the Tanzu Application package repository. If you have have an existing Contour installation, see Installing Cloud Native Runtimes with an Existing Contour Installation.
- By default, Tanzu Application Platform installs and uses a self-signed certificate authority for issuing TLS certificates to components by using ingress issuer. For more information, see Ingress Certificates. To install Cloud Native Runtimes, you must set the `shared.ingress_domain` or `cnrs.domain_name` property when you set `ingress_issuer`. For example:

  ```
  shared:
  ingress_domain: "foo.bar.com"
  ```

  or

  ```
  cnrs:
  domain_name: "foo.bar.com"
  ```

  If the domain name is not available or not what you want, you can set the domain name to any valid value if no process relies on the domain name resolving to the envoy IP. VMware discourages this for production environments. Another alternative to bypass setting domain name is to deactivate auto-TLS. For more information, see Disabling Automatic TLS Certificate Provisioning.

Install

To install Cloud Native Runtimes:

1. List version information for the package by running:

   ```
   tanzu package available list cnrs.tanzu.vmware.com --namespace tap-install
   ```

   For example:

   ```
   tanzu package available list cnrs.tanzu.vmware.com --namespace tap-install

     NAME                    VERSION  RELEASED-AT
     cnrs.tanzu.vmware.com   2.4.0    2023-06-05 19:00:00 -0500 -05
   ```

2. (Optional) Make changes to the default installation settings:

   1. Gather the values schema.

      ```
      tanzu package available get cnrs.tanzu.vmware.com/2.4.0 --values-schema -
      n tap-install
      ```

   2. Create a `cnr-values.yaml` file by using the following example as a guide to configure Cloud Native Runtimes:

      Note

      For most installations, you can leave the `cnr-values.yaml` empty, and use the default values.

      ```
      ---
      # Configures the domain that Knative Services will use
      domain_name: "mydomain.com"
      ```

      Configuration Notes:

      - If you are using a single-node cluster, such as kind, set the `lite.enable: true` option to lower CPU and memory requests for resources. To deactivate pod disruption budgets on Knative Serving, if high availability is not indispensable in your development environment, you can set `pbd.enable` to `false`.
      - Cloud Native Runtimes reuses the existing `tanzu-system-ingress` Contour installation for external and internal access when installed in the `full` profile. To use a separate Contour installation for system-internal traffic, set `cnrs.contour.internal.namespace` to the namespace of your separate Contour installation.
      - If you install Cloud Native Runtimes with the default value of `true` for the `allow_manual_configmap_update` configuration, you can only update some ConfigMaps manually. To update all ConfigMaps using overlays, change this value to `false`.

3. Install the package by running:

   ```
   tanzu package install cloud-native-runtimes \
     --package cnrs.tanzu.vmware.com \
     --version 2.4.0 \
     --namespace tap-install \
   ```

```
  --values-file cnr-values.yaml \
  --poll-timeout 30m
```

For example:

```
tanzu package install cloud-native-runtimes \
  --package cnrs.tanzu.vmware.com \
  --version 2.4.0 \
  --namespace tap-install \
  --values-file cnr-values.yaml \
  --poll-timeout 30m

| Installing package 'cnrs.tanzu.vmware.com'
| Getting package metadata for 'cnrs.tanzu.vmware.com'
| Creating service account 'cloud-native-runtimes-tap-install-sa'
| Creating cluster admin role 'cloud-native-runtimes-tap-install-cluster-role'
| Creating cluster role binding 'cloud-native-runtimes-tap-install-cluster-role
binding'
- Creating package resource
- Package install status: Reconciling

 Added installed package 'cloud-native-runtimes' in namespace 'tap-install'
```

4. Verify the package install by running:

```
tanzu package installed get cloud-native-runtimes -n tap-install
```

For example:

```
tanzu package installed get cloud-native-runtimes -n tap-install

Retrieving installation details for cloud-native-runtimes...
NAME:                    cloud-native-runtimes
PACKAGE-NAME:            cnrs.tanzu.vmware.com
PACKAGE-VERSION:         2.4.0
STATUS:                  Reconcile succeeded
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`.

Verify Your Installation

This topic tells you how to verify your Cloud Native Runtimes, commonly known as Cloud Native Runtimes, installation. You can verify that your Cloud Native Runtimes installation was successful by testing Knative Serving.

Prerequisites

1. Create a namespace and environment variable where you want to create Knative services. Run:

Note

This step covers configuring a namespace to run Knative services. If you rely on a SupplyChain to deploy Knative services into your cluster, skip this step because namespace configuration is covered when developer namespaces are set up to use installed packages. For more information, see Set up developer namespaces to use your installed packages. Otherwise, you must

complete the following steps for each namespace where you create Knative services.

```
export WORKLOAD_NAMESPACE='cnr-demo'
kubectl create namespace ${WORKLOAD_NAMESPACE}
```

2. Configure a namespace to use Cloud Native Runtimes. If you relocated images to another registry during Tanzu Application Platform installation, you must grant service accounts that run Knative services using Cloud Native Runtimes access to the image pull secrets. This includes the `default` service account in a namespace, which is created automatically but not associated with any image pull secrets. Without these credentials, attempts to start a service fail with a timeout and the pods report that they are unable to pull the `queue-proxy` image.

    1. Create an image pull secret in the namespace that Knative services run and fill it from the `tap-registry` secret mentioned in Add the Tanzu Application Platform package repository. Run the following commands to create an empty secret and annotate it as a target of the secretgen controller:

        ```
        kubectl create secret generic pull-secret --from-literal=.dockerconfigjso
        n={} --type=kubernetes.io/dockerconfigjson -n ${WORKLOAD_NAMESPACE}

        kubectl annotate secret pull-secret secretgen.carvel.dev/image-pull-secre
        t="" -n ${WORKLOAD_NAMESPACE}
        ```

    2. After you create a `pull-secret` secret in the same namespace as the service account, run the following command to add the secret to the service account:

        ```
        kubectl patch serviceaccount default -p '{"imagePullSecrets": [{"name":
        "pull-secret"}]}' -n ${WORKLOAD_NAMESPACE}
        ```

    3. Verify that a service account is correctly configured by running:

        ```
        kubectl describe serviceaccount default -n ${WORKLOAD_NAMESPACE}
        ```

        For example:

        ```
        kubectl describe sa default -n cnr-demo
        Name:                default
        Namespace:           cnr-demo
        Labels:              <none>
        Annotations:         <none>
        Image pull secrets:  pull-secret
        Mountable secrets:   default-token-xh6p4
        Tokens:              default-token-xh6p4
        Events:              <none>
        ```

        The service account has access to the `pull-secret` image pull secret.

Verify that `STATUS` is `Reconcile succeeded`.

Verify Knative Serving installation

To verify the installation of Knative Serving

1. Create a namespace and environment variable for the test. Run:

    ```
    export WORKLOAD_NAMESPACE='cnr-demo'
    ```

```
kubectl create namespace ${WORKLOAD_NAMESPACE}
```

2. Test Knative Serving by creating a test service. See Verifying Knative Serving.
3. Delete the namespace that you created for the demo. Run:

```
kubectl delete namespaces ${WORKLOAD_NAMESPACE}
unset WORKLOAD_NAMESPACE
```

Verify Knative Serving for Cloud Native Runtimes

This topic tells you how to verify that Knative Serving was installed for Cloud Native Runtimes, commonly known as CNRs.

Overview of verifying Knative Serving

To verify that Knative Serving was installed, create an example Knative service and test it.

The following procedure shows you how to create an example Knative service using the Cloud Native Runtimes sample app, `hello-yeti`. This sample is custom built for Cloud Native Runtimes and is stored in the VMware Harbor registry.

Note

If you do not have access to the Harbor registry, you can use the Hello World - Go sample app in the Knative documentation.

Prerequisites

Before you verify Knative Serving, you must have a namespace where you want to deploy Knative services. This namespace is referred to as `${WORKLOAD_NAMESPACE}` in this tutorial. See Verifying Your Installation.

Test Knative Serving

To create an example Knative service and use it to test Knative Serving:

1. If you are verifying on Tanzu Mission Control or vSphere 7.0 with Tanzu, then create a role binding in the `${WORKLOAD_NAMESPACE}` namespace. Run:

```
kubectl apply -n "${WORKLOAD_NAMESPACE}" -f - << EOF
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: ${WORKLOAD_NAMESPACE}-psp
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cnr-restricted
subjects:
- kind: Group
  name: system:serviceaccounts:${WORKLOAD_NAMESPACE}
EOF
```

2. Deploy the sample app using the `kn` CLI. Run:

```
kn service create hello-yeti -n ${WORKLOAD_NAMESPACE} \
  --image projects.registry.vmware.com/tanzu_serverless/hello-yeti@sha256:17d64
0edc48776cfc604a14fbabf1b4f88443acc580052eef3a753751ee31652 --env TARGET='hello
-yeti'
```

If you are verifying on Tanzu Mission Control or vSphere 7.0 with Tanzu, add `--user 1001` to the command above to run it as a non-root user.

3. Run one of these commands to retrieve the external address for your ingress, depending on your IaaS:

IaaS	Command
Tanzu Kubernetes Grid on AWS	`export EXTERNAL_ADDRESS=$(kubectl get service envoy -n tanzu-system-ingress -ojsonpath="{.status.loadBalancer.ingress[0].hostname}")`
Tanzu Mission Control on AWS	`export EXTERNAL_ADDRESS=$(kubectl get service envoy -n tanzu-system-ingress -ojsonpath="{.status.loadBalancer.ingress[0].hostname}")`
Amazon Elastic Kubernetes Service	`export EXTERNAL_ADDRESS=$(kubectl get service envoy -n tanzu-system-ingress -ojsonpath="{.status.loadBalancer.ingress[0].hostname}")`
vSphere 7.0 on Tanzu	`export EXTERNAL_ADDRESS=$(kubectl get service envoy -n tanzu-system-ingress -ojsonpath="{.status.loadBalancer.ingress[0].ip}")`
Tanzu Kubernetes Grid on vSphere, Azure, or Google Cloud Platform (GCP)	`export EXTERNAL_ADDRESS=$(kubectl get service envoy -n tanzu-system-ingress -ojsonpath="{.status.loadBalancer.ingress[0].ip}")`
Tanzu Kubernetes Grid Integrated Edition	`export EXTERNAL_ADDRESS=$(kubectl get service envoy -n tanzu-system-ingress -ojsonpath="{.status.loadBalancer.ingress[0].ip}")`
Tanzu Mission Control on vSphere	`export EXTERNAL_ADDRESS=$(kubectl get service envoy -n tanzu-system-ingress -ojsonpath="{.status.loadBalancer.ingress[0].ip}")`
Azure Kubernetes Service	`export EXTERNAL_ADDRESS=$(kubectl get service envoy -n tanzu-system-ingress -ojsonpath="{.status.loadBalancer.ingress[0].ip}")`
Google Kubernetes Engine	`export EXTERNAL_ADDRESS=$(kubectl get service envoy -n tanzu-system-ingress -ojsonpath="{.status.loadBalancer.ingress[0].ip}")`
Docker desktop	`export EXTERNAL_ADDRESS='localhost:8080'` And set up port-forwarding in a separate terminal: `kubectl -n tanzu-system-ingress port-forward svc/envoy 8080:80`

4. Connect to the app. Verify the URL for the Knative service.

   Run:

```
kn service list -n ${WORKLOAD_NAMESPACE}
```

   The result is something like this:

```
NAME           URL                                                        LAT
EST               AGE   CONDITIONS   READY   REASON
hello-yeti     https://hello-yeti.${WORKLOAD_NAMESPACE}.svc.cluster.local   hel
lo-yeti-00001     6s    3 OK / 3     True
```

Now, take the host name from the URL and set it in an environment variable `KSERVICE_HOSTNAME` like so:

```
export KSERVICE_HOSTNAME="hello-yeti.${WORKLOAD_NAMESPACE}.svc.cluster.local"
```

Then, connect to the app:

```
curl https://${KSERVICE_HOSTNAME} -k --resolve ${KSERVICE_HOSTNAME}:443:${EXTER
NAL_ADDRESS}
```

Note

If you configured DNS locally by using `/etc/hosts` or externally, the `--resolve` flag is omitted, or you can use a web browser.

On success, you see a reply from our mascot, Carl the Yeti.

Delete the Example Knative Service

After verifying your serving installation, delete the example Knative service and unset the environment variable:

1. Run:

   ```
   kn service delete hello-yeti -n ${WORKLOAD_NAMESPACE}
   unset EXTERNAL_ADDRESS
   unset KSERVICE_HOSTNAME
   ```

2. If you created port forwarding in step 3, terminate that process.

Knative Serving resource management

This topic tells you how to configure the memory and CPU allocation of resources in the `knative-serving` namespace.

Overview

By default, Knative deployments are allocated a predefined amount of CPU and memory. These default allocations support general use cases, but you might adjust the allocations for the following reasons:

- **Performance Optimization**: Customize resource allocations to help fine-tune the load for specific workloads to improve response times and throughput.
- **Cost Efficiency**: Customize resources to avoid over-provisioning or under-provisioning and verify efficient resource use.
- **Improved Stability**: Control resource allocation to prevent any deployment from consuming excessive resources and increase the stability of the entire cluster.

You can tailor the memory and CPU of Knative system controllers by using the Cloud Native Runtimes `resource_management` config option.

Configuring memory and cpu requests and limits for Knative Serving resources

To configure the memory and CPU allocation for the deployments in the `knative-serving` namespace, you must specify the `resource_management` config option on Cloud Native Runtimes. You can only configure the following deployments:

- `activator`
- `autoscaler`
- `autoscaler-hpa`
- `controller`
- `net-certmanager-controller`
- `net-certmanager-webhook`
- `net-contour-controller`
- `webhook`

Example: updating the activator deployment

The following example shows how to adjust the CPU and memory requests and limits by using the `resource_management` option of the Knative `activator` deployment. By default, the `activator` deployment has the following resources:

```
resources:
    requests:
        cpu: 300m
        memory: 60Mi
    limits:
        cpu: 1000m
        memory: 600Mi
```

Specify the following resources to the activator pod:

```
cnrs:
  resource_management:
    - name: NAME
      requests:
        memory: MEMORY
        cpu: CPU
      limits:
        memory: MEMORY
        cpu: CPU
```

Where:

- `NAME` represents the name of the deployment you want to configure.
- `requests` specifies the amount of resources that are allocated to the pods of this deployment:
    - `MEMORY` is how many megabytes of memory you want to request.
    - `CPU` is how many CPU cores, 100 millicpu units, you want to request.
- `limits` sets the maximum amount of resources that the pods can use:
    - `MEMORY` is the limit of how many megabytes of memory that the pods can use.
    - `CPU` is the limit of how many CPU cores, 100 millicpu units, pods can use.

Note

> You can provide any of the resource units supported by Kubernetes as explained in the Kubernetes documentation.

After updating Cloud Native Runtimes, get the deployment information to confirm the `activator` deployment updated:

```
kubectl get deployment activator -n knative-serving -o=jsonpath='{.spec.template.spec.
containers[?(@.name=="activator")].resources}'

{"limits":{"cpu":"1","memory":"1000Mi"},"requests":{"cpu":"100m","memory":"100Mi"}}
```

All fields, such as the request and limits of CPUs, and the memory's request and limits, are optional, except for the deployment's name, causing it to use the default values specified for these deployments.

For example, to change only the limit and the memory's request of the Knative `controller`, you can provide the following configuration to `resource_management`:

```
cnrs:
  resource_management:
    - name: "controller"
      requests:
        memory: "130M"
      limits:
        cpu: "0.5"
```

Upgrading Cloud Native Runtimes

This topic tells you how to upgrade Cloud Native Runtimes for Tanzu to the latest version.

Overview

New versions of Cloud Native Runtimes are available from the Tanzu Application Platform package repository, and you can upgrade as part of upgrading Tanzu Application Platform.

Prerequisites

The following prerequisites are required to upgrade Cloud Native Runtimes:

- An updated Tanzu Application Platform package repository with the version of Cloud Native Runtimes you want to upgrade to. For more information, see Upgrade your Tanzu Application Platform.

Upgrade Cloud Native Runtimes

To upgrade the Cloud Native Runtimes PackageInstall specifically, run:

```
tanzu package installed update cloud-native-runtimes \
  --package cnrs.tanzu.vmware.com \
  --version CNR-VERSION \
  --namespace tap-install \
  --values-file cnr-values.yaml
```

Where `CNR-VERSION` is the latest version of Cloud Native Runtimes available as part of the new Tanzu Application Platform package repository.

Uninstall Cloud Native Runtimes

This topic tells you how to uninstall Cloud Native Runtimes.

Overview

Cloud Native Runtimes is part of the Tanzu Application Platform package repository. For information about uninstalling the entire Tanzu Application Platform package repository, see Uninstall your Tanzu Application Platform by using Tanzu CLI.

Uninstall

To uninstall Cloud Native Runtimes:

1. Delete the installed package:

```
tanzu package installed delete cloud-native-runtimes --namespace tap-install
```

Cloud Native Runtimes AutoTLS how-to guides

This topic lists how-to guides related to AutoTLS.

Overview

This topic describes the scenarios related to autoTLS:

- Setting up AutoTLS
- Deactivate autoTLS
- Deactivate HTTP-to-HTTPS redirection
- Configure Custom Issuer or ClusterIssuer
- Configure wildcard certificates

Securing your web workloads in Cloud Native Runtimes

This topic gives you an overview of securing HTTP connections using TLS certificates in Cloud Native Runtimes, commonly known as CNRs, for VMware Tanzu Application Platform and helps you configure Transport Layer Security (TLS).

Prerequisites

Ensure that you have the Tanzu Application Platform, Cloud Native Runtimes for VMware Tanzu, Contour, and cert-manager installed.

Overview of Cloud Native Runtimes TLS configurations

This section describes default configuration, custom configuration, obtaining, and renewing TLS certificates with Cloud Native Runtimes.

Default TLS configuration in Cloud Native Runtimes

When installing Tanzu Application Platform by using profiles, the cert-manager package is used to facilitate the acquisition, management, and renewal of TLS certificates.

Cloud Native Runtimes automatically acquires TLS certificates for workloads through the shared ingress issuer integrated into the Tanzu Application Platform. The `shared.ingress_issuer` configuration value in Tanzu Application Platform specifies the ingress issuer and it refers to a `cert-manager.io/v1/ClusterIssuer`.

By default, the ingress issuer is self-signed and has limits. For more information about the shared ingress issuer, see the following Tanzu Application Platform documentation:

- Secure exposed ingress endpoints in Tanzu Application Platform

The following TLS features are in Cloud Native Runtimes by default:

- **Auto-TLS**

  Cloud Native Runtimes has the Auto-TLS feature enabled by default. It uses the cert-manager package to automate the process of certificate issuance and management. Auto-TLS takes care of requesting, renewing, and configuring TLS certificates for each domain that you configure in your Cloud Native Runtimes settings.

- **Automatic HTTPS Redirection**

  By default, Cloud Native Runtimes automatically redirects HTTP traffic to HTTPS for secured services. This ensures that all communication with your applications is encrypted and providing a secure experience for your users.

- **One certificate per hostname**

  Cloud Native Runtimes issues a unique certificate for each host name associated with a Knative Service.

Custom TLS configuration in Cloud Native Runtimes

While the default ingress issuer is suitable for testing and evaluation purposes, VMware recommends replacing it with your own issuer for production environments.

There are a few ways to customize TLS configuration in Cloud Native Runtimes:

Replace the shared ingress issuer at the Tanzu Application Platform's level

You have the flexibility to replace Tanzu Application Platform's default ingress issuer with any other `certificate authority` that is compliant with cert-manager ClusterIssuer. For information about how to replace the default ingress issuer, see Replacing the default ingress issuer.

Cloud Native Runtimes uses the issuer specified by the `shared.ingress_issuer` configuration value to issue certificates for your workload automatically.

Designate another ingress issuer for your workloads in Cloud Native Runtimes only

You can have a shared ingress issuer at the Tanzu Application Platform's level and designate another issuer used by Cloud Native Runtimes to issue TLS certificates for your workloads. This allows you to customize TLS settings for Cloud Native Runtimes while maintaining a global configuration for other components.

You can designate an ingress issuer for Cloud Native Runtimes by specifying the `cnrs.ingress_issuer` configuration value. The ingress or TLS configuration for Cloud Native Runtimes takes precedence over the shared ingress issuer.

For information about designating another ingress issuer for your workloads, see Configure Cloud Native Runtimes to use a custom Issuer or ClusterIssuer for details.

Provide an existing TLS certificate for your workloads in Cloud Native Runtimes

If you manually generated a TLS certificate and want to provide it to Cloud Native Runtimes instead of using an ingress issuer, you can follow the instructions in Use your existing TLS Certificate for Cloud Native Runtimes.

Resources on custom TLS configuration for Cloud Native Runtimes

The following resources are helpful for custom TLS configuration for Cloud Native Runtimes:

- Configure Cloud Native Runtimes to use a custom Issuer or ClusterIssuer
- Use wildcard certificates with Cloud Native Runtimes
- Use your existing TLS Certificate for Cloud Native Runtimes
- Deactivate HTTP-to-HTTPS redirection
- Opt out from any ingress issuer and deactivate automatic TLS feature

Opt out from an ingress issuer and deactivate automatic TLS feature

This topic tells you how to opt out from an ingress issuer and deactivate the automatic TLS feature for Cloud Native Runtimes, commonly known as CNRs.

Deactivate TLS

You can deactivate automatic TLS certificate provisioning in Cloud Native Runtimes by setting the `ingress_issuer` property to an empty string as follows:

```
cnrs:
    ingress_issuer: ""
```

Update your Tanzu Application Platform installation accordingly after following the step mentioned earlier.

```
tanzu package installed update tap -p tap.tanzu.vmware.com -v ${TAP-VERSION} --values-
file tap-values.yaml -n tap-install
```

Where `TAP-VERSION` is the version of Tanzu Application Platform you want to update to.

Deactivate HTTP-to-HTTPS redirection

This topic tells you how to deactivate HTTP-to-HTTPS redirection with Cloud Native Runtimes.

Overview

When you designate an ingress issuer for your workloads by setting either the `shared.ingress_issuer` or `cnrs.ingress_issuer` configuration value, in your `tap-values.yaml` file, the auto-TLS feature is enabled in Cloud Native Runtimes. When the auto-TLS is enabled, Cloud Native Runtimes automatically redirects traffic from HTTP to HTTPS. However, there can be situations where you want to opt out of this behavior and continue serving content over HTTP. To do this, you must deactivate the HTTPS redirection feature.

To deactivate HTTP-to-HTTPS redirection in Cloud Native Runtimes, you must edit your configuration values file:

1. Configure Cloud Native Runtimes to deactivate https redirection:

```
cnrs:
   https_redirection: false
```

2. Update your Tanzu Application Platform installation:

```
tanzu package installed update tap -p tap.tanzu.vmware.com -v ${TAP_VERSION} --
values-file tap-values.yaml -n tap-install
```

3. Verify that the HTTP-to-HTTPS redirection is deactivated by accessing your workload using the HTTP protocol. You can access the service without being redirected to HTTPS. Use a web browser or a tool like `curl` to test the behavior:

```
curl -I http://your-workload.your-domain.com
```

   The response shows an HTTP status code without redirection to HTTPS.

Configure Cloud Native Runtimes to use a custom Issuer or ClusterIssuer

This topic tells you how to configure Cloud Native Runtimes to use a custom Issuer or ClusterIssuer.

Overview

The ability to opt out of the shared ingress issuer and use a custom Issuer or ClusterIssuer for Cloud Native Runtimes provides greater flexibility, security, isolation, and integration with existing infrastructure, allowing you to tailor the TLS configurations to your specific needs and requirements.

The following example explains how to opt out of the shared ingress issuer and use Let's Encrypt with the HTTP01 challenge type. The HTTP01 challenge requires that your load balancer is reachable from the Internet by using HTTP. With the HTTP01 challenge type, a certificate is provisioned for each service.

Configure a custom issuer

You have the flexibility to replace Tanzu Application Platform's default ingress issuer with any other `certificate authority` that is compliant with cert-manager ClusterIssuer. For more information, see the cert-manager documentation. For information about how to replace the default ingress issuer, see Replacing the default ingress issuer.

To configure Cloud Native Runtimes to use a custom Issuer or ClusterIssuer with the HTTP01 challenge:

1. Create a custom Issuer or ClusterIssuer with the Certificate Authority (CA) that you want and configurations. Here's an example YAML configuration for a custom ClusterIssuer using Let's Encrypt with the HTTP01 challenge:

```
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-http01-issuer
spec:
  acme:
     email: YOUR-EMAIL
     server: https://acme-v02.api.letsencrypt.org/directory
     privateKeySecretRef:
```

```
      name: letsencrypt-http01-issuer-account-key
    solvers:
      - http01:
          ingress:
            class: contour
```

Where `YOUR-EMAIL` is your email address.

Specify the ingress class you are using in your Tanzu Application Platform cluster, which is `contour`.

2. Save the configuration from the previous step in a file called `issuer-letsencrypt-http01.yaml`.

Note

To test this feature, you can set `spec.acme.server` to `https://acme-staging-v02.api.letsencrypt.org/directory`. This is the staging URL, which generates self-signed certificates. It is useful for testing without worrying about hitting quotas for your actual domain.

3. Apply the Issuer or ClusterIssuer configuration to your cluster:

```
kubectl apply -f issuer-letsencrypt-http01.yaml
```

Configure Cloud Native Runtimes to use a custom issuer

To configure Cloud Native Runtimes to use a custom issuer:

1. Configure Cloud Native Runtimes to use a custom Issuer or ClusterIssuer for issuing certificates by updating your `tap-values.yaml` file with the following snippet of YAML.

```
cnrs:
   ingress_issuer: "letsencrypt-http01-issuer"
```

2. Update Tanzu Application Platform.

   To update the Tanzu Application Platform installation with the changes to the values file, run:

```
tanzu package installed update tap -p tap.tanzu.vmware.com -v ${TAP_VERSION} --
values-file tap-values.yaml -n tap-install
```

Verify the issuance of certificates

To verify your certificate:

1. Verify that your ClusterIssuer was created and properly issuing certificates, by running:

```
kubectl get clusterissuer letsencrypt-http01-issuer
```

2. Confirm the status of the certificate by running the following command. You see the certificate in a `Ready` state.

```
kubectl get certificate -n DEVELOPER-NAMESPACE
```

   You see the certificate in a `Ready` state.

Where `DEVELOPER-NAMESPACE` is the namespace that you want to use.

3. You can access your workload using the domain you specified with `curl` or a web browser and verify that it is using a TLS certificate issued by the custom Issuer or ClusterIssuer.

```
tanzu apps workload get WORKLOAD-NAME --namespace DEVELOPER-NAMESPACE
kubectl get ksvc WORKLOAD-NAME -n DEVELOPER-NAMESPACE -o jsonpath='{.status.ur
l}'
```

Where:

- `DEVELOPER-NAMESPACE` is the namespace that you want to use.
- `WORKLOAD-NAME` is the name of the workload you want to use.

For information about how to troubleshoot failures related to the certificate, see the cert-manager documentation.

Use wildcard certificates with Cloud Native Runtimes

This section describes how to configure, use, and verify wildcard certificates with Cloud Native Runtimes.

Cloud Native Runtimes uses the cert-manager package by default to automate the process of obtaining, managing, and renewing TLS certificates for your services.

cert-manager is a Kubernetes-native component that works with different `certificate authorities` (CAs) like Let's Encrypt and others, to manage certificates within your Kubernetes cluster. As part of Tanzu Application Platform predefined profile installation, cert-manager package is deployed to the cluster.

Configure an issuer for wildcard certificates

Any other `cert-manager.io/v1/ClusterIssuer` can replace Tanzu Application Platform's default ingress issuer. This topic uses Let's Encrypt as an example of how to set up a custom ingress issuer that issues wildcard certificates.

The Let's Encrypt documentation states that the DNS01 challenge is required to validate wildcard domains. This topic provides instructions for configuring Cloud Native Runtimes to use wildcard certificates with the DNS01 challenge only.

To configure an issuer for wildcard certificates:

1. Create a cert-manager custom Issuer or ClusterIssuer for the DNS01 challenge.

   You must create a custom Issuer or ClusterIssuer with the DNS01 solver configured for your specific DNS provider. For more information about supported DNS01 providers, see the cert-manager documentation for instructions for configuring cert-manager for all the supported DNS providers.

   The following example uses Let's Encrypt and Google Cloud DNS:

```
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-dns-wildcard
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    # This will register an issuer with LetsEncrypt.
    email: YOUR-EMAIL
    privateKeySecretRef:
```

```
      # Set privateKeySecretRef to any unused secret name.
      name: letsencrypt-dns-wildcard-account-key
    solvers:
    - dns01:
        cloudDNS:
          # Set this to your GCP project id
          project: PROJECT-ID
          # This is the secret used to access the service account
          serviceAccountSecretRef:
            name: cloud-dns-key
            key: key.json
```

Where:

- `YOUR-EMAIL` is the email associated with your DNS provider.
- `PROJECT-ID` is the ID of the GCP project.

2. Save the configuration from the previous step in a file called `issuer-wildcard.yaml`.

   When using cert-manager to obtain wildcard certificates, you typically must provide credentials, especially when using the DNS01 challenge. The DNS01 challenge requires cert-manager to create and delete DNS records for domain validation during the certificate issuance process. To perform these actions, cert-manager needs access to your DNS provider's API, which requires authentication using API keys, access tokens, or other credentials. See Supported DNS01 providers in the cert-manager documentation.

   Note

   To test this feature, you can set `spec.acme.server` to `https://acme-staging-v02.api.letsencrypt.org/directory`. This is the staging URL, which generates self-signed certificates. It is useful for testing without worrying about hitting quotas for your actual domain.

3. Apply the file you saved in the previous section to your cluster:

```
kubectl apply -f issuer-wildcard.yaml
```

Configure Cloud Native Runtimes to use wildcard certificates

To use wildcard certificates:

1. Configure Cloud Native Runtimes to use the custom Issuer or ClusterIssuer and indicate in which namespaces to create wildcard certificates.

   Note

   If no value is passed to `cnrs.namespace_selector`, only per service certificates are generated instead of wildcard certificates.

   You achieve this by adding the following YAML to your `tap-values.yaml` file.

```
cnrs:
  ingress_issuer: "letsencrypt-dns-wildcard"
  namespace_selector:
    matchExpressions:
```

```
    - key: apps.tanzu.vmware.com/tap-ns
      operator: Exists
```

This configuration tells Cloud Native Runtimes which custom Issuer or ClusterIssuer is used for issuing the wildcard certificate. When a Knative Service is created or updated with this configuration, Cloud Native Runtimes requests and uses the wildcard certificate from the specified custom Issuer or ClusterIssuer.

In Cloud Native Runtimes, the per-namespace certificate manager operates by using the namespace labels to verify which namespaces require a certificate to be generated. This example specifies that only namespaces labeled with `apps.tanzu.vmware.com/tap-ns` have corresponding wildcard certificates created for them.

If you are using the suggested namespace selector, label your developer namespace with `apps.tanzu.vmware.com/tap-ns`. Run:

```
kubectl label namespace DEV-NAMESPACE "apps.tanzu.vmware.com/tap-ns"
```

To remove a label from a namespace, run:

```
kubectl label namespace DEV-NAMESPACE "apps.tanzu.vmware.com/tap-ns"- --overwri
te
```

2. Update Tanzu Application Platform.

   To update the Tanzu Application Platform installation with the changes to the values file, run:

   ```
   tanzu package installed update tap -p tap.tanzu.vmware.com -v ${TAP_VERSION} --
   values-file tap-values.yaml -n tap-install
   ```

Verify the issuance of wildcard certificates

Verify that your ClusterIssuer was created and properly issuing certificates.

1. To verify your ClusterIssuer, run:

   ```
   kubectl get clusterissuer letsencrypt-dns-wildcard
   ```

2. Confirm the status of the certificate by running the following command. You see the certificate in a `Ready` state.

   ```
   kubectl get certificate -n DEVELOPER-NAMESPACE
   ```

   You see the certificate in a `Ready` state.

   Where `DEVELOPER-NAMESPACE` is the namespace you want to use.

3. You can access your workload using the domain you specified with `curl` or a web browser, and verify that it is using a TLS certificate issued by the custom Issuer or ClusterIssuer.

   ```
   tanzu apps workload get WORKLOAD-NAME --namespace DEVELOPER-NAMESPACE
   kubectl get ksvc WORKLOAD-NAME -n DEVELOPER-NAMESPACE -o jsonpath='{.status.ur
   l}'
   ```

   Where:

   - `DEVELOPER-NAMESPACE` is the namespace you want to use.
   - `WORKLOAD-NAME` is the name of the workload you want to use.

For details about how to troubleshoot failures related to the certificate, see the cert-manager documentation.

Configure garbage collection for the Knative revisions

This topic tells you about current Garbage collection defaults for Knative Revisions and you can update them in Tanzu Application Platform.

Overview

You can configure garbage collection with Knative. For more information, see the Knative documentation.

The current configuration sets the following defaults to mark Knative revisions for garbage collection:

```
  * `retain-since-create-time: "48h"`: Any revision created with an age of 2 days is c
onsidered for garbage collection.
  * `retain-since-last-active-time: "15h"`: Revision that was last active at least 15
hours ago is considered for garbage collection.
  * `min-non-active-revisions: "2"`: The minimum number of inactive Revisions to retai
n.
  * `max-non-active-revisions: "5"`: The maximum number of inactive Revisions to retai
n.
```

Update default values for Knative Garbage Collection

To update default values for Knative Garbage Collection:

1. Create an overlay with the following contents:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
   name: cnr-overlay-gc-cm
   namespace: tap-install #! namespace where tap is installed
   stringData:
   overlay-gc-cm.yaml: |
     #@ load("@ytt:overlay", "overlay")
     #@overlay/match by=overlay.subset({"kind":"ConfigMap","metadata":{"name":"con
   fig-gc"}})
     ---
     data:
       #@overlay/match missing_ok=True
       retain-since-create-time: "48h"
       #@overlay/match missing_ok=True
       retain-since-last-active-time: "15h"
       #@overlay/match missing_ok=True
       min-non-active-revisions: "2"
       #@overlay/match missing_ok=True
       max-non-active-revisions: "5"
   ```

2. Update Tanzu Application Platform values.

   ```
   package_overlays:
   - name: cnrs
     secrets:
       - name: cnr-overlay-gc-cm
   ```

3. Update Tanzu Application Platform Package.

```
tanzu package installed update tap -p tap.tanzu.vmware.com --values-file tap-va
lues.YAML  -n tap-install
```

4. Verify that the overlay was applied by running following commands:

```
kubectl get configmap config-gc --namespace knative-serving --output jsonpath="
{.data.retain-since-create-time}"
kubectl get configmap config-gc --namespace knative-serving --output jsonpath="
{.data.retain-since-last-active-time}"
kubectl get configmap config-gc --namespace knative-serving --output jsonpath="
{.data.min-non-active-revisions}"
kubectl get configmap config-gc --namespace knative-serving --output jsonpath="
{.data.max-non-active-revisions}"
```

Configure Cloud Native Runtimes with VMware NSX Advanced Load Balancer

This topic tells you how to configure Cloud Native Runtimes, commonly known as CNRs, with VMware NSX Advanced Load Balancer, formerly known as Avi Networks.

Overview

You can configure Cloud Native Runtimes to integrate with VMware NSX Advanced Load Balancer. VMware NSX Advanced Load Balancer is a multi-cloud platform that delivers features such as load balancing, security, and container ingress services.

The NSX Advanced Load Balancer Controller provides a control plane while the NSX Advanced Load Balance Service Engines provide a data plane. The Service Engines forward incoming traffic to your Kubernetes cluster's Envoy pods, which Contour creates and manages.

For information about VMware NSX Advanced Load Balancer, see VMware NSX Advanced Load Balancer Documentation.

Prerequisites

To integrate VMware NSX Advanced Load Balancer with Cloud Native Runtimes, you must first install Cloud Native Runtimes:

- If you have not already installed Cloud Native Runtimes, see Installing Cloud Native Runtimes.
- If you already have a Contour installation on your cluster, see Installing Cloud Native Runtimes with an Existing Contour Installation.

Integrate VMware NSX Advanced Load Balancer with Cloud Native Runtimes

To configure Cloud Native Runtimes with VMware NSX Advanced Load Balancer:

1. Deploy the NSX Advanced Load Balancer Controller on a supported infrastructure provider. For a list of NSX Advanced Load Balancer supported providers, see Installation Guides.
2. Deploy the Avi Kubernetes operator to your Kubernetes cluster where Cloud Native Runtimes is hosted. For more information, see the Avi Kubernetes Operator documentation.
3. Connect to a test app and verify that it is reachable. Run:

```
"curl -H KNATIVE-SERVICE-DOMAIN" ENVOY-IP
```

Where:

- `KNATIVE-SERVICE-DOMAIN` is the name of your domain.
- `ENVOY-IP` is the IP address of your Envoy instance.

For information about deploying a sample application and connecting to the application, see Test Knative Serving.

4. (Optional) Create a DNS record that configures your KService URL to point to the Avi Service Engines, and resolve to the external IP of the Envoy. You can create a DNS record on any platform that supports DNS services. For more information, see the documentation for your DNS service platform.

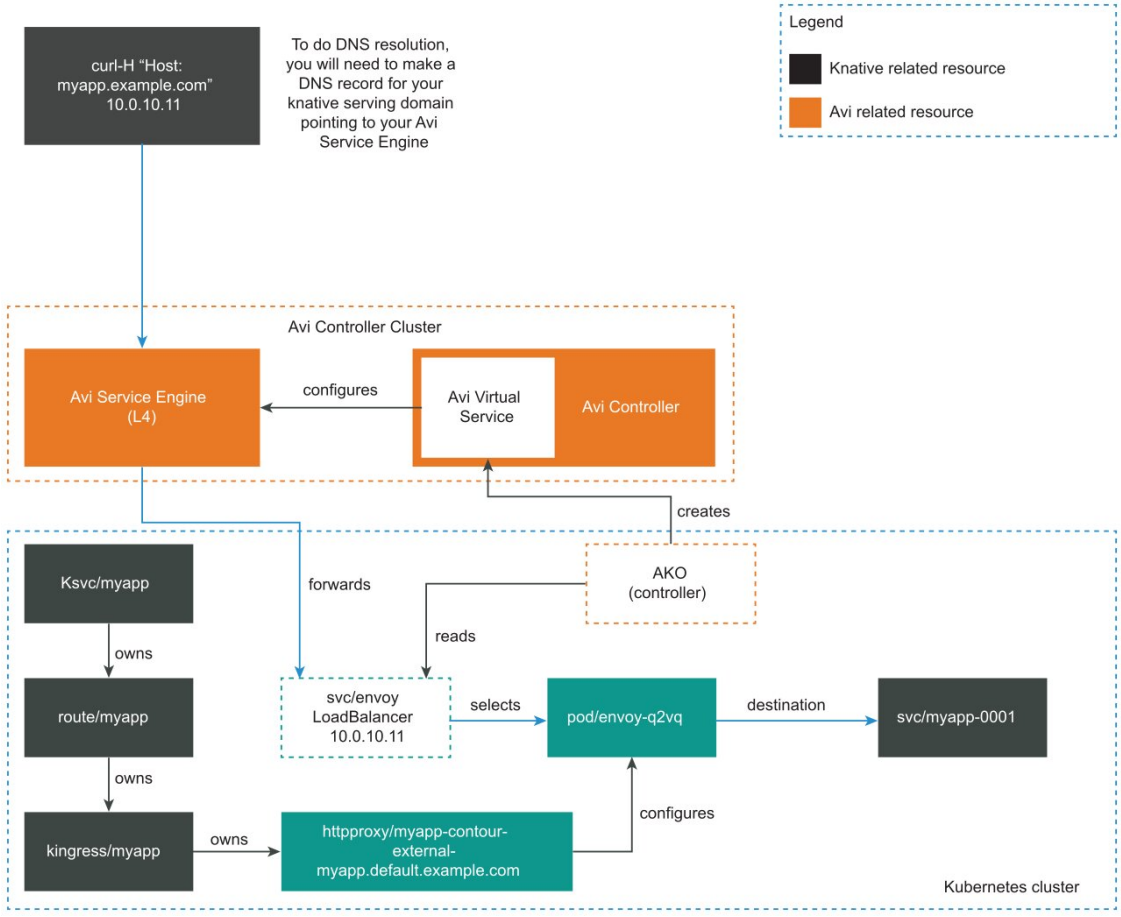
   To get the KService URL, run:

   ```
   kn route describe APP-NAME | grep "URL"
   ```

   To get Envoy's external IP, follow step 3 in Test Knative Serving.

About Routing with VMware NSX Advanced Load Balancer and Cloud Native Runtimes

The following diagram shows how VMware NSX Advanced Load Balancer integrates with Cloud Native Runtimes:



When Contour creates a Kubernetes LoadBalancer service for Envoy, the Avi Kubernetes operator (AKO) sees the new LoadBalancer service. Then NSX Advanced Load Balancer Controller creates a

Virtual Service. For information about LoadBalancer services, see Type LoadBalancer in the Kubernetes documentation.

For each Envoy service, NSX Advanced Load Balancer Controller creates a corresponding Virtual Service. See Virtual Services in the VMware NSX Advanced Load Balancer documentation.

After NSX Advanced Load Balancer Controller creates a Virtual Service, the Controller configures the Service Engines to forward traffic to the Envoy pods. The Envoy pods route traffic based on incoming requests, including traffic splitting and path based routing.

The NSX Advanced Load Balancer Controller provides Envoy with an external IP address so that apps are reachable by the app developer.

Note

VMware NSX Advanced Load Balancer does not interact directly with any Cloud Native Runtimes resources. VMware NSX Advanced Load Balancer forwards all incoming traffic to Envoy.

Installing Cloud Native Runtimes with your existing Contour installation

This topic describes how you can configure Cloud Native Runtimes, commonly known as CNRs, with your existing Contour instance. Cloud Native Runtimes uses Contour to manage internal and external access to the services in a cluster.

About using Contour with Cloud Native Runtimes

Follow the instructions in this topic if:

- You installed Contour as part of Tanzu Application Platform and want to configure Cloud Native Runtimes to use it.
- You see an error about `an existing Contour installation` when you install the Cloud Native Runtimes package.

Cloud Native Runtimes needs two instances of Contour:

- An instance for exposing services outside the cluster
- An instance for services that are private in your network

If installed as part of a Tanzu Application Platform profile, by default Cloud Native Runtimes use the Contour instance installed in the namespace `tanzu-system-ingress` for both internal and external traffic.

If you already use a Contour instance to route requests from clients outside and inside the cluster, you can use your own Contour if it matches the Contour version used by Tanzu Application Platform.

You can use the same single instance of Contour for both internal and external traffic. However, this causes Contour to handle internal and external traffic the same way. For example, if the Contour instance is configured to be accessible from clients outside the cluster, then any internal traffic is also accessible from outside the cluster. Tanzu Application Platform only supports using a single Contour instance for both internal and external traffic.

In all of these cases, Cloud Native Runtimes uses the Tanzu Application Platform's Contour `CustomResourceDefinition`s.

Prerequisites

The following prerequisites are required to configure Cloud Native Runtimes with an existing Contour installation:

- Contour version v1.25.2, the Contour version that is installed as part of Tanzu Application Platform. To identify your cluster's Contour version, see Identify Your Contour Version.
- Contour `CustomResourceDefinition`s versions:

Resource Name	Version
`contourdeployments.projectcontour.io`	v1alpha1
`contourconfigurations.projectcontour.io`	v1alpha1
`extensionservices.projectcontour.io`	v1alpha1
`httpproxies.projectcontour.io`	v1
`tlscertificatedelegations.projectcontour.io`	v1

Identify your Contour version

To identify your cluster's Contour version, run:

```
export CONTOUR_NAMESPACE=CONTOUR-NAMESPACE
export CONTOUR_DEPLOYMENT=$(kubectl get deployment --namespace $CONTOUR_NAMESPACE --ou
tput name)
kubectl get $CONTOUR_DEPLOYMENT --namespace $CONTOUR_NAMESPACE --output jsonpath="{.sp
ec.template.spec.containers[].image}"
kubectl get crds extensionservices.projectcontour.io --output jsonpath="{.status.store
dVersions}"
kubectl get crds httpproxies.projectcontour.io --output jsonpath="{.status.storedVersi
ons}"
kubectl get crds tlscertificatedelegations.projectcontour.io --output jsonpath="{.stat
us.storedVersions}"
```

Where `CONTOUR-NAMESPACE` is the namespace where Contour is installed on your Kubernetes cluster.

Install Cloud Native Runtimes on a cluster with your existing Contour instance

To install Cloud Native Runtimes on a cluster with an existing Contour instance, you can add values to your `cnr-values.yaml` file so that Cloud Native Runtimes uses your Contour instance.

An example of a `cnr-values.yaml` file where Cloud Native Runtimes uses the Contour version in a different namespace:

```
---
contour:
  external:
    namespace: "tanzu-system-ingress"
  internal:
    namespace: "tanzu-system-ingress"
```

Note

If your Contour instance is removed or configured incorrectly, apps running on Cloud Native Runtimes lose connectivity.

Customizing Cloud Native Runtimes

This topic tells you how to customize your Cloud Native Runtimes installation.

Overview

There are many package configuration options exposed through data values that allows you to customize your Cloud Native Runtimes installation.

To yield all the configuration options available in a Cloud Native Runtimes package version:

```
export CNR_VERSION=2.4.0
tanzu package available get cnrs.tanzu.vmware.com/${CNR_VERSION} --values-schema -n ta
p-install
```

Customizing Cloud Native Runtimes

Besides using the out-of-the-box options to configure your package, you can use ytt overlays to further customize your installation. For information about how to customize any Tanzu Application Platform package, see Customize your package installation.

The following example shows how to update the Knative ConfigMap `config-logging` to override the logging level of the Knative Serving controller to `debug`:

1. Create a Kubernetes secret containing the ytt overlay by applying the following configuration to your cluster.

   ```
   kubectl apply -n tap-install -f - << EOF
   apiVersion: v1
   kind: Secret
   metadata:
    name: cnrs-patch
   stringData:
    patch.yaml: |
      #@ load("@ytt:overlay", "overlay")
      #@overlay/match by=overlay.subset({"kind":"ConfigMap","metadata":{"name":"co
   nfig-logging","namespace":"knative-serving"}})
      ---
      data:
        #@overlay/match missing_ok=True
        loglevel.controller: "debug"
   EOF
   ```

   To learn more about the Carvel tool `ytt` and how to write overlays, see the ytt documentation.

2. Update your `tap-values.yaml` file to add the YAML below. This YAML informs the Tanzu Application Platform about the secret name where the overlay is stored and applies the overlay to the `cnrs` package.

   ```
   package_overlays:
   - name: cnrs
     secrets:
     - name: cnrs-patch
   ```

   You can retrieve your `tap-values.yaml` file by running:

   ```
   kubectl get secret tap-tap-install-values -n tap-install -ojsonpath="{.data.tap
   -values\.yaml}" | base64 -d
   ```

3. Update the Tanzu Application Platform installation.

```
tanzu package installed update tap -p tap.tanzu.vmware.com -v ${TAP_VERSION} --
values-file tap-values.yaml -n tap-install
```

4. Confirm your changes were applied to the corresponding ConfigMap.

   To confirm that your changes were applied to the ConfigMap `config-logging` by ensuring `loglevel.controller` is set to `debug`.

```
kubectl get configmap config-logging -n knative-serving -oyaml
```

Configure your external DNS with Cloud Native Runtimes

This topic tells you how to configure your external DNS with Cloud Native Runtimes, commonly known as CNRs.

Overview

Knative uses `svc.cluster.local` as the default domain.

Note

If you are setting up Cloud Native Runtimes for development or testing, you do not have to set up an external DNS. However, to access your workloads over the Internet, you must set up a custom domain and an external DNS.

Configure custom domain

To set up the custom domain and its external DNS record:

1. Configure your custom domain:

   When your workloads are created, Knative automatically creates URLs for each workload based on the configuration in the domain ConfigMap.

   - To set a default custom domain, edit your `cnr-values.yaml` file to contain the following:

```
---
domain_name: "mydomain.com"
```

     This edits the Knative domain ConfigMap to use `domain_name` as the default domain.

     Note

     `domain_name` must be a valid DNS subdomain.

   - **Advanced** To overwrite the domain ConfigMap entirely, edit your `cnr-values.yaml` file to contain the config-domain options that you want, similar to the following:

```
---
domain_config: |
 ---
 mydomain.com: |
```

```
mydomain.org: |
   selector:
      app: nonprofit
```

This replaces the body of the Knative domain ConfigMap with `domain_config`. This allows you to configure multiple custom domains, and configure a custom domain for a service depending on its labels.

See Changing the default domain for more information about the structure of the domain ConfigMap.

Note

`domain_config` must be valid YAML and a valid domain ConfigMap.

Note

You can only use one of `domain_config` or `domain_name` at a time. You can not use both.

2. Get the address of the cluster load balancer:

```
kubectl get service envoy -n EXTERNAL-CONTOUR-NS --output 'jsonpath={.status.lo
adBalancer.ingress}'
```

Where `EXTERNAL-CONTOUR-NS` is the namespace where a Contour serving external traffic is installed. If Cloud Native Runtimes was installed as part of a Tanzu Application Profile, this value is likely `tanzu-system-ingress`.

If this command returns a URL instead of an IP address, `ping` the URL to get the load balancer IP address.

3. Create a wildcard DNS `A` record that assigns the custom domain to the load balancer IP. Follow the instructions provided by your domain name registrar for creating records.

   The following is an example of the record you create:

```
*.DOMAIN IN A TTL LOADBALANCER-IP
```

   Where:

   - `DOMAIN` is the custom domain.
   - `TTL` is the time-to-live.
   - `LOADBALANCER-IP` is the load balancer IP.

   For example:

```
*.mydomain.com IN A 3600 198.51.100.6
```

   If you chose to configure multiple custom domains, you must create a wildcard DNS record for each domain.

Configure Knative service domain template

Knative uses domain template which specifies the golang text template string to use when constructing the Knative service's DNS name. The default value is `{{.Name}}.{{.Namespace}}.`

`{{.Domain}}`. Valid variables defined in the template include Name, Namespace, Domain, Labels, and Annotations.

To configure domain template for the created Knative Services, add the following to your `cnr-values.yaml` file:

```
---
domain_template: "{{.Name}}-{{.Namespace}}.{{.Domain}}"
```

This edits the Knative `domain-template` ConfigMap to use `domain_template` as the default domain template.

Changing this value might be necessary when the extra levels in the domain name generated are problematic for wildcard certificates that only support a single level of domain name added to the certificate's domain. In those cases you might consider using a value of `{{.Name}}-{{.Namespace}}.{{.Domain}}`, or removing the Namespace entirely from the template.

When choosing a new value, be thoughtful of the potential for conflicts, such as when users the use of characters like `-` in their service or namespace names.

You can use `{{.Annotations}}` or `{{.Labels}}` for any customization in the go template if needed.

VMware recommends to keeping namespace in the template to avoid domain name clashes. For example, `{{.Name}}-{{.Namespace}}.{{ index .Annotations "sub"}}.{{.Domain}}` and you have an annotation `{"sub":"foo"}`, the generated template is `{Name}-{Namespace}.foo.{Domain}`.

Use your existing TLS Certificate for Cloud Native Runtimes

This topic tells you how to use your existing TLS Certificate for Cloud Native Runtimes, commonly known as CNRs.

Overview

Configure secure HTTPS connections to enable your web workloads and routes to stop external TLS connections using an existing certificate.

You have the flexibility to provide your own TLS certificate to Cloud Native Runtimes *instead of* relying on the shared ingress issuer for your Knative workloads. To use the feature explained in this topic, you must configure Cloud Native Runtimes to bypass the cert-manager certificate issuer. For example, if you have set `cnrs.contour.default_tls_secret` in your `tap-values.yaml`, set the `cnrs.ingress_issuer` configuration to an empty value. For detailed instructions on how to opt out and deactivate the automatic TLS feature, see Opt out from any ingress issuer and deactivate automatic TLS feature.

Prerequisites

To configure TLS for Cloud Native Runtimes, you must first configure a Service Domain. For more information, see Configuring External DNS with Cloud Native Runtimes.

Configure TLS

To configure your TLS certificate for the created Knative Services, follow the steps:

1. Create a Kubernetes secret to hold your TLS Certificate:

   ```
   kubectl create -n DEVELOPER-NAMESPACE secret tls SECRET_NAME \
     --key key.pem \
     --cert cert.pem
   ```

2. Create a delegation. To do so, create a `tlscertdelegation.yaml` file:

   ```
   apiVersion: projectcontour.io/v1
   kind: TLSCertificateDelegation
   metadata:
     name: default-delegation
     namespace: DEVELOPER-NAMESPACE
   spec:
     delegations:
       - secretName: SECRET-NAME
         targetNamespaces:
           - "DEVELOPER-NAMESPACE"
   ```

   Where `SECRET-NAME` is the name of the Kubernetes secret you created earlier.

3. Apply the YAML file by running:

   ```
   kubectl apply -f tlscertdelegation.yaml
   ```

4. Include the following configuration in your `tap-values.yaml` file under Cloud Native Runtimes section and redeploy:

   ```
   cnrs:
     contour:
       default_tls_secret: "DEVELOPER-NAMESPACE/SECRET-NAME"
   ```

   Where `SECRET-NAME` is the name of the Kubernetes secret you created earlier.

   Where `DEVELOPER-NAMESPACE` is the name of the namespace where the secret was created.

5. Update Tanzu Application Platform.

   To update the Tanzu Application Platform installation with the changes to the values file, run:

   ```
   tanzu package installed update tap -p tap.tanzu.vmware.com -v ${TAP_VERSION} --
   values-file tap-values.yaml -n tap-install
   ```

   This edits the Knative `config-contour` ConfigMap to use `default_tls_secret` as the default TLS certificate.

   Your web workloads' URLs use the scheme `https` by default when this secret is provided.

Configuring observability for Cloud Native Runtimes

This topic tells you how to configure observability for Cloud Native Runtimes, commonly known as CNRs.

Overview

You can set up integrations with third-party observability tools to use logging, metrics, and tracing with Cloud Native Runtimes. These observability integrations allow you to monitor and collect detailed metrics from your clusters on Cloud Native Runtimes. You can collect logs and metrics for all workloads running on a cluster. This includes Cloud Native Runtimes components or any apps running on Cloud Native Runtimes. The integrations in this topic are recommended by VMware, however you can use any Kubernetes-compatible logging, metrics, and tracing platforms to monitor your cluster workload.

Logging

You can collect and forward logs for all workloads on a cluster, including Cloud Native Runtimes components and any apps running on Cloud Native Runtimes. You can use any logging platform that is compatible with Kubernetes to collect and forward logs for Cloud Native Runtimes workloads. VMware recommends using Fluent Bit to collect logs and forward logs to vRealize. The following sections describe configuring logging for Cloud Native Runtimes with Fluent Bit and vRealize as an example.

Configure Logging with Fluent Bit

You can use Fluent Bit to collect logs for all workloads on a cluster, including Cloud Native Runtimes components or any apps running on Cloud Native Runtimes. For more information about using Fluent Bit logs, see Fluent Bit Kubernetes Logging.

Fluent Bit lets you collect logs from Kubernetes containers, add Kubernetes metadata to these logs, and forward logs to third-party log storage services. For more information about collecting logs, see the Knative documentation.

If you are using Tanzu Mission Control (TMC), vSphere 7.0 with Tanzu, or Tanzu Kubernetes Cluster to manage your cloud-native environment, you must set up a role binding that grants required permissions to Fluent Bit containers to configure logging with any integration. Then, follow the instructions in the Fluent Bit documentation to complete the logging configuration. For more information about configuring Fluent Bit logging, see the Fluent Bit documentation.

To configure logging with Fluent Bit for your Cloud Native Runtimes environment:

1. VMware recommends that you add any integrations to the `ConfigMap` in your Knative Serving namespace. Follow the logging configuration steps in the Fluent Bit documentation to create the `Namespace`, `ServiceAccount`, `Role`, `RoleBinding`, and `ConfigMap`. To view these steps, see the Fluent Bit documentation.
2. If you are using TMC, vSphere with Tanzu, or Tanzu Kubernetes Cluster to manage your cloud-native environment, create a role binding in the Kubernetes namespace where your integration is deployed to grant permission for privileged Fluent Bit containers. For information about creating a role binding on a Tanzu platform, see Add a Role Binding. For information about viewing your Kubernetes namespaces, see the Kubernetes documentation.

   Create the following role binding:

   ```
   apiVersion: rbac.authorization.k8s.io/v1
   kind: RoleBinding
   metadata:
     name: fluentbit-psp-rolebinding
     namespace: FLUENTBIT-NAMESPACE
   roleRef:
     apiGroup: rbac.authorization.k8s.io
     kind: ClusterRole
     name:  PRIVILEGED-CLUSTERROLE
   subjects:
   - kind: ServiceAccount
     name: FLUENTBIT-SERVICEACCOUNT
     namespace: FLUENTBIT-NAMESPACE
   ```

   Where:

   - `FLUENTBIT-NAMESPACE` is your Fluent Bit namespace.
   - `PRIVILEGED-CLUSTERROLE` is the name of your privileged cluster role.
   - `FLUENTBIT-SERVICEACCOUNT` is your Fluent Bit service account.

3. To verify that you configured logging successfully, run the following to access logs through your web browser:

   ```
   kubectl port-forward --namespace logging service/log-collector 8080:80
   ```

   For more information about accessing Fluent Bit logs, see the Knative documentation.

Forward Logs to vRealize

After you configure log collection, you can forward logs to log management services. vRealize Log Insight is one service you can use with Cloud Native Runtimes. vRealize Log Insight is a scalable log management solution that provides log management, dashboards, analytics, and third-party extensibility for infrastructure and apps. For more information about vRealize Log Insight, see the VMware vRealize Log Insight Documentation.

To forward logs from your Cloud Native Runtimes environment to vRealize, you can use a new or existing instance of Tanzu Kubernetes Cluster. For information about how to configure log forwarding to vRealize from Tanzu Kubernetes Cluster, see the Configure Log forwarding from VMware Tanzu Kubernetes Cluster to vRealize Log Insight Cloud blog.

Metrics

Cloud Native Runtimes integrates with Prometheus and VMware Aria Operations for Applications (formerly known as Tanzu Observability by Wavefront) to collect metrics on components or apps. For more information about integrating with Prometheus, see the Prometheus documentation and the Wavefront documentation.

You can configure Prometheus endpoints on Cloud Native Runtimes components to collect metrics on your components or apps. For information about configuring this, see the Prometheus documentation.

You can use annotation based discovery with Prometheus to define which Kubernetes objects in your Cloud Native Runtimes environment to add metadata and collect metrics in a more automated way. For more information about using annotation based discovery, see the VMware Aria Operations for Applications documentation in GitHub.

You can then use the Wavefront Collector for Kubernetes collector to dynamically discover and scrape pods with the `prometheus.io/scrape` annotation prefix. For information about the Kubernetes collector, see the VMware Aria Operations for Applications documentation in GitHub.

Note

All Cloud Native Runtimes related metrics have the prefix `tanzu.vmware.com/cloud-native-runtimes.*`.

Tracing

Tracing is a method for understanding the performance of specific code paths in apps as they handle requests. You can configure tracing to collect performance metrics for your apps or Cloud Native Runtimes components. You can trace which aspects of Cloud Native Runtimes and workloads running on Cloud Native Runtimes are performing poorly.

Configuring Tracing

You can configure tracing for your applications on Cloud Native Runtimes. You configure tracing for Knative Serving by editing the ConfigMap `config-tracing` for your Knative namespaces.

VMware recommends that you add any integrations in your Serving namespaces. For information about how to enable request traces in each component, see the Knative documentation.

Forwarding trace data to an observability platform or data visualization tool

You can use the OpenTelemetry integration to forward trace data to a data visualization tool that can ingest data in Zipkin format. For more information about using Zipkin for tracing, see the Zipkin documentation.

VMware recommends integrating with VMware Aria Operations for Applications (formerly known as Tanzu Observability by Wavefront). For information about forwarding trace data, see the VMware Aria Operations for Applications documentation.

Sending trace data to VMware Aria Operations for Applications

You can send trace data to an observability and analytics platform such as VMware Aria Operations for Applications to view and monitor your trace data in dashboards. VMware Aria Operations for Applications offers several deployment options. During development, a single proxy is often sufficient for all data sources. For more information about other deployment options, see the VMware Aria Operations for Applications documentation.

To configure Cloud Native Runtimes to send traces to the Wavefront proxy and then, configure the Wavefront proxy to consume Zipkin spans:

1. Deploy the Wavefront Proxy. For more information about Wavefront proxies, see Install and Manage Wavefront Proxies.
2. Configure the namespace where the Wavefront Proxy was deployed with proper credentials to its container image registry.

   The following example uses the Namespace Provisioner package to automatically configure namespaces labeled with `apps.tanzu.vmware.com/tap-ns`.

   ```
   export WF_NAMESPACE=default
   kubectl label namespace ${WF_NAMESPACE} apps.tanzu.vmware.com/tap-ns=""

   export WF_REGISTRY_HOSTNAME=HOSTNAME
   export WF_REGISTRY_USERNAME=USERNAME
   export WF_REGISTRY_PASSWORD=PASSWORD
   tanzu secret registry add registry-credentials \
    --username ${WF_REGISTRY_USERNAME} --password ${WF_REGISTRY_PASSWORD} \
    --server ${WF_REGISTRY_HOSTNAME} \
    --export-to-all-namespaces --yes --namespace tap-install
   ```

   Where:

   - `WF_NAMESPACE` is the namespace where you deployed the Wavefront Proxy.
   - `HOSTNAME` is the image registry where the Wavefront Proxy image is located.
   - `USERNAME` is your user name to access the image registry to pull the Wavefront Proxy image.
   - `PASSWORD` is your password to access the image registry to pull the Wavefront Proxy image.

   For more information about how to set up developer namespaces, see Provision developer namespaces. Provision developer namespaces.
3. Configure the Wavefront Proxy to allow Zipkin/Istio traces.

You can uncomment the lines indicated in the YAML file for the Wavefront Deployment to enable consumption of Zipkin traces. Edit the Wavefront Deployment to set the `WAVEFRONT_PROXY_ARGS` environment variable to the value `--traceZipkinListenerPorts 9411`. Also, edit the Wavefront Deployment to expose the containerPort `9411`.

4. Confirm that the Wavefront Proxy is running and working.

   Verify that pods are running. For more information about how to test a proxy, see the VMware Aria Operations for Applications documentation.

   ```
   kubectl get pods -n ${WF_NAMESPACE}
   ```

   Where `WF_NAMESPACE` is the namespace where you deployed the Wavefront Proxy.

5. Edit the Serving ConfigMap `config-tracing` to enable the Zipkin tracing integration.

   You can configure Cloud Native Runtimes to send traces to the Wavefront proxy by editing the `zipkin-endpoint` property in the ConfigMap to point to the Wavefront proxy URL. You can configure the Wavefront proxy to consume Zipkin spans by listening to port `9411`.

   Note

   There are two ways of editing a Knative ConfigMap on Cloud Native Runtimes. Depending on your installation, you can edit the ConfigMap directly on the cluster or by using overlays. For information about how to edit ConfigMaps using overlays, see Configuring Cloud Native Runtimes.

   The following example of a Kubernetes secret contains a ytt overlay with the suggested changes to the ConfigMap `config-tracing`:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
    name: cnrs-patch
   stringData:
    patch.yaml: |
      #@ load("@ytt:overlay", "overlay")
      #@overlay/match by=overlay.subset({"kind":"ConfigMap","metadata":{"name":"co
   nfig-tracing","namespace":"knative-serving"}})
      ---
      data:
        #@overlay/match missing_ok=True
        backend: "zipkin"
        #@overlay/match missing_ok=True
        zipkin-endpoint: "http://wavefront-proxy.default.svc.cluster.local:9411/ap
   i/v2/spans"
   ```

   After you follow the steps in Customizing Cloud Native Runtimes to configure your installation to use an overlay, you can examine the ConfigMap on the cluster to confirm that the changes were applied.

   ```
   kubectl get configmap config-tracing --namespace knative-serving --output yaml
   ```

   The ConfigMap looks like this example:

   ```
   apiVersion: v1
   kind: ConfigMap
   metadata:
   name: config-tracing
   data:
   ```

```
_example: |
    ...
backend: "zipkin"
zipkin-endpoint: "http://wavefront-proxy.default.svc.cluster.local:9411/api/v2/
spans"
```

Other resources:

- OpenTelemetry Integration
- Wavefront Proxies
- Deploy a Wavefront Proxy in Kubernetes (Manual Install)
- Managing API Tokens on Wavefront

Configuring Cloud Native Runtimes with Tanzu Service Mesh

This topic tells you how to configure Cloud Native Runtimes, commonly known as CNRs, with Tanzu Service Mesh.

Overview

You cannot install Cloud Native Runtimes on a cluster that has Tanzu Service Mesh attached.

This workaround describes how you can configure Tanzu Service Mesh to ignore the Cloud Native Runtimes. This allows Contour to provide ingress routing for the Knative workloads, while Tanzu Service Mesh continues to satisfy other connectivity concerns.

Note

Cloud Native Runtimes workloads are unable to use Tanzu Service Mesh features like Global Namespace, Mutual Transport Layer Security authentication (mTLS), retries, and timeouts.

For information about Tanzu Service Mesh, see Tanzu Service Mesh Documentation.

Prerequisites

The following prerequisites are required:

- A cluster attached to Tanzu Service Mesh
- You haven't installed Cloud Native Runtimes

Note

If you installed Cloud Native Runtimes on a cluster that has Tanzu Service Mesh attached before doing the following procedure, pods fail to start. To fix this problem, follow the procedure in this topic and then delete all pods in the excluded namespaces.

Run Cloud Native Runtimes on a cluster attached to Tanzu Service Mesh

Configure Tanzu Service Mesh to ignore namespaces related to Cloud Native Runtimes:

1. Navigate to the **Cluster Overview** tab in the Tanzu Service Mesh UI.
2. On the cluster where you want to install Cloud Native Runtimes, click **...**.
3. Click **Edit Cluster...**.
4. Create an Is Exactly rule for each of the following namespaces:
    - CONTOUR-NS
    - knative-serving
    - tap-install
    - rabbitmq-system
    - kapp-controller
    - The namespace or namespaces where you plan to run Knative workloads.

    Where `CONTOUR-NS` is the one or more namespaces where Contour is installed on your cluster. If Cloud Native Runtimes was installed as part of a Tanzu Application Profile, this value is likely `tanzu-system-ingress`.

Next steps

After configuring Tanzu Service Mesh, install Cloud Native Runtimes and verify your installation:

1. Install Cloud Native Runtimes. See Installing Cloud Native Runtimes.
2. Verify your installation. See Verifying Your Installation.

Note

You must create all Knative workloads in the namespace or namespaces where you plan to run these Knative workloads. If you do not, your pods fail to start.

Troubleshooting Cloud Native Runtimes

This topic tells you how to troubleshoot Cloud Native Runtimes, commonly known as CNRs, installation, or configuration.

Installation fails to reconcile app/cloud-native-runtimes

Symptom

When installing Cloud Native Runtimes, you see one of the following errors:

```
11:41:16AM: ongoing: reconcile app/cloud-native-runtimes (kappctrl.k14s.io/v1alpha1) n
amespace: cloud-native-runtime
11:41:16AM:  ^ Waiting for generation 1 to be observed
kapp: Error: Timed out waiting after 15m0s
```

Or,

```
3:15:34PM:  ^ Reconciling
3:16:09PM: fail: reconcile app/cloud-native-runtimes (kappctrl.k14s.io/v1alpha1) names
pace: cloud-native-runtimes
3:16:09PM:  ^ Reconcile failed:  (message: Deploying: Error (see .status.usefulErrorMe
ssage for details))
```

```
kapp: Error: waiting on reconcile app/cloud-native-runtimes (kappctrl.k14s.io/v1alpha
1) namespace: cloud-native-runtimes:
  Finished unsuccessfully (Reconcile failed:  (message: Deploying: Error (see .status.
usefulErrorMessage for details)))
```

Explanation

The `cloud-native-runtimes` deployment app installs the subcomponents of Cloud Native Runtimes. Error messages about reconciling indicate that one or more subcomponents have failed to install.

Solution

Use the following procedure to examine logs:

1. Get the logs from the `cloud-native-runtimes` app by running:

   ```
   kubectl get app/cloud-native-runtimes -n cloud-native-runtimes -o jsonpath="{.s
   tatus.deploy.stdout}"
   ```

   > **Note**
   >
   > If the command does not return log entries, kapp-controller is not installed or is not running correctly.

2. Review the output for sub component deployments that have failed or are still ongoing. See the following examples for suggestions on resolving common problems.

Example 1: The Cloud Provider does not support the creation of Service type LoadBalancer

Follow these steps to identify and resolve the problem of the cloud provider not supporting services of type `LoadBalancer`:

1. Search the log output for `Load balancer`, for example, by running:

   ```
   kubectl -n cloud-native-runtimes get app cloud-native-runtimes -ojsonpath="{.st
   atus.deploy.stdout}" | grep "Load balancer" -C 1
   ```

2. If the output looks similar to the following, ensure that your cloud provider supports services of type `LoadBalancer`. For more information, see Access Tanzu Developer Portal.

   ```
   6:30:22PM: ongoing: reconcile service/envoy (v1) namespace: CONTOUR-NS
   6:30:22PM:  ^ Load balancer ingress is empty
   6:30:29PM: ---- waiting on 1 changes [322/323 done] ----
   ```

   Where `CONTOUR-NS` is the namespace where Contour is installed on your cluster. If Cloud Native Runtimes was installed as part of a Tanzu Application Profile, this value is likely `tanzu-system-ingress`.

Example 2: The webhook deployment failed

Follow these steps to identify and resolve the problem of the `webhook` deployment failing in the `knative-serving` namespace:

1. Review the logs for output similar to the following:

   ```
   10:51:58PM: ok: reconcile customresourcedefinition/httpproxies.projectcontour.i
   o (apiextensions.k8s.io/v1) cluster
   ```

```
10:51:58PM: fail: reconcile deployment/webhook (apps/v1) namespace: knative-ser
ving
10:51:58PM:  ^ Deployment is not progressing: ProgressDeadlineExceeded (messag
e: ReplicaSet "webhook-6f5d979b7d" has timed out progressing.)
```

2. Run `kubectl get pods` to find the name of the pod:

   ```
   kubectl get pods --show-labels -n NAMESPACE
   ```

   Where `NAMESPACE` is the namespace associated with the reconcile error, for example, `knative-serving`.

   For example,

   ```
   $ kubectl get pods --show-labels -n knative-serving
   NAME                       READY   STATUS    RESTARTS   AGE   LABELS
   webhook-6f5d979b7d-cxr9k   0/1     Pending   0          44h   app=webhook,kapp.
   k14s.io/app=1626302357703846007,kapp.k14s.io/association=v1.9621e0a793b4e925077
   dd557acedbcfe,pod-template-hash=6f5d979b7d,role=webhook
   ```

3. Run `kubectl logs` and `kubectl describe`:

   ```
   kubectl logs PODNAME -n NAMESPACE
   kubectl describe pod PODNAME -n NAMESPACE
   ```

   Where:

   - `PODNAME` is found in the output of step 3. For example, `webhook-6f5d979b7d-cxr9k`.
   - `NAMESPACE` is the namespace associated with the reconcile error, for example, `knative-serving`.

   For example:

   ```
   $ kubectl logs webhook-6f5d979b7d-cxr9k -n knative-serving

   $ kubectl describe pod webhook-6f5d979b7d-cxr9k  -n knative-serving
   Events:
   Type     Reason            Age                 From               Message
   ----     ------            ----                ----               -------
   Warning  FailedScheduling  80s (x14 over 14m)  default-scheduler  0/1 nodes are
   available: 1 Insufficient cpu.
   ```

4. Review the output from the `kubectl logs` and `kubectl describe` commands and take further action.

   For this example of the webhook deployment, the output indicates that the scheduler does not have enough CPU to run the pod. In this case, the solution is to add nodes or CPU cores to the cluster. If you are using Tanzu Mission Control (TMC), increase the number of workers in the node pool to three or more through the TMC UI. See Edit a Node Pool, in the TMC documentation.

Knative Service Fails to Come up Due to Invalid HTTPPRoxy

Symptom

When creating a Knative Service, it does not reach ready status. The corresponding Route resource has the status `Ready=Unknown` with `Reason=EndpointsNotReady`. When you inspect the logs for the `net-contour-controller`, you see an error like this:

{"severity":"ERROR","timestamp":"2022-12-08T16:27:08.320604183Z","logger":"net-contour-controller","caller":"ingress/reconciler.go:313","message":"Returned an error","commit":"041f9e3","knative.dev/controller":"knative.dev.net-contour.pkg.reconciler.contour.Reconciler","knative.dev/kind":"networking.internal.knative.dev.Ingress","knative.dev/traceid":"9d615387-f552-449c-a8cd-04c69dd1849e","knative.dev/key":"cody/foo-java","targetMethod":"ReconcileKind","error":"HTTPProxy.projectcontour.io \"foo-java-contour-5f549ae3e6f584a5f33d069a0650c0d8foo-java.cody.\" is invalid: metadata.name: Invalid value: \"foo-java-contour-5f549ae3e6f584a5f33d069a0650c0d8foo-java.cody.\": a lowercase RFC 1123 subdomain must consist of lower case alphanumeric characters, '-' or '.', and must start and end with an alphanumeric character (e.g. 'example.com', regex used for validation is '[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*')","stacktrace":"knative.dev/networking/pkg/client/injection/reconciler/networking/v1alpha1/ingress.(*reconcilerImpl).Reconcile\n\tknative.dev/networking@v0.0.0-20221012062251-58f3e6239b4f/pkg/client/injection/reconciler/networking/v1alpha1/ingress/reconciler.go:313\nknative.dev/pkg/controller.(*Impl).processNextWorkItem\n\tknative.dev/pkg@v0.0.0-20221011175852-714b7630a836/controller/controller.go:542\nknative.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg@v0.0.0-20221011175852-714b7630a836/controller/controller.go:491"}

Solution

Because the `ChildName` function produces invalid resource names, certain combinations of Name, Namespace, and Domain yield invalid names for HTTPProxy resources because of the way the name is hashed and trimmed to fit the size requirement. It can have non-alphanumeric characters at the end of the name.

Resolving this is unique to each Knative service. It is likely to involve renaming your app to be shorter so that after the hash and trim procedure, the name is cut to end on an alphanumeric character.

For example, `foo-java.cody.iterate.tanzu-azure-lab.winterfell.fun` is hashed and trimmed into `foo-java-contour-5f549ae3e6f584a5f33d069a0650c0d8foo-java.cody.`, leaving an invalid `.` at the end.

However, changing the app name to `foo-jav` causes `foo-jav-contour-SOME-DIFFERENT-HASH-foo-jav.cody.it`, which is a valid name.

Cloud Native Runtimes Reference

The following topics give you reference documentation for Cloud Native Runtimes:

- Compatibility Matrix

Integrations you can use with Cloud Native Runtimes

This topic lists integrations you can use with Cloud Native Runtimes.

Cloud Native Runtimes integrations

Cloud Native Runtimes Integration	Version	Documentation
VMware Tanzu Observability	Supported	Configuring Observability for Cloud Native Runtimes
Avi Vantage	Supported	Configuring Cloud Native Runtimes with Avi Vantage
Tanzu Service Mesh	Supported	Configuring Cloud Native Runtimes with Tanzu Service Mesh

For tools and software compatibility, see Prerequisites for installing Tanzu Application Platform.

Overview of Contour

Contour is an ingress controller for Kubernetes that supports dynamic configuration updates and multi-team ingress delegation. It provides the control plane for the Envoy edge and service proxy.

Document organization

The Contour component documentation consists of the following subsections that relate to what you want to achieve:

- How-to guides: To find a set of steps to solve a specific problem acting as a certain user persona.
- Reference: To find specific information such as Contour's APIs.

Overview of Contour

Contour is an ingress controller for Kubernetes that supports dynamic configuration updates and multi-team ingress delegation. It provides the control plane for the Envoy edge and service proxy.

Document organization

The Contour component documentation consists of the following subsections that relate to what you want to achieve:

- How-to guides: To find a set of steps to solve a specific problem acting as a certain user persona.
- Reference: To find specific information such as Contour's APIs.

Install Contour

This topic tells you how to install Contour from the Tanzu Application Platform (commonly known as TAP) package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Contour. For more information about profiles, see Components and installation profiles.

To install Contour from the Tanzu Application Platform package repository without a profile:

1. Install cert-manager.
2. List version information for the package by running:

   ```
   tanzu package available list contour.tanzu.vmware.com -n tap-install
   ```

   For example:

   ```
   $  tanzu package available list contour.tanzu.vmware.com -n tap-install
     NAME                      VERSION       RELEASED-AT
     contour.tanzu.vmware.com  1.24.4        2023-05-22 19:00:00 -0500 -05
   ```

3. Create a file named `contour-rbac.yaml` by using the following sample and apply the configuration:

   ```
   apiVersion: rbac.authorization.k8s.io/v1
   kind: ClusterRole
   metadata:
   ```

```
  name: contour-tap-install-cluster-admin-role
rules:
- apiGroups:
  - '*'
  resources:
  - '*'
  verbs:
  - '*'
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: contour-tap-install-cluster-admin-role-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: contour-tap-install-cluster-admin-role
subjects:
- kind: ServiceAccount
  name: contour-tap-install-sa
  namespace: tap-install
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: contour-tap-install-sa
  namespace: tap-install
```

4. Apply the configuration by running:

```
kubectl apply -f contour-rbac.yaml
```

5. Create a file named `contour-install.yaml` by using the following sample and apply the configuration:

Note

The following configuration installs the Contour package with default options. To make changes to the default installation settings, go to the next step.

```
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: contour
  namespace: tap-install
spec:
  serviceAccountName: contour-tap-install-sa
  packageRef:
    refName: contour.tanzu.vmware.com
    versionSelection:
      constraints: "VERSION-NUMBER"
      prereleases: {}
```

Where `VERSION-NUMBER` is the version of the package listed earlier.

6. (Optional) Make changes to the default installation settings:

    1. Gather values schema by running:

    ```
    tanzu package available get contour.tanzu.vmware.com/1.24.4 --values-sche
    ```

```
ma -n tap-install
```

For example:

```
$ tanzu package available get contour.tanzu.vmware.com/1.24.4 --values-sc
hema -n tap-install

  KEY                                                    DEFAULT
TYPE     DESCRIPTION
  kubernetes_distribution
string   The distribution of Kubernetes, used to determine if distributio
n-specific

configurations need to be applied. Options are empty and openshift. If ru
nning

on an OpenShift cluster, this must be set to openshift. When set to opens
hift,

a Role and RoleBinding are created to associate Contour's controllers wit
h the

appropriate OpenShift Security Context Constraint resource.
  kubernetes_version                                     0.0.0
string   The version of Kubernetes being used, for enabling version-speci
fic behaviors.

Accept any valid major.minor.patch version of Kubernetes. This field is

optional. Currently only has effect when kubernetes_distribution is set t
o

openshift.
  namespace                                              tanzu-system-
ingress      string   The namespace in which to deploy Contour and Envoy.
  registry_secret_names                                  [contour-reg-
creds]       array    The names of the placeholder secrets that will cont
ain registry credentials to

pull the Contour and Envoy images.
  certificates.duration                                  8760h
string   How long the certificates should be valid for.
  certificates.renewBefore                               360h
string   How long before expiration the certificates should be renewed.
  contour.pspNames                                       vmware-system
-restricted  string   Pod security policy names to apply to Contour.
  contour.replicas                                       2
integer  How many Contour pod replicas to have.
  contour.resources.contour.requests.memory
string   Memory request to apply to the contour container.
  contour.resources.contour.requests.cpu
string   CPU request to apply to the contour container.
  contour.resources.contour.limits.cpu
string   CPU limit to apply to the contour container.
  contour.resources.contour.limits.memory
string   Memory limit to apply to the contour container.
  contour.useProxyProtocol                               false
boolean  Whether to enable PROXY protocol for all Envoy listeners.
  contour.configFileContents                             <nil>
<nil>    The YAML contents of the Contour config file. See

https://projectcontour.io/docs/1.24/configuration/#configuration-file for
more

information.
  contour.logLevel                                       info
```

```
string   The Contour log level. Valid options are 'info' and 'debug'.
  envoy.pspNames
string   Pod security policy names to apply to Envoy.
  envoy.service.externalTrafficPolicy
string   The external traffic policy for the Envoy service. If type is 'C
lusterIP', this

field is ignored. Otherwise, defaults to 'Cluster' for vsphere and 'Loca
l' for

others.
  envoy.service.loadBalancerIP
string   The desired load balancer IP. If type is not 'LoadBalancer', thi
s field is

ignored. It is up to the cloud provider whether to honor this request. If
not

specified, then load balancer IP will be assigned by the cloud provider.
  envoy.service.nodePorts.http                                    0
integer  The node port number to expose Envoy's HTTP listener on. If not
specified, a

node port will be auto-assigned by Kubernetes.
  envoy.service.nodePorts.https                                   0
integer  The node port number to expose Envoy's HTTPS listener on. If not
specified, a

node port will be auto-assigned by Kubernetes.
  envoy.service.type
string   The type of Kubernetes service to provision for Envoy. Valid val
ues are

'LoadBalancer', 'NodePort', and 'ClusterIP'. If not specified, will defau
lt to

'NodePort' for vsphere and 'LoadBalancer' for others.
  envoy.service.annotations                                       <nil>
<nil>    Annotations to set on the Envoy service.
  envoy.service.aws.LBType                                        classic
string   The type of AWS load balancer to provision. Options are 'classi
c' and 'nlb'.
  envoy.service.disableWait                                       false
boolean  This setting is no longer supported and is included in the schem
a for backwards

compatibility only.
  envoy.terminationGracePeriodSeconds                             300
integer  The termination grace period, in seconds, for the Envoy pods.
  envoy.workload.replicas                                         2
integer  The number of Envoy replicas to deploy when 'type' is set to 'De
ployment'. If

not specified, it will default to '2'.
  envoy.workload.resources.envoy.limits.cpu
string   CPU limit to apply to the envoy container.
  envoy.workload.resources.envoy.limits.memory
string   Memory limit to apply to the envoy container.
  envoy.workload.resources.envoy.requests.cpu
string   CPU request to apply to the envoy container.
  envoy.workload.resources.envoy.requests.memory
string   Memory request to apply to the envoy container.
  envoy.workload.resources.shutdownManager.limits.cpu
string   CPU limit to apply to the shutdown-manager container.
  envoy.workload.resources.shutdownManager.limits.memory
string   Memory limit to apply to the shutdown-manager container.
```

```
  envoy.workload.resources.shutdownManager.requests.cpu
string   CPU request to apply to the shutdown-manager container.
  envoy.workload.resources.shutdownManager.requests.memory
string   Memory request to apply to the shutdown-manager container.
  envoy.workload.type                                        DaemonSet
string   The type of Kubernetes workload Envoy is deployed as. Options ar
e 'Deployment'

or 'DaemonSet'. If not specified, will default to 'DaemonSet'.
  envoy.hostNetwork                                          false
boolean  Whether to enable host networking for the Envoy pods.
  envoy.hostPorts.https                                      443
integer  If enable == true, the host port number to expose Envoy's HTTPS
listener on.
  envoy.hostPorts.enable                                     true
boolean  Whether to enable host ports. If false, http & https are ignore
d.
  envoy.hostPorts.http                                       80
integer  If enable == true, the host port number to expose Envoy's HTTP l
istener on.
  envoy.logLevel                                             info
string   The Envoy log level.
  infrastructure_provider                                    vsphere
string   The underlying infrastructure provider. Options are vsphere, aw
s, and azure.

This field is not required, but enables better validation and defaulting
if

provided.
```

2. Create a `contour-install.yaml` file by using the following sample as a guide:

Note

This sample is for installation in an AWS public cloud with `LoadBalancer` services.

```
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: contour
  namespace: tap-install
spec:
  serviceAccountName: contour-tap-install-sa
  packageRef:
    refName: contour.tanzu.vmware.com
    versionSelection:
      constraints: 1.24.4
      prereleases: {}
  values:
  - secretRef:
      name: contour-values
---
apiVersion: v1
kind: Secret
metadata:
  name: contour-values
  namespace: tap-install
stringData:
  values.yaml: |
    envoy:
      service:
```

```
        type: LoadBalancer
    infrastructure_provider: aws
```

The LoadBalancer type is appropriate for most installations, but local clusters such as `kind` can fail to complete the package install if LoadBalancer services are not supported.

For local clusters, you can configure `contour.envoy.service.type` to be `NodePort`. If your local cluster is set up with extra port mappings on the nodes, you might also need configure `envoy.service.nodePorts.http` and `envoy.service.nodePorts.https` to match the port mappings from your local machine into one of the nodes of your local cluster.

Contour provides an Ingress implementation by default. If you have another Ingress implementation in your cluster, you must explicitly specify an IngressClass to select a particular implementation.

Cloud Native Runtimes programs Contour HTTPRoutes are based on the installed namespace. The default installation of Cloud Native Runtimes uses a single Contour to provide internet-visible services. You can install a second Contour instance with service type `ClusterIP` if you want to expose some services to only the local cluster. The second instance must be installed in a separate namespace. You must set the Cloud Native Runtimes value `ingress.internal.namespace` to point to this namespace.

7. Install the package by running:

```
kubectl apply -f contour-install.yaml
```

8. Verify the package install by running:

```
tanzu package installed get contour -n tap-install
```

For example:

```
$ tanzu package installed get contour -n tap-install

NAME:                    contour
PACKAGE-NAME:            contour.tanzu.vmware.com
PACKAGE-VERSION:         1.24.4
STATUS:                  Reconcile succeeded
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`

9. Verify the installation by running:

```
kubectl get po -n tanzu-system-ingress
```

For example:

```
$  kubectl get po -n tanzu-system-ingress
NAME                       READY   STATUS    RESTARTS   AGE
contour-689789679f-cl8cg   1/1     Running   0          6d16h
contour-689789679f-g2xjn   1/1     Running   0          6d16h
envoy-2blkl                2/2     Running   0          6d16h
envoy-2rjtb                2/2     Running   0          6d16h
envoy-9sljz                2/2     Running   0          6d16h
envoy-cv5fc                2/2     Running   0          6d16h
```

```
envoy-gvtlf                2/2     Running   0          6d16h
envoy-qtrxd                2/2     Running   0          6d16h
```

Ensure that all pods are `Running` with all containers ready.

Contour how-to guides

This section contains the following how-to guides for Contour:

- Configure Cipher Suites and TLS version in Contour
- Configure Contour to propagate header with Domain Mapping
- Configure Contour to support TLS termination at an AWS LoadBalancer
- Troubleshoot Contour

Configure Cipher Suites and TLS version in Contour

This topic tells you how to configure TLS options for Contour in Tanzu Application Platform (commonly known as TAP).

Contour provides configuration options for TLS version and Cipher Suites. Rather than directly exposed through a top level key in the pacakge, they fall into the category of advanced Contour configurations by using the `contour.configFileContents` key. For more information about these configuration options, see Contour documentation.

To configure TLS options for Contour in Tanzu Application Platform, edit the `contour` section of your TAP values file as follows:

```
contour:
  # ... there maybe some configuration already here
  contour:
    configFileContents:
      tls:
        minimum-protocol-version: "1.2"
        cipher-suites:
        - '[ECDHE-ECDSA-AES128-GCM-SHA256|ECDHE-ECDSA-CHACHA20-POLY1305]'
        - '[ECDHE-RSA-AES128-GCM-SHA256|ECDHE-RSA-CHACHA20-POLY1305]'
        - 'ECDHE-ECDSA-AES256-GCM-SHA384'
        - 'ECDHE-RSA-AES256-GCM-SHA384'
```

Expect to see the following Cipher Suites and TLS version data in the Contour configmap:

```
$ kubectl -n tanzu-system-ingress get configmap contour -oyaml
apiVersion: v1
data:
  contour.yaml: |
    tls:
      minimum-protocol-version: "1.2"
      cipher-suites:
      - '[ECDHE-ECDSA-AES128-GCM-SHA256|ECDHE-ECDSA-CHACHA20-POLY1305]'
      - '[ECDHE-RSA-AES128-GCM-SHA256|ECDHE-RSA-CHACHA20-POLY1305]'
      - ECDHE-ECDSA-AES256-GCM-SHA384
      - ECDHE-RSA-AES256-GCM-SHA384
kind: ConfigMap
metadata:
...
```

Important

To update the configmap, you must configure it through Tanzu Application Platform values file. If you change it directly in the configmap, kapp-controller reverts all the changes you made.

Configure Contour to propagate header with Domain Mapping

This topic tells you how to configure Contour for header propagation with Knative Domain Mapping and autoTLS certs.

After `autoTLS` is enabled, a request for a Knative service with Domain Mapping might not maintain the `x-forwaded-proto` header. This can cause issues when the request is forwarded to a system expecting this header to be set to `https`, for example, Spring Security SSO.

You can solve this issue by configuring Contour with `num-trusted-hops: 1`. This allows the header to correctly propagate by using Domain Mapping.

Important

To update the configmap, you must configure it in the Tanzu Application Platform values file. If you change it directly in the configmap, kapp-controller reverts all the changes you made.

Example configuration for Contour by using the Tanzu Application Platform values file (commonly called `tap-values.yaml`):

```
contour:
  contour:
    configFileContents:
      num-trusted-hops: 1
```

Configure Contour to support TLS termination against AWS Elastic Load Balancing

This topic tells you how to configure Contour to accept traffic from an AWS LoadBalancer that stops TLS traffic.

For more information, see the AWS Knowledge Center documentation.

About certificates and domains

This topic focuses on using a wildcard certificate for your domain. For example, if your domain is `bigbiz.com`, the certificate must be for the wildcard domain `*.bigbiz.com`, and everything routed by using Contour must fit that domain pattern.

However, in Tanzu Application Platform, the default URL pattern for Web Workloads, for example, Knative Services, is `{{.Name}}.{{.Namespace}}.{{.Domain}}`.

The wildcard certificate mentioned earlier does not apply to these URLs. To address this, this topic describes how to configure Cloud Native Runtimes so that Web Workload URLs also fit the wildcard domain.

You can keep the default URL pattern if your `DOMAIN` includes the namespace where Web Workloads are deployed.

Prerequisites

The following are required before proceeding with the configuration:

- An EKS cluster.
- The Contour package installed on the cluster, either as part of Tanzu Application Platform or from the standalone component installation. For more information, see Install Contour.
- Access to AWS Certificate Manager.
- A domain registered in Route53 or elsewhere. This topic refers to this domain as `DOMAIN`.

Create a TLS certificate in ACM

Create a public TLS certificate for `*.DOMAIN` by using AWS Certificate Manager (ACM). For more information, see the AWS documentation.

Important

Record the `ARN` of the created certificate, which is required in the following steps.

Configure Tanzu Application Platform

Follow these steps to configure your Tanzu Application Platform:

1. Create the following overlay in `overlay-contour-envoy-secret.yaml`:

```
apiVersion: v1
kind: Secret
metadata:
name: overlay-contour-envoy
namespace: tap-install
stringData:
overlay-contour-envoy.yml: |
  #@ load("@ytt:overlay", "overlay")

  #@overlay/match by=overlay.subset({"kind": "Service", "metadata": {"name": "e
nvoy"}})
  ---
  spec:
    ports:
    #@overlay/match by=overlay.subset({"name":"https"})
    -
      targetPort: 8080
```

2. Apply the overlay to your cluster:

```
kubectl apply -f overlay-contour-envoy-secret.yaml
```

3. Add the following configurations to your Tanzu Application Platform values file:

```
shared:
  ingress_issuer: ""
  ingress_domain: DOMAIN

tap_gui:
  app_config:
    app:
      baseUrl: https://tap-gui.DOMAIN #! note the change in scheme
    backend:
```

```
      baseUrl: https://tap-gui.DOMAIN #! note the change in scheme
      reading:
        allow:
        - host: "*.DOMAIN"

cnrs:
  default_external_scheme: "https"
  ingress_issuer: ""
  domain_template: "{{.Name}}-{{.Namespace}}.{{.Domain}}"

contour:
  infrastructure_provider: aws
  envoy:
    service:
      aws:
        LBType: nlb
      annotations:
        service.beta.kubernetes.io/aws-load-balancer-ssl-cert: ARN
        service.beta.kubernetes.io/aws-load-balancer-backend-protocol: http

package_overlays:
- name: contour
  secrets:
  - name: overlay-contour-envoy
```

Where `ARN` is the ARN recorded in Create a TLS certificate in ACM.

To use the Classic LoadBalancer, remove `contour.envoy.service.aws.LBType: nlb`.

4. Update your Tanzu Application Platform installation:

```
tanzu package installed update tap -n tap-install --values-file tap-values.yaml
-p tap.tanzu.vmware.com -v VERSION
```

Where `VERSION` is the version of Tanzu Application Platform in use, which must be in the form of `X.X.X`.

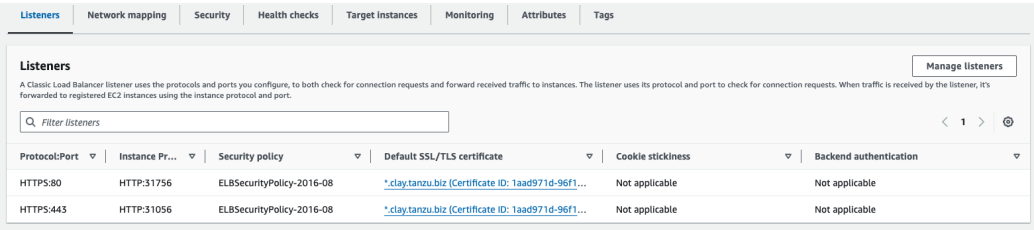
Configure AWS

Follow these steps to configure your AWS:

1. (Optional) Find the load balancer associated with Contour's Envoy LoadBalancer Service and verify the configuration of HTTPS listeners.

   You must have an AWS LoadBalancer pointing to Contour's Envoys.

   There are two listeners, 80 and 443, on your LoadBalancer. Both listeners use the certificate associated with the `ARN` in your tap-values file.



Note

AWS adds SSL or HTTPS listeners for all ports on the Kubernetes service by default. In the context of this topic, it indicates that port 80 is also configured to listen on HTTPS. To leave port 80 open to HTTP traffic, you can add the annotation `service.beta.kubernetes.io/aws-load-balancer-`

`ssl-ports: "https"` to your tap-values file at `contour.envoy.service.annotations`. The annotation adds an SSL or HTTPS listener for the `https` port on the service. For more information about the annotations, see the AWS cloud provider documentation.

2. Configure the domain name system (DNS).
    1. Get the External IP of the Envoy service:

        ```
        kubectl get svc envoy -n NAMESPACE
        ```

        Where `NAMESPACE` is the namespace where Contour is installed. The default value is `tanzu-system-ingress` unless configured otherwise.

        The result resembles the following:

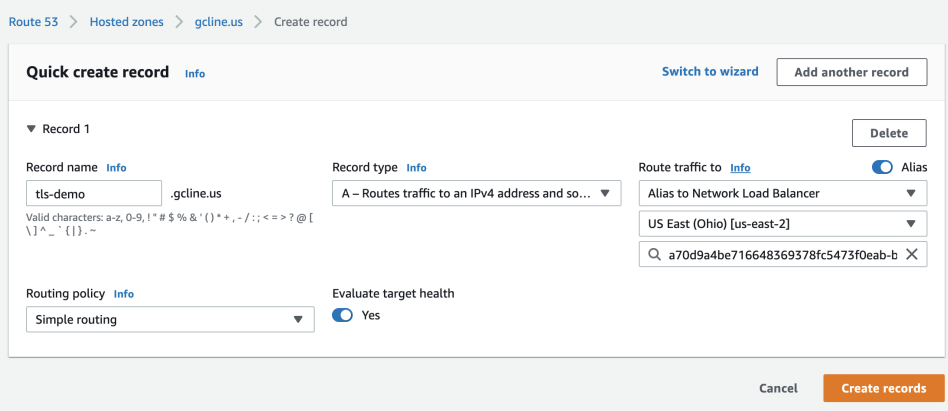
        ```
        NAME     TYPE           CLUSTER-IP      EXTERNAL-IP
        PORT(S)             AGE
         envoy   LoadBalancer   10.100.24.154   a7ea2bbde8a164036a7e4c1ed5700cdf-
        154fb911d990bb1f.elb.us-east-2.amazonaws.com   443:31606/TCP      40d
        ```

    2. Create a DNS record pointing from `DOMAIN` to the NLB Domain, which is the External IP address from the earlier step in this procedure.

        In the **Route traffic to** section, you must set:

        - Alias to Network LoadBalancer.
        - The appropriate region for your NLB.
        - The name of your NLB domain from the previous step.

        It resembles the following:

Important

If not using AWS Route53, you must create a CNAME entry in your DNS provider. Otherwise, with AWS Route53, you can create an “A” record type, and alias it to the Network LoadBalancer.

Verify the configuration

If you have a Web Workload on the cluster, you can verify that it is available at `https://appname-appnamespace.DOMAIN`.

Troubleshoot Contour

This topic tells you how to troubleshoot Contour installation and configuration.

Envoy pods fail with malformed IP address errors

Symptom:

The Contour package fails to reconcile, and the Envoy pods fail with errors like:

```
23-11-23 01:55:11.503][1][warning][config] [source/common/config/grpc_subscription_imp
l.cc:128] gRPC config for type.googleapis.com/envoy.config.listener.v3.Listener reject
ed: Error adding/updating listener(s) ingress_http: malformed IP address: ::
```

Explanation:

By default, the Contour package programs Envoy's listeners to use IPv6 addresses. In most cases, this is OK. However, some clusters, such as TKGI, only support IPv4.

Solution:

Important

This solution only applies to Contour v1.26 or later, which is included in Tanzu Application Platform v1.8.0 and later.

To use IPv4 addresses on the listeners, update the Tanzu Application Platform values file (commonly called `tap-values.yaml`).

The following is an example of the Contour section of the values file:

```
contour:
  envoy:
    listenIPFamily: IPv4
```

Contour reference

This section contains reference documentation for using Contour.

Open source Documentation

The Contour open source documentation contains more detailed Contour-specific reference information for advanced use, including:

- API reference
- Configuration reference
- Compatability matrix

Important

There may be some features or configurations that are not accessible by using the Contour package API in Tanzu Application Platform.

Overview of Crossplane

Crossplane is an open source, Cloud Native Computing Foundation (CNCF) project built on the foundation of Kubernetes. Tanzu Application Platform (commonly known as TAP) uses Crossplane to power a number of capabilities, such as dynamic provisioning of services instances with Services Toolkit and the Bitnami Services.

Note

If installing Tanzu Application Platform to a cluster that already has Crossplane installed, see Use your existing Crossplane installation.

Crossplane with Tanzu Application Platform

Tanzu Application Platform includes a Carvel package named `crossplane.tanzu.vmware.com`, which is included by default in the full, iterate, and run profiles. The package installs Upbound Universal Crossplane (UXP).

In addition, the package includes two pre-configured Crossplane providers: provider-helm and provider-kubernetes. Both of providers provide useful Managed Resources that you can use as part of Composition. These are both used by Tanzu Application Platform's Bitnami Services.

The package installs UXP and the providers to the `crossplane-system` namespace.

Getting started

To learn about working with Crossplane in general, see the Crossplane documentation. To learn about how Tanzu Application Platform integrates with Crossplane, see one of the following tutorials to get started.

For apps teams:

- Tutorial: Working with Bitnami Services

For ops teams:

- Tutorial: Setup Dynamic Provisioning of Service Instances
- How-to guide: Use your existing Crossplane installation
- How-to guide: Delete Crossplane resources when you uninstall Tanzu Application Platform

Alternatively, see the reference material.

Overview of Crossplane

Crossplane is an open source, Cloud Native Computing Foundation (CNCF) project built on the foundation of Kubernetes. Tanzu Application Platform (commonly known as TAP) uses Crossplane to power a number of capabilities, such as dynamic provisioning of services instances with Services Toolkit and the Bitnami Services.

Note

If installing Tanzu Application Platform to a cluster that already has Crossplane installed, see Use your existing Crossplane installation.

Crossplane with Tanzu Application Platform

Tanzu Application Platform includes a Carvel package named `crossplane.tanzu.vmware.com`, which is included by default in the full, iterate, and run profiles. The package installs Upbound Universal Crossplane (UXP).

In addition, the package includes two pre-configured Crossplane providers: provider-helm and provider-kubernetes. Both of providers provide useful Managed Resources that you can use as part of Composition. These are both used by Tanzu Application Platform's Bitnami Services.

The package installs UXP and the providers to the `crossplane-system` namespace.

Getting started

To learn about working with Crossplane in general, see the Crossplane documentation. To learn about how Tanzu Application Platform integrates with Crossplane, see one of the following tutorials to get started.

For apps teams:

- Tutorial: Working with Bitnami Services

For ops teams:

- Tutorial: Setup Dynamic Provisioning of Service Instances
- How-to guide: Use your existing Crossplane installation
- How-to guide: Delete Crossplane resources when you uninstall Tanzu Application Platform

Alternatively, see the reference material.

Install Crossplane

This topic tells you how to install Crossplane from the Tanzu Application Platform (commonly known as TAP) package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Crossplane. For more information about profiles, see Components and installation profiles.

If installing Tanzu Application Platform to a cluster that already has Crossplane installed, see Use your existing Crossplane installation.

Prerequisites

Before installing Crossplane:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.

Install Crossplane

To install Crossplane:

1. See what versions of Crossplane are available to install by running:

   ```
   tanzu package available list -n tap-install crossplane.tanzu.vmware.com
   ```

   For example:

```
$ tanzu package available list -n tap-install crossplane.tanzu.vmware.com
  NAME                              VERSION           RELEASED-AT
  crossplane.tanzu.vmware.com       0.1.1             2023-03-10 14:24:35 +000
0 UTC
```

2. Install Crossplane by running:

   ```
   tanzu package install crossplane \
     --package crossplane.tanzu.vmware.com \
     --version VERSION-NUMBER \
     --namespace tap-install
   ```

   Where `VERSION-NUMBER` is the Crossplane version you want to install. For example, `0.1.1`.

3. Verify that the package installed by running:

   ```
   tanzu package installed get crossplane -n tap-install
   ```

   In the output, confirm that the `STATUS` value is `Reconcile succeeded`.

   For example:

   ```
   $ tanzu package installed get crossplane -n tap-install
   NAMESPACE:          tap-install
   NAME:               crossplane
   PACKAGE-NAME:       crossplane.tanzu.vmware.com
   PACKAGE-VERSION:    0.1.1
   STATUS:             Reconcile succeeded
   CONDITIONS:         - type: ReconcileSucceeded
     status: "True"
     reason: ""
     message: ""
   ```

Crossplane how-to guides

This section contains how-to guides for Crossplane.

In this section:

- Use your existing Crossplane installation
- Delete Crossplane resources when you uninstall Tanzu Application Platform
- Troubleshoot Crossplane

Use your existing Crossplane installation

This topic describes how to install Tanzu Application Platform (commonly known as TAP) to a cluster that already has Crossplane installed. This also applies if you want to install Crossplane without using the Tanzu application Platform package, for example, through Helm install.

About installing the Crossplane package

The `crossplane.tanzu.vmware.com` package is included with the Full, Iterate and Run profiles. By default, any installation of Tanzu Application Platform using one of these profiles installs Crossplane to the cluster through the package. In most cases, this is desirable because Crossplane is necessary to support many of the service related capabilities that Tanzu Application Platform offers.

There are some cases where installing the Crossplane package by default is not desirable. For example, if you plan to install Tanzu Application Platform to a cluster that already has Crossplane

installed on it, or if you want to install Crossplane by some other means.

Exclude the Crossplane package from the installation

If you want to install Crossplane by other means, exclude the Crossplane package from the Tanzu Application Platform installation.

To do so, add the `crossplane.tanzu.vmware.com` package to the `excluded_packages` array in the `tap-values.yaml` file as follows:

```
# tap-values.yaml

excluded_packages:
- crossplane.tanzu.vmware.com
```

Important

Other Tanzu Application Platform components might depend on the Crossplane package, for example, Bitnami Services. If you exclude the Crossplane package, you must also exclude any other Tanzu Application Platform package that depends on it.

Delete Crossplane resources when you uninstall Tanzu Application Platform

This topic describes how to configure the Crossplane package to either retain or delete Crossplane resources when you uninstall Tanzu Application Platform.

About deleting Crossplane resources

By default, the `crossplane.tanzu.vmware.com` package is configured to retain all Crossplane CRDs, providers, and managed resources when the package is uninstalled. This is in the interest of caution in relation to accidental deletion of stateful data.

Consider the following scenario. An organization is using Tanzu Application Platform. They have installed the AWS Provider for Crossplane and are offering development teams self-serve access to AWS RDS databases through Dynamic Provisioning. Development teams have created a number of AWS RDS databases and have bound them to their application workloads. The AWS RDS databases now contain stateful data for the applications. If an operator decides to uninstall Tanzu Application Platform, what happens to the AWS RDS databases?

The answer depends on whether you view the life cycle of external services provisioned through the platform to be tied to the life cycle of the platform itself. By default, Tanzu Application Platform does not tie the life cycle of the platform to the life cycle of external services. This means that you must delete any external resources after deleting Tanzu Application Platform. However, Tanzu Application Platform provides the package value `orphan_resources` to override this default behavior.

Override the default retention behavior

You can configure Tanzu Application Platform to delete Crossplane resources to avoid orphaned resources. To do so, update the `tap-values.yaml` as follows:

```
# tap-values.yaml
```

```
crossplane:
  orphan_resources: false
```

Troubleshoot Crossplane

This topic explains how you troubleshoot issues related to Crossplane on Tanzu Application Platform (commonly known as TAP).

For the limitations of Crossplane, see Crossplane limitations.

Resource already exists error when installing Crossplane

Symptom:

Installation of Crossplane, or a Tanzu Application Platform profile that includes Crossplane, fails with the error:

```
Resource already exists
```

Explanation:

Crossplane is already installed on the cluster. You cannot install the Crossplane package on a cluster that already has Crossplane installed on it by using another method, such as Helm install.

Solution:

Exclude the Crossplane package in the `tap-values.yaml` file. For more information, see Use your existing Crossplane installation.

The `validatingwebhookconfiguration` is not removed when you uninstall the Crossplane Package

Symptom:

The Crossplane Package deploys a `validatingwebhookconfiguration` named `crossplane` during installation. This resource is not deleted when you uninstall the Package.

Solution:

Delete the `validatingwebhookconfiguration` manually by running:

```
kubectl delete validatingwebhookconfiguration crossplane
```

Crossplane reference

This section provides reference documentation for Crossplane.

In this section:

- Package values
- Version matrix
- Crossplane limitations

Package values for Crossplane

This topic lists the keys and values you can use to configure the behavior of the Crossplane package. Configuration is split between configuration specific to Crossplane in Tanzu Application

Platform (commonly known as TAP) and configuration of the Upbound Universal Crossplane (UXP) Helm Chart.

If you are applying configuration to Tanzu Application Platform through the use of profiles and the `tap-values.yaml`, all configuration must exist under the `crossplane` top-level key.

For example:

```
crossplane:
  replicas: 3
```

Tanzu Application Platform configuration

The following table lists configuration specific to Crossplane in Tanzu Application Platform.

KEY	DEFAULT	TYPE	DESCRIPTION
kubernetes_version	`""`	string	Optional: The Kubernetes version. Valid values are `''` or take the form `'1.25.0'`
kubernetes_distribution	`""`	string	Optional: The Kubernetes distribution. Valid values are `''` or `'openshift'`
orphan_resources	`true`	boolean	Optional: Whether to orphan Crossplane CRDs, providers, and managed resources when the package is uninstalled. If `false` the resources are deleted.
adopt_resources	`false`	boolean	Optional: Whether to adopt existing Crossplane CRDs and resources when the package is installed.
ca_cert_data	`""`	string	Optional: PEM-encoded certificate data for Crossplane to trust TLS connections with a custom CA certificate.

Standard Crossplane configuration

The following table lists configuration for the UXP Helm Chart.

KEY	DEFAULT	TYPE	DESCRIPTION
replicas	`1`	integer	
serviceAccount.customAnnotations	`'{}'`	object	
bootstrapper.resources	`'{}'`	object	
bootstrapper.config.args		array	
bootstrapper.config.debugMode	`false`	boolean	
bootstrapper.config.envVars	`'{}'`	object	
bootstrapper.image.pullPolicy	`IfNotPresent`	string	
bootstrapper.image.repository	`xpkg.upbound.io/upbound/uxp-bootstrapper`	string	

KEY	DEFAULT	TYPE	DESCRIPTION
bootstrapper.image.tag	`v1.11.0-up.1`	string	
maxReconcileRate	`""`	string	How frequently Crossplane reconciles its resources in seconds.
metrics.enabled	`false`	boolean	
securityContextCrossplane.allowPrivilegeEscalation	`false`	boolean	
securityContextCrossplane.readOnlyRootFilesystem	`true`	boolean	
securityContextCrossplane.runAsGroup	`65532`	integer	
securityContextCrossplane.runAsUser	`65532`	integer	
xfn.securityContext.allowPrivilegeEscalation	`false`	boolean	
xfn.securityContext.capabilities.add		array	
xfn.securityContext.readOnlyRootFilesystem	`true`	boolean	
xfn.securityContext.runAsGroup	`65532`	integer	
xfn.securityContext.runAsUser	`65532`	integer	
xfn.securityContext.seccompProfile.type	`Unconfined`	string	
xfn.args	`'{}'`	object	
xfn.cache.configMap	`""`	string	
xfn.cache.medium	`""`	string	
xfn.cache.pvc	`""`	string	
xfn.cache.sizeLimit	`1Gi`	string	
xfn.enabled	`false`	boolean	

KEY	DEFAULT	TYPE	DESCRIPTION
xfn.extraEnvVars	`'{}'`	object	
xfn.image.pullPolicy	`IfNotPresent`	string	
xfn.image.repository	`crossplane/xfn`	string	
xfn.image.tag	`v1.11.0-up.1`	string	
xfn.resources.requests.cpu	`1000m`	string	
xfn.resources.requests.memory	`1Gi`	string	
xfn.resources.limits.memory	`2Gi`	string	
xfn.resources.limits.cpu	`2000m`	string	
resourcesCrossplane.limits.cpu	`500m`	string	
resourcesCrossplane.limits.memory	`1Gi`	string	
resourcesCrossplane.requests.cpu	`250m`	string	
resourcesCrossplane.requests.memory	`768Mi`	string	
resourcesRBACManager.limits.memory	`768Mi`	string	
resourcesRBACManager.limits.cpu	`100m`	string	
resourcesRBACManager.requests.cpu	`100m`	string	
resourcesRBACManager.requests.memory	`512Mi`	string	
configuration.packages		array	
deploymentStrategy	`RollingUpdate`	string	
extraEnvVarsCrossplane	`'{}'`	object	List of extra environment variables to set in the Crossplane deployment. For example, `extraEnvironmentVars: sample.key: value1 ANOTHER.KEY: value2` results with `- name: sample_key value: "value1" - name: ANOTHER_KEY value: "value2"`
registryCaBundleConfig	`'{}'`	object	

KEY	DEFAULT	TYPE	DESCRIPTION
billing.awsMarketplace.iamRoleARN	`arn:aws:iam::ACCOUNT-ID>:role/ROLE-NAME`	string	
billing.awsMarketplace.enabled	`false`	boolean	
image.pullPolicy	`IfNotPresent`	string	
image.repository	`upbound/crossplane`	string	
image.tag	`v1.11.0-up.1`	string	
packageCache.configMap	`""`	string	
packageCache.medium	`""`	string	
packageCache.pvc	`""`	string	
packageCache.sizeLimit	`5Mi`	string	
securityContextRBACManager.allowPrivilegeEscalation	`false`	boolean	
securityContextRBACManager.readOnlyRootFilesystem	`true`	boolean	
securityContextRBACManager.runAsGroup	`65532`	integer	
securityContextRBACManager.runAsUser	`65532`	integer	
rbacManager.managementPolicy	`Basic`	string	
rbacManager.nodeSelector	`'{}'`	object	
rbacManager.affinity	`'{}'`	object	
rbacManager.args	`'{}'`	object	
rbacManager.deploy	`true`	boolean	
rbacManager.leaderElection	`true`	boolean	

KEY	DEFAULT	TYPE	DESCRIPTION
rbacManager.replicas	`1`	integer	
rbacManager.skipAggregatedClusterRoles	`true`	boolean	
rbacManager.tolerations		array	
customAnnotations	`'{}'`	object	Custom annotations to add to the Crossplane deployment and pod.
customLabels	`'{}'`	object	Custom labels to add into metadata.
leaderElection	`true`	boolean	
nodeSelector	`'{}'`	object	
podSecurityContextCrossplane	`'{}'`	object	
webhooks.enabled	`false`	boolean	
args	`'{}'`	object	
podSecurityContextRBACManager	`'{}'`	object	
provider.packages		array	
tolerations		array	
upbound.controlPlane.permission	`edit`	string	
affinity	`'{}'`	object	
extraEnvVarsRBACManager	`'{}'`	object	List of extra environment variables to set in the Crossplane RBAC manager deployment. For example, `extraEnvironmentVars: sample.key: value1 ANOTHER.KEY: value2` results with `- name: sample_key value: "value1" - name: ANOTHER_KEY value: "value2"`
nameOverride	`crossplane`	string	
priorityClassName	`""`	string	

Version matrix for Crossplane

This topic provides you with a version matrix for the Crossplane package and its open source components in Tanzu Application Platform v1.9 (commonly known as TAP).

To view this information for another Tanzu Application Platform version, select the version from the drop-down menu at the top of this page.

The following table has the component versions for the Crossplane package.

Component	Version
Crossplane package	0.5.0
UXP	1.15.0-up.1
Upbound Crossplane	1.15.0-up.1
provider-helm	0.15.0
provider-kubernetes	0.12.1

Note

Tanzu Application Platform patch releases are only added to the table when there is a change to one or more of the other versions in the table. Otherwise, the corresponding versions remain the same for each Tanzu Application Platform patch release.

Crossplane limitations

This topic tells you about the limitations related to Crossplane on Tanzu Application Platform (commonly known as TAP).

For troubleshooting guidance, see Troubleshoot Crossplane.

Cluster performance degradation due to large number of CRDs

Take care before installing extra Crossplane Providers into Tanzu Application Platform. Some Providers install hundreds of additional CRDs into the cluster.

This is particularly true of the Providers for AWS, Azure, and GCP. For the number of CRDs installed with these Providers, see:

- provider-aws CRDs
- provider-azure CRDs
- provider-gcp CRDs

You must ensure that your cluster has sufficient resource to support this number of additional CRDs if you choose to install them.

Workaround:

To address this issue, Upbound have released a new feature named Provider Families. For more information, see the Crossplane blog.

Overview of Default roles for Tanzu Application Platform

Tanzu Application Platform (commonly known as TAP) v1.9 includes:

- Documentation for using your existing identity management solution for Kubernetes authentication. For more information, see Set up authentication for your Tanzu Application Platform deployment.
- Six default roles for providing authorization for users and service accounts within a namespace on a cluster that runs one of the Tanzu Application Platform profiles. For more

information about service accounts, see the Kubernetes documentation.

Default roles

Tanzu Application Platform has resources that are namespace scoped, such as workloads, pipelineruns, builds and resources that are cluster scoped, such as clusterbuilders, clustersupplychains, clusterimagetemplates. Tanzu Application Platform provides four roles that include both a namespace scoped role and a cluster scoped role:

Role name	Role description	Namespace scoped role name	Cluster scoped role name
app-editor	View, create, edit, and delete a Tanzu workload or deliverable.	app-editor	app-editor-cluster-access
app-viewer	Read-only for a Tanzu workload or deliverable.	app-viewer	app-view-cluster-access
app-operator	View, create, edit, and delete supply chain resources.	app-operator	app-operator-cluster-access
service-operator	View, create, edit, and delete service instances, service instance classes, and resource claim policies.	service-operator	service-operator-cluster-access

For more information about the different roles and their permissions, see Role descriptions for Tanzu Application Platform.

Additionally, the following two roles are available for service accounts for namespace scoped resources associated with Tanzu Supply Chains:

Role name	Namespace scoped role name
workload	workload
deliverable	deliverable

The default roles provide an opinionated starting point for the most common permissions that users need when using Tanzu Application Platform. However, as described in the Kubernetes documentation about RBAC, you can create customized roles and permissions that better meet your needs. Aggregated cluster roles are used to build VMware Tanzu Application Platform default roles.

Cluster admins must be careful when creating Roles or ClusterRoles. When changing roles or adding new roles that carry one of the labels used by the default roles, the roles are automatically updated to the aggregation state. It can lead to unintentional changes in functions and permissions to all users.

The default roles are installed with every Tanzu Application Platform profile except for `view`.

Assigning with roles using kubectl

For more information about assigning roles to users, see Bind a user or group to a default role.

Disclaimer

Tanzu Developer Portal does not make use of the described roles. Instead, it provides the user with view access for each cluster.

Overview of Default roles for Tanzu Application Platform

Tanzu Application Platform (commonly known as TAP) v1.9 includes:

- Documentation for using your existing identity management solution for Kubernetes authentication. For more information, see Set up authentication for your Tanzu Application Platform deployment.
- Six default roles for providing authorization for users and service accounts within a namespace on a cluster that runs one of the Tanzu Application Platform profiles. For more information about service accounts, see the Kubernetes documentation.

Default roles

Tanzu Application Platform has resources that are namespace scoped, such as workloads, pipelineruns, builds and resources that are cluster scoped, such as clusterbuilders, clustersupplychains, clusterimagetemplates. Tanzu Application Platform provides four roles that include both a namespace scoped role and a cluster scoped role:

Role name	Role description	Namespace scoped role name	Cluster scoped role name
app-editor	View, create, edit, and delete a Tanzu workload or deliverable.	app-editor	app-editor-cluster-access
app-viewer	Read-only for a Tanzu workload or deliverable.	app-viewer	app-view-cluster-access
app-operator	View, create, edit, and delete supply chain resources.	app-operator	app-operator-cluster-access
service-operator	View, create, edit, and delete service instances, service instance classes, and resource claim policies.	service-operator	service-operator-cluster-access

For more information about the different roles and their permissions, see Role descriptions for Tanzu Application Platform.

Additionally, the following two roles are available for service accounts for namespace scoped resources associated with Tanzu Supply Chains:

Role name	Namespace scoped role name
workload	workload
deliverable	deliverable

The default roles provide an opinionated starting point for the most common permissions that users need when using Tanzu Application Platform. However, as described in the Kubernetes documentation about RBAC, you can create customized roles and permissions that better meet your needs. Aggregated cluster roles are used to build VMware Tanzu Application Platform default roles.

Cluster admins must be careful when creating Roles or ClusterRoles. When changing roles or adding new roles that carry one of the labels used by the default roles, the roles are automatically updated to the aggregation state. It can lead to unintentional changes in functions and permissions to all users.

The default roles are installed with every Tanzu Application Platform profile except for `view`.

Assigning with roles using kubectl

For more information about assigning roles to users, see Bind a user or group to a default role.

Disclaimer

Tanzu Developer Portal does not make use of the described roles. Instead, it provides the user with view access for each cluster.

Set up authentication for your Tanzu Application Platform deployment

There are multiple ways to set up authentication for your Tanzu Application Platform (commonly known as TAP) deployment. You can manage authentication at the infrastructure level with your Kubernetes provider, such as Tanzu Kubernetes Grid, EKS, AKS, or GKE.

VMware recommends Pinniped for integrating your identity management into Tanzu Application Platform on multicloud. It provides many supported integrations for widely used identity providers. To use Pinniped, see Installing Pinniped on Tanzu Application Platform and Log in by using Pinniped.

See Integrating Azure Active Directory for Azure Active Directory Integration.

Tanzu Kubernetes Grid

For Tanzu Kubernetes Grid clusters, Pinniped is the default identity solution and is installed as a core package. For more information, see Core Packages and Enable Identity Management in an Existing Deployment in the Tanzu Kubernetes Grid documentation.

Set up authentication for your Tanzu Application Platform deployment

There are multiple ways to set up authentication for your Tanzu Application Platform (commonly known as TAP) deployment. You can manage authentication at the infrastructure level with your Kubernetes provider, such as Tanzu Kubernetes Grid, EKS, AKS, or GKE.

VMware recommends Pinniped for integrating your identity management into Tanzu Application Platform on multicloud. It provides many supported integrations for widely used identity providers. To use Pinniped, see Installing Pinniped on Tanzu Application Platform and Log in by using Pinniped.

See Integrating Azure Active Directory for Azure Active Directory Integration.

Tanzu Kubernetes Grid

For Tanzu Kubernetes Grid clusters, Pinniped is the default identity solution and is installed as a core package. For more information, see Core Packages and Enable Identity Management in an Existing Deployment in the Tanzu Kubernetes Grid documentation.

Install Pinniped on Tanzu Application Platform

Pinniped is used to support authentication on Tanzu Application Platform (commonly known as TAP). This topic tells you how to install Pinniped on a single cluster of Tanzu Application Platform.

Note

This topic only provides an example of one possible installation method for Pinniped on Tanzu Application Platform by using the default Contour ingress controler

included in the platform. See Pinniped documentation for more information about the specific installation method that suits your environment.

Use this topic to learn how to deploy two Pinniped components into the cluster:

- **Pinniped Supervisor:** An OIDC server which allows users to authenticate with an external identity provider (IDP). It hosts an API for the concierge component to fulfill authentication requests.
- **Pinniped Concierge:** A credential exchange API that takes a credential from an identity source, for example, Pinniped Supervisor, proprietary IDP, as input. The Pinniped Concierge authenticates the user by using the credential, and returns another credential that is parsable by the host Kubernetes cluster or by an impersonation proxy that acts on behalf of the user.

Prerequisites

Meet these prerequisites:

- Install the package `certmanager`. This is included in Tanzu Application Platform.
- Install the package `contour`. This is included in Tanzu Application Platform.
- Create a `workspace` directory to function as your workspace.

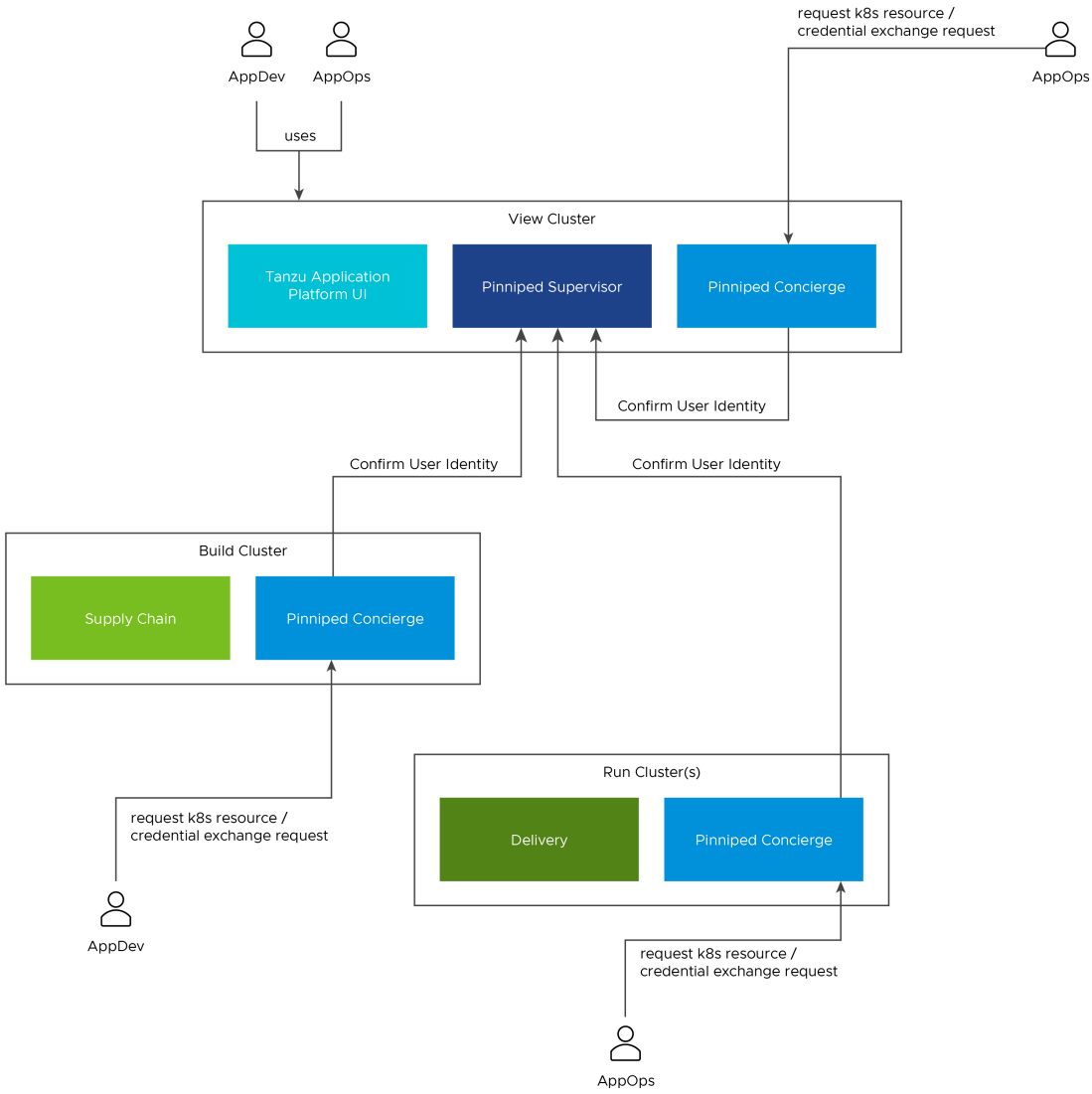
Environment planning

If you run Tanzu Application Platform on a single cluster, both Pinniped Supervisor and Pinniped Concierge are installed to this cluster.

When running a multicluster setup, you must decide which cluster to deploy the Supervisor onto. Furthermore, every cluster must have the Concierge deployed. Pinniped Supervisor runs as a central component that is consumed by multiple Pinniped Concierge instances. As a result, Pinniped Supervisor must be deployed to a single cluster that meets the prerequisites. You can deploy Pinniped Supervisor to the View Cluster of your Tanzu Application Platform, because it is a central single instance cluster. For more information, see Overview of multicluster Tanzu Application Platform.

You must deploy the Pinniped Concierge to every cluster that you want to enable authentication for, including the View Cluster itself.

See the following diagram for a possible deployment model:



For more information about the Pinniped architecture and deployment model, see Pinniped documentation.

Install Pinniped Supervisor by using Let's Encrypt

Follow these steps to install `pinniped-supervisor`:

1. Switch tooling to the desired cluster.
2. Create the necessary certificate files.
3. Create the Ingress resources.
4. Create the `pinniped-supervisor` configuration.
5. Apply these resources to the cluster.

Create Certificates (`letsencrypt` or `cert-manager`)

Choose a fully qualified domain name (FQDN) that can resolve to the Contour instance in the `tanzu-system-ingress` namespace. The FQDN `pinniped-supervisor.example.com` is used in the following sections.

Create a `ClusterIssuer` for `letsencrypt` and a TLS certificate resource for Pinniped Supervisor by creating the following resources and saving them into `workspace/pinniped-`

`supervisor/certificates.yaml`:

```
---
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-staging
  namespace: cert-manager
spec:
  acme:
    email: "EMAIL"
    privateKeySecretRef:
      name: letsencrypt-staging
    server: https://acme-staging-v02.api.letsencrypt.org/directory
    solvers:
    - http01:
        ingress:
          class: contour

---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: pinniped-supervisor-cert
  namespace: pinniped-supervisor
spec:
  secretName: pinniped-supervisor-tls-cert
  dnsNames:
  - "DNS-NAME"
  issuerRef:
    name: letsencrypt-staging
    kind: ClusterIssuer
```

Where:

- `EMAIL` is the user email address for `letsencrypt`. For example, `your-mail@example.com`
- `DNS-NAME` is the domain in which the `pinniped-supervisor` is published. For example, `pinniped-supervisor.example.com`

Create Ingress resources

Create a Service and Ingress resource to make the `pinniped-supervisor` accessible from outside the cluster.

To do so, create the following resources and save them into `workspace/pinniped-supervisor/ingress.yaml`:

```
---
apiVersion: v1
kind: Service
metadata:
  name: pinniped-supervisor
  namespace: pinniped-supervisor
spec:
  ports:
  - name: pinniped-supervisor
    port: 8443
    protocol: TCP
    targetPort: 8443
  selector:
    app: pinniped-supervisor

---
apiVersion: projectcontour.io/v1
```

```
kind: HTTPProxy
metadata:
  name: pinniped-supervisor
  namespace: pinniped-supervisor
spec:
  virtualhost:
    fqdn: "DNS-NAME"
    tls:
      passthrough: true
  tcpproxy:
    services:
    - name: pinniped-supervisor
      port: 8443
```

Where:

- `DNS-NAME` is the domain in which the `pinniped-supervisor` is published. For example, `pinniped-supervisor.example.com`
- `tls.passthrough: true` specifies that the TLS connection is forwarded to and terminated in the supervisor pod.

Create the `pinniped-supervisor` configuration

Create a `FederationDomain` to link the concierge to the supervisor instance and configure an `OIDCIdentityProvider` to connect the supervisor to your OIDC Provider. The following example uses `auth0` as the `OIDCIdentityProvider`. For more information about how to configure different identity providers, including OKTA, GitLab, OpenLDAP, Dex, Microsoft AD and more, see Pinniped documentation.

To create the `pinniped-supervisor` configuration, create the following resources and save them into `workspace/pinniped-supervisor/oidc_identity_provider.yaml`:

```
apiVersion: idp.supervisor.pinniped.dev/v1alpha1
kind: OIDCIdentityProvider
metadata:
  namespace: pinniped-supervisor
  name: auth0
spec:
  # Specify the upstream issuer URL associated with your auth0 application.
  issuer: https://"APPLICATION-SUBDOMAIN".auth0.com/

  # Specify how to form authorization requests.
  authorizationConfig:
    additionalScopes: ["openid", "email"]
    allowPasswordGrant: false

  # Specify how claims are mapped to Kubernetes identities. This varies by provider.
  claims:
    username: email
    groups: groups

  # Specify the name of the Kubernetes Secret that contains your
  # application's client credentials (created as follows).
  client:
    secretName: auth0-client-credentials

---
apiVersion: v1
kind: Secret
metadata:
  namespace: pinniped-supervisor
  name: auth0-client-credentials
type: secrets.pinniped.dev/oidc-client
```

```
stringData:
  clientID: "AUTH0-CLIENT-ID"
  clientSecret: "AUTH0-CLIENT-SECRET"

---
apiVersion: config.supervisor.pinniped.dev/v1alpha1
kind: FederationDomain
metadata:
  name: pinniped-supervisor-federation-domain
  namespace: pinniped-supervisor
spec:
  issuer: "DNS-NAME"
  tls:
    secretName: pinniped-supervisor-tls-cert
```

Where:

- `APPLICATION-SUBDOMAIN` is the application specific subdomain that is assigned after the application registration.
- `AUTH0-CLIENT-ID` and `AUTH0-CLIENT-SECRET` are the credentials retrieved from the application registration.
- `DNS-NAME` is the domain in which the `pinniped-supervisor` is published. For example, `pinniped-supervisor.example.com`

Apply the resources

After creating the resource files, you can install them into the cluster. Follow these steps to deploy them as a kapp application:

1. Install the `pinniped-supervisor` by running:

   ```
   kapp deploy -y --app pinniped-supervisor -f pinniped-supervisor -f https://get.
   pinniped.dev/v0.22.0/install-pinniped-supervisor.yaml
   ```

   Note

   To keep the security patches up to date, you must install the most recent version of Pinniped. See Vmware Tanzu Pinniped Releases in GitHub for more information.

2. Get the external IP address of Ingress by running:

   ```
   kubectl -n tanzu-system-ingress get svc/envoy -o jsonpath='{.status.loadBalance
   r.ingress[0].ip}'
   ```

3. If not already covered by the Tanzu Application Platform wildcard DNS entry, add an entry to the DNS system to bind the external IP address with.

Switch to production issuer (`letsencrypt` or `cert-manager`)

Follow these steps to switch to a `letsencrypt` production issuer so the generated TLS certificate is recognized as valid by web browsers and clients:

1. Edit the ClusterIssuer for `letsencrypt` and add TLS certificate resource for `pinniped-supervisor` by creating or updating the following resources and saving them into `workspace/pinniped-supervisor/certificates.yaml`:

```
---
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-prod
  namespace: cert-manager
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    email: "EMAIL"
    privateKeySecretRef:
      name: letsencrypt-prod
    solvers:
    - http01:
        ingress:
          class: contour

---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: pinniped-supervisor-cert
  namespace: pinniped-supervisor
spec:
  secretName: pinniped-supervisor-tls-cert
  dnsNames:
  - "DNS-NAME"
  issuerRef:
    name: letsencrypt-prod
    kind: ClusterIssuer
```

Where:

- `EMAIL` is the user email address for `letsencrypt`. For example, `your-mail@example.com`
- `DNS-NAME` is the domain in which the `pinniped-supervisor` is published. For example, `pinniped-supervisor.example.com`

2. Create or update the `pinniped-supervisor` kapp application:

```
kapp deploy -y --app pinniped-supervisor -f pinniped-supervisor -f https://get.
pinniped.dev/v0.22.0/install-pinniped-supervisor.yaml
```

Install Pinniped Supervisor Private CA

Follow these steps to install `pinniped-supervisor`:

1. Switch tooling to the desired cluster.
2. Create the necessary certificate files.
3. Create the Ingress resources.
4. Create the `pinniped-supervisor` configuration.
5. Apply these resources to the cluster.

Create Certificate Secret

Choose a fully qualified domain name (FQDN) that can resolve to the Contour instance in the `tanzu-system-ingress` namespace. Create a certificate by using a CA that the clients trust. This FQDN can be under the `ingress_domain` in the TAP values file, or a dedicated DNS entry. The FQDN `pinniped-supervisor.example.com` is used in the following sections.

After the certificate files are available, they must be encoded to base64 format in a single-line layout. For example, you can encode the certificate file `my.crt` by running:

```
cat my.crt | base64 -w 0
```

Create the following resource and save it into `workspace/pinniped-supervisor/ingress.yaml`:

```
---
apiVersion: v1
kind: Secret
metadata:
  name: pinniped-supervisor-tls-cert
  namespace: pinniped-supervisor
type: kubernetes.io/tls
data:
  tls.crt: PRIVATE-KEY
  tls.key: PUBLIC-KEY
```

Where:

- `PRIVATE-KEY` is the base64 encoded public key.
- `PUBLIC-KEY` is the base64 encoded public key.

Create Ingress resources

Create a Service and Ingress resource to make the `pinniped-supervisor` accessible from outside the cluster.

To do so, create the following resources and save them into `workspace/pinniped-supervisor/ingress.yaml`:

```
---
apiVersion: v1
kind: Service
metadata:
  name: pinniped-supervisor
  namespace: pinniped-supervisor
spec:
  ports:
  - name: pinniped-supervisor
    port: 8443
    protocol: TCP
    targetPort: 8080
  selector:
    app: pinniped-supervisor

---
apiVersion: projectcontour.io/v1
kind: HTTPProxy
metadata:
  name: pinniped-supervisor
  namespace: pinniped-supervisor
spec:
  virtualhost:
    fqdn: "DNS-NAME"
    tls:
      passthrough: true
  tcpproxy:
    services:
    - name: pinniped-supervisor
      port: 8443
```

Where:

- `DNS-NAME` is the domain in which the `pinniped-supervisor` is published. For example, `pinniped-supervisor.example.com`
- `tls.passthrough: true` specifies that the TLS connection is forwarded to and terminated in the supervisor pod.

Create the `pinniped-supervisor` configuration

Create a `FederationDomain` to link the concierge to the supervisor instance and configure an `OIDCIdentityProvider` to connect the supervisor to your OIDC Provider. The following example uses `auth0` as the `OIDCIdentityProvider`. For more information about how to configure different identity providers, including OKTA, GitLab, OpenLDAP, Dex, Microsoft AD and more, see Pinniped documentation.

To create the `pinniped-supervisor` configuration, create the following resources and save them into `workspace/pinniped-supervisor/oidc_identity_provider.yaml`:

```
apiVersion: idp.supervisor.pinniped.dev/v1alpha1
kind: OIDCIdentityProvider
metadata:
  namespace: pinniped-supervisor
  name: auth0
spec:
  # Specify the upstream issuer URL associated with your auth0 application.
  issuer: https://"APPLICATION-SUBDOMAIN".auth0.com/

  # Specify how to form authorization requests.
  authorizationConfig:
    additionalScopes: ["openid", "email"]
    allowPasswordGrant: false

  # Specify how claims are mapped to Kubernetes identities. This varies by provider.
  claims:
    username: email
    groups: groups

  # Specify the name of the Kubernetes Secret that contains your
  # application's client credentials (created as follows).
  client:
    secretName: auth0-client-credentials

---
apiVersion: v1
kind: Secret
metadata:
  namespace: pinniped-supervisor
  name: auth0-client-credentials
type: secrets.pinniped.dev/oidc-client
stringData:
  clientID: "AUTH0-CLIENT-ID"
  clientSecret: "AUTH0-CLIENT-SECRET"

---
apiVersion: config.supervisor.pinniped.dev/v1alpha1
kind: FederationDomain
metadata:
  name: pinniped-supervisor-federation-domain
  namespace: pinniped-supervisor
spec:
  issuer: "DNS-NAME"
  tls:
    secretName: pinniped-supervisor-tls-cert
```

Where:

- `APPLICATION-SUBDOMAIN` is the application specific subdomain that is assigned after the application registration.
- `AUTH0-CLIENT-ID` and `AUTH0-CLIENT-SECRET` are the credentials retrieved from the application registration.
- `DNS-NAME` is the domain in which the pinniped-supervisor is published. For example, `pinniped-supervisor.example.com`

Apply the resources

After creating the resource files, you can install them into the cluster. Follow these steps to deploy them as a kapp application:

1. Install the supervisor by running:

```
kapp deploy -y --app pinniped-supervisor -f pinniped-supervisor -f https://get.
pinniped.dev/v0.22.0/install-pinniped-supervisor.yaml
```

> **Note**
>
> To keep the security patches up to date, you must install the most recent version of Pinniped. See Vmware Tanzu Pinniped Releases in GitHub for more information.

2. Get the external IP address of Ingress by running:

```
kubectl -n tanzu-system-ingress get svc/envoy -o jsonpath='{.status.loadBalance
r.ingress[0].ip}'
```

3. If not already covered by a Tanzu Application Platform wildcard DNS entry, add an entry to the DNS system to bind the external IP address with.

Install Pinniped Concierge

To install Pinniped Concierge:

1. Switch tooling to the desired cluster.
2. Deploy the Pinniped Concierge by running:

```
kapp deploy -y --app pinniped-concierge \
  -f https://get.pinniped.dev/v0.22.0/install-pinniped-concierge.yaml
```

3. Get the CA certificate of the supervisor by running the following command against the cluster running the `pinniped-supervisor`:

```
kubectl get secret pinniped-supervisor-tls-cert -n pinniped-supervisor -o 'go-t
emplate={{index .data "tls.crt"}}'
```

> **Note**
>
> The `tls.crt` contains the entire certificate chain including the CA certificate for `letsencrypt` generated certificates.

4. Create the following resource to `workspace/pinniped-concierge/jwt_authenticator.yaml`:

```
---
apiVersion: authentication.concierge.pinniped.dev/v1alpha1
kind: JWTAuthenticator
metadata:
  name: pinniped-jwt-authenticator
spec:
  issuer: "DNS-NAME"
  audience: concierge
  tls:
    certificateAuthorityData: "CA-DATA"
```

If you use the `letsencrypt` production issuer, you can omit the `tls` section:

```
---
apiVersion: authentication.concierge.pinniped.dev/v1alpha1
kind: JWTAuthenticator
metadata:
  name: pinniped-jwt-authenticator
spec:
  issuer: "DNS-NAME"
  audience: concierge
```

Where:

- `DNS-NAME` is the domain in which the `pinniped-supervisor` is published. For example, `pinniped-supervisor.example.com`
- `CA-DATA` is the public key of the signing CA or the public key of the Pinniped httpproxy certificate.

5. Deploy the resource by running:

```
kapp deploy -y --app pinniped-concierge-jwt --into-ns pinniped-concierge -f pin
niped-concierge/jwt_authenticator.yaml
```

Log in to the cluster

See Log in by using Pinniped.

Integrate your Azure Active Directory

This topic tells you how to integrate your Azure Active Directory (commonly known as AD).

Integrate Azure AD with a new or existing AKS without Pinniped

Perform the following procedures to integrate Azure AD with a new or existing AKS without Pinniped.

Prerequisites

Download and install the Azure CLI.

Set up a platform operator

To set up a platform operator:

1. Navigate to the **Azure Active Directory Overview** page.
2. Select **Groups** under the **Manage** side menu.

3. Identify or create an admin group for the AKS cluster.
4. Retrieve the object ID of the admin group.
5. Take one of the following actions.
    - Create an AKS Cluster with Azure AD enabled by running:

      ```
      az group create --name RESOURCE-GROUP --location LOCATION
      az aks create -g RESOURCE-GROUP -n MANAGED-CLUSTER --enable-aad --aad-adm
      in-group-object-ids OBJECT-ID
      ```

      Where:

        - `RESOURCE-GROUP` is your resource group
        - `LOCATION` is your location
        - `MANAGED-CLUSTER` is your managed cluster
        - `OBJECT-ID` is the object ID
    - Enable Azure AD integration on the existing cluster by running:

      ```
      az aks update -g RESOURCE-GROUP -n MANAGED-CLUSTER --enable-aad --aad-adm
      in-group-object-ids OBJECT-ID
      ```

      Where:

        - `RESOURCE-GROUP` is your resource group
        - `MANAGED-CLUSTER` is your managed cluster
        - `OBJECT-ID` is the object ID
6. Add **Platform Operators** to the admin group.
7. Log in to the AKS cluster by running:

   ```
   az aks get-credentials --resource-group RESOURCE-GROUP --name MANAGED-CLUSTER -
   -admin
   ```

   Where:

    - `RESOURCE-GROUP` is your resource group
    - `MANAGED-CLUSTER` is your managed cluster

Set up a Tanzu Application Platform default role group

To set up a Tanzu Application Platform default role group:

1. Navigate to the **Azure Active Directory Overview** page.
2. Select **Groups** under the **Manage** side menu.
3. Identify or create a list of groups in the Azure AD for each of the Tanzu Application Platform default roles (`app-operator`, `app-viewer`, and `app-editor`).
4. Retrieve the corresponding object IDs for each group.
5. Add users to the groups accordingly.
6. For each object ID retrieved earlier, use the Tanzu CLI RBAC plug-in to bind the `object id` group to a role by running:

   ```
   tanzu rbac binding add -g OBJECT-ID -r TAP-ROLE -n NAMESPACE
   ```

Where:

- `OBJECT-ID` is the object ID
- `TAP-ROLE` is the Tanzu Application Platform role
- `NAMESPACE` is the namespace

Set up kubeconfig

To set up kubeconfig:

1. Set up the `kubeconfig` to point to the AKS cluster by running:

   ```
   az aks get-credentials --resource-group RESOURCE-GROUP --name MANAGED-CLUSTER
   ```

   Where:

   - `RESOURCE-GROUP` is your resource group
   - `MANAGED-CLUSTER` is your managed cluster

2. Run any kubectl command to trigger a browser login. For example:

   ```
   kubectl get pods
   ```

Integrate Azure AD with Pinniped

Perform the following procedures to set up Azure AD with Pinniped.

Prerequisites

Install Pinniped supervisor and concierge on the cluster without setting up the OIDCIdentityProvider and secret.

Set up the Azure AD app

To set up the Azure AD app:

1. Navigate to the **Azure Active Directory Overview** page.
2. Select **App registrations** under the **Manage** side menu.
3. Select **New Registration**.
4. Enter the name of the application. For example, `gke-pinniped-supervisor-app`.
5. Under **Supported account types**, select **Accounts in this organisational directory only (VMware, Inc. only - Single tenant)**.
6. Under **Redirect URI**, select **Web** as the platform.
7. Enter the call URI to the supervisor. For example, `https://pinniped-supervisor.example.com/callback`.
8. Select **Register** to create the app.
9. If not already redirected, navigate to the app settings page.
10. Select **Token configuration** under the **Manage** menu.
11. Select **Add groups claim** > **All groups (includes distribution lists but not groups assigned to the application)**.
12. Select **Add** to create the group claim.

13. Select the app name in the breadcrumb navigation to return to the app settings page.
14. Select the **Endpoints** tab and record the value in the **OpenID Connect metadata document** field.
15. Return to the app settings page.
16. Record the **Application (client) ID**.
17. Select **Certificates & secrets** under the **Manage** menu.
18. Create a new client secret and record this value.
19. Add the following YAML to `oidc_identity_provider.yaml`.

    ```
    ---
    apiVersion: idp.supervisor.pinniped.dev/v1alpha1
    kind: OIDCIdentityProvider
    metadata:
      namespace: pinniped-supervisor
      name: azure-ad
    spec:
      # Specify the upstream issuer URL.
      issuer: ISSUER-URL

      authorizationConfig:
        additionalScopes: ["openid", "email", "profile"]
        allowPasswordGrant: false

      # Specify how claims are mapped to Kubernetes identities.
      claims:
        username: preferred_username
        groups: groups

      # Specify the name of the Kubernetes Secret that contains your
      # application's client credentials (created below).
      client:
        secretName: azure-ad-client-credentials
    ---
    apiVersion: v1
    kind: Secret
    metadata:
      namespace: pinniped-supervisor
      name: azure-ad-client-credentials
    type: secrets.pinniped.dev/oidc-client
    stringData:
      clientID: "AZURE-AD-CLIENT-ID"
      clientSecret: "AZURE-AD-CLIENT-SECRET"
    ```

    Where:

    - `ISSUER-URL` is the OpenID Connect metadata document URL you recorded earlier, but without the trailing `/.well-known/openid-configuration`
    - `AZURE-AD-CLIENT-ID` is the Azure AD client ID you recorded earlier
    - `AZURE-AD-CLIENT-SECRET` is the Azure AD client secret you recorded earlier

20. Apply your changes from the kubectl CLI by running:

    ```
    kubectl apply workspace/pinniped-supervisor/oidc_identity_provider.yaml
    ```

Set up the Tanzu Application Platform default role group

To set up a Tanzu Application Platform default role group:

1. Navigate to the **Azure Active Directory Overview** page.
2. Select **Groups** under the **Manage** side menu.
3. Identify or create a list of groups in the Azure AD for each of the Tanzu Application Platform default roles (`app-operator`, `app-viewer`, and `app-editor`).
4. Retrieve the corresponding object IDs for each group.
5. Add users to the groups accordingly.
6. For each object ID retrieved earlier, use kubectl to bind the `object id` group to a role by running:

   ```
   kubectl apply -n DEVELOPER_NAMESPACE -f - << EOF
   ---
   apiVersion: rbac.authorization.k8s.io/v1
   kind: RoleBinding
   metadata:
     name: TAP-ROLE
     namespace: $DEVELOPER_NAMSPACE
   roleRef:
     apiGroup: rbac.authorization.k8s.io
     kind: ClusterRole
     name: TAP-ROLE
   subjects:
   - apiGroup: rbac.authorization.k8s.io
     kind: Group
     name: OBJECT-ID
   EOF
   ---
   apiVersion: rbac.authorization.k8s.io/v1
   kind: ClusterRoleBinding
   metadata:
     name: TAP-ROLE-cluster-access
   roleRef:
     apiGroup: rbac.authorization.k8s.io
     kind: ClusterRole
     name: TAP-ROLE
   subjects:
   - apiGroup: rbac.authorization.k8s.io
     kind: Group
     name: OBJECT-ID
   ```

   Where:

   - `OBJECT-ID` is the object ID.
   - `TAP-ROLE` is the Tanzu Application Platform role.
   - `DEVELOPER-NAMESPACE` is the namespace for scoping the group.

For more information about binding users or groups to roles, see Bind a user or group to a default role.

Set up kubeconfig

Follow these steps to set up kubeconfig:

1. Set up `kubeconfig` using the Pinniped CLI by running:

   ```
   pinniped get kubeconfig --kubeconfig-context YOUR-KUBECONFIG-CONTEXT > /tmp/con
   cierge-kubeconfig
   ```

   Where `YOUR-KUBECONFIG-CONTEXT` is your your kubeconfig context.

2. Run any kubectl command to trigger a browser login. For example:

```
export KUBECONFIG="/tmp/concierge-kubeconfig"
kubectl get pods
```

Role descriptions for Tanzu Application Platform

This topic is a high level overview of each default role. For more information about the specific permissions of each role for every Tanzu Application Platform (commonly known as TAP) component, see Detailed role permissions for Tanzu Application Platform.

app-editor

The app-editor role can create, edit, and delete a Tanzu workload or deliverable.

Assign this role to a user, for example an app developer, to give permissions to create running workloads on the cluster. This allows them to deploy their applications. This role allows the user to:

- View, create, update, or delete a Tanzu workload or deliverable. This includes viewing the logs of the pods spun up through the Tanzu workload and tracing a commit through the build process.
- Download the images associated with their Tanzu workload so they can test images locally, or create a Tanzu workload from it instead of starting from source code in a repository.
- View and use Application Accelerator templates.
- View, create, update, or delete a Tanzu workload binding with an existing service.

app-viewer

The app-viewer role cannot create, edit, or delete a Tanzu workload or deliverable.

This role has a subset of the permissions of the app-editor role. Use it if you do not want a user to create, edit, or delete a Tanzu workload or deliverable, but they need to view its status. For example, give these permissions to an application developer that requires visibility into the state of their Tanzu workload or micro-service, but does not have the permissions to deploy it, such as to production or staging environments. This role cannot bind services with a Tanzu workload.

app-operator

The app-operator role can create, edit, and delete supply chain resources.

Assign this role to a user who defines the activities within a supply chain or the path to production. For example, building, testing, or scanning. This role can view, create, update, or delete Tanzu supply chain resources, including Tanzu Build Service control plane resources such as:

- kpack's builder, stack, and store
- Scanning resources
- Grype
- The metadata store

If this person must create Tanzu workloads, you can bind the user with the app-editor role as well.

service-operator

The service-operator role can create, edit, and delete service instances, service instance classes, and resource claim policies to permit the claimability of service instances across one or more namespaces.

Assign this role to a user who is responsible for the life cycle (create, edit and delete) of service instances. This role can also view resource claims across all namespaces as well as query for the list of claimable service instances in a given namespace.

workload

This role provides the service account associated with the Tanzu workload the permissions needed to execute the activities in the supply chain. This role is for a "robot" versus a user.

deliverable

This role gives the delivery “robot” service account the permissions needed to create running workloads. This role is not for a user.

Role descriptions for Tanzu Application Platform

This topic is a high level overview of each default role. For more information about the specific permissions of each role for every Tanzu Application Platform (commonly known as TAP) component, see Detailed role permissions for Tanzu Application Platform.

app-editor

The app-editor role can create, edit, and delete a Tanzu workload or deliverable.

Assign this role to a user, for example an app developer, to give permissions to create running workloads on the cluster. This allows them to deploy their applications. This role allows the user to:

- View, create, update, or delete a Tanzu workload or deliverable. This includes viewing the logs of the pods spun up through the Tanzu workload and tracing a commit through the build process.
- Download the images associated with their Tanzu workload so they can test images locally, or create a Tanzu workload from it instead of starting from source code in a repository.
- View and use Application Accelerator templates.
- View, create, update, or delete a Tanzu workload binding with an existing service.

app-viewer

The app-viewer role cannot create, edit, or delete a Tanzu workload or deliverable.

This role has a subset of the permissions of the app-editor role. Use it if you do not want a user to create, edit, or delete a Tanzu workload or deliverable, but they need to view its status. For example, give these permissions to an application developer that requires visibility into the state of their Tanzu workload or micro-service, but does not have the permissions to deploy it, such as to production or staging environments. This role cannot bind services with a Tanzu workload.

app-operator

The app-operator role can create, edit, and delete supply chain resources.

Assign this role to a user who defines the activities within a supply chain or the path to production. For example, building, testing, or scanning. This role can view, create, update, or delete Tanzu

supply chain resources, including Tanzu Build Service control plane resources such as:

- kpack's builder, stack, and store
- Scanning resources
- Grype
- The metadata store

If this person must create Tanzu workloads, you can bind the user with the app-editor role as well.

service-operator

The service-operator role can create, edit, and delete service instances, service instance classes, and resource claim policies to permit the claimability of service instances across one or more namespaces.

Assign this role to a user who is responsible for the life cycle (create, edit and delete) of service instances. This role can also view resource claims across all namespaces as well as query for the list of claimable service instances in a given namespace.

workload

This role provides the service account associated with the Tanzu workload the permissions needed to execute the activities in the supply chain. This role is for a "robot" versus a user.

deliverable

This role gives the delivery "robot" service account the permissions needed to create running workloads. This role is not for a user.

Detailed role permissions for Tanzu Application Platform

This topic tells you the specific permissions of each role for every Tanzu Application Platform (commonly known as TAP) component.

Native Kubernetes Resources

`apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"`

```
- apiGroups: [""]
  resources: ["configmaps","endpoints","events","persistentvolumeclaims","pods","pods/
log","resourcequotas","services"]
  verbs: ["get","list","watch"]
- apiGroups: ["apps"]
  resources: ["deployments","replicasets","statefulsets"]
  verbs: ["get","list","watch"]
- apiGroups: ["batch"]
  resources: ["cronjobs","jobs"]
  verbs: ["get","list","watch"]
- apiGroups: ["events.k8s.io"]
  resources: ["events"]
  verbs: ["get","list","watch"]
- apiGroups: ["networking.k8s.io"]
  resources: ["ingresses"]
  verbs: ["get","list","watch"]
```

`apps.tanzu.vmware.com/aggregate-to-app-operator: "true"`

```
- apiGroups: [""]
  resources: ["configmaps","secrets"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
```

App Accelerator

apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"

```
- apiGroups: ["accelerator.apps.tanzu.vmware.com"]
  resources: ["accelerators"]
  verbs: ["get","list","watch"]
```

apps.tanzu.vmware.com/aggregate-to-app-operator: "true"

```
- apiGroups: ["accelerator.apps.tanzu.vmware.com"]
  resources: ["accelerators"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
```

Cartographer

apps.tanzu.vmware.com/aggregate-to-app-editor: "true"

```
- apiGroups: ["carto.run"]
  resources: ["deliverables","workloads"]
  verbs: ["create","patch","update","delete","deletecollection"]
```

apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"

```
- apiGroups: ["carto.run"]
  resources: ["deliverables","runnables","workloads"]
  verbs: ["get","list","watch"]
```

apps.tanzu.vmware.com/aggregate-to-app-viewer-cluster-access: "true"

```
- apiGroups: ["carto.run"]
  resources: ["clusterconfigtemplates","clusterconfigtemplates","clusterdeliveries","c
lusterdeploymenttemplates","clusterimagetemplates","clusterruntemplates","clustersourc
etemplates","clustersupplychains","clustertemplates"]
  verbs: ["get","list","watch"]
```

apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access

```
- apiGroups: ["carto.run"]
  resources: ["clusterconfigtemplates","clusterconfigtemplates","clusterdeliveries","c
lusterdeploymenttemplates","clusterimagetemplates","clusterruntemplates","clustersourc
etemplates","clustersupplychains","clustertemplates"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
```

Cloud Native Runtimes

apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"

```
- apiGroups: ["apps"]
  resources: ["deployments","replicasets","statefulsets"]
  verbs: ["get","list","watch"]
- apiGroups: ["batch"]
  resources: ["cronjobs","jobs"]
  verbs: ["get","list","watch"]
- apiGroups: ["networking.k8s.io"]
  resources: ["ingresses"]
  verbs: ["get","list","watch"]
- apiGroups: ["serving.knative.dev"]
  resources: ["configurations","services","revisions","routes"]
  verbs: ["get","list","watch"]
- apiGroups: ["sources.*"]
  resources: ["(many)"]
  verbs: ["get","list","watch"]
```

Convention Service

apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"

```
- apiGroups: ["conventions.carto.run"]
  resources: ["podintents"]
  verbs: ["get","list","watch"]
- apiGroups: ["conventions.apps.tanzu.vmware.com"]
  resources: ["podintents"]
  verbs: ["get","list","watch"]
```

apps.tanzu.vmware.com/aggregate-to-app-viewer-cluster-access: "true"

```
- apiGroups: ["conventions.carto.run"]
  resources: ["clusterpodconventions"]
  verbs: ["get","list","watch"]
- apiGroups: ["conventions.apps.tanzu.vmware.com"]
  resources: ["clusterpodconventions"]
  verbs: ["get","list","watch"]
```

apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access

```
- apiGroups: ["conventions.carto.run"]
  resources: ["clusterpodconventions"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["conventions.apps.tanzu.vmware.com"]
  resources: ["clusterpodconventions"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
```

Developer Conventions

apps.tanzu.vmware.com/aggregate-to-app-editor: "true"

```
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get","list","watch"]
- apiGroups: [""]
  resources: ["pods/exec","pods/portforward"]
  verbs: ["get","list","create"]
- apiGroups: ["carto.run"]
```

```
  resources: ["workloads"]
  verbs: ["get","list","watch"]
```

OOTB Templates

apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"

```
- apiGroups: [""]
  resources: ["configmaps"]
  verbs: ["get","list","watch"]
- apiGroups: ["carto.run"]
  resources: ["deliverables","runnables"]
  verbs: ["get","list","watch"]
- apiGroups: ["conventions.carto.run"]
  resources: ["podintents"]
- apiGroups: ["conventions.apps.tanzu.vmware.com"]
  resources: ["podintents"]
  verbs: ["get","list","watch"]
- apiGroups: ["kappctrl.k14s.io"]
  resources: ["apps"]
  verbs: ["get","list","watch"]
- apiGroups: ["kpack.io"]
  resources: ["images"]
  verbs: ["get","list","watch"]
- apiGroups: ["scanning.apps.tanzu.vmware.com"]
  resources: ["imagescans","sourcescans"]
  verbs: ["get","list","watch"]
- apiGroups: ["servicebinding.io"]
  resources: ["servicebindings"]
  verbs: ["get","list","watch"]
- apiGroups: ["services.apps.tanzu.vmware.com"]
  resources: ["resourceclaims"]
  verbs: ["get","list","watch"]
- apiGroups: ["serving.knative.dev"]
  resources: ["services"]
  verbs: ["get","list","watch"]
- apiGroups: ["source.apps.tanzu.vmware.com"]
  resources: ["imagerepositories","mavenartifacts"]
  verbs: ["get","list","watch"]
- apiGroups: ["source.toolkit.fluxcd.io"]
  resources: ["gitrepositories"]
  verbs: ["get","list","watch"]
- apiGroups: ["tekton.dev"]
  resources: ["pipelineruns","taskruns"]
  verbs: ["get","list","watch"]
```

apps.tanzu.vmware.com/aggregate-to-workload: "true"

```
- apiGroups: ["carto.run"]
  resources: ["deliverables","runnables"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["conventions.carto.run"]
  resources: ["podintents"]
- apiGroups: ["conventions.apps.tanzu.vmware.com"]
  resources: ["podintents"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["kpack.io"]
  resources: ["images"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["scanning.apps.tanzu.vmware.com"]
  resources: ["imagescans","sourcescans"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["source.apps.tanzu.vmware.com"]
```

```
  resources: ["imagerepositories","mavenartifacts"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["source.toolkit.fluxcd.io"]
  resources: ["gitrepositories"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["tekton.dev"]
  resources: ["pipelineruns","taskruns"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
```

`apps.tanzu.vmware.com/aggregate-to-deliverable: "true"`

```
- apiGroups: [""]
  resources: ["configmaps"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["kappctrl.k14s.io"]
  resources: ["apps"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["servicebinding.io"]
  resources: ["servicebindings"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["services.apps.tanzu.vmware.com"]
  resources: ["resourceclaims"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["serving.knative.dev"]
  resources: ["services"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["source.apps.tanzu.vmware.com"]
  resources: ["imagerepositories","mavenartifacts"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["source.toolkit.fluxcd.io"]
  resources: ["gitrepositories"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
```

Service Bindings

`apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"`

```
- apiGroups: ["servicebinding.io"]
  resources: ["servicebindings"]
  verbs: ["get","list","watch"]
```

Services Toolkit

`apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"`

```
- apiGroups: ["services.apps.tanzu.vmware.com"]
  resources: ["resourceclaims"]
  verbs: ["get","list","watch"]
```

`apps.tanzu.vmware.com/aggregate-to-app-viewer-cluster-access: "true"`

```
- apiGroups: ["services.apps.tanzu.vmware.com"]
  resources: ["clusterresources"]
  verbs: ["get","list","watch"]
```

`apps.tanzu.vmware.com/aggregate-to-app-operator: "true"`

```
- apiGroups: ["services.apps.tanzu.vmware.com"]
  resources: ["resourceclaims"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
```

`apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access`

```
- apiGroups: ["services.apps.tanzu.vmware.com"]
  resources: ["clusterresources"]
  verbs: ["get","list","watch"]
```

Source Controller

`apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"`

```
- apiGroups: ["source.apps.tanzu.vmware.com"]
  resources: ["imagerepositories","mavenartifacts"]
  verbs: ["get","list","watch"]
```

Supply Chain Security Tools — Scan

`apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"`

```
- apiGroups: ["scanning.apps.tanzu.vmware.com"]
  resources: ["imagescans","scanpolicies","scantemplates","sourcescans"]
  verbs: ["get","list","watch"]
```

`apps.tanzu.vmware.com/aggregate-to-app-operator: "true"`

```
- apiGroups: ["scanning.apps.tanzu.vmware.com"]
  resources: ["scanpolicies","scantemplates"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
```

Tanzu Build Service

`apps.tanzu.vmware.com/aggregate-to-app-editor: "true"`

```
- apiGroups: ["kpack.io"]
  resources: ["builds"]
  verbs: ["patch"]
```

`apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"`

```
- apiGroups: ["kpack.io"]
  resources: ["builds","builders","images"]
  verbs: ["get","list","watch"]
```

`apps.tanzu.vmware.com/aggregate-to-app-viewer-cluster-access: "true"`

```
- apiGroups: ["kpack.io"]
  resources: ["clusterbuilders","clusterstacks","clusterstores"]
  verbs: ["get","list","watch"]
```

`apps.tanzu.vmware.com/aggregate-to-app-operator: "true"`

```
- apiGroups: ["kpack.io"]
  resources: ["builders"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
```

`apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access`

```
- apiGroups: ["kpack.io"]
  resources: ["clusterbuilders","clusterstacks","clusterstores"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
```

Tekton

`apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"`

```
- apiGroups: ["tekton.dev"]
  resources: ["pipelineresources","pipelineruns","pipelines","taskruns","tasks"]
  verbs: ["get","list","watch"]
```

`apps.tanzu.vmware.com/aggregate-to-app-viewer-cluster-access: "true"`

```
- apiGroups: ["tekton.dev"]
  resources: ["clustertasks"]
  verbs: ["get","list","watch"]
```

`apps.tanzu.vmware.com/aggregate-to-app-operator: "true"`

```
- apiGroups: ["tekton.dev"]
  resources: ["pipelineresources","pipelines","tasks"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
```

`apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access`

```
- apiGroups: ["tekton.dev"]
  resources: ["clustertasks"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
```

Bind a user or group to a default role

This topic tells you how to use Kubernetes manifests to create the binding between users or groups and roles. Using a manifest allows you to add or remove users or groups after the initial creation. You can update the manifest and re-apply it by using kubectl.

After configuring your identity provider for Kubernetes, users can be authenticated to Kubernetes. However, they must be authorized to perform actions on resources. You can use kubectl to create role bindings that binds a user to the roles. For more information, see Role descriptions for Tanzu Application Platform.

Prerequisites

Meet these prerequisites:

1. Configure kubectl and have admin access to the cluster.
2. Configure an authentication solution for the cluster. You can use Pinniped or the authentication service native to your Kubernetes distribution.

Manage a user or groups mapping to a role

A Kubernetes RoleBinding or ClusterRoleBinding grants a user or group mapping to a role. A RoleBinding grants a role or ClusterRole to a specific namespace. Alternatively, a ClusterBinding maps a clusterrole to all namespaces. For more information, see the Kubernetes documentation.

Each of the four roles for users have both a role for namespace scoped resources and a role for cluster scoped resources. To ensure that the roles have access to the resources at the correct scoping, you must create bindings for both the namespace scoped resources and the cluster scoped resources. For a listing of the roles and cluster roles for each out-of-the-box Tanzu Application Platform role, see Default roles.

The following is an example of mapping the user `developer-1` to the `app-editor` role:

```
kubectl apply -n DEVELOPER_NAMESPACE -f - << EOF
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: app-editor
  namespace: $DEVELOPER_NAMSPACE
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: app-editor
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: developer-1
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: app-editor-cluster-access
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: app-editor-cluster-access
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: developer-1
EOF
```

Where `DEVELOPER-NAMESPACE` is the namespace for scoping the developers access.

To simplify the management of user to role membership, you can bind a group to a role in Kubernetes and manage a users group membership with an enteprise identity provider. For an example of Azure AD, see Integrate your Azure Active Directory.

The following example demonstrates how to add the group `developers` to the `app-editor` role:

```
kubectl apply -n DEVELOPER_NAMESPACE -f - << EOF
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: app-editor
```

```
  namespace: $DEVELOPER_NAMSPACE
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: app-editor
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: developers
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: app-editor-cluster-access
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: app-editor-cluster-access
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: developers
EOF
```

Where `DEVELOPER-NAMESPACE` is the namespace for scoping the developers access.

To add additional users or groups to the role, you can add another subject to the list. For example, to add the user `developer-2`, the `subjects` key looks like the following:

```
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: developer-1
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: developer-2
```

Additionally, you can use a combination of users and groups in the `subject` key as follows:

```
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: developers
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: developer-1
```

Log in to Tanzu Application Platform by using Pinniped

This topic tells you how to log in to your Tanzu Application Platform (commonly known as TAP) by using Pinniped.

As a prerequisite, the administrator must provide authorization for users to resources by using `rolebindings`. For more information, see Bind a user or group to a default role.

To log in to your cluster by using Pinniped, follow these steps:

1. Install the Pinniped CLI.

   For more information, see Pinniped documentation.

   Important

> The latest compatible version of Pinniped CLI is required not only for the administrator to generate the `kubeconfig`, but also for the user to log in with the provided configuration.

2. Generate and distribute `kubeconfig` to users.
3. Login with the provided `kubeconfig`.

Download the Pinniped CLI

You must use a Pinniped CLI version that matches the installed Concierge or Supervisor. Use one of the following links to download the Pinniped CLI version `0.22.0`:

- Mac OS with AMD64
- Linux with AMD64
- Windows with AMD64

You must install the command-line tool on your `$PATH`, such as `/usr/local/bin` on macOS or Linux. You must also mark the file as executable.

Generate and distribute kubeconfig to users

As an administrator, you can generate the kubeconfig by using the following command:

```
pinniped get kubeconfig --kubeconfig-context <your-kubeconfig-context>  > /tmp/concier
ge-kubeconfig
```

Distribute this `kubeconfig` to your users so they can login by using `pinniped`.

Login with the provided kubeconfig

As a user of the cluster, you need the `kubeconfig` provided by your admin and the Pinniped CLI installed on your local machine to log in. Logging in is required to request information from the cluster. You can execute any resource request with kubectl to enter the authentication flow. For example:

```
kubectl --kubeconfig /tmp/concierge-kubeconfig get pods
```

If you do not want to explicitly use `--kubeconfig` in every command, you can also export an environment variable to set the `kubeconfig` path in your shell session.

```
export KUBECONFIG="/tmp/concierge-kubeconfig"
kubectl get pods
```

This command enables `pinniped` to print a URL for you to visit in the browser. You can then log in, copy the authentication code and paste it back to the terminal. After the login succeeds, you either see the resources or a message indicating that you have no permission to access the resources.

If you use a Windows machine, the command referenced in the generated `kubeconfig` might not work. In this case, you must change the path under `user.exec.command` in the `kubeconfig` to point to the install path of the Pinniped CLI.

Additional resources about Tanzu Application Platform authentication and authorization

Use this topic to learn additional information about authentication and authorization for Tanzu Application Platform (commonly known as TAP).

See Default roles for Tanzu Application Platform overview to get started.

Install

Default roles are released as part of Tanzu Application Platform. Alternatively, you can also install default roles independently. See Install default roles independently for more information.

Additional resources about Tanzu Application Platform authentication and authorization

Use this topic to learn additional information about authentication and authorization for Tanzu Application Platform (commonly known as TAP).

See Default roles for Tanzu Application Platform overview to get started.

Install

Default roles are released as part of Tanzu Application Platform. Alternatively, you can also install default roles independently. See Install default roles independently for more information.

Install default roles independently for your Tanzu Application Platform

This topic tells you how to install default roles for Tanzu Application Platform (commonly known as TAP) without deploying a TAP profile.

Note

Follow the steps in this topic if you do not want to use a profile to install default roles. For more information about profiles, see Components and installation profiles.

Prerequisites

Before installing default roles, complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.

Install

To install default roles:

1. List version information for the package by running:

   ```
   tanzu package available list tap-auth.tanzu.vmware.com --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package available list tap-auth.tanzu.vmware.com --namespace tap-instal
   l
   - Retrieving package versions for tap-auth.tanzu.vmware.com...
     NAME                          VERSION       RELEASED-AT
     tap-auth.tanzu.vmware.com     1.0.1
   ```

2. Install the package by running:

```
tanzu package install tap-auth \
  --package tap-auth.tanzu.vmware.com \
  --version VERSION \
  --namespace tap-install
```

Where:

- `VERSION` is the package version number. For example, `1.0.1`.

For example:

```
$ tanzu package install tap-auth \
  --package tap-auth.tanzu.vmware.com \
  --version 1.0.1 \
  --namespace tap-install
```

Overview of Developer Conventions

Developer Conventions is a set of conventions that enable your workloads to support live-update and debug operations in Tanzu Application Platform (commonly known as TAP).

Prerequisites

- Tanzu CLI Apps plug-in
- Tanzu Dev Tools for VSCode IDE extension.

Features

Enabling Live Updates

Developer Conventions modifies your workload to enable live updates in either of the following situations:

- You deploy a workload by using the Tanzu CLI Apps plug-in and include the flag `--live-update=true`. For more information about how to deploy a workload with the CLI, see Create or update a workload.
- You deploy a workload by using the `Tanzu: Live Update Start` option through the Tanzu Developer Tools for VS Code extension. For more information about live updating with the extension, see Overview of Tanzu Developer Tools for Visual Studio Code.

When either of the preceding actions take place, the convention behaves as follows:

1. Looks for the `apps.tanzu.vmware.com/live-update=true` annotation on a PodTemplateSpec associated with a workload.
2. Verifies that the image to which conventions are applied contains a process that can be live updated.
3. Adds annotations to the PodTemplateSpec to modify the Knative properties `minScale` & `maxScale` such that the minimum and maximum number of pods is 1. This ensures the eventual running pod is not scaled down to 0 during a live update session.

After these changes are made, you can use the Tanzu Dev Tools extension or the Tilt CLI to make live update changes to source code directly on the cluster.

Enabling debugging

Developer Conventions modifies your workload to enable debugging in either of the following situations:

- You deploy a workload by using the Tanzu CLI Apps plug-in and include the flag `--debug=true`. For more information about how to deploy a workload with the CLI, see Create or update a workload.
- You deploy a workload by using the `Tanzu Java Debug Start` option through the Tanzu Developer Tools for VS Code extension. For more information about debugging with the extension, see Overview of Tanzu Developer Tools for Visual Studio Code.

When either of the preceding actions take place, the convention behaves as follows:

1. It looks for the `apps.tanzu.vmware.com/debug=true` annotation on a PodTemplateSpec associated with a workload.
2. It checks for the `debug-8` or `debug-9` labels on the image configuration's bill of materials (BOM).
3. It sets the TimeoutSeconds of the Liveness, Readiness, and Startup probes to 600 if currently set to a lower number.
4. It adds annotations to the PodTemplateSpec to modify the Knative properties `minScale` & `maxScale` such that the minimum and maximum number of pods is 1. This ensures the eventual running pod won't be scaled down to 0 during a debug session.

After these changes are made, you can use the Tanzu Dev Tools extension or other CLI-based debuggers to debug your workload directly on the cluster.

Note

Currently, Developer Conventions only supports debug operations for Java applications.

Next steps

- Install Developer Conventions

Overview of Developer Conventions

Developer Conventions is a set of conventions that enable your workloads to support live-update and debug operations in Tanzu Application Platform (commonly known as TAP).

Prerequisites

- Tanzu CLI Apps plug-in
- Tanzu Dev Tools for VSCode IDE extension.

Features

Enabling Live Updates

Developer Conventions modifies your workload to enable live updates in either of the following situations:

- You deploy a workload by using the Tanzu CLI Apps plug-in and include the flag `--live-update=true`. For more information about how to deploy a workload with the CLI, see

Create or update a workload.

- You deploy a workload by using the `Tanzu: Live Update Start` option through the Tanzu Developer Tools for VS Code extension. For more information about live updating with the extension, see Overview of Tanzu Developer Tools for Visual Studio Code.

When either of the preceding actions take place, the convention behaves as follows:

1. Looks for the `apps.tanzu.vmware.com/live-update=true` annotation on a PodTemplateSpec associated with a workload.
2. Verifies that the image to which conventions are applied contains a process that can be live updated.
3. Adds annotations to the PodTemplateSpec to modify the Knative properties `minScale` & `maxScale` such that the minimum and maximum number of pods is 1. This ensures the eventual running pod is not scaled down to 0 during a live update session.

After these changes are made, you can use the Tanzu Dev Tools extension or the Tilt CLI to make live update changes to source code directly on the cluster.

Enabling debugging

Developer Conventions modifies your workload to enable debugging in either of the following situations:

- You deploy a workload by using the Tanzu CLI Apps plug-in and include the flag `--debug=true`. For more information about how to deploy a workload with the CLI, see Create or update a workload.
- You deploy a workload by using the `Tanzu Java Debug Start` option through the Tanzu Developer Tools for VS Code extension. For more information about debugging with the extension, see Overview of Tanzu Developer Tools for Visual Studio Code.

When either of the preceding actions take place, the convention behaves as follows:

1. It looks for the `apps.tanzu.vmware.com/debug=true` annotation on a PodTemplateSpec associated with a workload.
2. It checks for the `debug-8` or `debug-9` labels on the image configuration's bill of materials (BOM).
3. It sets the TimeoutSeconds of the Liveness, Readiness, and Startup probes to 600 if currently set to a lower number.
4. It adds annotations to the PodTemplateSpec to modify the Knative properties `minScale` & `maxScale` such that the minimum and maximum number of pods is 1. This ensures the eventual running pod won't be scaled down to 0 during a debug session.

After these changes are made, you can use the Tanzu Dev Tools extension or other CLI-based debuggers to debug your workload directly on the cluster.

Note

Currently, Developer Conventions only supports debug operations for Java applications.

Next steps

- Install Developer Conventions

Install Developer Conventions

This document tells you how to install Developer Conventions from the Tanzu Application Platform (commonly known as TAP) package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Developer Conventions. For more information about profiles, see About Tanzu Application Platform components and profiles.

Prerequisites

Before installing Developer Conventions:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- Install Supply Chain Choreographer.
- Create conventions with Cartographer Conventions.

Install

To install Developer Conventions:

1. Get the exact name and version information for the Developer Conventions package to be installed by running:

   ```
   tanzu package available list developer-conventions.tanzu.vmware.com --namespace
   tap-install
   ```

   For example:

   ```
   $ tanzu package available list developer-conventions.tanzu.vmware.com --namespa
   ce tap-install
   - Retrieving package versions for developer-conventions.tanzu.vmware.com
     NAME                                    VERSION        RELEASED-AT
     developer-conventions.tanzu.vmware.com  0.3.0          2021-10-19T00:00:00Z
   ```

2. Install the package by running:

   ```
   tanzu package install developer-conventions \
     --package developer-conventions.tanzu.vmware.com \
     --version 0.3.0 \
     --namespace tap-install
   ```

3. Verify the package install by running:

   ```
   tanzu package installed get developer-conventions --namespace tap-install
   ```

   For example:

   ```
   tanzu package installed get developer-conventions -n tap-install
   | Retrieving installation details for developer-conventions...
   NAME:                    developer-conventions
   PACKAGE-NAME:            developer-conventions.tanzu.vmware.com
   PACKAGE-VERSION:         0.3.0
   STATUS:                  Reconcile succeeded
   ```

```
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`

Resource limits

The following resource limits are set on the Developer Conventions service:

```
resources:
  limits:
  cpu: 100m
  memory: 256Mi
  requests:
  cpu: 100m
  memory: 20Mi
```

Uninstall

To uninstall Developer Conventions, follow the guide for Uninstall Tanzu Application Platform packages. The package name for developer conventions is `developer-conventions`.

Run Developer Conventions on an OpenShift cluster

This topic tells you about considerations for running Developer Conventions on OpenShift.

On OpenShift clusters, Developer Conventions must run with a custom SecurityContextConstraint (SCC) to enable compliance with restricted Kubernetes pod security standards. Tanzu Application Platform configures the following SCC for the Developer Convention's webhook when you configure the `kubernetes_distribution: openshift` key in the `tap-values.yaml` file.

Specification follows:

```
---
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  name: developer-conventions-scc
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: false
allowPrivilegedContainer: false
defaultAddCapabilities: null
fsGroup:
  type: RunAsAny
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities: null
runAsUser:
  type: MustRunAsNonRoot
seLinuxContext:
  type: MustRunAs
supplementalGroups:
  type: RunAsAny
volumes:
  - secret
seccompProfiles: []
```

```
groups:
  - system:serviceaccounts:developer-conventions
```

Use External Secrets Operator in Tanzu Application Platform

The External Secrets Operator is a Kubernetes operator that integrates with external secret management systems, for example, Google Secrets Manager and Hashicorp Vault. It reads information from external APIs and automatically injects the values into a Kubernetes secret.

Tanzu Application Platform (commonly known as TAP) uses the External Secrets Operator to simplify Kubernetes secret life cycle management. The `external-secrets` plug-in, which is available in the Tanzu CLI, interacts with the External Secrets Operator API. Users can use this CLI plug-in to create and view External Secrets Operator resources on a Kubernetes cluster.

External Secrets Operator is available in Tanzu Application Platform packages with a Carvel Package named `external-secrets.apps.tanzu.vmware.com`. It is not part of any installation profile.

Where to start

To learn more about managing secrets with External Secrets in general, see the official External Secrets Operator documentation. For installing the External Secrets Operator and the CLI plug-in see the following documentation. Also, see the example integration of External-Secrets with Hashicorp Vault.

- Installing External Secrets Operator in TAP
- Installing Tanzu CLI
- External-Secrets with Hashicorp Vault
- Tanzu External Secrets CLI plug-in Reference

Use External Secrets Operator in Tanzu Application Platform

The External Secrets Operator is a Kubernetes operator that integrates with external secret management systems, for example, Google Secrets Manager and Hashicorp Vault. It reads information from external APIs and automatically injects the values into a Kubernetes secret.

Tanzu Application Platform (commonly known as TAP) uses the External Secrets Operator to simplify Kubernetes secret life cycle management. The `external-secrets` plug-in, which is available in the Tanzu CLI, interacts with the External Secrets Operator API. Users can use this CLI plug-in to create and view External Secrets Operator resources on a Kubernetes cluster.

External Secrets Operator is available in Tanzu Application Platform packages with a Carvel Package named `external-secrets.apps.tanzu.vmware.com`. It is not part of any installation profile.

Where to start

To learn more about managing secrets with External Secrets in general, see the official External Secrets Operator documentation. For installing the External Secrets Operator and the CLI plug-in see the following documentation. Also, see the example integration of External-Secrets with Hashicorp Vault.

- Installing External Secrets Operator in TAP
- Installing Tanzu CLI

- External-Secrets with Hashicorp Vault
- Tanzu External Secrets CLI plug-in Reference

Install External Secrets Operator in Tanzu Application Platform

This topic tells you how to install the External Secrets Operator from the Tanzu Application Platform (commonly known as TAP) package repository.

Important

External Secrets Operator is not included or installed with any Tanzu Application Platform profile.

Prerequisites

Before installing External Secrets Operator:

- Complete all prerequisites to install the Tanzu Application Platform. For more information, see Prerequisites.

Install

To install External Secrets Operator:

1. List version information for the package by running:

   ```
   tanzu package available list external-secrets.apps.tanzu.vmware.com -n tap-inst
   all
   ```

   For example:

   ```
   NAME                                     VERSION         RELEASED-AT
   external-secrets.apps.tanzu.vmware.com   0.9.4+tanzu.2   2023-11-10 00:00:00 -050
   0 EST
   ```

2. (Optional) For Tanzu Application Platform v1.7.1 and later, you can create a YAML values file to specify optional configuration for the External Secrets Operator. In this release, there is only one setting: `kubernetes_distribution`, which you can set to `"openshift"` or `""`.
   - To run External Secrets Operator on an OpenShift cluster, set the `kubernetes_distribution` value to `openshift`. This setting removes some of the default security settings in External Secrets Operator so that OpenShift can replace the security settings with its own.
   - If you are running the External Secrets Operator on any other platform, leave `kubernetes_distribution` as an empty string or omit the value entirely.

   Note

   The `kubernetes_distribution` setting is available as of External Secrets Operator v0.9.4+tanzu.2, which is available in Tanzu Application Platform v1.7.1. External Secrets Operator v0.9.4+tanzu.1 and earlier might not be able to run on OpenShift clusters.

3. Install the package by running:

```
tanzu package install external-secrets \
 --package external-secrets.apps.tanzu.vmware.com \
 --version VERSION-NUMBER \
 --values-file VALUES-FILE.yaml \ # The use of this file is optional
 --namespace tap-install
```

Where:

- `VERSION-NUMBER` is the package version you retrieved earlier.
- (Optional) `VALUES-FILE` is the YAML file you created earlier containing the values used to configure External Secrets Operator.

For example:

```
$ tanzu package install external-secrets \
   --package external-secrets.apps.tanzu.vmware.com \
   --version 0.6.1+tap.6 \
   --values-file external-secrets-values.yaml \
   --namespace tap-install

\ Installing package 'external-secrets.apps.tanzu.vmware.com'
| Getting package metadata for 'external-secrets.apps.tanzu.vmware.com'
| Creating service account 'external-secrets-tap-install-sa'
/ Creating cluster admin role 'external-secrets-tap-install-cluster-role'
| Creating cluster role binding 'external-secrets-tap-install-cluster-rolebindi
n
/ Creating cluster role binding 'external-secrets-tap-install-cluster-rolebindi
n
\ Creating cluster role binding 'external-secrets-tap-install-cluster-rolebindi
n
| Creating cluster role binding 'external-secrets-tap-install-cluster-rolebindi
ng'
\ Creating package resource
Waiting for 'PackageInstall' reconciliation for 'external-secrets'
'PackageInstall' resource install status: Reconciling
'PackageInstall' resource install status: ReconcileSucceeded

Added installed package 'external-secrets'
```

4. Verify the package installation by running:

```
tanzu package installed get external-secrets --namespace tap-install
```

For example:

```
tanzu package installed get external-secrets -n tap-install

NAME:                    external-secrets
PACKAGE-NAME:            external-secrets.apps.tanzu.vmware.com
PACKAGE-VERSION:         0.9.4+tanzu.2
STATUS:                  Reconcile succeeded
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Integrate External Secrets Operator with HashiCorp Vault in Tanzu Application Platform

This topic shows you how to integrate External Secrets Operator with HashiCorp Vault in Tanzu Application Platform.

The operator synchronizes secret data from external APIs to a Kubernetes secret resource. For more information about Kubernetes secret resources, see the Kubernetes documentation.

Important

This example integration is constructed to showcase the features available and must not be considered in a production environment.

Prerequisites

Before proceeding with this example, you must:

- Install External Secrets Operator. For more information, see Install External Secrets Operator.
- Install the Tanzu CLI. The Tanzu CLI includes the plug-in `external-secrets`. For Tanzu CLI installation, see Tanzu CLI.
- Have a running instance of HashiCorp Vault. In this instance, there is a secret defined with the key `eso-demo/reg-cred`.

Set up the integration

To set up the External Secrets Operator integration with HashiCorp Vault:

1. Create a `Secret` with the vault token. For example:

```
VAULT_TOKEN="vault-token-value"

cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Secret
metadata:
name: vault-token
stringData:
token: $VAULT_TOKEN
EOF
```

2. Create a `SecretStore` resource referencing the `vault-token` secret. For example:

Caution

When creating the `SecretStore`, ensure that you match the Vault KV secret engine version. This is either `v1` or `v2`. The default is `v2`. For more information, see Vault KV Secrets Engine documentation.

```
VAULT_SERVER="http://my.vault.server:8200"
VAULT_PATH="eso-demo"

cat <<EOF | tanzu external-secrets store create -y -f -
---
apiVersion: external-secrets.io/v1beta1
kind: SecretStore
metadata:
  name: vault-secret-store
spec:
  provider:
    vault:
```

```
      server: $VAULT_SERVER
      path: $VAULT_PATH
      version: v2
      auth:
        tokenSecretRef:
          name: "vault-token" # vault-token created in the previous step
          key: "token"
EOF
```

Important

If you are using a secret store service with a custom CA certificate then you must provide this certificate to External Secret Operator directly by including the CA `SecretStore` or `ClusterSecretStore` resource.

The Tanzu Application Platform distribution of External Secret Operator does not support the Tanzu Application Platform field `shared.ca_cert_data`. For more information about setting the CA in the ESO configuration, see the ESO documentation.

3. Verify that the status of the `SecretStore` resource is `Valid` by running:

```
tanzu external-secrets store list
```

Example output:

```
NAMESPACE  NAME                PROVIDER         STATUS
default    vault-secret-store  Hashicorp Vault  Valid
```

4. Create an `ExternalSecret` resource that uses the `SecretStore` you just created by running:

```
cat <<EOF | tanzu external-secrets secret create -y -f -
---
apiVersion: external-secrets.io/v1beta1
kind: ExternalSecret
metadata:
  name: vault-secret-example
spec:
  refreshInterval: 15m
  secretStoreRef:
    name: vault-secret-store
    kind: SecretStore
  target:
    name: registry-secret
    template:
      type: kubernetes.io/dockerconfigjson
      data:
        .dockerconfigjson: "{{ .registryCred | toString }}"
    creationPolicy: Owner
  data:
  - secretKey: registryCred
    remoteRef:
      key: $VAULT_PATH/eso-demo
      property: reg-cred
EOF
```

5. Verify that the status of the `ExternalSecret` resource is `Valid` by running:

```
tanzu external-secrets secret list
```

Example output:

```
NAMESPACE  NAME                  SECRET NAME      STORE               REFRESH I
NTERVAL  STATUS             LAST UPDATED  LAST REFRESH
default    vault-secret-example  registry-secret  vault-secret-store  15m
SecretSynced  21s           10m
```

6. After the resource has reconciled, a Kubernetes `secret` resource is created. Look for a secret named `registry-secret` created by the referenced `ExternalSecret`. For example:

```
kubectl get secrets registry-secret -o="jsonpath={.data.\.dockerconfigjson}" |
base64 -D
{"auths":{"my-registry.example:8200":{"username":"foo","password":"bar4","emai
l":"foo@bar.example","auth":"Zm9vOmJhcjQ="}}}
```

Tanzu External Secrets CLI plug-in command reference

The Tanzu External Secrets CLI plug-in command reference has moved to the Tanzu CLI Command Reference documentation.

Overview of Flux CD Source Controller

Flux CD Source Controller provides you with APIs for acquiring resources such as Git, Helm repositories and S3 buckets on the cluster. For more information, see Flux CD Source Controller documentation.

The source-controller implements the source.toolkit.fluxcd.io API in GitHub and is a core component of the GitOps toolkit.

Overview of Flux CD Source Controller

Flux CD Source Controller provides you with APIs for acquiring resources such as Git, Helm repositories and S3 buckets on the cluster. For more information, see Flux CD Source Controller documentation.

The source-controller implements the source.toolkit.fluxcd.io API in GitHub and is a core component of the GitOps toolkit.

Install Flux CD Source Controller

This topic tells you how to install Flux CD Source Controller from the Tanzu Application Platform (commonly known as TAP) package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Flux CD Source Controller. For more information about profiles, see Components and installation profiles.

Prerequisites

Before installing Flux CD Source Controller:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- Install cert-manager on the cluster. For more information, see Install cert-manager.

Configuration

The Flux CD Source Controller package has no configuration values.

Installation

To install Flux CD Source Controller from the Tanzu Application Platform package repository:

1. List version information for the package by running:

   ```
   tanzu package available list fluxcd.source.controller.tanzu.vmware.com -n tap-i
   nstall
   ```

   For example:

   ```
   $ tanzu package available list fluxcd.source.controller.tanzu.vmware.com -n tap
   -install
       \ Retrieving package versions for fluxcd.source.controller.tanzu.vmware.co
   m...
         NAME                                          VERSION  RELEASED-AT
         fluxcd.source.controller.tanzu.vmware.com  1.1.2    2024-03-11 00:00:00 -
   0500 -05
   ```

2. Install the package by running:

   ```
   tanzu package install fluxcd-source-controller -p fluxcd.source.controller.tanz
   u.vmware.com -v VERSION-NUMBER -n tap-install
   ```

   Where:

   - `VERSION-NUMBER` is the version of the package listed in step 1.

   For example:

   ```
   tanzu package install fluxcd-source-controller -p fluxcd.source.controller.tanz
   u.vmware.com -v 1.1.2 -n tap-install
   \ Installing package 'fluxcd.source.controller.tanzu.vmware.com'
   | Getting package metadata for 'fluxcd.source.controller.tanzu.vmware.com'
   | Creating service account 'fluxcd-source-controller-tap-install-sa'
   | Creating cluster admin role 'fluxcd-source-controller-tap-install-cluster-rol
   e'
   | Creating cluster role binding 'fluxcd-source-controller-tap-install-cluster-r
   olebinding'
   | Creating package resource
   - Waiting for 'PackageInstall' reconciliation for 'fluxcd-source-controller'
   | 'PackageInstall' resource install status: Reconciling

     Added installed package 'fluxcd-source-controller'
   ```

   This package creates a new namespace called `flux-system`. This namespace hosts all the elements of fluxcd.

3. Verify the package install by running:

   ```
   tanzu package installed get fluxcd-source-controller -n tap-install
   ```

   For example:

   ```
   tanzu package installed get fluxcd-source-controller -n tap-install
   \ Retrieving installation details for fluxcd-source-controller...
   NAME:                    fluxcd-source-controller
   PACKAGE-NAME:            fluxcd.source.controller.tanzu.vmware.com
   ```

```
PACKAGE-VERSION:          0.16.0
STATUS:                   Reconcile succeeded
CONDITIONS:               [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`.

```
kubectl get pods -n flux-system
```

For example:

```
$ kubectl get pods -n flux-system
NAME                                READY   STATUS    RESTARTS   AGE
source-controller-69859f545d-ll8fj   1/1     Running   0          3m38s
```

Verify that `STATUS` is `Running`.

Try fluxcd-source-controller

1. Verify the main components of `fluxcd-source-controller` were installed by running:

   ```
   kubectl get all -n flux-system
   ```

   Expect to see the following outputs or similar:

   ```
   NAME                                      READY   STATUS    RESTARTS   AGE
   pod/source-controller-7684c85659-2zfxb    1/1     Running   0          40m

   NAME                          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
   AGE
   service/source-controller     ClusterIP   10.108.138.74   <none>        80/TCP
   40m

   NAME                                  READY   UP-TO-DATE   AVAILABLE   AGE
   deployment.apps/source-controller     1/1     1            1           40m

   NAME                                             DESIRED   CURRENT   READY   AGE
   replicaset.apps/source-controller-7684c85659     1         1         1       40m
   ```

2. Verify all the CRD were installedby running:

   ```
   kubectl get crds -n flux-system | grep ".fluxcd.io"
   buckets.source.toolkit.fluxcd.io                          2022-03-07T19:20:14Z
   gitrepositories.source.toolkit.fluxcd.io                  2022-03-07T19:20:14Z
   helmcharts.source.toolkit.fluxcd.io                       2022-03-07T19:20:14Z
   helmrepositories.source.toolkit.fluxcd.io                 2022-03-07T19:20:14Z
   ```

   Note

   You will communicate with `fluxcd-source-controller` through its CRDs.

3. If you are using a Git repository with a custom CA certificate, provide this certificate to the Flux CD Source Controller directly by including the CA in the service account used by the supply chain.

   The Tanzu Application Platform distribution of Flux CD Source Controller does not support the Tanzu Application Platform `shared.ca_cert_data` field. For more information about setting the CA in the service account, see Use Git authentication with Supply Chain Choreographer.

4. Follow these steps to consume a `GitRepository` object:

    1. Create the following `gitrepository-sample.yaml` file:

        ```
        apiVersion: source.toolkit.fluxcd.io/v1
        kind: GitRepository
        metadata:
          name: gitrepository-sample
        spec:
          interval: 1m
          url: https://github.com/vmware-tanzu/application-accelerator-samples
          ref:
            branch: main
        ```

    2. Apply the created conf:

        ```
        kubectl apply -f gitrepository-sample.yaml
        gitrepository.source.toolkit.fluxcd.io/gitrepository-sample created
        ```

    3. Verify the git-repository was fetched correctly:

        ```
        kubectl get GitRepository
        NAME                    URL
        READY   STATUS
        AGE
        gitrepository-sample   https://github.com/vmware-tanzu/application-accele
        rator-samples   True    Fetched revision: main/132f4e719209eb10b9485302f8
        593fc0e680f4fc   4s
        ```

    For more examples, see the samples directory on fluxcd/source-controller/samples in GitHub.

Configure resource limits

Each system has its unique resource requirements, and the default resources provided by Flux Source Controller might not meet the needs of your specific use case.

Flux Source Controller uses the following resource limits by default:

```
resources:
  limits:
    cpu: 1000m
    memory: 1Gi
  requests:
    cpu: 50m
    memory: 64Mi
```

You can configure the resource limits by following these steps:

1. Create a `Secret` with the following `ytt` overlay:

    ```
    apiVersion: v1
    kind: Secret
    metadata:
      name: fluxcd-source-overlay-secret
      namespace: tap-install
    stringData:
      fluxcd-source-overlay.yaml: |
          #@ load("@ytt:data", "data")
          #@ load("@ytt:overlay", "overlay")
          #@overlay/match by=overlay.subset({"kind":"Deployment", "metadata":{"nam
    e":"fluxcd-source-controller", "namespace": "flux-system"}}), expects="1+"
          ---
    ```

```
      spec:
        template:
          spec:
            containers:
            #@overlay/match by="name"
            - name: manager
              resources:
                limits:
                  cpu: "4"
                  memory: "4Gi"
                requests:
                  cpu: "2"
                  memory: "2Gi"
```

2. Update the Tanzu Application Platform values file by including a `package_overlays` field:

```
package_overlays:
- name: fluxcd-source-controller
  secrets:
  - name: fluxcd-source-overlay-secret
```

3. Update Tanzu Application Platform by running:

```
tanzu package installed update tap -p tap.tanzu.vmware.com -v 1.9.0 --values-fi
le tap-values.yaml -n tap-install
```

For more information about the package customization, see Customize your package installation.

Documentation

For documentation specific to fluxcd-source-controller, see the main repository fluxcd/source-controller in GitHub.

Use Flux CD Source Controller

To build workload by using the `GitRepository` API of Flux CD Source Controller, see Building from source with Supply Chain Choreographer.

For authenticating to Git repositories by using Flux CD Source Controller in Supply Chain Choreographer, see Git authentication with Supply Chain Choreographer.

Overview of Learning Center for Tanzu Application Platform

Learning Center is no longer part of Tanzu Application Platform. Use Tanzu Academy instead for all Tanzu Application Platform learning and education needs.

Overview of Local Source Proxy

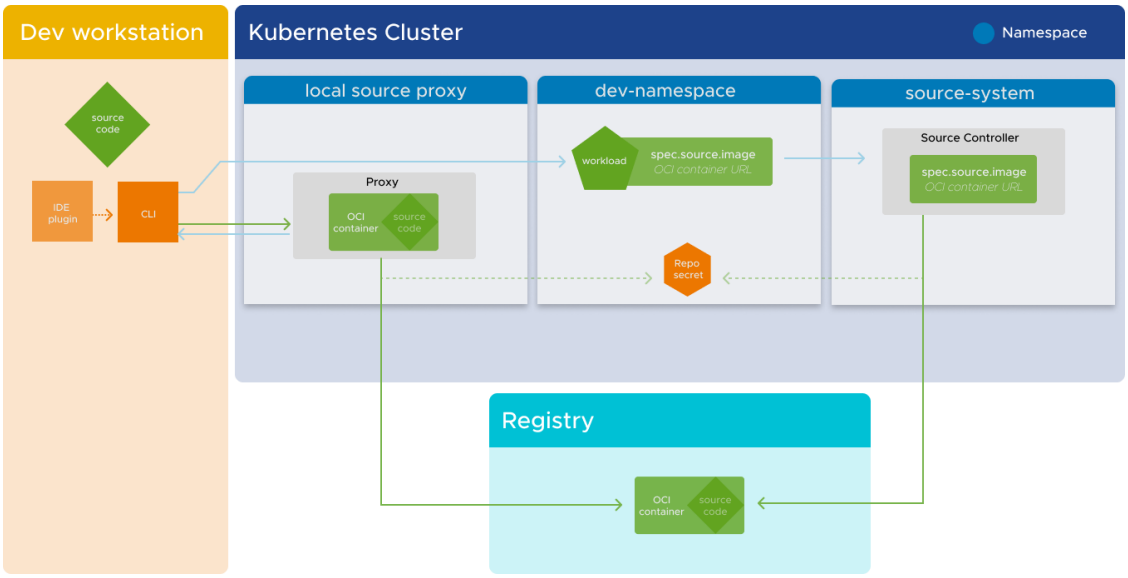
Local Source Proxy (LSP) offers developers a secure and user-friendly approach to seamlessly upload their local source code to a Tanzu Application Platform cluster. This enables developers to navigate their code smoothly through a predefined production pathway using supply chains.

With Local Source Proxy, developers can:

- Interact with external registries without needing to know registry specifics, such as endpoints, credentials, and certificates. This eliminates the burden of platform and app operators having to distribute registry credentials to developer workstations.

- Deploy their workloads from a local source through any mechanism, including IDE extensions, without providing the source image location. Developers can seamlessly deploy their applications without managing registry credentials on their local machines or keeping track of where their local source is uploaded.

Consequently, the `--source-image` flag in the Apps CLI plug-in becomes optional, and there is no longer a need for Docker registry credentials on the developer's local machine. Local Source Proxy abstracts these details, making deployment more streamlined and user-friendly for developers.



By removing the necessity for specific registry information and credentials, Local Source Proxy simplifies the developer experience. This enables developers to focus on developing their applications instead of managing registry-related complexities.

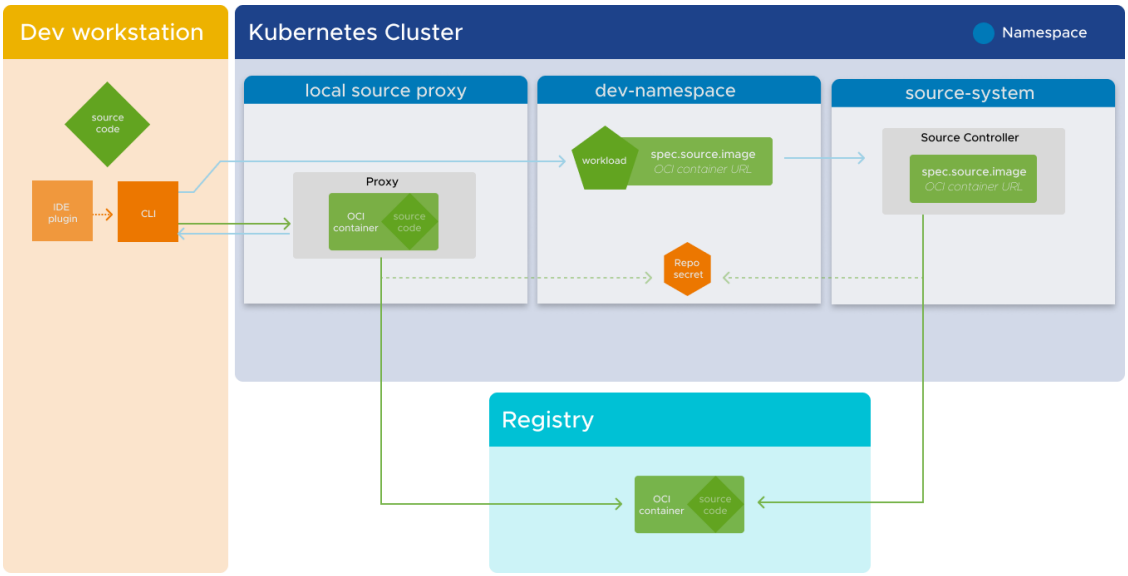
Overview of Local Source Proxy

Local Source Proxy (LSP) offers developers a secure and user-friendly approach to seamlessly upload their local source code to a Tanzu Application Platform cluster. This enables developers to navigate their code smoothly through a predefined production pathway using supply chains.

With Local Source Proxy, developers can:

- Interact with external registries without needing to know registry specifics, such as endpoints, credentials, and certificates. This eliminates the burden of platform and app operators having to distribute registry credentials to developer workstations.
- Deploy their workloads from a local source through any mechanism, including IDE extensions, without providing the source image location. Developers can seamlessly deploy their applications without managing registry credentials on their local machines or keeping track of where their local source is uploaded.

Consequently, the `--source-image` flag in the Apps CLI plug-in becomes optional, and there is no longer a need for Docker registry credentials on the developer's local machine. Local Source Proxy abstracts these details, making deployment more streamlined and user-friendly for developers.



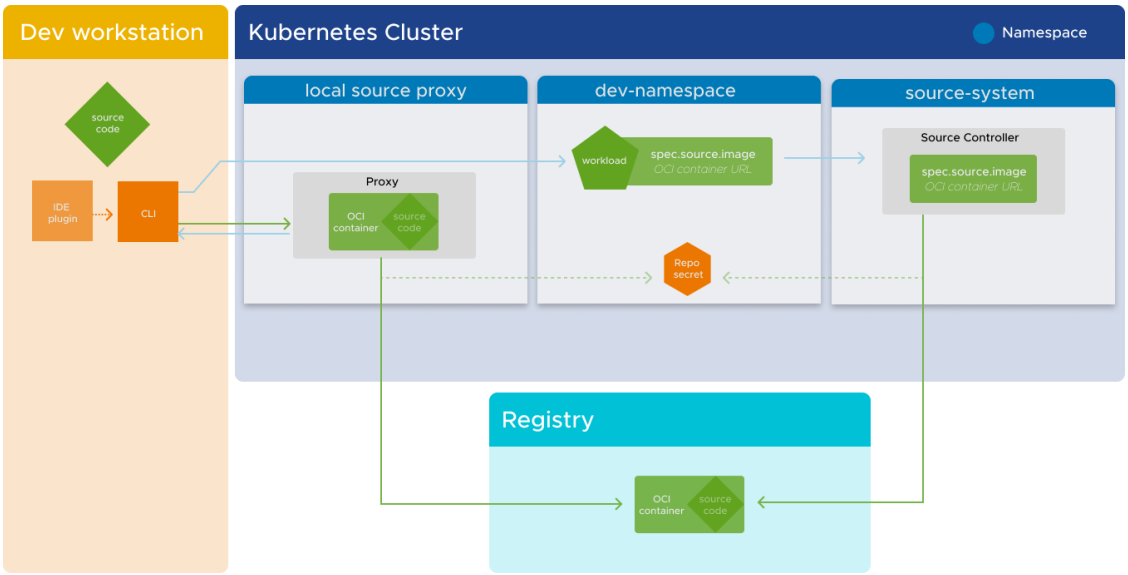
By removing the necessity for specific registry information and credentials, Local Source Proxy simplifies the developer experience. This enables developers to focus on developing their applications instead of managing registry-related complexities.

Design of Local Source Proxy

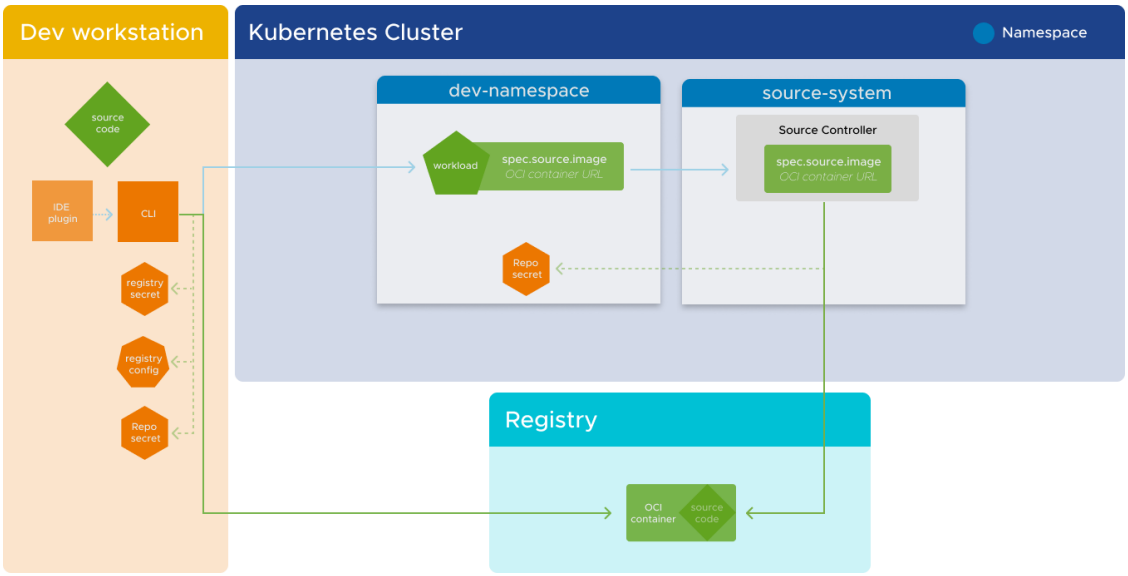
Local Source Proxy (LSP) serves as a proxy registry server with OCI (Open Container Initiative) compatibility. Its main purpose is to handle image push requests by forwarding them to an external registry server, which is configured through the `tap-values.yaml` file.

Local Source Proxy takes care of authentication and authorization against the external registry provider internally. This ensures that the external registry remains transparent to the user.

By functioning as a proxy registry server, Local Source Proxy simplifies the process of interacting with external registry servers. Local Source Proxy provides a centralized and transparent approach for image push requests, handling authentication and authorization seamlessly.



If you don't use Local Source Proxy, your developer workstation requires more configuration, and the process of interacting with external registry servers is more complicated.



The Apps CLI plug-in generates requests that adhere to the OCI distribution standard to push artifacts to Local Source Proxy instances. The Kubernetes API server handles user authentication and authorization.

Consequently this default mechanism becomes the primary way to push a developer's local source code to the Tanzu Application Platform cluster. This renders the `--source-image` flag optional. By leveraging Local Source Proxy, developers can seamlessly deploy their code without explicitly specifying the source image.

However, the system remains backward-compatible to accommodate different scenarios.

The `local-source-proxy` mechanism is completely bypassed if either of the following is true:

- The `--source-image` flag is provided
- The workload has a defined source image that Local Source Proxy didn't set

This ensures compatibility with existing workflows and allows developers to continue using their preferred methods for deployment.

By default, the `iterate` and `full` profiles include the installation of this package. To suppress this behavior, exclude the package by adding `local-source-proxy.apps.tanzu.vmware.com` to the list of excluded packages. For how to do so, see Exclude packages from a Tanzu Application Platform profile.

Prerequisites for Local Source Proxy

You need the following prerequisites before you can install Local Source Proxy (LSP):

- The Tanzu Application Platform profile `full` or `iterate`. For more information, see Components and installation profiles for Tanzu Application Platform.
- A registry server with a repository capable of accepting and hosting OCI artifacts, such as Google Artifact Registry, JFrog Artifactory, Harbor, Elastic Container Registry, and so on.
- Either IaaS-specific trust for Kubernetes service accounts to access the registry, or a secret with sufficient privileges to push and pull artifacts from that repository.
- If you install Local Source Proxy on a Kubernetes cluster that a cloud provider manages (such as EKS, AKS, or GKE), ensure that TCP port 5002 is open between control plane nodes and your worker nodes.

The rest of this topic tells you how to obtain these prerequisites.

Using Tanzu CLI

All registries except ECR can use the following code:

```
tanzu secret registry add lsp-push-credentials \
--username USERNAME-VALUE --password PASSWORD-VALUE \
--server REGISTRY-SERVER \
--namespace tap-install --yes
```

Declarative syntax

For declarative syntax:

```
---
apiVersion: v1
kind: Secret
metadata:
  name: lsp-push-credentials
  namespace: tap-install
type: kubernetes.io/dockerconfigjson
stringData:
  .dockerconfigjson: BASE64-ENCODED-DOCKER-CONFIG-JSON
```

The `dockerconfigjson` structure is as follows:

```
{"auths":{"REGISTRY-SERVER":{"username":"USERNAME-VALUE","password": "PASSWORD-VALU
E"}}}
```

If you're using the Tanzu Application Platform GitOps installer using Secrets OPerationS (SOPS), after using SOPS to encrypt the secret put the secret in the `clusters/CLUSTER-NAME/cluster-config/config/lsp` directory in your GitOps repository.

If you're using the Tanzu Application Platform GitOps installer using ESO, create a secret as follows:

```
#@ load("@ytt:data", "data")
#@ load("@ytt:json", "json")

#@ def config():
#@  return {
#@    "auths": {
#@      data.values.tap_value.{path-to-registry-host}: {
#@       "username": data.values.tap_values.{path-to-registry-username},
#@       "password": data.values.tap_values.{path-to-registry-password}
#@      }
#@    }
#@  }
#@ end
---
apiVersion: v1
kind: Secret
metadata:
  name: lsp-push-credentials
  namespace: tap-install
type: kubernetes.io/dockerconfigjson
stringData:
  .dockerconfigjson: #@ json.encode(config())
```

The procedure you use to obtain a secret with sufficient privileges depends on whether your registry is Elastic Container Registry (ECR) or something else.

Using AWS

If you're using Elastic Container Registry as your registry, you must create the container repository ahead of time. Additionally you require an AWS Identity Access and Management (IAM) role Amazon Resource Name (ARN) that possesses the necessary privileges to push and pull artifacts to the ECR repository. This is limited in scope to the service account for Local Source Proxy.

1. Export the variables by running:

```
export AWS_ACCOUNT_ID=012345678901  # Your AWS account ID
export AWS_REGION=us-west-2         # The AWS region you are going to deploy
to
export EKS_CLUSTER_NAME=tap-on-aws  # The name of your Elastic Kubernetes Ser
vice Cluster
```

2. Create the repository by running:

```
aws ecr create-repository --repository-name tap-lsp --region $AWS_REGION
```

3. Output the files, and then use the policy documents to create the IAM roles, by running:

```
export OIDCPROVIDER=$(aws eks describe-cluster --name $EKS_CLUSTER_NAME --reg
ion $AWS_REGION \
--output json | jq '.cluster.identity.oidc.issuer' | tr -d '"' | sed 's/http
s:\/\///')

cat << EOF > local-source-proxy-trust-policy.json
{
  "Version": "2012-10-17",
  "Statement": [
      {
          "Effect": "Allow",
          "Principal": {
              "Federated": "arn:aws:iam::${AWS_ACCOUNT_ID}:oidc-provider/${OI
DCPROVIDER}"
          },
          "Action": "sts:AssumeRoleWithWebIdentity",
          "Condition": {
              "StringEquals": {
                  "${OIDCPROVIDER}:aud": "sts.amazonaws.com"
              },
              "StringLike": {
                  "${OIDCPROVIDER}:sub": [
                      "system:serviceaccount:tap-local-source-system:proxy-ma
nager"
                  ]
              }
          }
      }
  ]
}
EOF

cat << EOF > local-source-proxy-policy.json
{
  "Version": "2012-10-17",
  "Statement": [
      {
          "Action": [
              "ecr:GetAuthorizationToken"
          ],
          "Resource": "*",
          "Effect": "Allow",
          "Sid": "TAPLSPGlobal"
      },
```

```
      {
          "Effect": "Allow",
          "Action": [
              "ecr:BatchCheckLayerAvailability",
              "ecr:GetDownloadUrlForLayer",
              "ecr:GetRepositoryPolicy",
              "ecr:DescribeRepositories",
              "ecr:ListImages",
              "ecr:DescribeImages",
              "ecr:BatchGetImage",
              "ecr:GetLifecyclePolicy",
              "ecr:GetLifecyclePolicyPreview",
              "ecr:ListTagsForResource",
              "ecr:DescribeImageScanFindings",
              "ecr:InitiateLayerUpload",
              "ecr:UploadLayerPart",
              "ecr:CompleteLayerUpload",
              "ecr:PutImage"
          ],
          "Resource": [
              "arn:aws:ecr:${AWS_REGION}:${AWS_ACCOUNT_ID}:repository/tap-ls
p"
          ],
          "Sid": "TAPLSPScoped"
      }
  ]
}
EOF

# Create the TAP Local Source Proxy Role
aws iam create-role --role-name tap-local-source-proxy --assume-role-policy-d
ocument \
file://local-source-proxy-trust-policy.json
# Attach the Policy to the tap-local-source-proxy Role created above
aws iam put-role-policy --role-name tap-local-source-proxy --policy-name tapL
ocalSourcePolicy \
--policy-document file://local-source-proxy-policy.json
```

Using a secret with pull privileges only

You can use a secret with only pull privileges if you prefer to have a dedicated credential with a least-privilege policy, specifically for downloading artifacts instead of reusing credentials with higher privileges.

The secret containing this credential is distributed across developer namespaces by using the Secretgen `SecretExport` resource. Namespace Provisioner automatically imports it to the developer namespace. However, for development purposes, you can skip this step and use the same secret for both pushing and pulling artifacts.

To use a secret with pull privileges only, run:

```
# For all registries except ECR
tanzu secret registry add lsp-pull-credentials \
--username USERNAME-VALUE --password 'PASSWORD-VALUE' \
--server REGISTRY-SERVER \
--namespace tap-install --yes
```

Install Local Source Proxy

This topic tells you how to install and customize Local Source Proxy (LSP).

Prerequisites

Meet the prerequisites before attempting to install Local Source Proxy.

Install

Install Local Source Proxy by updating `tap-values.yaml` with the following details:

```
local_source_proxy:
  repository: REGISTRY-SERVER/REPOSITORY-PATH

  push_secret:
   name: lsp-push-credentials
   namespace: tap-install
   create_export: true

  pull_secret:
   name: lsp-pull-credentials
   namespace: tap-install
   create_export: true
```

Where:

- `local_source_proxy.repository` is required. This is the repository where all your source code will be uploaded. Examples of commonly used repositories include:
    - Harbor, which has the form `my-harbor.io/my-project/local-source`.
    - Docker Hub, which has the form `my-dockerhub-user/local-source` or `index.docker.io/my-user/local-source`.
    - Google Artifact Registry, which has the form `MY-REGISTRY-REGION-docker.pkg.dev/my-project/local-source/image`.
    - Google Cloud Registry, which has the form `gcr.io/my-project/local-source`.

  You are not restricted to using a repository from this list.
- `local_source_proxy.push_secret` is required. This is the push secret reference that has the permission to push artifacts to the repository mentioned in `local_source_proxy.repository`.
- `local_source_proxy.push_secret.create_export` is `false` by default. Set it as `true` to tell Local Source Proxy to create a `SecretExport` for the referenced secret and own it. Do one of the following if you're reusing the secret that already existed in your cluster:
    - Ensure that it's not exported by any other process
    - Set this flag to `false` and ensure that it is exportable to the `tap-local-source-system` namespace by using a `SecretExport` resource
- `local_source_proxy.pull_secret` is optional, but recommended for production. Use the credential that has pull permissions. You can re-use `lsp-push-credentials` that have pull access if you chose not to create a separate request for pulling.
- `local_source_proxy.pull_secret.create_export` is `false` by default. Set it as `true` to tell Local Source Proxy to create a SecretExport for the referenced secret and own it.

In a production installation of Tanzu Application Platform, do not share the registry secret that has write access across developer namespaces. Instead, distribute a separate registry secret that only has read access.

In such cases, `pull_secret` can be specified, and `source-controller` uses the `pull_secret` to pull source artifacts for deployment.

The `pull_secret` uses a Docker registry credentials secret referenced by its `name` and `namespace`.

To enable the Tanzu Application Platform installer to automatically create a `SecretExport` resource in the specified `namespace` for exporting the secret to developer namespaces, set `pull_secret.create_export` to `true`.

Alternatively, if `create_export` is set to `false`, you must manually create the `SecretExport` resource in the referenced `namespace`.

Do one of the following if you're reusing the secret that already existed in your cluster:

- Ensure that it's not exported by any other process
- Set this flag to `false` and ensure that it is exportable to developer namespaces by using a `SecretExport` resource

Create the SecretExport resource

If you left `create_export` as `true` in `tap-values.yaml`, skip this procedure. If you set the `create_export` to `false`, apply the following YAML in `tap-values.yaml` to create the `SecretExport` resource:

Use this YAML for `push-secret`:

```
---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretExport
metadata:
  name: lsp-push-credentials
  namespace: tap-install
spec:
  toNamespace: tap-local-source-system
```

Use this YAML for `pull-secret`:

```
---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretExport
metadata:
  name: lsp-pull-credentials
  namespace: tap-install
spec:
  toNamespace: *
```

Customize the installation

You can configure specific Local Source Proxy resources by using the following properties in `tap-values.yaml`, as described in the following sections.

Override default RBAC permissions to access the proxy service

When this section is not provided, the default behavior is to employ the `system:authenticated` group as the chosen option.

```
local_source_proxy:
  rbac_subjects_for_proxy_access:
  - kind: "Group"
    name: "system:authenticated"
    apiGroup: "rbac.authorization.k8s.io"
```

To grant access for a specific user or group to push images through Local Source Proxy, use the `rbac_subjects_for_proxy_access` property in the `local_source_proxy` configuration:

- Set `rbac_subjects_for_proxy_access.kind` to either `"User"` or `"Group"`.
- Set `rbac_subjects_for_proxy_access.name` to the user name or group name that requires authorization.
- Set `rbac_subjects_for_proxy_access.apiGroup` to either `rbac.authorization.k8s.io` or the custom `apiGroup` associated with the specified kind.

For more information about RoleBinding, see the Kubernetes documentation. These settings enable you to customize RBAC permissions for accessing the proxy service according to your specific user or group requirements.

Important

If you define the `rbac_subjects_for_proxy_access` configuration in the `tap-values.yaml` file, it supersedes the default `system:authenticated` role binding with the one specified in `tap-values.yaml`. This allows platform operators to restrict access to the proxy server exclusively to a predefined set of groups, users, or service accounts.

For example:

```
local_source_proxy:
  rbac_subjects_for_proxy_access:
  - kind: "Group"
    name: "alpha-group"
    apiGroup: "rbac.authorization.k8s.io"
```

Override default CPU and memory limits for controller pods

To configure the compute resource limits for the Local Source Proxy server, you can use the `proxy_configuration` section within the `local_source_proxy` configuration in `tap-values.yaml`.

Specify the following properties to set the resource limits:

- `proxy_configuration.limits.cpu`: Set the maximum CPU limit for the Local Source Proxy server.
- `proxy_configuration.limits.memory`: Configure the maximum memory limit available for the Local Source Proxy server.

For example:

```
local_source_proxy:
  proxy_configuration:
    limits:
      cpu: 500m
      memory: 100Mi
```

Use AWS Identity and Access Management (IAM) roles for ECR

If your Tanzu Application Platform installation is both

- On AWS with Amazon Elastic Kubernetes Service (EKS)
- Using the AWS Elastic Container Registry (ECR) for storing local source code

then write the IAM role that has push and pull permissions for `aws_iam_role_arn`. For example:

```
local_source_proxy:
  repository: 123456789012.dkr.ecr.us-east-1.amazonaws.com/local-source
```

```
  push_secret:
    aws_iam_role_arn: arn:aws:iam::123456789012:role/tap-local-source-proxy
```

You specify this IAM role in `aws_iam_role_arn` to assign it to the Kubernetes service account that the Local Source Proxy server uses. For the steps to create an IAM role and add the Amazon Resource Name (ARN) to the Kubernetes service account used by Local Source Proxy, see Prerequisites for Local Source Proxy.

Doing this allows the Local Source Proxy server to handle incoming image push requests with the appropriate IAM role-based permissions.

Increase or decrease the number of replicas

By default, the Local Source Proxy instance is configured with one replica in the deployment. You can increase or decrease the number of replicas by changing the `proxy_configuration.replicas` value.

For example:

```
local_source_proxy:
  proxy_configuration:
    replicas: 3
```

Specify CA certificate data for registries that use self-signed certificates

If you have a custom TLS certificate or a self-signed certificate for your registry server, and it uses a private certificate authority, you can set up the public CA certificate data by providing the necessary configuration under `local_source_proxy.ca_cert_data` in the `tap.values.yaml` file.

This allows you to provide the necessary certificate information for secure communication with the registry server. For example:

```
local_source_proxy:
  ca_cert_data: |
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
```

Local Source Proxy detects when `shared.ca_cert_data` is provided in `tap-values.yaml` and imports the CA certificate data from that key.

Troubleshoot Local Source Proxy

This topic helps you troubleshoot issues you might encounter with Local Source Proxy (LSP).

View Local Source Proxy server logs

Symptom

You encounter an error and need to view the Local Source Proxy server logs to investigate it.

Solution

Run

```
kubectl -n tap-local-source-system logs deployments/local-source-proxy
```

Use `-f` to follow the log output.

View Apps CLI plug-in health messages

Symptom

You need to read the Apps CLI plug-in health messages to assess the status of Local Source Proxy and its connectivity with the upstream repository.

Solution

Run

```
tanzu apps lsp health
```

Example:

```
$ tanzu apps lsp health
user_has_permission: true
reachable: true
upstream_authenticated: true
overall_health: true
message: All health checks passed
```

User does not have RBAC permission to list services

Symptom

You encounter any of these error messages:

```
$ tanzu apps workload apply
Error: Either Local Source Proxy is not installed on the Cluster or you don't have per
missions to access it
Reason: The current user does not have permission to access the local source proxy.
Messages:
- services "http:local-source-proxy:5001" is forbidden: User "abc@example.com" cannot
get resource "services/proxy" in API group "" in the namespace "tap-local-source-syste
m": requires one of ["container.services.proxy"] permission(s).
```

```
$ tanzu apps lsp health
user_has_permission: false
reachable: false
upstream_authenticated: false
overall_health: false
message: |-
  The current user does not have permission to access the local source proxy.
  Messages:
  - services "http:local-source-proxy:5001" is forbidden: User "abc@example.com" canno
t get resource "services/proxy" in API group "" in the namespace "tap-local-source-sys
tem": requires one of ["container.services.proxy"] permission(s).
```

Cause

Typically, this situation arises when a custom user or group is specified within the `rbac_subjects_for_proxy_access` section of `tap-values.yaml`.

Solution

Ensure that the user or group listed is valid. For more information about overriding default RBAC permissions to access the proxy service, see Override default RBAC permissions to access the proxy service.

Missing repository in Tanzu Application Platform values

Symptom

You encounter one of these error messages:

```
$ tanzu apps workload apply
Error: Local source proxy failed to upload source to the repository
Reason: Local source proxy is not healthy.
Messages:
- registry server configuration in the cluster is invalid
```

```
$ tanzu apps lsp health
user_has_permission: true
reachable: true
upstream_authenticated: false
overall_health: false
message: |
  Local source proxy is not healthy.
  Messages:
  - registry server configuration in the cluster is invalid
```

Cause

The cause might be that `tap-values.yaml` lacks a valid value for the repository.

Solution

Add a valid repository value to `tap-values.yaml` and wait for the app reconciliation to complete.

Missing or misconfigured registry secret

Symptom

You encounter one of these error messages:

```
$ tanzu apps workload apply
Error: Local source proxy failed to upload source to the repository
Reason: Local source proxy was unable to authenticate against the target registry.
Messages:
- GET https://gcr.io/v2/token?scope=repository:abc-playground/lsp-source:pull,push&ser
vice=gcr.io: UNAUTHORIZED: You don't have the needed permissions to perform this opera
tion, and you may have invalid credentials. To authenticate your request, follow the s
teps in: https://cloud.google.com/container-registry/docs/advanced-authentication
```

```
$ tanzu apps lsp health
user_has_permission: true
reachable: true
upstream_authenticated: false
overall_health: false
message: |-
  Local source proxy was unable to authenticate against the target registry.
  Messages:
  - GET https://gcr.io/v2/token?scope=repository:abc-playground/lsp-source:pull,push&s
ervice=gcr.io: UNAUTHORIZED: You don't have the
```

```
 needed permissions to perform this operation, and you may have invalid credentials. T
o authenticate your request, follow the steps in: https:/
/cloud.google.com/container-registry/docs/advanced-authentication
```

Cause

Potential causes include:

- A missing registry secret:
  - `push_secret` information is not available in the `local_source_proxy` section of `tap-values.yaml`.
  - `image_registry.secret` information is not available in the `shared` section of `tap-values.yaml`.
- If `push_secret` is used, the secret was not exported to the Local Source Proxy namespace. The credentials used in the secret do not match the configured external registry.

Solution

1. Ensure that at least one of the following entries is found in `tap-values.yaml`:
   - `push_secret` information in the `local_source_proxy` section
   - `image_registry.secret` information in the `shared` section
2. If `push_secret` is used, make sure that it can be exported to the Local Source Proxy namespace.
3. Ensure that the credentials used in the secret match the configured external registry.

Invalid credentials

Symptom

You encounter one of these error messages:

```
$ tanzu apps workload apply
Error: Local source proxy failed to upload source to the repository
Reason: Local source proxy was unable to authenticate against the target registry.
Messages:
- GET https://gcr.io/v2/token?scope=repository:abc-playground/lsp-source:pull,push&ser
vice=gcr.io: UNAUTHORIZED: Not Authorized.
```

```
$ tanzu apps lsp health # when using Harbor
user_has_permission: true
reachable: true
upstream_authenticated: false
overall_health: false
message: |-
  Local source proxy was unable to authenticate against the target registry.
  Messages:
  - 401 Unauthorized
```

```
$ tanzu apps lsp health # when using GCR
user_has_permission: true
reachable: true
upstream_authenticated: false
overall_health: false
message: |-
  Local source proxy was unable to authenticate against the target registry.
  Messages:
```

```
  - GET https://gcr.io/v2/token?scope=repository:abc-playground/lsp-source:pull,push&s
ervice=gcr.io: UNAUTHORIZED: Not Authorized.
```

Cause

The cause is the use of invalid credentials.

Solution

Change the credentials used in the secret to match those in the configured external registry.

Local Source Proxy doesn't automatically detect changes to `podspec`

Symptom

Local Source Proxy doesn't automatically detect changes to `podspec`.

Cause

AWS Elastic Container Registry (ECR) is configured as the external registry in `tap-values.yaml`.

Solution

Delete the old pods so that the new pods can mount the expected `podspec`, enabling access to the registry through the Identity and Access Management (IAM) role Amazon Resource Name (ARN).

Error: unknown command "lsp" for "apps"

Symptom

When running `tanzu apps lsp health` the CLI returns the error message

```
Error: unknown command "lsp" for "apps"
```

Cause

Tanzu CLI and the apps plug-in are out of date.

Solution

1. Install Tanzu CLI v0.12.0 or later.
2. Upgrade the apps plug-in by running:

   ```
   tanzu plugin upgrade apps
   ```

Error: i/o timeout

Symptom

When you run `tanzu apps lsp health` or `tanzu apps workload apply`, after a few minutes the CLI returns the following error message:

```
connect: i/o timeout
```

Cause

TCP port 5002 is not open between your control plane nodes and your worker nodes.

Solution

1. Open the port.
2. Run the command again.

Reference for Local Source Proxy

This topic gives you reference information for Local Source Proxy (LSP).

Default resources

Local Source Proxy comes with the standard `iterate` and `full` Tanzu Application Platform installation profiles. The default set of resources provisioned in a namespace is mostly predefined. However, certain resources are defined based on the configuration entries in `tap-values.yaml` under the `local_source_proxy` section.

For more information about installation profiles, see Installation profiles in Tanzu Application Platform.

The following table shows the list of resources that are templated in the installation of applicable profiles:

Namespace	Kind	Name	Reconcile
tap-local-source-system	Service	local-source-proxy	No
tap-local-source-system	Secret	local-source-proxy-values	Yes
local_source_proxy.push_secret.namespace (tap-values.yaml)	SecretExport	local_source_proxy.push_secret.name	Yes
tap-local-source-system	SecretImport	local_source_proxy.push_secret.name	Yes
tap-local-source-system	Deployment	local-source-proxy	No
tap-local-source-system	Role	local-source-proxy-manager	No
tap-local-source-system	RoleBinding	proxy-manager	No
tap-local-source-system	ServiceAccount	proxy-manager	No
tap-local-source-system	Role	push-artifact	No
tap-local-source-system	RoleBinding	service-proxy-role-binding	Yes

Overview of Namespace Provisioner

Namespace Provisioner provides a secure, automated way for you to provision namespaces with the resources and namespace-level privileges required for your workloads to function as intended in Tanzu Application Platform (commonly known as TAP).

Description

Namespace Provisioner enables platform operators to add additional customized namespace-scoped resources using GitOps to meet their organization's requirements and provides continuous

reconciliation using the kapp-controller to maintain the desired state of the namespace-scoped resources.

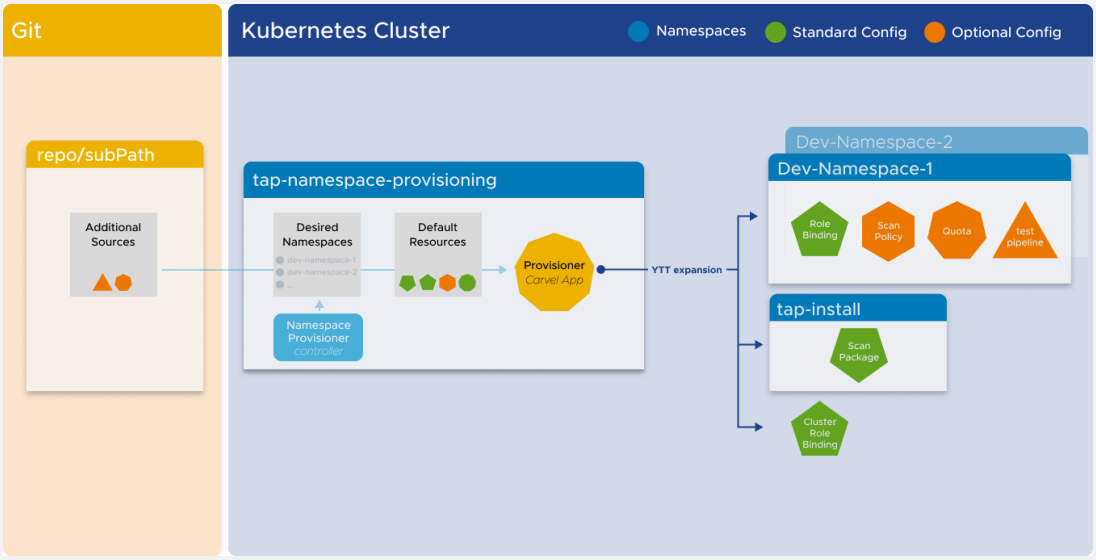
Namespace Provisioner enables operators that are new to Kubernetes to automate the provisioning of multiple developer namespaces in a shared cluster. For organizations that have already adopted Kubernetes, Namespace Provisioner is also compatible with existing Kubernetes tooling.

Modes

Use Namespace Provisioner with one of the following modes:

Controller mode

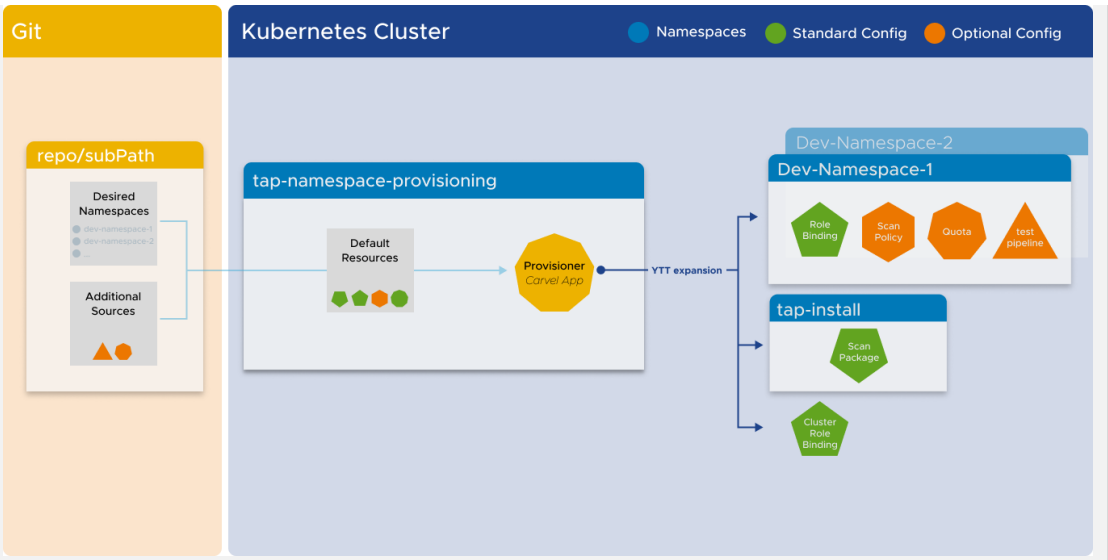
Controller mode has the following characteristics:



- The list of developer namespaces is managed by the Namespace Provisioner controller using a label selector `apps.tanzu.vmware.com/tap-ns=""`
- Namespace Provisioner creates default resources in all managed namespaces.
- Namespace Provisioner creates additional Platform Operator templated resources stored in Git repository locations specified under the `additional_sources` section in Namespace Provisioner configuration. For more information, see Customize Installation of Namespace Provisioner.

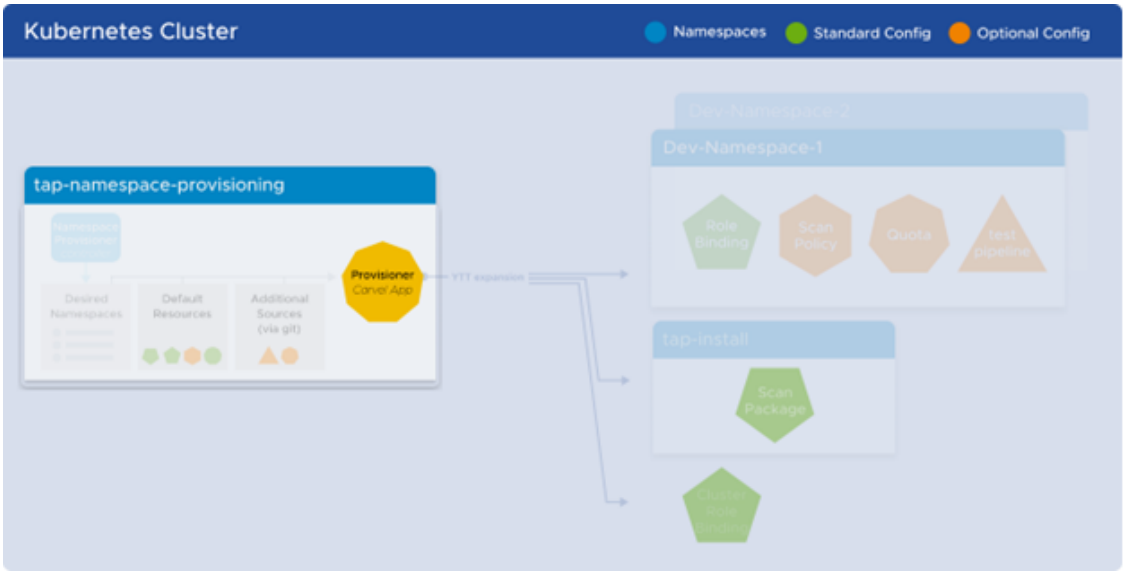
GitOps mode

Gitops mode has the following characteristics



- The list of developer namespaces is managed in a Git repository that is specified in the `gitops_install` section of the Namespace Provisioner configuration.
- Namespace Provisioner creates default resources that are shipped Out of the Box in all managed namespaces.
- Namespace Provisioner creates additional Platform Operator templated resources stored in Git repositories specified under `additional_sources` in Namespace Provisioner configuration. For more information, see Customize Installation of Namespace Provisioner.

Provisioner Carvel application



Namespace Provisioner consists of a Carvel application called `provisioner` that facilitates the creation of resources in the managed developer namespaces. The `provisioner` application uses ytt to templatize a set of resources into installations in multiple namespaces.

Desired namespaces

The following section describes how the list of desired developer namespaces is managed in controller and GitOps modes.

Controller mode
In controller mode, the list of desired namespaces used by the `provisioner` application to create resources in, is maintained in the `desired-namespaces` ConfigMap. This ConfigMap is managed by the Namespace Provisioner controller and it provides a declarative way to indicate which namespaces should be populated with resources. The ConfigMap consists of a list of namespace objects, with a required `name` parameter, and optional additional parameters which are used as `data.values` for customizing defined resources.

For example,

```
---
apiVersion: v1
kind: ConfigMap
metadata:
    name: desired-namespaces
    namespace: tap-namespace-provisioning
    annotations:
        kapp.k14s.io/create-strategy: fallback-on-update
        namespace-provisioner.apps.tanzu.vmware.com/no-overwrite: "" #! This annotat
ion tells the provisioner app to not override this configMap as this is your desired
state.
data:
    namespaces.yaml: |
        #@data/values
        ---
        namespaces:
        - name: dev-ns1
        # additional parameters about dev-ns1 added via label/annotations or GitOps
        - name: dev-ns2
        # additional parameters about dev-ns1 added via label/annotations or GitOps
```

GitOps mode
In the GitOps mode, the list of desired namespaces used by the `provisioner` application to create resources in, is maintained in a Git repository as a ytt data values file as shown in this sample file. This file provides a declarative way to indicate which namespaces should be populated with resources. For more information, see the `Options if using GitOps` section in Customize Install.

Namespace Provisioner controller

The Namespace Provisioner controller (controller) is installed by default and manages the content contained in the `desired-namespaces` ConfigMap. The controller watches namespaces in the cluster and updates the `desired-namespaces` ConfigMap with a list of all namespaces that match the namespace label selector.The default namespace label selector is `apps.tanzu.vmware.com/tap-ns`. For more information, see Use a different label selector than default.

Overview of Namespace Provisioner

Namespace Provisioner provides a secure, automated way for you to provision namespaces with the resources and namespace-level privileges required for your workloads to function as intended in Tanzu Application Platform (commonly known as TAP).

Description

Namespace Provisioner enables platform operators to add additional customized namespace-scoped resources using GitOps to meet their organization's requirements and provides continuous reconciliation using the kapp-controller to maintain the desired state of the namespace-scoped resources.

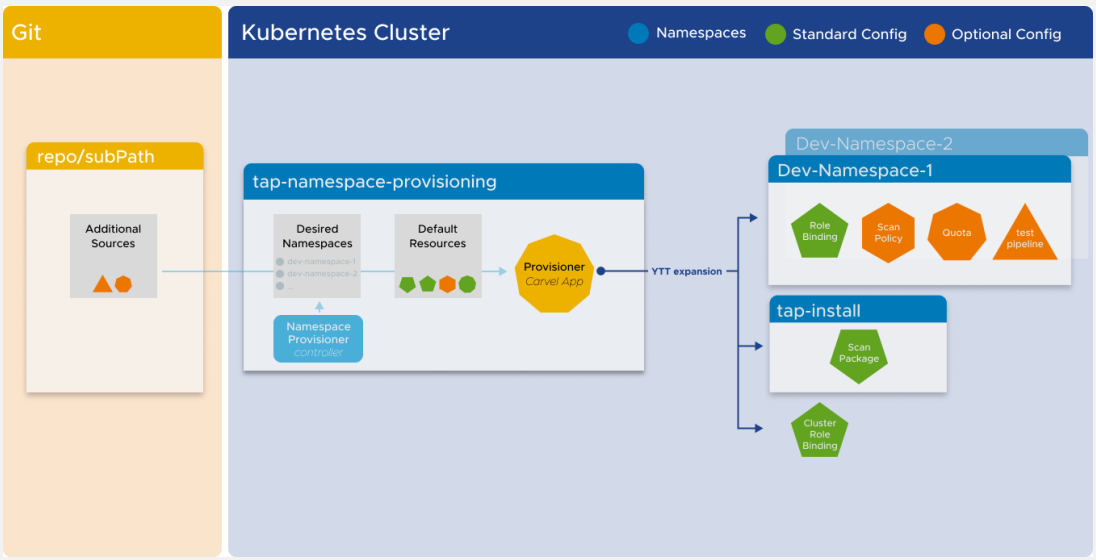
Namespace Provisioner enables operators that are new to Kubernetes to automate the provisioning of multiple developer namespaces in a shared cluster. For organizations that have already adopted Kubernetes, Namespace Provisioner is also compatible with existing Kubernetes tooling.

Modes

Use Namespace Provisioner with one of the following modes:

Controller mode

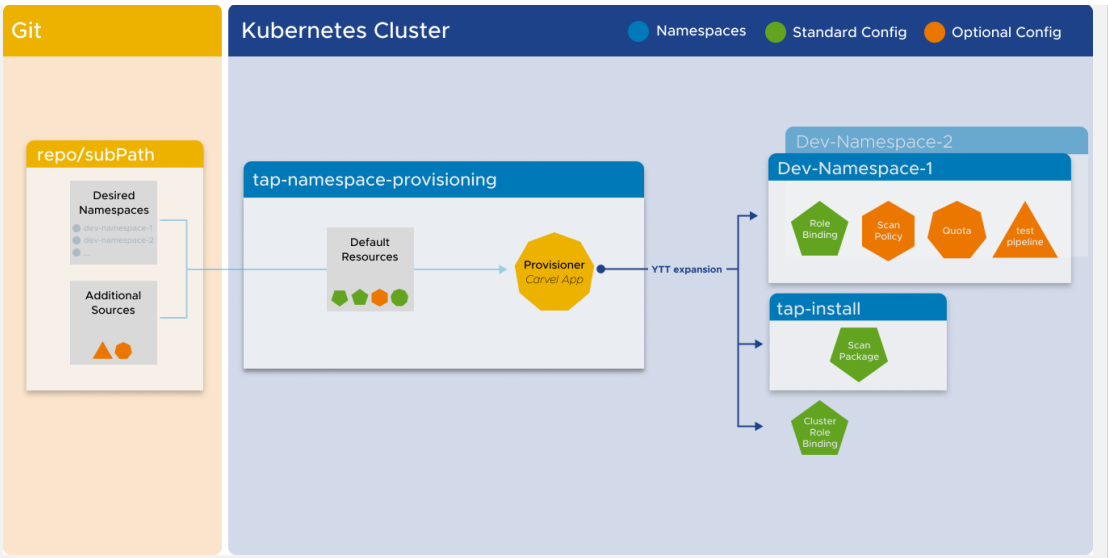
Controller mode has the following characteristics:



- The list of developer namespaces is managed by the Namespace Provisioner controller using a label selector `apps.tanzu.vmware.com/tap-ns=""`
- Namespace Provisioner creates default resources in all managed namespaces.
- Namespace Provisioner creates additional Platform Operator templated resources stored in Git repository locations specified under the `additional_sources` section in Namespace Provisioner configuration. For more information, see Customize Installation of Namespace Provisioner.

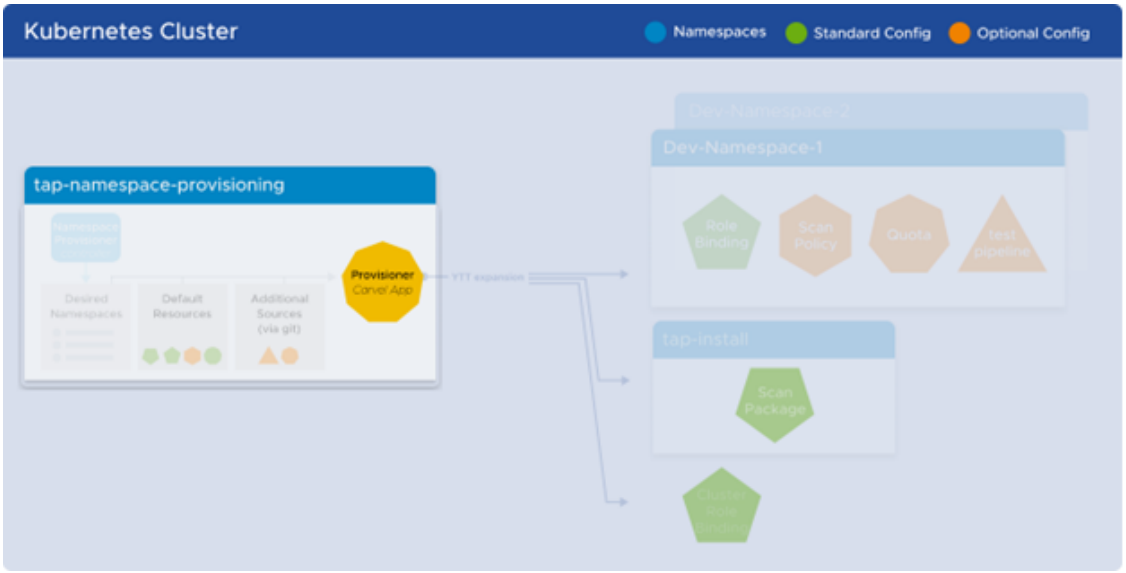
GitOps mode

Gitops mode has the following characteristics



- The list of developer namespaces is managed in a Git repository that is specified in the `gitops_install` section of the Namespace Provisioner configuration.
- Namespace Provisioner creates default resources that are shipped Out of the Box in all managed namespaces.
- Namespace Provisioner creates additional Platform Operator templated resources stored in Git repositories specified under `additional_sources` in Namespace Provisioner configuration. For more information, see Customize Installation of Namespace Provisioner.

Provisioner Carvel application



Namespace Provisioner consists of a Carvel application called `provisioner` that facilitates the creation of resources in the managed developer namespaces. The `provisioner` application uses ytt to templatize a set of resources into installations in multiple namespaces.

Desired namespaces

The following section describes how the list of desired developer namespaces is managed in controller and GitOps modes.

Controller mode
In controller mode, the list of desired namespaces used by the `provisioner` application to create resources in, is maintained in the `desired-namespaces` ConfigMap. This ConfigMap is managed by the Namespace Provisioner controller and it provides a declarative way to indicate which namespaces should be populated with resources. The ConfigMap consists of a list of namespace objects, with a required `name` parameter, and optional additional parameters which are used as `data.values` for customizing defined resources.

For example,

```
---
apiVersion: v1
kind: ConfigMap
metadata:
    name: desired-namespaces
    namespace: tap-namespace-provisioning
    annotations:
        kapp.k14s.io/create-strategy: fallback-on-update
        namespace-provisioner.apps.tanzu.vmware.com/no-overwrite: "" #! This annotat
ion tells the provisioner app to not override this configMap as this is your desired
state.
data:
    namespaces.yaml: |
        #@data/values
        ---
        namespaces:
        - name: dev-ns1
        # additional parameters about dev-ns1 added via label/annotations or GitOps
        - name: dev-ns2
        # additional parameters about dev-ns1 added via label/annotations or GitOps
```

GitOps mode
In the GitOps mode, the list of desired namespaces used by the `provisioner` application to create resources in, is maintained in a Git repository as a ytt data values file as shown in this sample file. This file provides a declarative way to indicate which namespaces should be populated with resources. For more information, see the `Options if using GitOps` section in Customize Install.

Namespace Provisioner controller

The Namespace Provisioner controller (controller) is installed by default and manages the content contained in the `desired-namespaces` ConfigMap. The controller watches namespaces in the cluster and updates the `desired-namespaces` ConfigMap with a list of all namespaces that match the namespace label selector.The default namespace label selector is `apps.tanzu.vmware.com/tap-ns`. For more information, see Use a different label selector than default.

Get started with Namespace Provisioner

This topic provides a list of topics to help you get started with Namespace Provisioner.

Provision Developer Namespaces

Customize Installation of Namespace Provisioner

Setup for OOTB Supply Chains

Provision developer namespaces in Namespace Provisioner

This topic tells you how to use Namespace Provisioner to provision developer namespaces in Tanzu Application Platform (commonly known as TAP).

Prerequisite

- The Namespace Provisioner package is installed and reconciled.
- The registry-credential secret referenced by the supply chain components for pulling and pushing images is added to **tap-install** and exported to all namespaces.

Example secret creation, exported to all namespaces:

```
tanzu secret registry add registry-credentials --server REGISTRY-SERVER --username REG
ISTRY-USERNAME --password REGISTRY-PASSWORD --export-to-all-namespaces --yes --namespa
ce tap-install
```

Important

Namespace Provisioner creates a secret called registries-credentials in each managed namespace which is a placeholder secret filled indirectly by secretgen-controller with all the registry credentials exported for that managed namespace.

Manage a list of developer namespaces

There are two ways to manage the list of developer namespaces that are managed by Namespace Provisioner.

Using Namespace Provisioner Controller

`tap-values.yaml` configuration example:

```
namespace_provisioner:
  controller: true
```

The imperative way is to create the namespace using kubectl or using other means and label it using the default selector.

1. Create a namespace using kubectl or any other means

   ```
   kubectl create namespace YOUR-NEW-DEVELOPER-NAMESPACE
   ```

2. Label your new developer namespace with the default *namespace_selector* `apps.tanzu.vmware.com/tap-ns=""`.

   ```
   kubectl label namespaces YOUR-NEW-DEVELOPER-NAMESPACE apps.tanzu.vmware.com/t
   ap-ns=""
   ```

   - This label tells the Namespace Provisioner controller to add this namespace to the desired-namespaces ConfigMap.
   - By default, the label's value can be anything, including "".

- If required, you can change the default label selector, see Customize Installation of Namespace Provisioner.

3. Run the following command to verify the default resources have been created in the namespace:

```
kubectl get secrets,serviceaccount,rolebinding,pods,workload,configmap,limitr
ange -n YOUR-NEW-DEVELOPER-NAMESPACE
```

For example:

```
NAME                              TYPE                                  DATA   AGE
secret/app-tls-cert               kubernetes.io/tls                     3      19s
secret/registries-credentials     kubernetes.io/dockerconfigjson        1      26s
secret/scanner-secret-ref         kubernetes.io/dockerconfigjson        1      20s

NAME                              SECRETS   AGE
serviceaccount/default            1         4h7m
serviceaccount/grype-scanner      2         20s

NAME                                                                  ROLE
AGE
rolebinding.rbac.authorization.k8s.io/default-permit-deliverable   ClusterRol
e/deliverable   26s
rolebinding.rbac.authorization.k8s.io/default-permit-workload      ClusterRol
e/workload      26s

NAME                            DATA   AGE
configmap/kube-root-ca.crt      1      38h

NAME                              CREATED AT
limitrange/dev-lr   2023-03-08T04:18:58Z
```

Using GitOps

The GitOps approach provides a fully declarative way to create developer namespaces managed by Namespace Provisioner.

`tap-values.yaml` configuration example:

```
namespace_provisioner:
  controller: false
  gitops_install:
    ref: origin/main
    subPath: ns-provisioner-samples/gitops-install
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
```

This GitOps configuration does the following things:

- `controller: false` - The Namespace Provisioner package does not install the controller. The list of namespaces is managed in a GitOps repository instead.
- The `gitops-install` directory specified as the `subPath` value includes two files:
    1. desired-namespace.yaml contains the list of developer namespaces in a ytt data.values format.
    2. namespaces.yaml contains a Kubernetes namespace object.

Note

If you have another tool like Tanzu Mission Control or some other process that is taking care of creating namespaces for you, and you don't want a Namespace

Provisioner to create the namespaces, you can delete this file from your GitOps install repository.

Important The `tap-values.yaml` configuration example above creates the following two namespaces: `dev` and `qa`. If these namespaces already exist in your cluster, remove them or rename the namespaces in your GitOps repository so they do not conflict with existing resources.

Run the following command to verify the default resources are created in the namespace:

```
kubectl get secrets,serviceaccount,rolebinding,pods,workload,configmap,limitrange -n
dev
```

For example:

```
NAME                              TYPE                                  DATA   AGE
secret/app-tls-cert               kubernetes.io/tls                     3      52s
secret/registries-credentials     kubernetes.io/dockerconfigjson        1      59s
secret/scanner-secret-ref         kubernetes.io/dockerconfigjson        1      53s

NAME                              SECRETS   AGE
serviceaccount/default            1         59s
serviceaccount/grype-scanner      2         53s

NAME                                                                  ROLE
AGE
rolebinding.rbac.authorization.k8s.io/default-permit-deliverable      ClusterRole/deliv
erable   59s
rolebinding.rbac.authorization.k8s.io/default-permit-workload         ClusterRole/workl
oad      59s

NAME                            DATA   AGE
configmap/kube-root-ca.crt      1      59s

NAME                  CREATED AT
limitrange/dev-lr     2023-03-08T04:22:20Z
```

For more information, see the GitOps section of Customize Installation of Namespace Provisioner.

Enable additional users with Kubernetes RBAC

Namespace Provisioner does not support enabling additional users with Kubernetes RBAC. Support is planned for an upcoming release. Until Namespace Provisioner support is provided, follow the instructions in Enable additional users with Kubernetes RBAC.

Customize Namespace Provisioner installation

This topic tells you how to customize a standard installation of Namespace Provisioner in Tanzu Application Platform (commonly known as TAP).

Namespace Provisioner is packaged and distributed using a set of Carvel tools. The Namespace Provisioner package is installed as part of all the standard installation profiles except the View profile. For more information about installation profiles, see Installation profiles in Tanzu Application Platform.

The default set of resources provisioned in a namespace is based on a combination of the Tanzu Application Platform installation profile employed and the supply chain that is installed on the

cluster. For a list of what resources are created for different profile and supply chain combinations, see the Default Resources mapping table.

To see the Namespace Provisioner Package Schema for all configurable values, run:

```
tanzu package available get namespace-provisioner.apps.tanzu.vmware.com/0.3.0 --values
-schema -n tap-install
```

Different package customization options are available depending on what method you use to manage the list of developer namespaces:

Add additional resources to your namespaces from your GitOps repository

- `additional_sources`: This is an array of Git repository locations that contain Platform Operator templated resources to create in the provisioned namespaces, in addition to the default resources. The format of the Git repository locations must follow the “fetch” section of the kapp controller App specification, and only the Git type fetch is supported.
- `additional_sources[].git` This entry can include a secretRef specified for providing authentication details for connecting to a private Git repository. For more information, see Git Authentication for Private repository. The following parameters are available:
    - `name`: The name of the secret to be imported and used as valuesFrom in kapp.
    - `namespace`: The namespace where the secret exists.
    - `create_export`: A Boolean flag that controls the creation of a SecretExport resource in the namespace. The default value is false. If the secret is already exported, make sure that it is exported to the `tap-namespace-provisioning` namespace.
    - `path`: (**Optional**) This must start with the prefix `_ytt_lib/`. Namespace Provisioner mounts all the additional sources as a ytt library so it can expand the manifests in the additional sources for all managed namespaces using the logic in the expansion template. The path after the `_ytt_lib` prefix can be any string value, and must be unique across all additional sources. If you do not provide a `path`, Namespace Provisioner generates a `path` using `url` and `subPath`.

Important

Namespace Provisioner relies on kapp-controller for any tasks involving communication with external services, such as registries or Git repositories. When operating in air-gapped environments or other scenarios where external services are secured by a Custom CA certificate, you must configure kapp-controller with the CA certificate data to prevent X.509 certificate errors. For more information, see Deploy onto Cluster in the Cluster Essentials for VMware Tanzu documentation.

If the additional sources contain a resource that is scoped to a specific namespace, it is created in that namespace with a modified name that includes the developer namespace name. For example, the resource name will be “{resource name}-{developer namespace name}”.

If the additional sources include resources without any specified namespaces, and these resources are not cluster-scoped, Namespace Provisioner creates those resources in all of the namespaces it manages. However, if the resource is cluster-scoped, only a single instance of the resource is created.

Sample `tap-values.yaml` configuration:

Using Namespace Provisioner Controller

The Git repository is configured under `additional_sources`.

```
namespace_provisioner:
  controller: true
  additional_sources:
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/testing-scanning-supplychain
      url: https://github.com/vmware-tanzu/application-accelerator-samples.git
      # secretRef section is only needed if connecting to a Private Git repo
      secretRef:
        name: git-auth
        namespace: tap-install
        create_export: true
    path: _ytt_lib/testing-scanning-supplychain-setup
```

Using GitOps

The Git repository is configured under `additional_sources`.

```
namespace_provisioner:
  controller: false
  additional_sources:
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/testing-scanning-supplychain
      url: https://github.com/vmware-tanzu/application-accelerator-samples.git
      # secretRef section is only needed if connecting to a Private Git repo
      secretRef:
        name: git-auth
        namespace: tap-install
        create_export: true
    path: _ytt_lib/testing-scanning-supplychain-setup
  gitops_install:
    ref: origin/main
    subPath: ns-provisioner-samples/gitops-install
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
```

If a path is not specified in the `additional_sources` configuration, Namespace Provisioner automatically generates a path as follows: `_ytt_lib/applicaton-accelerator-samples-git-ns-provisioner-samples-testing-scaning-supplychain-0`

For more information, see Git Authentication for Private repository.

Adjust sync period of Namespace Provisioner

The `sync_period` parameter is the interval at which the Namespace Provisioner reconciles. It must be specified in the format of time + unit. The minimum allowed `sync_period` is 30 seconds. If a value lower than 30 seconds is specified in the `tap-values.yaml` file, Namespace Provisioner automatically sets the `sync_period` to 30 seconds. If no value is specified, the default `sync_period` is `1m0s`.

Sample `tap-values.yaml` configuration:

Using Namespace Provisioner Controller

Use the `sync_period` key.

```
namespace_provisioner:
  sync_period: 2m0s
```

Using GitOps
Use the `sync_period` key.

```
namespace_provisioner:
  controller: false
  sync_period: 1m0s
  gitops_install:
    ref: origin/main
    subPath: ns-provisioner-samples/gitops-install
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
```

Import user defined secrets in YAML format as ytt data.values

The `import_data_values_secrets` is an array of additional secrets in YAML format that can be imported into the Namespace Provisioner as data.values under the `data.values.imported` key. Namespace Provisioner creates a SecretImport for each secret listed in the array in the `tap-namespace-provisioning` namespace. Alternatively, you can manually create a SecretExport for the same secrets and export them to the `tap-namespace-provisioning` namespace. The following parameters are available:

- `name`: The name of the secret to be imported to use as valuesFrom in kapp.
- `namespace`: The namespace where the secret exists.
- `create_export`: A Boolean flag that indicates whether a `SecretExport` resource is created in the namespace. The default value is `false`. If the secret is already exported, ensure that it is exported to the `tap-namespace-provisioning` namespace.

Note

The `stringData` key of the secret must have `.yaml` or `.yml` suffix.

Example secret:

```
# Format of the secret that is importable under data.values.imported
apiVersion: v1
kind: Secret
metadata:
  name: user-defined-secrets
type: Opaque
stringData:
  # Key needs to have .yaml or .yml at the end
  content.yaml: |
    key1: value1
    key2: value2
```

Sample `tap-values.yaml` configuration:

Using Namespace Provisioner Controller
The list of secrets are imported under `import_data_values_secrets`.

```
namespace_provisioner:
  controller: true
  import_data_values_secrets:
  - name: user-defined-secrets
```

```
    namespace: tap-install
    create_export: true
```

Using GitOps

The list of secrets are imported under `import_data_values_secrets`.

```
namespace_provisioner:
  controller: false
  import_data_values_secrets:
  - name: user-defined-secrets
    namespace: tap-install
    create_export: true
  gitops_install:
    ref: origin/main
    subPath: ns-provisioner-samples/gitops-install
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
```

Use AWS IAM roles

If you are installing Tanzu Application Platform on Amazon Elastic Kubernetes Service (EKS), you can use the IAM Role specified in `aws_iam_role_arn` to configure the Kubernetes service account used by the workload and the supply chain components. For additional details on the process of creating the IAM Role for workloads and obtaining the `aws_iam_role_arn`, see Create AWS Resources for Tanzu Application Platform.

Sample `tap-values.yaml` configuration:

Using Namespace Provisioner Controller

Add the AWS IAM Role to `aws_iam_role_arn`.

```
namespace_provisioner:
  controller: yes
  aws_iam_role_arn: "arn:aws:iam::123456789012:role/EKSIAMRole"
```

Using GitOps

Add the AWS IAM Role to `aws_iam_role_arn`.

```
namespace_provisioner:
  controller: false
  aws_iam_role_arn: "arn:aws:iam::123456789012:role/EKSIAMRole"
  gitops_install:
    ref: origin/main
    subPath: ns-provisioner-samples/gitops-install
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
```

Apply default parameters to all namespaces

The `default_parameters` is an array of parameters that are applied to all namespaces. Use these parameters as ytt, such as `data.values.default_parameters` for templating default and additional resources.

Sample `tap-values.yaml` configuration:

Using Namespace Provisioner Controller

Use the `default_parameters` with the desired parameter.

```
namespace_provisioner:
  controller: yes
  default_parameters:
    limits:
      default:
        cpu: 1.7
        memory: 1Gi
      defaultRequest:
        cpu: 100m
        memory: 1Gi
```

Using GitOps

Use the `default_parameters` with the desired parameter.

```
namespace_provisioner:
  controller: false
  default_parameters:
    limits:
      default:
        cpu: 1.7
        memory: 1Gi
      defaultRequest:
        cpu: 100m
        memory: 1Gi
  gitops_install:
    ref: origin/main
    subPath: ns-provisioner-samples/gitops-install
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
```

Import overlay secrets

The `overlay_secrets` is a list of secrets that contains Carvel ytt overlay definitions. These overlays are applied to the resources created by the Namespace Provisioner. If the secrets are located in a different namespace, they are imported to the `namespace-provisioner` namespace.

Note

The `stringData` key of the secret must have `.yaml` or `.yml` suffix.

Sample secret with overlay to be used:

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: grype-package-overlay
  namespace: tap-install
  annotations:
    kapp.k14s.io/change-rule: "delete after deleting tap"
stringData:
  grype-package-overlay.yaml: |
    #@  load("@ytt:overlay", "overlay")
    #@
    #@  def matchGrypeScanners(index, left, right):
    #@    if left["apiVersion"] != "packaging.carvel.dev/v1alpha1" or left["kind"] !=
"PackageInstall":
    #@      return False
    #@    end
    #@    return "metadata" in left and "name" in left["metadata"] and left["metadat
```

```
a"]["name"].startswith("grype-scanner")
    #@  end

    #@overlay/match by=matchGrypeScanners, expects="0+"
    ---
    metadata:
      annotations:
        #@overlay/match expects="0+"
        ext.packaging.carvel.dev/ytt-paths-from-secret-name.0: my-grype-overlay-secret
EOF
```

Sample `tap-values.yaml` configuration:

Using Namespace Provisioner Controller

The list of secrets with the overlay are set under `overlay_secrets`.

```
namespace_provisioner:
  controller: true
  overlay_secrets:
  - name: grype-package-overlay
    namespace: tap-install
    create_export: true
```

Using GitOps

The list of secrets with the overlay are set under `overlay_secrets`.

```
namespace_provisioner:
  controller: false
  overlay_secrets:
  - name: grype-package-overlay
    namespace: tap-install
    create_export: true
  gitops_install:
    ref: origin/main
    subPath: ns-provisioner-samples/gitops-install
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
```

Furthermore, you have the following options for customization:

Options if using Controller

If you are using the controller to manage the list of developer namespaces, you have the following additional customization options available:

- Use a different label selector than default
- Override the default CPU and memory limits for controller pods
- Customize the label and annotation prefixes that controller watches

Use a different label selector than default

Use the `namespace_selector` to specify the label selector used by the controller to identify the namespaces that will be included in the desired-namespaces ConfigMap.

Sample `tap-values.yaml` configuration:

```
namespace_provisioner:
  controller: true
  namespace_selector:
    matchExpressions:
    - key: apps.tanzu.vmware.com/tap-ns
      operator: Exists
```

Override the default CPU and memory limits for controller pods

To configure the compute resources for the Namespace Provisioner controllers, you can use the `controller_resources` section in the Namespace Provisioner configuration in `tap-values.yaml`.

To set the maximum CPU and memory limits for the controllers, edit the `controller_resources.resources.limits.cpu` and `controller_resources.resources.limits.memory` values.

Similarly, you can configure the minimum CPU capacity and memory requests for the controllers by adjusting the `controller_resources.resources.requests.cpu` and `controller_resources.resources.requests.memory` settings.

Sample `tap-values.yaml` configuration:

```
namespace_provisioner:
  controller: true
  controller_resources:
    resources:
      limits:
        cpu: 500m
        memory: 100Mi
      requests:
        cpu: 100m
        memory: 20Mi
```

Customize the label and annotation prefixes that controller watches

The `parameter_prefixes` is an array of label and annotation prefixes that the Namespace Provisioner controller uses to identify and include namespace-specific parameters in the desired-namespaces ConfigMap. These parameters can then be used as ytt data.values for templating both default and additional resources.

For example, if the value `tap.tanzu.vmware.com` is specified in `parameter_prefixes`, the Namespace Provisioner controller searches for annotations or labels in a provisioned namespace that begin with the prefix `tap.tanzu.vmware.com/`. It extracts those annotations or labels and uses them as parameters for further configuration and customization.

Sample `tap-values.yaml` configuration:

```
namespace_provisioner:
  controller: yes
  parameter_prefixes:
  - tmc.cloud.vmware.com
  - tap.tanzu.vmware.com
```

Options if using GitOps

If you are using GitOps to manage the list of developer namespaces, you have the following customization option:

Use GitOps to manage developer namespaces list

`gitops_install` is a Git repository configuration with the list of namespaces to be provisioned.

Only use the gitops_install section when `controller: false` is set. If this section is used in conjunction with `controller: true`, the Namespace Provisioner package fails to reconcile, resulting in an error message stating `controller: false when using 'gitops_install' in provided values`.

Files in the Git repository must have a `.yaml` or `.yml` extension.

The `gitops_install` section can have the following entries:

- `url`: The Git repository URL (required)

- `subPath`: The Git repository subpath where the file is
- `ref`: The Git repository reference, the default is origin/main
- `secretRef`: If the repository needs authentication, the reference to the secret is set here
    - `name`: The name of the secret to be used for the repository authentication, see Git Authentication for Private repository.
    - `namespace`: The namespace where the secret is created. Namespace Provisioner creates a Carvel secretgen SecretImport from this namespace to the Namespace Provisioner namespace.
    - `create_export`: A Boolean flag to create a Carvel secretgen SecretExport from the given namespace to Namespace Provisioner namespace. The default value is `false`.

  Sample `gitops_install` repository file:

  Note

  The Carvel data header (#@data/values) is required in this file.

```
#@data/values
---
namespaces:
- name: dev
- name: qa
```

```
#@ load("@ytt:data", "data")
#! This loop will now loop over the namespace list in
#! in ns.yaml and will create those namespaces.
#@ for ns in data.values.namespaces:
---
apiVersion: v1
kind: Namespace
metadata:
  name: #@ ns.name
#@ end
```

This file in the sample repository creates the namespaces in the namespaces list so no manual intervention is required.

Sample `tap-values.yaml` configuration:

```
namespace_provisioner:
  controller: false
  gitops_install:
    ref: origin/main
    subPath: ns-provisioner-samples/gitops-install
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
```

Set up Out of the Box Supply Chains in Namespace Provisioner

This topic tells you how to set up Namespace Provisioner to automate resource creation needed for workloads to run on Out of the Box Supply Chain Basic and Out of the Box Supply Chain with Testing.

Out of the Box Supply Chain Basic

To create a developer namespace, see Provision Developer Namespaces.

Namespace Provisioner creates a set of default resources in all managed namespaces which are sufficient to run a workload through the Out of the Box Supply Chain Basic.

Run the following Tanzu CLI command to create a workload in your developer namespace.

Using the Tanzu CLI:

- Create workload using Tanzu apps CLI command:

```
tanzu apps workload apply tanzu-java-web-app \
--git-repo https://github.com/sample-accelerators/tanzu-java-web-app \
--git-branch main \
--type web \
--app tanzu-java-web-app \
--namespace DEVELOPER-NAMESPACE \
--tail \
--yes
```

Using a workload YAML:

- Create a `workload.yaml` file:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  labels:
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/workload-type: web
  name: tanzu-java-web-app
  namespace: DEVELOPER-NAMESPACE
spec:
  source:
    git:
      ref:
        branch: main
      url: https://github.com/sample-accelerators/tanzu-java-web-app
```

Out of the Box Supply Chain with Testing

The Out of the Box Supply Chain with Testing adds the **source-tester** step in the supply chain which tests the source code pulled by the supply chain. For source code testing to work in the supply chain, Tekton Pipelines must exist in the same namespace as the Workload so that the Tekton PipelineRun object that is created to run the tests can reference the developer-provided Pipeline.

By default, the workload is matched to the corresponding pipeline to run using labels. Pipelines must have the label `apps.tanzu.vmware.com/pipeline: test`. This provides a default match if no other labels are provided, but you can add additional labels. The pipeline expects two parameters:

- `source-url` is an HTTP address with a `.tar.gz` file containing all the source code to test.
- `source-revision` is the revision of the commit or image reference, such as setting `workload.spec.source.image` instead of `workload.spec.source.git`.

For example:

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
```

```
  name: tekton-pipeline-java
  labels:
    apps.tanzu.vmware.com/pipeline: test      # (!) required
spec:
  params:
    - name: source-url                        # (!) required
    - name: source-revision                   # (!) required
  tasks:
    - name: test
      params:
        - name: source-url
          value: $(params.source-url)
        - name: source-revision
          value: $(params.source-revision)
      taskSpec:
        params:
          - name: source-url
          - name: source-revision
        steps:
          - name: test
            image: gradle
            script: |-
              cd `mktemp -d`
              wget -qO- $(params.source-url) | tar xvz -m
              ./mvnw test
```

Add Java Tekton Pipelines to your developer namespace

To create a developer namespace, see the Provision Developer Namespaces.

Namespace Provisioner can automate the creation of a Tekton pipeline needed for the workload to run on an Out of the Box Supply Chain with Testing. You can create an example pipeline in your GitOps repository and add your GitOps repository as an additional source in Namespace Provisioner configuration in `tap-values.yaml`. See Customize Installation of Namespace Provisioner.

Add the following configuration to `tap-values.yaml` to add this example java pipeline to your developer namespace:

Using Namespace Provisioner Controller:

- Example `tap-values.yaml` configuration:

```
namespace_provisioner:
  controller: true
  additional_sources:
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/testing-supplychain
      url: https://github.com/vmware-tanzu/application-accelerator-samples.git
```

Using GitOps:

- Example `tap-values.yaml` configuration:

```
namespace_provisioner:
  controller: false
  additional_sources:
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/testing-supplychain
      url: https://github.com/vmware-tanzu/application-accelerator-samples.git
  gitops_install:
    ref: origin/main
```

```
    subPath: ns-provisioner-samples/gitops-install
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
```

The example pipeline resource has the following ytt logic which creates this pipeline only if the following conditions are met:

- `supply_chain` in your `tap-values.yaml` file is either `testing` or `testing_scanning`
- `profile` in your `tap-values.yaml` file is either `full, iterate`, or `build`.

```
#@ load("@ytt:data", "data")
#@ def in_list(key, list):
#@  return hasattr(data.values.tap_values, key) and (data.values.tap_values[key] in li
st)
#@ end
#@ if/end in_list('supply_chain', ['testing', 'testing_scanning']) and in_list('profil
e', ['full', 'iterate', 'build']):
```

After adding the additional source to your `tap-values.yaml` file, you can see the `tekton-pipeline-java` created in your developer namespace. To verify that the pipeline is created correctly:

```
kubectl get pipeline.tekton.dev -n DEVELOPER-NAMESPACE
```

Where `DEVELOPER-NAMESPACE` is the name of the new developer namespace you want to use.

Run the following Tanzu CLI command to create a workload in your developer namespace:

Using the Tanzu CLI:

- Create workload using Tanzu apps CLI command.

```
tanzu apps workload apply tanzu-java-web-app \
--git-repo https://github.com/sample-accelerators/tanzu-java-web-app \
--git-branch main \
--type web \
--app tanzu-java-web-app \
--label apps.tanzu.vmware.com/has-tests="true" \
--namespace DEVELOPER-NAMESPACE \
--tail \
--yes
```

Using workload YAML:

- Create a `workload.yaml` file:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  labels:
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/has-tests: "true"
    apps.tanzu.vmware.com/workload-type: web
  name: tanzu-java-web-app
  namespace: DEVELOPER-NAMESPACE
spec:
  source:
    git:
      ref:
        branch: main
      url: https://github.com/sample-accelerators/tanzu-java-web-app
```

Out of the Box Supply Chain with Testing and Scanning

The Out of the Box Supply Chain with Testing and Scanning adds the `source-tester`, `source-scanner`, and `image-scanner` steps in the supply chain. See Scan Types for Supply Chain Security Tools - Scan. These steps test the source code pulled by the supply chain and scans for CVEs on the source and the image built by the supply chain. For these new testing and scanning steps to work, the following additional resources must exist in the same namespace as the workload:

- `Pipeline:` defines how to run the tests on the source code pulled by the supply chain and which image to use that has the tools to run those tests.
- `ScanTemplate`: defines how to run a scan, you can change how the scan is run, either for images or source code.
    - A ScanTemplate defines the PodTemplateSpec used by a Job to run a particular scan, such as an image or source. When the supply chain initiates an ImageScan or SourceScan, they reference these templates which must be in the same namespace as the workload.
    - Grype scanner is automatically installed in all the namespaces managed by Namespace Provisioner. For more information, see About Source and Image Scans.
- `ScanPolicy` defines how to evaluate whether the artifacts scanned are compliant. For example, allowing one to be restrictive about particular vulnerabilities found.
    - When an ImageScan or a SourceScan is created to run a scan, they reference a policy, the policy name must match the following example ScanPolicy.
    - See Writing Policy Templates.

Add Java Tekton Pipelines Grype Scan Policy to your developer namespace

To create a developer namespace, see Provision Developer Namespaces.

Namespace Provisioner can automate the creation of a Tekton pipeline and a ScanPolicy that is needed for the workload to run on an Out of the Box Supply Chain with Testing and Scanning. Create an example Pipeline and a ScanPolicy in your GitOps repository and add your GitOps repository as an additional source in Namespace Provisioner configuration in `tap-values.yaml`. See Customize Installation of Namespace Provisioner.

Add the following configuration to your `tap-values.yaml` file to add the example java pipeline and grype scan policy to your developer namespace. See application-accelerator-samples in GitHub.

Using Namespace Provisioner Controller:

- Sample `tap-values.yaml` configuration:

```
namespace_provisioner:
  controller: true
  additional_sources:
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/testing-scanning-supplychain
      url: https://github.com/vmware-tanzu/application-accelerator-samples.git
```

Using GitOps:

- Example `tap-values.yaml` configuration:

```
namespace_provisioner:
  controller: false
  additional_sources:
  - git:
      ref: origin/main
```

```
      subPath: ns-provisioner-samples/testing-scanning-supplychain
      url: https://github.com/vmware-tanzu/application-accelerator-samples.git
  gitops_install:
    ref: origin/main
    subPath: ns-provisioner-samples/gitops-install
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
```

The example Pipeline resource have the following ytt logic which creates this pipeline only if

- `supply_chain` in your `tap-values.yaml` file is either `testing` or `testing_scanning`
- `profile` in your `tap-values.yaml` file is either `full, iterate`, or `build`.

```
#@ load("@ytt:data", "data")
#@ def in_list(key, list):
#@  return hasattr(data.values.tap_values, key) and (data.values.tap_values[key] in li
st)
#@ end
#@ if/end in_list('supply_chain', ['testing', 'testing_scanning']) and in_list('profil
e', ['full', 'iterate', 'build']):
```

The example ScanPolicy resource have the following ytt logic which creates this pipeline only if

- `supply_chain` in your `tap-values.yaml` file is `testing_scanning`
- `profile` in your `tap-values.yaml` file is either `full` or `build`.

After adding the additional source to your `tap-values.yaml` file, you can see the `tekton-pipeline-java and scan-policy` created in your developer namespace. To verify that the pipeline is created correctly:

```
kubectl get pipeline.tekton.dev,scanpolicies -n DEVELOPER-NAMESPACE
```

Where `DEVELOPER-NAMESPACE` is the name of the new developer namespace you want to use.

Run the following Tanzu CLI command to create a workload in your developer namespace:

Using the Tanzu CLI:

- Create workload using Tanzu apps CLI command.

  ```
  tanzu apps workload apply tanzu-java-web-app \
  --git-repo https://github.com/sample-accelerators/tanzu-java-web-app \
  --git-branch main \
  --type web \
  --app tanzu-java-web-app \
  --label apps.tanzu.vmware.com/has-tests="true" \
  --namespace DEVELOPER-NAMESPACE \
  --tail \
  --yes
  ```

Using a workload YAML:

- Create a workload.yaml file:

  ```
  apiVersion: carto.run/v1alpha1
  kind: Workload
  metadata:
    labels:
      app.kubernetes.io/part-of: tanzu-java-web-app
      apps.tanzu.vmware.com/has-tests: "true"
      apps.tanzu.vmware.com/workload-type: web
    name: tanzu-java-web-app
    namespace: DEVELOPER-NAMESPACE
  spec:
    source:
  ```

```
    git:
      ref:
        branch: main
      url: https://github.com/sample-accelerators/tanzu-java-web-app
```

Namespace Provisioner use cases and examples

Review the following Namespace Provisioner uses cases:

Use multiple tekton pipelines and Scan policies in the same namespace in Namespace Provisioner

Add Tekton pipelines and scan policies using namespace parameters in Namespace Provisioner

Working with private Git repositories in Namespace Provisioner

Customize default resources in Namespace Provisioner

Install multiple scanners in the developer namespace in Namespace Provisioner

Apply ScanTemplate overlays in air-gapped environments in Namespace Provisioner

Use multiple Tekton pipelines and scan policies in the same namespace in Namespace Provisioner

This topic tells you how to use Namespace Provisioner to configure developer namespaces to include multiple Tekton pipelines and ScanPolices in Tanzu Application Platform (commonly known as TAP).

For information about, how to create a developer namespace, see Provision Developer Namespaces.

This sample GitOps location has a Java, Python and a Golang testing pipeline as well as a Strict and a Lax grype ScanPolicy.

Using Namespace Provisioner Controller

Add the following configuration to your `tap-values.yaml` file to add multiple tekton pipelines and scan policies to your developer namespace:

```
namespace_provisioner:
  controller: true
  additional_sources:
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/testing-scanning-supplychain-polyglot
      url: https://github.com/vmware-tanzu/application-accelerator-samples.git
```

Using GitOps

Add the following configuration to your `tap-values.yaml` file to add multiple tekton pipelines and scan policies to your developer namespace:

```
namespace_provisioner:
  controller: false
  additional_sources:
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/testing-scanning-supplychain-polyglot
      url: https://github.com/vmware-tanzu/application-accelerator-samples.git
  gitops_install:
    ref: origin/main
```

```
    subPath: ns-provisioner-samples/gitops-install
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
```

The sample Pipeline resource have the following ytt logic which creates this pipeline only if

- `supply_chain` in your `tap-values.yaml` file is either `testing` or `testing_scanning`
- `profile` in your `tap-values.yaml` file is either `full, iterate` or `build`.

```
#@ load("@ytt:data", "data")
#@ def in_list(key, list):
#@  return hasattr(data.values.tap_values, key) and (data.values.tap_values[key] in li
st)
#@ end
#@ if/end in_list('supply_chain', ['testing', 'testing_scanning']) and in_list('profil
e', ['full', 'iterate', 'build']):
```

All pipelines have an additional label `apps.tanzu.vmware.com/language` to differentiate between them.

The sample ScanPolicy resource have the following ytt logic which creates this pipeline only if

- `supply_chain` in your `tap-values.yaml` file is `testing_scanning`
- `profile` in your `tap-values.yaml` file is either `full` or `build`.

The strict ScanPolicy does not allow any workloads that have Critical and High vulnerabilities to pass through the supply chain whereas the lax ScanPolicy allows the workloads to pass regardless of CVEs detected. The allowed severity level is configured using the `notAllowedSeverities := []` part of the rego file section of ScanPolicy.

Caution

The lax ScanPolicy is just added for tutorial purposes but it is not advised to use such a policy in Production workloads.

After adding the additional source to your `tap-values.yaml` file, you should be able to see the `tekton-pipeline-java, tekton-pipeline-golang, tekton-pipeline-python`, `scan-policy` and `lax-scan-policy` created in your developer namespace. Run the following command to see if the pipelines are created correctly.

```
kubectl get pipeline.tekton.dev,scanpolicies -n YOUR-NEW-DEVELOPER-NAMESPACE
```

Run the following Tanzu CLI command to create a workload in your developer namespace:

Using Tanzu CLI

Create workload using tanzu apps CLI command

```
tanzu apps workload apply tanzu-java-web-app \
--git-repo https://github.com/sample-accelerators/tanzu-java-web-app \
--git-branch main \
--type web \
--app tanzu-java-web-app \
--label apps.tanzu.vmware.com/has-tests="true" \
--param-yaml testing_pipeline_matching_labels='{"apps.tanzu.vmware.com/language": "j
ava"}' \
--param scanning_source_policy="lax-scan-policy" \
--param scanning_image_policy="lax-scan-policy" \
--namespace YOUR-NEW-DEVELOPER-NAMESPACE \
```

```
--tail \
--yes
```

Using workload yaml

Create a workload.yaml file with the details as below.

```
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  generation: 1
  labels:
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/has-tests: "true"
    apps.tanzu.vmware.com/workload-type: web
  name: tanzu-java-web-app
  namespace: YOUR-NEW-DEVELOPER-NAMESPACE
spec:
  params:
  - name: scanning_source_policy
    value: lax-scan-policy
  - name: scanning_image_policy
    value: lax-scan-policy
  - name: testing_pipeline_matching_labels
    value:
      apps.tanzu.vmware.com/language: java
  source:
    git:
      ref:
        branch: main
      url: https://github.com/sample-accelerators/tanzu-java-web-app
```

Note

`--param-yaml testing_pipeline_matching_labels` tells the supply chain to use the selector that matches the Java pipeline. To use the Python or Golang pipelines, use the selector that matches the language label in those resources.`--param scanning_source_policy="lax-scan-policy"` tells the supply chain to use the lax ScanPolicy for the workload.

Add Tekton pipelines and scan policies using namespace parameters in Namespace Provisioner

This topic tells you how to use Namespace Provisioner to parameterize your additional resources and pass those parameters to namespaces in Tanzu Application Platform (commonly known as TAP).

Instead of creating all the pipelines in all provisioned namespaces, create a Tekton pipeline and ScanPolicy that is bespoke to namespaces that are running workloads using a specific language stack.

For information about, how to create a developer namespace, see Provision Developer Namespaces.

This use case looks at the pipelines and ScanPolicies in this sample GitOps location.

Using Namespace Provisioner controller

Use controller to pass the parameters to a namespace via labels and annotations on the namespace. To enable this, set the `parameter_prefixes` in `tap-values.yaml`. The controller looks for labels and annotations starting with that prefix to populate parameters for a given namespace. For more information, see Customize the label and annotation prefixes that controller watches.

Add the following configuration to your `tap-values.yaml` file to add parameterized Tekton pipelines and scan policies to your developer namespace:

```
namespace_provisioner:
  controller: true
  additional_sources:
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/testing-scanning-supplychain-parameterized
      url: https://github.com/vmware-tanzu/application-accelerator-samples.git
  parameter_prefixes:
  - tap.tanzu.vmware.com
```

Note

This example adds `tap.tanzu.vmware.com` as a parameter_prefixes in Namespace Provisioner configuration. This tells the Namespace Provisioner controller to look for the annotations and labels on a provisioned namespace that start with the prefix `tap.tanzu.vmware.com` and use those as parameters.

The sample pipelines have the following ytt logic which creates this pipeline only if

- `supply_chain` in your `tap-values.yaml` file is either `testing` or `testing_scanning`
- `profile` in your `tap-values.yaml` file is either`full, iterate` or `build. pipeline` parameter that matches the language for which the pipeline is for.

```
#@ load("@ytt:data", "data")
#@ def in_list(key, list):
#@  return hasattr(data.values.tap_values, key) and (data.values.tap_values[key] in
list)
#@ end
#@ if/end in_list('supply_chain', ['testing', 'testing_scanning']) and in_list('prof
ile', ['full', 'iterate', 'build']) and hasattr(data.values, 'pipeline') and data.va
lues.pipeline == 'java':
```

The sample ScanPolicy resource have the following ytt logic which creates this pipeline only if

- `supply_chain` in your `tap-values.yaml` file is `testing_scanning`
- `profile` in your `tap-values.yaml` file is either `full` or `build`.
- `scanpolicy`parameter matches either `strict` or `lax`

```
#@ load("@ytt:data", "data")
#@ def in_list(key, list):
#@  return hasattr(data.values.tap_values, key) and (data.values.tap_values[key] in
list)
#@ end
#@ if/end in_list('supply_chain', ['testing_scanning']) and in_list('profile', ['ful
l', 'build']) and hasattr(data.values, 'scanpolicy') and data.values.scanpolicy ==
'lax':
```

Label your developer namespace using the `parameter_prefixes` with the parameter to be used in the `additional_sources` as follows:

```
kubectl label namespaces YOUR-NEW-DEVELOPER-NAMESPACE tap.tanzu.vmware.com/scanpolic
y=lax
```

```
kubectl label namespaces YOUR-NEW-DEVELOPER-NAMESPACE tap.tanzu.vmware.com/pipeline=
java
```

Using GitOps

Pass the parameters to a namespace by adding them to the `data.values` file located in the GitOps repository. Use this sample file as an example.

Add the following configuration to your `tap-values.yaml` file to add parameterized Tekton pipelines and scan policies to your developer namespace:

```
namespace_provisioner:
  controller: false
  additional_sources:
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/testing-scanning-supplychain-parameterized
      url: https://github.com/vmware-tanzu/application-accelerator-samples.git
  gitops_install:
    ref: origin/main
    subPath: ns-provisioner-samples/gitops-install-with-params
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
```

`gitops_install` uses this sample GitOps location to create the namespaces and manage the desired namespaces from GitOps. For more information, see GitOps section of Customize Installation of Namespace Provisioner.

Sample of `gitops_install` files:

```
#@data/values
---
namespaces:
- name: dev
  scanpolicy: lax
  pipeline: java
- name: qa
  scanpolicy: strict
  pipeline: java
```

```
#@ load("@ytt:data", "data")
#! This loop will now loop over the namespace list in
#! in ns.yaml and will create those namespaces.
#@ for ns in data.values.namespaces:
---
apiVersion: v1
kind: Namespace
metadata:
  name: #@ ns.name
#@ end
```

The sample pipelines have the following ytt logic which creates this pipeline only if the following conditions are met:

- `supply_chain` in your `tap-values.yaml` file is either `testing` or `testing_scanning`
- `profile` in your `tap-values.yaml` file is either`full, iterate` or `build`.
- `pipeline` parameter that matches the language for which the pipeline is for.

```
#@ load("@ytt:data", "data")
#@ def in_list(key, list):
#@  return hasattr(data.values.tap_values, key) and (data.values.tap_values[key] in
list)
#@ end
#@ if/end in_list('supply_chain', ['testing', 'testing_scanning']) and in_list('prof
ile', ['full', 'iterate', 'build']) and hasattr(data.values, 'pipeline') and data.va
lues.pipeline == 'java':
```

The sample ScanPolicy resource have the following ytt logic which creates this pipeline only if the following conditions are me:

- `supply_chain` in your `tap-values.yaml` file is `testing_scanning`
- `profile` in your `tap-values.yaml` file is either `full` or `build`.
- `scanpolicy`parameter matches either `strict` or `lax`

```
#@ load("@ytt:data", "data")
#@ def in_list(key, list):
#@  return hasattr(data.values.tap_values, key) and (data.values.tap_values[key] in
list)
#@ end
#@ if/end in_list('supply_chain', ['testing_scanning']) and in_list('profile', ['ful
l', 'build']) and hasattr(data.values, 'scanpolicy') and data.values.scanpolicy ==
'lax':
```

Run the following Tanzu CLI command to create a workload in your developer namespace:

Using Tanzu CLI

Create a workload using Tanzu Apps CLI plug-in command

```
tanzu apps workload apply tanzu-java-web-app \
--git-repo https://github.com/sample-accelerators/tanzu-java-web-app \
--git-branch main \
--type web \
--app tanzu-java-web-app \
--label apps.tanzu.vmware.com/has-tests="true" \
--namespace YOUR-NEW-DEVELOPER-NAMESPACE \
--tail \
--yes
```

Using workload yaml

Create a workload.yaml file with the details as below.

```
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  labels:
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/has-tests: "true"
    apps.tanzu.vmware.com/workload-type: web
  name: tanzu-java-web-app
  namespace: YOUR-NEW-DEVELOPER-NAMESPACE
spec:
  source:
    git:
      ref:
        branch: main
      url: https://github.com/sample-accelerators/tanzu-java-web-app
```

Run the following command to verify the resources have been created in the namespace:

```
kubectl get secrets,serviceaccount,rolebinding,pods,workload,configmap,limitrange,pipeline,scanpolicies -n YOUR-NEW-DEVELOPER-NAMESPACE
```

Work with private Git repositories in Namespace Provisioner

This topic tells you how to configure Namespace Provisioner to use private Git repositories for storing GitOps based installation files, and platform operator templated resources that you want to create in your developer namespace in Tanzu Application Platform (commonly known as TAP).

Git Authentication for using a private Git repository

Authentication is provided using a secret in the `tap-namespace-provisioning` namespace, or an existing secret in another namespace referred to in the secretRef in the `additional_sources`. For more details, see Customize Installation of Namespace Provisioner.

Create the Git Authentication secret in tap-namespace-provisioning namespace

The secrets for Git authentication allow the following keys: ssh-privatekey, ssh-knownhosts, username, and password. If ssh-knownhosts is not specified, Git does not perform strict host checking.

Important

Namespace Provisioner relies on kapp-controller for any tasks involving communication with external services, such as registries or Git repositories. When operating in air-gapped environments or other scenarios where external services are secured by a Custom CA certificate, you must configure kapp-controller with the CA certificate data to prevent X.509 certificate errors. For more information, see Deploy onto Cluster in the Cluster Essentials for VMware Tanzu documentation.

1. Create the Git secret:

   Using HTTP(s) based Authentication
   If you are using Username and Password for authentication:

   ```
   cat << EOF | kubectl apply -f -
   apiVersion: v1
   kind: Secret
   metadata:
     name: git-auth
     namespace: tap-namespace-provisioning
   type: Opaque
   stringData:
     username: GIT-USERNAME
     password: GIT-PASSWORD
   EOF
   ```

   Using SSH based Authentication
   If you are using SSH private key for authentication:

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: git-auth
  namespace: tap-namespace-provisioning
type: Opaque
stringData:
  ssh-privatekey: |
      -----BEGIN OPENSSH PRIVATE KEY-----
      ..
      -----END OPENSSH PRIVATE KEY-----
EOF
```

2. Add the `secretRef` section to the `additional_sources` and the `gitops_install` section of your `tap-values.yaml` file:

Using Namespace Provisioner Controller

Description

```
namespace_provisioner:
  controller: true
  additional_sources:
  - git:
      ref: origin/main
      subPath: sources
      # This example URL is for SSH auth. Use https:// path if using HTTPS au
th
      url: git@github.com:private-repo-org/repo.git
      secretRef:
        name: git-auth
```

Using GitOps

Description

In this example, the location where the list of namespaces resides is also a private repository. So you must create a secret named `git-auth-install` with the same authentication details.

```
namespace_provisioner:
  controller: false
  additional_sources:
  - git:
      ref: origin/main
      subPath: tekton-pipelines
      # This example URL is for SSH auth. Use https:// path if using HTTPS au
th
      url: git@github.com:private-repo-org/repo.git
      secretRef:
        name: git-auth
  gitops_install:
    ref: origin/main
    subPath: gitops-install
    # This example URL is for SSH auth. Use https:// path if using HTTPS auth
    url: git@github.com:private-repo-org/repo.git
    secretRef:
      name: git-auth-install
```

Import from another namespace

If you already have a Git secret created in a namespace other than `tap-namespace-provisioning` namespace and you want to refer to that, the `secretRef` section should have the namespace mentioned with the `create_export` flag. The default value for `create_export` is false as it assumes the Secret is already exported for `tap-namespace-provisioning` namespace, but allows you to specify if you want the Namespace Provisioner to create a `Carvel SecretExport` for that secret.

The example refers to `git-auth` secret from `tap-install` in the secretRef section.

Using Namespace Provisioner Controller

Description

```
namespace_provisioner:
  controller: true
  additional_sources:
  - git:
      ref: origin/main
      subPath: sources
      #! This example URL is for SSH auth. Use https:// path if using HTTPS auth
      url: git@github.com:private-repo-org/repo.git
      secretRef:
          name: git-auth
          namespace: tap-install
          #! If this secret is already exported for this namespace, you can ignore t
he create_export key as it defaults to false
          create_export: true
```

Using GitOps

Description

```
namespace_provisioner:
  controller: false
  additional_sources:
  - git:
      ref: origin/main
      subPath: tekton-pipelines
      #! This example URL is for SSH auth. Use https:// path if using HTTPS auth
      url: git@github.com:private-repo-org/repo.git
      secretRef:
          name: git-auth
          namespace: tap-install
          #! If this secret is already exported for this namespace, you can ignore t
he create_export key as it defaults to false
          create_export: true
  gitops_install:
    ref: origin/main
    subPath: gitops-install
    #! This example URL is for SSH auth. Use https:// path if using HTTPS auth
    url: git@github.com:private-repo-org/repo.git
    secretRef:
      name: git-auth-install
      namespace: tap-install
      #! If this secret is already exported for this namespace, you can ignore the c
reate_export key as it defaults to false
      create_export: true
```

After reconciling, Namespace Provisioner creates:

- SecretExport for the secret in the provided namespace (tap-install in the above example) to the Namespace Provisioner namespace.

- SecretImport for the secret in Namespace Provisioning namespace (tap-namespace-provisioning) so Carvel secretgen-controller can create the required secret for the Namespace Provisioner to connect to the private Git repository.

Git Authentication for Private Repository for Workloads and Supply chain

To either fetch or push source code from or to a repository that requires credentials, you must provide those through a Kubernetes secret object referenced by the intended Kubernetes object created for performing the action. The following sections provide details about how to appropriately set up Kubernetes secrets for carrying those credentials forward to the proper resources.

This section provides instructions on how to configure the `default` service account to work with private Git repositories for workloads and supply chain using Namespace Provisioner.

To configure the service account to work with private Git repositories, follow the steps below:

1. Create a secret in the `tap-install` namespace or any namespace of your preference, that contains the Git credentials in YAML format.
    - `host`, `username`, and `password`, or `personal access token` values for HTTP based Git Authentication.
    - `ssh-privatekey`, `identity`, `identity_pub`, and `known_hosts` for SSH based Git Authentication.

Note

The `stringData` key of the secret must have **.yaml** or **.yml** suffix at the end.

Using HTTP(s) based Authentication

If using Username and Password for authentication.

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: workload-git-auth
  namespace: tap-install
type: Opaque
stringData:
  content.yaml: |
    git:
      #! For HTTP Auth. Recommend using https:// for the git server.
      host: GIT-SERVER
      username: GIT-USERNAME
      password: GIT-PASSWORD
EOF
```

Using SSH based Authentication

If you are using SSH private key for authentication, create the Git secret with authentication details as follows:

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: workload-git-auth
  namespace: tap-install
```

```
type: Opaque
stringData:
  content.yaml: |
    git:
      host: GIT-SERVER
      #! For SSH Auth
      ssh_privatekey: SSH-PRIVATE-KEY
      identity: SSH-PRIVATE-KEY
      identity_pub: SSH-PUBLIC-KEY
      known_hosts: GIT-SERVER-PUBLIC-KEYS
EOF
```

2. To create a secret that will be added to the service account in the developer namespace within the GitOps repository, use this example or follow the example provided below.

   Instead of directly including the actual user name and password in the Git repository secret, use the `data.values.imported` keys to add references to the values from the git-auth secret created in Step 1.

   This secret represents the actual Git secret that will be created by the Namespace Provisioner in each managed namespace. It should be included in your Git repository linked in the `additional_sources` section of `tap-values.yaml` mentioned in Step 4.

   Using HTTP(s) based Authentication
   If using Username and Password for authentication.

   ```
   #@ load("@ytt:data", "data")
   ---
   apiVersion: v1
   kind: Secret
   metadata:
     name: git
     annotations:
       tekton.dev/git-0: #@ data.values.imported.git.host
   type: kubernetes.io/basic-auth
   stringData:
     username: #@ data.values.imported.git.username
     password: #@ data.values.imported.git.token
   ```

   Using SSH based Authentication
   If using SSH private key for authentication:

   ```
   #@ load("@ytt:data", "data")
   ---
   apiVersion: v1
   kind: Secret
   metadata:
     name: git
     annotations:
       tekton.dev/git-0: #@ data.values.imported.git.host
   type: kubernetes.io/ssh-auth
   stringData:
     identity: #@ data.values.imported.git.identity
     identity.pub: #@ data.values.imported.git.identity_pub
     known_hosts: #@ data.values.imported.git.known_hosts
     ssh-privatekey: #@ data.values.imported.git.ssh_privatekey
   ```

3. Combine this `tap-values.yaml`:

   Using Namespace Provisioner Controller

Add the following configuration to `tap-values.yaml`:

```
namespace_provisioner:
  controller: true
  additional_sources:
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/credentials
      url: https://github.com/vmware-tanzu/application-accelerator-samples.gi
t
  import_data_values_secrets:
  - name: workload-git-auth
    namespace: tap-install
    create_export: true
  default_parameters:
    supply_chain_service_account:
      secrets:
      - git
```

Using GitOps

Add the following configuration to `tap-values.yaml`:

```
namespace_provisioner:
  controller: false
  additional_sources:
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/credentials
      url: https://github.com/vmware-tanzu/application-accelerator-samples.gi
t
  gitops_install:
    ref: origin/main
    subPath: ns-provisioner-samples/gitops-install
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
  import_data_values_secrets:
  - name: workload-git-auth
    namespace: tap-install
    create_export: true
  default_parameters:
    supply_chain_service_account:
      secrets:
      - git
```

- First additional source points to the location where the templated Git secret resides which will be created in all developer namespaces.
- Import the newly created `workload-git-auth` secret into Namespace Provisioner to use in `data.values.imported` by adding the secret to the `import_data_values_secrets`.
- Add the secret to be added to the ServiceAccount in the `default_parameters`. For more information, see Customize service accounts.

Note

`create_export` is set to `true` in `import_data_values_secrets`, as a result, a SecretExport is created for the `workload-git-auth` secret in the `tap-install` namespace automatically by Namespace Provisioner. After the changes are reconciled, the secret named `git` is in all provisioned

namespaces and is also added to the default service account of those namespaces.

4. In the `ootb_supply_chain_*.gitops.credentials_secret` and `ootb_supply_chain_*.source.credentials_secret` section of your `tap-values.yaml` file, edit the name of the Git secret that contains the credentials. This is necessary for the supply chain to include the `secretRef` when creating the Flux `GitRepository` resource. Here is an example:

```
ootb_supply_chain_testing_scanning:
  gitops:
    credentials_secret: git  # Replace with the actual name of your GitOps Git
secret for the workload, if different.
  source:
    credentials_secret: git  # Replace with the actual name of your source Git
secret for the workload, if different.
```

By providing this configuration, the supply chain associates the created GitRepository resource with the specified Git secret managed by the Namespace Provisioner.

Customize default resources in Namespace Provisioner

This topic tells you how to deactivate Grype in Namespace Provisioner and how to configure the `default` service account to work with private Git repositories in Tanzu Application Platform (commonly known as TAP).

Deactivate Grype install

Grype is installed with Namespace Provisioner by default. If you prefer to use a different scanner for namespaces instead of Grype, you can deactivate the installation of the default Grype scanner.

Deactivate Grype for all namespaces

To deactivate the default installation of Grype for all namespaces managed by the Namespace Provisioner, set the `skip_grype` parameter to `true` in the `default_parameters` section of the `tap-values.yaml`:

```
namespace_provisioner:
  default_parameters:
    skip_grype: true
```

By enabling the `skip_grype: true` setting, the PackageInstall and the secret `grype-scanner-{namespace}` are not generated in the `tap-install` namespace for any namespaces that are managed by the Namespace Provisioner.

Deactivate Grype for a specific namespace

Using Namespace Provisioner Controller
To deactivate the installation of Grype for a specific namespace, annotate or label the namespace by setting the reserved parameter `skip_grype` to `true`. Use the default or customized `parameter_prefixes`. For more information, see Customize the label and annotation prefixes that controller watches.

```
kubectl annotate ns YOUR-NEW-DEVELOPER-NAMESPACE param.nsp.tap/skip_grype=true
```

Using GitOps

Add the parameter `skip_grype` with the value `true` in the namespaces file in the GitOps repository.

```
#@data/values
---
namespaces:
- name: dev
  skip_grype: true
- name: qa
```

Customize service accounts

This section provides instructions on how to configure the `default` service account to work with private Git repositories for workloads and supply chain using Namespace Provisioner.

To configure the service account to work with private Git repositories, follow the steps below:

1. Create a secret in the `tap-install` namespace, or any namespace that contains the Git credentials in the YAML format.
    - `host`, `username`, and `password` values for HTTP based Git Authentication.
    - `ssh-privatekey`, `identity`, `identity_pub`, and `known_hosts` for SSH based Git Authentication.

    Note

    stringData key of the secret must have **.yaml** or **.yml** suffix at the end.

    ```
    #! Example shows HTTP as well as SSH based authentication
    cat << EOF | kubectl apply -f -
    apiVersion: v1
    kind: Secret
    metadata:
      name: workload-git-auth
      namespace: tap-install
    type: Opaque
    stringData:
      content.yaml: |
        git:
          #! For HTTP Auth. Recommend using https:// for the git server.
          host: GIT-SERVER
          username: GIT-USERNAME
          token: GIT-PASSWORD
          #! For SSH Auth
          ssh_privatekey: SSH-PRIVATE-KEY
          identity: SSH-PRIVATE-KEY
          identity_pub: SSH-PUBLIC-KEY
          known_hosts: GIT-SERVER-PUBLIC-KEYS
    EOF
    ```

2. Create a scaffolding of a Git secret, this must be added to the service account in your developer namespace in your GitOps repository. A sample secret is available in the vmware-tanzu/application-accelerator-samples Git repository. Instead of putting the user name and password in the secret in your Git repository, use the `data.values.imported` keys to put the reference to the values in the git-auth secret created in step 1. For example:

```
#@ load("@ytt:data", "data")
#@ load("@ytt:base64", "base64")
---
apiVersion: v1
kind: Secret
metadata:
  name: git
  annotations:
    tekton.dev/git-0: #@ data.values.imported.git.host
type: kubernetes.io/basic-auth
stringData:
  username: #@ base64.encode(data.values.imported.git.username)
  password: #@ base64.encode(data.values.imported.git.token)
```

3. Complete the process by customizing the SupplyChain ServiceAccount for all or specific namespaces as described in the following sections.

Update ServiceAccount for all namespaces

Customize the SupplyChain ServiceAccount by adding additional `secrets` or `imagePullSecrets` for all namespaces managed by the Namespace Provisioner. Edit the `supply_chain_service_account` parameter in the `default_parameters` section of the `tap-values.yaml` file. If you have a separate Service Account for delivery purposes, configure it using the `delivery_service_account` parameter. For example:

```
namespace_provisioner:
  controller: true
  additional_sources:
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/credentials
      url: https://github.com/vmware-tanzu/application-accelerator-samples.git
  import_data_values_secrets:
  - name: workload-git-auth
    namespace: tap-install
    create_export: true
  default_parameters:
    supply_chain_service_account:
      secrets:
      - git
      imagePullSecrets: [] #! optional
    delivery_service_account: #! Not required, specify only if the Service account is
different from the Supply chain service account.
      secrets: [] #! optional
      imagePullSecrets: [] #! optional
```

- This adds the `git` secret to the Service Account mentioned in `ootb_supply_chain_*.service_account`. If not specified, it takes the `default` service account.
- `additional_sources` points to the location where the templated Git secret resides which will be created in all developer namespaces.
- Import the newly created `workload-git-auth` secret into Namespace Provisioner to use in `data.values.imported` by adding the secret to the `import_data_values_secrets`.
- Add the secret to be added to the ServiceAccount in the `default_parameters`

Update ServiceAccount for a specific namespace

To customize the SupplyChain ServiceAccount for a specific namespace managed by the Namespace Provisioner and include additional `secrets` or `imagePullSecrets`, use the

`supply_chain_service_account` parameter. This parameter allows you to edit the ServiceAccount and add any required `secrets` or `imagePullSecrets`.

If you have a separate ServiceAccount for delivery purposes, you can also configure it using the `delivery_service_account` parameter.

Using Namespace Provisioner Controller

Add the following configuration to your `tap-values.yaml` file:

```
namespace_provisioner:
  controller: true
  additional_sources:
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/credentials
      url: https://github.com/vmware-tanzu/application-accelerator-samples.git
  import_data_values_secrets:
  - name: workload-git-auth
    namespace: tap-install
    create_export: true
```

- `additional_sources` points to the location where the templated Git secret resides which will be created in all developer namespaces.
- Import the newly created `workload-git-auth` secret into Namespace Provisioner to use in `data.values.imported` by adding the secret to the `import_data_values_secrets`.

Annotate the namespace with the parameter so the ServiceAccount is updated

```
kubectl annotate ns dev param.nsp.tap/supply_chain_service_account.secrets='["git"]'
```

The `desired-namespaces` ConfigMap will look like:

```
#@data/values
---
namespaces:
- name: dev
  supply_chain_service_account:
    secrets:
    - git
```

If the ServiceAccount for delivery is different, then:

```
kubectl annotate ns dev param.nsp.tap/delivery_service_account.secrets='["git"]'
```

The `desired-namespaces` ConfigMap will look like:

```
#@data/values
---
namespaces:
- name: dev
  supply_chain_service_account:
    secrets:
    - git
  delivery_service_account:
    secrets:
    - git
```

Using GitOps

Add the following configuration to your `tap-values.yaml` file:

```
namespace_provisioner:
  controller: false
  additional_sources:
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/credentials
      url: https://github.com/vmware-tanzu/application-accelerator-samples.git
  gitops_install:
    ref: origin/main
    subPath: ns-provisioner-samples/gitops-install-params-sa
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
  import_data_values_secrets:
  - name: workload-git-auth
    namespace: tap-install
    create_export: true
```

- `additional_sources` points to the location where the templated Git secret resides which will be created in all developer namespaces.
- Configure the desired namespaces yaml in the GitOps repository with the parameter in the namespace. Check the sample file where you are adding the `git` secret to the supply chain service account.
- Import the newly created `workload-git-auth` secret into Namespace Provisioner to use in `data.values.imported` by adding the secret to the `import_data_values_secrets`.

Note

`create_export` is set to `true` in `import_data_values_secrets` meaning that a SecretExport is created for the `workload-git-auth` secret in the `tap-install` namespace automatically by Namespace Provisioner. After the changes are reconciled, the secret named **git** is in all provisioned namespaces and is also added to the default service account of those namespaces.

Customize Limit Range defaults

Namespace Provisioner creates the LimitRange resources on Run clusters in all Namespace Provisioner managed namespaces. For more information, see Default Resources.

You can opt-in to have the LimitRange resource created on Full and Iterate clusters. For more information, see Set/Update LimitRange defaults for all namespaces and Set/Update LimitRange defaults for a specific namespace.

Namespace Provisioner does not create LimitRange resource in Build and View clusters.

Default values in LimitRange resource are as follows:

```
limits:
  default:
    cpu : 1500m
    memory : 1Gi
  defaultRequest:
    cpu : 100m
    memory : 1Gi
```

Set or Update LimitRange defaults for all namespaces

To update the values in LimitRange for all Namespace Provisioner managed namespaces, specify the `default_parameters` configuration in `tap-values.yaml` as follows:

```
namespace_provisioner:
  default_parameters:
    # overwrite default limits set by the OOTB LimitRange (in Run Cluster) for all nam
espaces
    # set default limits for Full and Iterate Cluster in all namespaces
    limits:
      default:
        cpu: 1000m
        memory: 1Gi
      defaultRequest:
        cpu: 200m
        memory: 500Mi
```

Set or Update LimitRange defaults for a specific namespace

Override the LimitRange for specific namespaces as follows:

Using Namespace Provisioner Controller

Annotate or label a namespace using the default *parameter_prefix* `param.nsp.tap/` followed by the YAML path to CPU or memory limits as follows:

```
kubectl annotate ns YOUR-NEW-DEVELOPER-NAMESPACE param.nsp.tap/limits.default.cpu=11
00m

kubectl annotate ns YOUR-NEW-DEVELOPER-NAMESPACE param.nsp.tap/limits.default.memory
=2Gi

kubectl annotate ns YOUR-NEW-DEVELOPER-NAMESPACE param.nsp.tap/limits.defaultReques
t.cpu=1500m

kubectl annotate ns YOUR-NEW-DEVELOPER-NAMESPACE param.nsp.tap/limits.defaultReques
t.memory=1Gi
```

- The controller detects the annotations and labels with the `param.nsp.tap/` prefix, and adds the keys and values in the desired-namespace ConfigMaps as parameters for that namespace.
- If you want the controller to search for a custom prefix, instead of the default `param.nsp.tap`, prefix, use the `parameter_prefixes` configuration option in the tap-values.yaml file. For more information, see Customize the label and annotation prefixes that controller watches.

Note

Labels take precedence over annotations if the same key is provided in both.

Using GitOps

Add the following configuration to your `tap-values.yaml` file to add parameterized limits to your developer namespace:

```
namespace_provisioner:
  controller: false
  gitops_install:
    ref: origin/main
    subPath: ns-provisioner-samples/gitops-install-with-params
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
```

This adds `gitops_install` with this sample GitOps location to create the namespaces and manage the desired namespaces from GitOps. For more information, see the GitOps tab in Customize Installation of Namespace Provisioner.

Sample of `gitops_install` files:

```
#@data/values
---
namespaces:
- name: dev
  limits:
    default:
      cpu: 1200m
      memory: 1.5Gi
    defaultRequest:
      cpu: 300m
      memory: 30Mi
- name: qa
```

```
#@ load("@ytt:data", "data")
#! This loop will now loop over the namespace list in
#! in ns.yaml and will create those namespaces.
#@ for ns in data.values.namespaces:
---
apiVersion: v1
kind: Namespace
metadata:
  name: #@ ns.name
#@ end
```

The Namespace Provisioner creates a LimitRange with default values for `qa` namespace and with the given values for `dev` namespace.

Deactivate LimitRange Setup

The Namespace Provisioner generates a Kubernetes LimitRange object as a default resource in the namespaces it manages within the Run profile clusters. Additionally, the Namespace Provisioner offers the capability for Platform operators to enable LimitRange object stamping in Full and Iterate profile clusters using namespace parameters. The following options are available to deactivate the installation of the default LimitRange object:

Deactivate for all namespaces

To exclude the installation of the default LimitRange, set the `skip_limit_range` parameter to `true` in the `default_parameters` section of the `tap-values.yaml` file as shown here:

```
namespace_provisioner:
  default_parameters:
    skip_limit_range: true
```

Deactivate for a specific namespace

Using Namespace Provisioner Controller
To deactivate the LimitRange for a specific developer namespace, annotate or label the namespace using the parameter `skip_grype` and set its value to `true`. Use the default or customized `parameter_prefixes`, for more information, as explained in the Customize the label and annotation prefixes that controller watches section.

```
kubectl annotate ns YOUR-NEW-DEVELOPER-NAMESPACE param.nsp.tap/skip_limit_range=true
```

Using GitOps

Add the parameter `skip_limit_range` with the value `true` in the namespaces file in the GitOps repository as shown below:

```
#@data/values
---
namespaces:
- name: dev
  skip_limit_range: true
  limits:
    default:
      cpu: 1200m
      memory: 1.5Gi
    defaultRequest:
      cpu: 300m
      memory: 30Mi
- name: qa
```

Install multiple scanners in the developer namespace in Namespace Provisioner

This topic tells you how to use Namespace Provisioner to automate multiple scanner installations in the developer namespace in Tanzu Application Platform (commonly known as TAP).

Grype scanner is installed by default in all namespaces managed by Namespace Provisioner.

The following steps describe how to install Snyk scanner and Grype in the developer namespace and use both together in the supply chain. Grype is used for Source scans and Snyk is used for Image scans.

For information about, how to create a developer namespace, see Provision Developer Namespaces.

1. Create a secret in the `tap-install` namespace or any namespace of your preference that contains the Snyk token in YAML format. It must have .yaml or .yml in the key as shown here:

   ```
   cat << EOF | kubectl apply -f -
   apiVersion: v1
   kind: Secret
   metadata:
     name: scanner-auth
     namespace: tap-install
   type: Opaque
   stringData:
     content.yaml: |
       scanners:
         snyk_api_token: "" # Paste your snyk API token here
   EOF
   ```

2. Add the following configuration to your `tap-values.yaml` file to create the supply-chain and scanners:

Using Namespace Provisioner Controller

Add the following configuration to your `tap-values.yaml` file:

```
namespace_provisioner:
  controller: true
  additional_sources:
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/testing-scanning-supplychain-multiple-scanners
      url: https://github.com/vmware-tanzu/application-accelerator-samples.git
  import_data_values_secrets:
  - name: scanner-auth
    namespace: tap-install
    create_export: true
```

Using GitOps

Add the following configuration to your `tap-values.yaml` file:

```
namespace_provisioner:
  controller: false
  additional_sources:
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/testing-scanning-supplychain-multiple-scanners
      url: https://github.com/vmware-tanzu/application-accelerator-samples.git
  import_data_values_secrets:
  - name: scanner-auth
    namespace: tap-install
    create_export: true
  gitops_install:
    ref: origin/main
    subPath: ns-provisioner-samples/gitops-install
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
```

Additional source points to the location of the sample GitOps repo which has the following custom resources:

- `tekton-pipeline-java.yaml`: Use this to create a Tekton pipeline for running tests on the Java workload.
- `scanpolicy-grype.yaml and scanpolicy-snyk.yaml`: Use to create Scan policies for Grype and Snyk scanners.
- `snyk-token-secret.yaml`: This is a Snyk token secret that must be created in the developer namespace. Instead of putting the actual Snyk token in the secret in the Git repository, put the reference to the values in the scanner-auth secret created in Step 1 by using the `data.values.imported` keys.
- `snyk-scanner-install.yaml`: This contains the PackageInstall for installing the Snyk package for the developer namespace. The namespace `tap-install` is mentioned in the PackageInstall resource. This causes Namespace Provisioner to create a PackageInstall resource for all provisioned namespaces in the same namespace and add `-{namespace}` as the suffix in the name to avoid name collisions.

Run the following command to apply a workload in your developer namespace that uses Grype for source scan and Snyk for Image scan:

Using Tanzu CLI

Create workload using tanzu apps CLI command

```
tanzu apps workload apply tanzu-java-web-app \
--git-repo https://github.com/sample-accelerators/tanzu-java-web-app \
--git-branch main \
--type web \
```

```
--app tanzu-java-web-app \
--label apps.tanzu.vmware.com/has-tests="true" \
--param scanning_image_policy=snyk-scan-policy \
--param scanning_image_template=snyk-private-image-scan-template \
--namespace YOUR-NEW-DEVELOPER-NAMESPACE \
--tail \
--yes
```

Using workload yaml

Create a `workload.yaml` file:

```
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  labels:
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/has-tests: "true"
    apps.tanzu.vmware.com/workload-type: web
  name: tanzu-java-web-app
  namespace: YOUR-NEW-DEVELOPER-NAMESPACE
spec:
  params:
  - name: scanning_image_policy
    value: snyk-scan-policy
  - name: scanning_image_template
    value: snyk-private-image-scan-template
  source:
    git:
      ref:
        branch: main
      url: https://github.com/sample-accelerators/tanzu-java-web-app
```

Apply ScanTemplate overlays in air-gapped environments in Namespace Provisioner

This topic tells you how to use Namespace Provisioner to customize the ScanTemplates created by the `grype-scanner` PackageInstall in Namespace Provisioner. Use annotations to apply an overlay to the ScanTemplates in Tanzu Application Platform (commonly known as TAP).

Namespace Provisioner includes a pre-configured `grype-scanner` PackageInstall for each developer namespace. For more information about default resources, see Default resources.

If you require customization of the ScanTemplate created by the PackageInstall, you must apply overlays to the ScanTemplate through package customization as Namespace Provisioner does not directly create the ScanTemplate. For more information on how to customize a package installation, see Customize a package.

For information about potential customizations of the `grype-scanner` and troubleshooting tips, see Use vulnerability scanning in offline and air-gapped environments.

1. To enable updates to the ScanTemplates, create an overlay specifically designed for this purpose. When the package is processed, the overlay is applied to the ScanTemplate. It is done by the reference to this overlay in the annotation `ext.packaging.carvel.dev/ytt-paths-from-secret-name`:

   ```
   cat << EOF | kubectl apply -f -
   apiVersion: v1
   kind: Secret
   metadata:
   ```

```
  name: grype-airgap-override-stale-db-overlay
  namespace: tap-install #! namespace where tap is installed
stringData:
  patch.yaml: |
    #@ load("@ytt:overlay", "overlay")

    #@overlay/match by=overlay.subset({"kind":"ScanTemplate"}),expects="1+"
    ---
    spec:
      template:
        initContainers:
          #@overlay/match by=overlay.subset({"name": "scan-plugin"}), expects
="1+"
          - name: scan-plugin
            #@overlay/match missing_ok=True
            env:
              #@overlay/append
              - name: GRYPE_DB_MAX_ALLOWED_BUILT_AGE #! see note on best practi
ces
                value: "240h"
EOF
```

2. To enhance the functionality of the `grype-scanner` PackageInstall created by the Namespace Provisioner, create an overlay that adds the `ext.packaging.carvel.dev/ytt-paths-from-secret-name` annotation. This annotation enables the PackageInstall to retrieve information from the created secret with the overlay and apply it to the ScanTemplate.

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: grype-airgap-override-stale-db-overlay-for-nsp
  namespace: tap-install # or any other namespaces from where nsp will import t
he secret
stringData:
  patch-grype-install-in-nsp.yaml: |
    #@ load("@ytt:overlay", "overlay")
    #@ def matchGrypeScanners(index, left, right):
      #@ if left["apiVersion"] != "packaging.carvel.dev/v1alpha1" or left["kin
d"] != "PackageInstall":
        #@ return False
      #@ end
      #@ return left["metadata"]["name"].startswith("grype-scanner")
    #@ end
    #@overlay/match by=matchGrypeScanners, expects="0+"
    ---
    metadata:
      annotations:
        #@overlay/match missing_ok=True
        ext.packaging.carvel.dev/ytt-paths-from-secret-name.0: grype-airgap-ove
rride-stale-db-overlay
        #! The value of the above annotation is the name of the secret that con
tains the grype overlay
EOF
```

3. Update the `tap-values.yaml` file as follows so the overlay is applied to the PackageInstall. For more information, see Import overlay secrets.

```
namespace_provisioner:
  overlay_secrets:
  - create_export: true
    name: grype-airgap-override-stale-db-overlay-for-nsp
```

```
    namespace: tap-install # or any other namespaces from where nsp will import
the secret
```

Work with Git repositories in air-gapped environments with Namespace Provisioner

This topic provides instructions for configuring Namespace Provisioner to use air-gapped Git repositories. This allows you to store GitOps-based installation files and platform operator-templated resources intended for creation in your developer namespace in Tanzu Application Platform (TAP).

Git authentication

Authentication is established through a secret in the `tap-namespace-provisioning` namespace or an existing secret in another namespace referenced in the `secretRef` in `additional_sources`. For more details, refer to Customize Installation of Namespace Provisioner.

Create the Git authentication secret in `tap-namespace-provisioning` namespace

The Git authentication secrets support the following keys: `ssh-privatekey`, `ssh-knownhosts`, `username`, and `password`. If `ssh-knownhosts` is not specified, Git does not perform strict host checking.

Important

In air-gapped environments or other scenarios where external services are secured by a Custom CA certificate, configure kapp-controller with the CA certificate data to prevent X.509 certificate errors. For more information, see Deploy onto Cluster in the Cluster Essentials for VMware Tanzu documentation.

1. Create the Git secret:

   Using HTTP(s) based authentication
   If you are using user name and password for authentication:

   ```
   cat << EOF | kubectl apply -f -
   apiVersion: v1
   kind: Secret
   metadata:
     name: git-auth
     namespace: tap-namespace-provisioning
   type: Opaque
   stringData:
     username: GIT-USERNAME
     password: GIT-PASSWORD
   EOF
   ```

   Using SSH based authentication
   If you are using SSH private key for authentication:

   ```
   cat << EOF | kubectl apply -f -
   apiVersion: v1
   kind: Secret
   metadata:
     name: git-auth
   ```

```
  namespace: tap-namespace-provisioning
type: Opaque
stringData:
  ssh-privatekey: |
      -----BEGIN OPENSSH PRIVATE KEY-----
      ...
      -----END OPENSSH PRIVATE KEY-----
EOF
```

2. Add the `secretRef` section to the `additional_sources` and the `gitops_install` section of your `tap-values.yaml` file:

Using Namespace Provisioner Controller
Description

```
namespace_provisioner:
  controller: true
  additional_sources:
  - git:
      ref: origin/main
      subPath: sources
      # This example URL is for SSH auth. Use https:// path if using HTTPS au
th
      url: git@git-airgap-server:private-repo-org/repo.git
      secretRef:
        name: git-auth
```

Using GitOps
Description

Caution In kapp-controller v0.46.0 and earlier, there is a limitation that prevents the reuse of the same Git secret multiple times. If you have multiple additional sources using repositories with identical credentials, you must create distinct secrets, each with the same authentication details.

In this example, the list of namespaces resides in a repository. Therefore, you must create a secret named `git-auth-install` with the same authentication details for this location.

```
namespace_provisioner:
  controller: false
  additional_sources:
  - git:
      ref: origin/main
      subPath: tekton-pipelines
      # This example URL is for SSH auth. Use https:// path if using HTTPS au
th
      url: git@git-airgap-server:private-repo-org/repo.git
      secretRef:
        name: git-auth
  gitops_install:
    ref: origin/main
    subPath: gitops-install
    # This example URL is for SSH auth. Use https:// path if using HTTPS auth
    url: git@git-airgap-server:private-repo-org/repo.git
    secretRef:
      name: git-auth-install
```

Import from another namespace

If you already have a Git secret created in a namespace other than the `tap-namespace-provisioning` namespace and you want to refer to it, the `secretRef` section must include the namespace and the `create_export` flag. The default value for `create_export` is `false`, assuming the secret is already exported for the `tap-namespace-provisioning` namespace. However, you can specify if you want Namespace Provisioner to create a Carvel `SecretExport` for that secret.

In this example, the `secretRef` section refers to the `git-auth` secret from the `tap-install` namespace.

Using Namespace Provisioner Controller

Description

```
namespace_provisioner:
  controller: true
  additional_sources:
  - git:
      ref: origin/main
      subPath: sources
      #! This example URL is for SSH auth. Use https:// path if using HTTPS auth
      url: git@git-airgap-server:private-repo-org/repo.git
      secretRef:
          name: git-auth
          namespace: tap-install
          #! If this secret is already exported for this namespace, you can ignore t
he create_export key as it defaults to false
          create_export: true
```

Using GitOps

Description

```
namespace_provisioner:
  controller: false
  additional_sources:
  - git:
      ref: origin/main
      subPath: tekton-pipelines
      #! This example URL is for SSH auth. Use https:// path if using HTTPS auth
      url: git@git-airgap-server:private-repo-org/repo.git
      secretRef:
          name: git-auth
          namespace: tap-install
          #! If this secret is already exported for this namespace, you can ignore t
he create_export key as it defaults to false
          create_export: true
  gitops_install:
    ref: origin/main
    subPath: gitops-install
    #! This example URL is for SSH auth. Use https:// path if using HTTPS auth
    url: git@git-airgap-server:private-repo-org/repo.git
    secretRef:
      name: git-auth-install
      namespace: tap-install
      #! If this secret is already exported for this namespace, you can ignore the c
reate_export key as it defaults to false
      create_export: true
```

After reconciliation, Namespace Provisioner creates:

- SecretExport for the secret in the provided namespace, exporting it to the Namespace Provisioner namespace, for example, `tap-install`
- SecretImport for the secret in the `tap-namespace-provisioning` namespace. This enables Carvel secretgen-controller to create the required secret, allowing Namespace Provisioner to connect to the Git repository.

Git authentication for workloads and supply chain

When fetching or pushing source code to a repository that requires credentials, it's essential to provide those credentials through a Kubernetes secret object referenced by the corresponding Kubernetes object created for the action. The following sections describe setting up Kubernetes secrets to securely pass these credentials to the relevant resources. This procedure provides the steps to configure the `default` service account to interact with Git repositories for workloads and supply chain using Namespace Provisioner.

Set up the service account to interact with Git repositories:

1. Create a secret in the `tap-install` namespace or any preferred namespace, containing Git credentials in YAML format.
    - `host`, `username`, `caFile` and `password` or `personal access token` values for HTTP-based Git authentication.
    - `ssh-privatekey`, `identity`, `identity_pub`, and `known_hosts` for SSH-based Git authentication.

Note

The `stringData` key of the secret must end with either the `.yaml` or `.yml` suffix.

Using HTTP(s) based authentication
If using user name and password for authentication.

In this configuration for an air-gapped environment, the Git repository server has a custom certificate of authority that cannot be verified against public issuers, so you must provide the `caFile` content to log in against it.

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: workload-git-auth
  namespace: tap-install
type: Opaque
stringData:
  content.yaml: |
    git:
      #! For HTTP Auth. Recommend using https:// for the git server.
      host: GIT-SERVER
      username: GIT-USERNAME
      password: GIT-PASSWORD
      caFile: |
        -----BEGIN CERTIFICATE-----
        ...
        -----END CERTIFICATE-----
EOF
```

Using SSH based authentication
To use an SSH private key for authentication, create the Git secret with the authentication details as follows:

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: workload-git-auth
  namespace: tap-install
type: Opaque
stringData:
  content.yaml: |
    git:
      host: GIT-SERVER
      #! For SSH Auth
      ssh_privatekey: SSH-PRIVATE-KEY
      identity: SSH-PRIVATE-KEY
      identity_pub: SSH-PUBLIC-KEY
      known_hosts: GIT-SERVER-PUBLIC-KEYS
EOF
```

2. To create a secret to be added to the service account in the developer namespace in the GitOps repository, use this example for HTTP-based or this example for `setings.xml`-based, or follow the example below.

   Rather than directly including the actual user name and password in the Git repository secret, use the `data.values.imported` keys to add references to the values from the `git-auth` secret created in the previous step.

   This secret represents the Git secret that is created by the Namespace Provisioner in each managed namespace. It must be included in your Git repository linked in the `additional_sources` section of `tap-values.yaml` mentioned in the next step.

   Using HTTP(s) based authentication
   If using user name and password for authentication.

   In this configuration for an air-gapped environment, the Git repository server has a custom certificate of authority that cannot be verified against public issuers, so you must provide the `caFile` content to log in against it.

   ```
   #@ load("@ytt:data", "data")
   ---
   apiVersion: v1
   kind: Secret
   metadata:
    name: git
    annotations:
      tekton.dev/git-0: #@ data.values.imported.git.host
   type: kubernetes.io/basic-auth
   stringData:
    username: #@ data.values.imported.git.username
    password: #@ data.values.imported.git.token
    caFile: #@ data.values.imported.git.caFile
   ```

   Using settings.xml based authentication for Java applications
   If using user name and password for authentication.

   ```
   #@ load("@ytt:data", "data")
   ```

```
apiVersion: v1
kind: Secret
metadata:
 name: settings-xml
type: service.binding/maven
stringData:
 type: maven
 provider: sample
 #@yaml/text-templated-strings
 settings.xml: |
   <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0" xmlns:xsi="htt
p://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 https://mave
n.apache.org/xsd/settings-1.0.0.xsd">
       <mirrors>
           <mirror>
               <id>reposilite</id>
               <name>Accelerator samples</name>
               <url>(@= data.values.imported.git.host @)/vmware-tanzu/applica
tion-accelerator-samples</url>
               <mirrorOf>*</mirrorOf>
           </mirror>
       </mirrors>
       <servers>
           <server>
               <id>reposilite</id>
               <username>(@= data.values.imported.git.username @)</username>
               <password>(@= data.values.imported.git.password @)</password>
           </server>
       </servers>
   </settings>
```

Using SSH based authentication

If using SSH private key for authentication:

```
#@ load("@ytt:data", "data")
---
apiVersion: v1
kind: Secret
metadata:
  name: git
  annotations:
    tekton.dev/git-0: #@ data.values.imported.git.host
type: kubernetes.io/basic-auth
stringData:
  identity: #@ data.values.imported.git.identity
  identity.pub: #@ data.values.imported.git.identity_pub
  known_hosts: #@ data.values.imported.git.known_hosts
  ssh-privatekey: #@ data.values.imported.git.ssh_privatekey
```

3. Combine this `tap-values.yaml`:

Using Namespace Provisioner Controller

Add the following configuration to `tap-values.yaml`:

```
namespace_provisioner:
  controller: true
  additional_sources:
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/credentials
      url: https://git-airgap-server/application-accelerator-samples.git
  import_data_values_secrets:
```

```
  - name: workload-git-auth
    namespace: tap-install
    create_export: true
  default_parameters:
    supply_chain_service_account:
      secrets:
      - git
```

Where `https://git-airgap-server/application-accelerator-samples.git` is a fork of the application-accelerator-samples repository.

Using GitOps

Add the following configuration to `tap-values.yaml`:

```
namespace_provisioner:
  controller: false
  additional_sources:
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/credentials
      url: https://git-airgap-server/vmware-tanzu/application-accelerator-sam
ples.git
  gitops_install:
    ref: origin/main
    subPath: ns-provisioner-samples/gitops-install
    url: https://git-airgap-server/vmware-tanzu/application-accelerator-sampl
es.git
  import_data_values_secrets:
  - name: workload-git-auth
    namespace: tap-install
    create_export: true
  default_parameters:
    supply_chain_service_account:
      secrets:
      - git
```

Where `https://git-airgap-server/application-accelerator-samples.git` is a fork of the application-accelerator-samples repository.

- `additional sources` points to the location where the templated Git secret resides, which is created in all developer namespaces.
- Import the newly created `workload-git-auth` secret into Namespace Provisioner to use in `data.values.imported` by adding the secret to `import_data_values_secrets`.
- Add the secret to be included in the ServiceAccount in the `default_parameters`. For more information, see Customize service accounts.

Note

`create_export` is set to `true` in `import_data_values_secrets`. As a result, a SecretExport is automatically created for the `workload-git-auth` secret in the `tap-install` namespace by Namespace Provisioner. After the changes are reconciled, the secret named `git` is present in all provisioned namespaces and is also added to the default service account of those namespaces.

4. In the `ootb_supply_chain_*.gitops.credentials_secret` and `ootb_supply_chain_*.source.credentials_secret` section of your `tap-values.yaml` file, edit

the name of the Git secret containing the credentials. This is necessary for the supply chain to include the `secretRef` when creating the Flux `GitRepository` resource. For example:

```
ootb_supply_chain_testing_scanning:
gitops:
  credentials_secret: git  # Replace with the actual name of your GitOps Git se
cret for the workload, if different.
source:
  credentials_secret: git  # Replace with the actual name of your source Git se
cret for the workload, if different.
```

By providing this configuration, the supply chain associates the created `GitRepository` resource with the specified Git secret managed by Namespace Provisioner.

5. Create the workload:

Using HTTP/HTTPS or SSH-based
If using user name and password for authentication.

```
tanzu apps workload apply APP-NAME \
--git-repo GIT-REPO \
--git-branch BRANCH \
--type web \
--app APP-NAME \
--label apps.tanzu.vmware.com/has-tests="true" \
--namespace DEV-NAMESPACE \
--tail \
--yes
```

Using settings.xml based authentication for Java applications
If using user name and password for authentication.

```
tanzu apps workload apply APP-NAME \
--git-repo GIT_REPO \
--git-branch BRANCH \
--type web \
--app APP-NAME \
--label apps.tanzu.vmware.com/has-tests="true" \
--namespace DEV-NAMESPACE \
--param-yaml buildServiceBindings='[{"name": "settings-xml", "kind": "Secre
t"}]'
--tail \
--yes
```

Troubleshoot Namespace Provisioner

This topic tells you how to troubleshoot Namespace Provisioner in Tanzu Application Platform (commonly known as TAP).

Air-gapped installation

Namespace Provisioner relies on kapp-controller for any tasks involving communication with external services, such as registries or Git repositories. When operating in air-gapped environments or other scenarios where external services are secured by a Custom CA certificate, you must configure kapp-controller with the CA certificate data to prevent X.509 certificate errors. For more information, see Deploy onto Cluster in the Cluster Essentials for VMware Tanzu documentation.

View controller logs

To get the logs when using the controller workflow, run the following kubectl command:

```
kubectl -n tap-namespace-provisioning logs deployments/controller-manager
```

Use `-f` to follow the log output.

Provisioner application error

After the Namespace Provisioner is installed in the Tanzu Application Platform cluster, the main resource to check is the provisioner Carvel Application in the `tap-namespace-provisioning` namespace. To check for the status of the Application, run the following kubectl command:

```
kubectl -n tap-namespace-provisioning get app/provisioner --template={{.status.usefulE
rrorMessage}}
```

Common errors

You might encounter one of the following errors:

Namespace selector malformed

When using the controller and customizing the `namespace_selector` from `tap_values.yaml`, the match expression must be compliant with the Kubernetes label selector. If it is not compliant, the Namespace Provisioner controller fails and log an error message in the controller logs.

For example, if the configured `namespace_selector` is as follows:

```
namespace_provisioner:
  controller: true
  namespace_selector:
    matchExpressions:
    - key: apps.tanzu.vmware.com/tap-ns
      operator: exists
```

This is not compliant as the operator must be `Exist` instead of `exists`. When labeling the namespace `dev` with `apps.tanzu.vmware.com/tap-ns`, the controller produces an error message similar to the following, (followed by some reconciliation messages)

```
{"level":"error","ts":"2022-12-14T15:41:44.639402794Z","logger":".0.1.NamespaceSelecto
rReconciler","msg":"unable to sync","controller":"namespace","controllerGroup":"","con
trollerKind":"Namespace","Namespace":{"name":"dev"},"namespace":"","name":"dev","recon
cileID":"26395d34-418b-446d-9b5e-a4a73cc657ed","resourceType":"/v1, Kind=Namespace","e
rror":"\"exists\" is not a valid pod selector operator","stacktrace":"..."}
```

Debugging ytt templating errors in additional sources

When working with ytt, templating errors in the additional sources in your GitOps repository can cause the Provisioner Carvel application to go into `Reconcile Failed` state. To debug the Application, run the following command:

```
kubectl -n tap-namespace-provisioning get app/provisioner --template={{.status.usefulE
rrorMessage}}
```

Sample Error message from Application when there is a ytt templating error:

```
ytt: Error:
- library.eval: Evaluating library 'witherror':
    in <toplevel>
      template.yaml:155 | #@          instances = overlay.apply(instance.eval(), custo
mize())
    reason:
     - struct has no .gl_secret_user field or method
         in <toplevel>
           _ytt_lib/witherror/secrets.yaml:12 |   username: #@ data.values.gl_secret_u
ser
```

Unable to delete namespace

When a user tries to delete a namespace that was managed by Namespace Provisioner, it gets stucks in the `Terminating` status.

Possible Cause 1: When a provisioned namespace that has a Cartographer Workload in it is deleted, the namespace will likely remain in the Terminating state because some resources can not be deleted. One of the causes of this behavior is that the Cartographer Workload using the Out of the Box supply chains and delivery creates a Carvel Kapp App for the workload that references the ServiceAccount in the namespace. Deleting the namespace deletes the Service Account that Kapp relies on before the App itself is deleted. As a result, the Carvel App blocks the namespace termination while waiting for the ServiceAccount to exist with a `finalizer` (`finalizers.kapp-ctrl.k14s.io/delete`) message.

Solution: Remove the Kapp App finalizer in the Kapp App

Possible Cause 2: If you try to delete a namespace that was previously managed by the controller, and the namespace was not cleaned up before disabling the controller, it gets stuck in the `Terminating` state. This happens because the controller adds a `finalizer` to the namespaces (`namespace-provisioner.apps.tanzu.vmware.com/finalizer`) it manages, and is no longer there to clean up that finalizer as it was deactivated by the user.

Solution: Remove manually the finalizer in the namespace

Namespace Provisioner reference

This section tells you about known issues and limitations, default resources, and parameters in Namespace Provisioner.

Known limitations and issues

Default resources

Parameters

Known limitations and issues in Namespace Provisioner

This topic tells you about the known issues and limitations for Namespace Provisioner in Tanzu Application Platform (commonly known as TAP).

- If you are using a GitOps repository to manage the list of namespaces, all the namespaces in the list must exist in the cluster. The provisioner application fails to reconcile if the namespaces do not exist on the cluster.
- To use different private repositories, the secret used for each entry must be a unique name, for example, `gitops_install`, or `additional_sources`. Reusing the same secret is not supported due to a limitation in kapp-controller.

  Note

This limitation is resolved in Tanzu Cluster Essentials v1.6.0 and later.

- Before performing any operations, such as uninstalling the Namespace Provisioner or changing from Controller mode to GitOps mode, ensure that you edit the namespaces managed by the Namespace Provisioner in the `namespace-provisioner.apps.tanzu.vmware.com/finalizer` finalizer. Either remove the label used to set the namespaces as managed or edit the namespace manifest and remove the finalizer entry. For more information, see Manage a list of developer namespaces.

Default resources

This topic tells you about the resources that are templated in the default-resources secret for the different installation profile and supply chain value combinations in Tanzu Application Platform (commonly known as TAP).

Namespace Provisioner is installed as part of the full, iterate, build, and run installation profiles. The default set of resources provisioned in a namespace is based on a combination of the installation profile employed and the supply chain that is installed on the cluster. For more information about installation profiles, see Installation profiles in Tanzu Application Platform.

The following table shows the list of resources that are templated in the default-resources secret for each installation profile and supply chain value combination:

Namespace	Kind	Name	supply_chain	Install Profile	Reconcile
tap-install	PackageInstall	grype-scanner-{ns}	testing_scanning	full, build	Yes
tap-install	Secret	grype-scanner-{ns}	testing_scanning	full, build	Yes
Developer Namespace	Secret	registries-credentials	n/a	full, iterate, build, run	Yes
Developer Namespace	ServiceAccount	From: ootb_supply_chain_{supply_chain}.service_account (default: “default”)	n/a	full, iterate, build, run	No
Developer Namespace	ServiceAccount	From: ootb_delivery_basic.service_account (default: “default”)	n/a	full, iterate, run	No
Developer Namespace	RoleBinding	default-permit-deliverable	n/a	full, iterate, run	Yes
Developer Namespace	RoleBinding	default-permit-workload	n/a	full, iterate, build	Yes
Developer Namespace	LimitRange	{namespace}-lr	n/a	run	Yes

For installing additional resources for OOTB Supply Chain with Testing and Scanning, see Supply Chain Security Tools - Scan.

Customize namespaces in Namespace Provisioner

This topic tells you how to use Namespace Provisioner to customize namespaces in controller mode in Tanzu Application Platform (commonly known as TAP).

When managing multiple developer namespaces in a cluster, it is often necessary to customize each namespace individually. To customize a namespace in controller mode, add parameters to a

namespace through labels and annotations using either the default prefix or a custom-defined prefix.

In GitOps mode, you can configure these parameters in the GitOps file. For more information, see Customize the label and annotation prefixes that controller watches.

Caution

If a parameter is initially created through annotations and later a label with the same key is used, the annotation is overwritten.

Parameter key

- The format for specifying the parameter is: `<prefix>/<parameter-key>=<parameter-value>`.
- The parameter key can be a single string or a pseudo JSON-path structure, for example, `kye1.inner-key1.inner-key3.inner-key4`. This is translated into a structured format in the values.
- The label value can only be a string.
- If you need to pass a list or another object as the parameter value, annotations should be used instead. Annotations support using `[]` to define lists and `{}` to define objects
- All parameters created with labels and annotations can be utilized when using templates for resources in ytt `additional_sources` from `data.values`.

Examples:

1. To define a list of tools used by the namespace:

   ```
   kubectl annotate ns dev param.nsp.tap/project.tools='["git", "maven"]'
   ```

   The `desired-namespaces` ConfigMap will look like:

   ```
   #@data/values
   ---
   namespaces:
   - name: dev
    project:
      tools:
      - git
      - maven
   ```

2. To add a list of objects:

   ```
   kubectl annotate ns dev param.nsp.tap/volume.claims='[{"name": "logs", "mountPa
   th": "/var/logs/app"}, {"name": "truststore", "mountPath": "/opt/app/ssl"}]
   ```

   The `desired-namespaces` ConfigMap will look like:

   ```
   #@data/values
   ---
   namespaces:
   - name: dev
    volume:
      claims:
      - name: logs
      mountPath: /var/logs/app
   ```

```
    - name: truststore
    mountPath: /opt/app/ssl
```

3. Simple key-value:

   ```
   kubectl annotate ns dev param.nsp.tap/scanpolicy=relaxed
   ```

   The `desired-namespaces` ConfigMap will look like:

   ```
   #@data/values
   ---
   namespaces:
   - name: dev
    scanpolicy: relaxed
   ```

4. Object as value:

   ```
   kubectl annotate ns dev param.nsp.tap/maven.values='{"username":"user", "passwo
   rd":"my-pass","repo":"myrepo","version":"0.1.1-alpha.0"}'
   ```

   The `desired-namespaces` ConfigMap will look like:

   ```
   #@data/values
   ---
   namespaces:
   - name: dev
    maven:
      values:
        username: user
        password: my-pass
        repo: myrepo
        Version: 0.1.1-alpha.0
   ```

Reserved Namespace Parameters

Namespace Provisioner reserves certain parameters for its use. The following is a list of parameters used by the Namespace Provisioner, which apply to both the `default_parameters` in `tap-values.yaml` and the namespace parameters through labels and annotations:

- `limits` (object): Use to configure the LimitRange. For more information, see Customize Limit Range defaults.
- `skip_limit_range` (boolean): Use to determine if the LimitRange should be created. For more information, see Customize Limit Range defaults.
- `skip_grype` (boolean): Use to determine if Grype scanner resources are going to be created. For more information, see Deactivate Grype install.
- `supply_chain_service_account` (object): Contains the secrets and imagePullSecrets to be added to the Supply Chain ServiceAccount. For more information, see Customize service accounts.
- `delivery_service_account` (object): Contains the secrets and imagePullSecrets to be added to the delivery ServiceAccount. For more information, see Customize service accounts.

Overview of Service Bindings

This topic tells you about using Service Bindings in Tanzu Application Platform (commonly know as TAP).

Supported service binding specifications

Service Bindings packages the Service Binding for Kubernetes open source project.

It implements the Service Binding Specification for Kubernetes v1.0.

This implementation provides support for:

- Provisioned Service
- Workload Projection
- Service Binding
- Direct Secret Reference
- Role-Based Access Control (RBAC)

The following are not supported:

- Workload Resource Mapping
- Extensions including:
    - Binding Secret Generation Strategies

Overview of Service Bindings

This topic tells you about using Service Bindings in Tanzu Application Platform (commonly know as TAP).

Supported service binding specifications

Service Bindings packages the Service Binding for Kubernetes open source project.

It implements the Service Binding Specification for Kubernetes v1.0.

This implementation provides support for:

- Provisioned Service
- Workload Projection
- Service Binding
- Direct Secret Reference
- Role-Based Access Control (RBAC)

The following are not supported:

- Workload Resource Mapping
- Extensions including:
    - Binding Secret Generation Strategies

Install Service Bindings

This topic tells you how to install Service Bindings from the Tanzu Application Platform (commonly known as TAP) package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Service Bindings. For more information about profiles, see Components and installation profiles.

Prerequisites

Before installing Service Bindings:

- Complete all prerequisites to install Tanzu Application Platform (commonly knows as TAP). For more information, see Prerequisites.

Install Service Bindings

Use the following procedure to install Service Bindings:

1. List version information for the package by running:

```
tanzu package available list servicebinding.tanzu.vmware.com --namespace tap-in
stall
```

For example:

```
$ tanzu package available list servicebinding.tanzu.vmware.com --namespace tap-
install
- Retrieving package versions for servicebinding.tanzu.vmware.com...
  NAME                              VERSION  RELEASED-AT
  servicebinding.tanzu.vmware.com   0.10.3   2023-12-05T08:26:41Z
```

2. Install the package by running:

```
tanzu package install service-bindings -p servicebinding.tanzu.vmware.com  -v
0.10.3 -n tap-install
```

Example output:

```
/ Installing package 'servicebinding.tanzu.vmware.com'
| Getting namespace 'tap-install'
- Getting package metadata for 'servicebinding.tanzu.vmware.com'
| Creating service account 'service-bindings-tap-install-sa'
| Creating cluster admin role 'service-bindings-tap-install-cluster-role'
| Creating cluster role binding 'service-bindings-tap-install-cluster-rolebindi
ng'
\ Creating package resource
| Package install status: Reconciling

 Added installed package 'service-bindings' in namespace 'tap-install'
```

3. Verify the package install by running:

```
tanzu package installed get service-bindings -n tap-install
```

Example output:

```
- Retrieving installation details for service-bindings...
NAME:                    service-bindings
PACKAGE-NAME:            servicebinding.tanzu.vmware.com
PACKAGE-VERSION:         0.10.3
STATUS:                  Reconcile succeeded
```

```
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

4. Run the following command:

   ```
   kubectl get pods -n service-bindings
   ```

   For example:

   ```
   $ kubectl get pods -n service-bindings
   NAME                                                READY   STATUS    RESTARTS
   AGE
   servicebinding-controller-manager-7856497ddd-bmgv2   1/1     Running   0
   5m59s
   ```

   Verify that `STATUS` is `Running`

Troubleshoot Service Bindings

This topic tells you how to troubleshoot Service Bindings in Tanzu Application Platform (commonly known as TAP).

Collect logs

To help identify issues when troubleshooting, you can retrieve and examine logs from the service binding manager.

To retrieve pod logs from the `manager` running in the `service-bindings` namespace, run:

```
kubectl -n service-bindings logs -l role=manager
```

For example:

```
$ kubectl -n service-bindings logs -l role=manager

2021/11/05 15:25:28 Registering 3 clients
2021/11/05 15:25:28 Registering 3 informer factories
2021/11/05 15:25:28 Registering 7 informers
2021/11/05 15:25:28 Registering 8 controllers
{"severity":"INFO","timestamp":"2021-11-05T15:25:28.483823208Z","caller":"logging/nfi
g.go:116","message":"Successfully created the logger."}
{"severity":"INFO","timestamp":"2021-11-05T15:25:28.48392361Z","caller":"logging/confi
g.go:117","message":"Logging level set to: info"}
{"severity":"INFO","timestamp":"2021-11-05T15:25:28.483999911Z","caller":"logging/conf
ig.go:79","message":"Fetch GitHub commit ID from kodata failed","error":"open /var/ru
n/ko/HEAD: no such file or directory"}
{"severity":"INFO","timestamp":"2021-11-05T15:25:28.484035711Z","logger":"webhook","ca
ller":"profiling/server.go:64","message":"Profiling enabled: false"}
{"severity":"INFO","timestamp":"2021-11-05T15:25:28.522884909Z","logger":"webhook","ca
ller":"leaderelection/context.go:46","message":"Running with Standard leader electio
n"}
{"severity":"INFO","timestamp":"2021-11-05T15:25:28.523358615Z","logger":"webhook","ca
ller":"provisionedservice/controller.go:31","message":"Setting up event handlers."}
...
{"severity":"ERROR","timestamp":"2021-11-17T12:30:24.557178813Z","logger":"webhook","c
aller":"controller/controller.go:548","message":"Reconcile error","duration":"276.504µ
s","error":"deployments.apps \"spring-petclinic\" not found","stacktrace":"knative.de
v/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90db
b0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkIte
m\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\n
knative.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg@v0.0.0-20210331
065221-952fdd90dbb0/controller/controller.go:468"}
```

```
{"severity":"ERROR","timestamp":"2021-11-17T12:47:04.558217679Z","logger":"webhook","c
aller":"controller/controller.go:548","message":"Reconcile error","duration":"249.103µ
s","error":"deployments.apps \"spring-petclinic\" not found","stacktrace":"knative.de
v/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90db
b0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkIte
m\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\n
knative.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg@v0.0.0-20210331
065221-952fdd90dbb0/controller/controller.go:468"}
{"severity":"ERROR","timestamp":"2021-11-17T13:03:44.558683121Z","logger":"webhook","c
aller":"controller/controller.go:548","message":"Reconcile error","duration":"177.403µ
s","error":"deployments.apps \"spring-petclinic\" not found","stacktrace":"knative.de
v/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90db
b0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkIte
m\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\n
knative.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg@v0.0.0-20210331
065221-952fdd90dbb0/controller/controller.go:468"}
{"severity":"ERROR","timestamp":"2021-11-17T13:20:24.559192644Z","logger":"webhook","c
aller":"controller/controller.go:548","message":"Reconcile error","duration":"223.203µ
s","error":"deployments.apps \"spring-petclinic\" not found","stacktrace":"knative.de
v/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90db
b0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkIte
m\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\n
knative.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg@v0.0.0-20210331
065221-952fdd90dbb0/controller/controller.go:468"}
{"severity":"ERROR","timestamp":"2021-11-17T13:37:04.559648412Z","logger":"webhook","c
aller":"controller/controller.go:548","message":"Reconcile error","duration":"173.003µ
s","error":"deployments.apps \"spring-petclinic\" not found","stacktrace":"knative.de
v/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90db
b0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkIte
m\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\n
knative.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg@v0.0.0-20210331
065221-952fdd90dbb0/controller/controller.go:468"}
{"severity":"ERROR","timestamp":"2021-11-17T13:53:44.56010516Z","logger":"webhook","ca
ller":"controller/controller.go:548","message":"Reconcile error","duration":"182.402µ
s","error":"deployments.apps \"spring-petclinic\" not found","stacktrace":"knative.de
v/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90db
b0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkIte
m\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\n
knative.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg@v0.0.0-20210331
065221-952fdd90dbb0/controller/controller.go:468"}
{"severity":"ERROR","timestamp":"2021-11-17T14:10:24.560536033Z","logger":"webhook","c
aller":"controller/controller.go:548","message":"Reconcile error","duration":"155.603µ
s","error":"deployments.apps \"spring-petclinic\" not found","stacktrace":"knative.de
v/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90db
b0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkIte
m\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\n
knative.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg@v0.0.0-20210331
065221-952fdd90dbb0/controller/controller.go:468"}
{"severity":"ERROR","timestamp":"2021-11-17T14:27:04.560960243Z","logger":"webhook","c
aller":"controller/controller.go:548","message":"Reconcile error","duration":"171.002µ
s","error":"deployments.apps \"spring-petclinic\" not found","stacktrace":"knative.de
v/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90db
b0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkIte
m\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\n
knative.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg@v0.0.0-20210331
065221-952fdd90dbb0/controller/controller.go:468"}
{"severity":"ERROR","timestamp":"2021-11-17T14:43:44.56142548Z","logger":"webhook","ca
ller":"controller/controller.go:548","message":"Reconcile error","duration":"179.203µ
s","error":"deployments.apps \"spring-petclinic\" not found","stacktrace":"knative.de
v/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90db
b0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkIte
m\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\n
knative.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg@v0.0.0-20210331
065221-952fdd90dbb0/controller/controller.go:468"}
{"severity":"ERROR","timestamp":"2021-11-17T15:00:24.561881861Z","logger":"webhook","c
aller":"controller/controller.go:548","message":"Reconcile error","duration":"167.902µ
```

```
s","error":"deployments.apps \"spring-petclinic\" not found","stacktrace":"knative.de
v/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90db
b0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkIte
m\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\n
knative.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg@v0.0.0-20210331
065221-952fdd90dbb0/controller/controller.go:468"}
```

Service Bindings resource specification

This topic tells you about the Service Bindings resource specification in Tanzu Application Platform (commonly known as TAP).

The `ServiceBinding` resource shape and behavior is defined by the following specification:

```
apiVersion: servicebinding.io/v1alpha3
kind: ServiceBinding
metadata:
  name: account-db
spec:
  service:
    apiVersion: mysql.example/v1alpha1
    kind: MySQL
    name: account-db
  workload:
    apiVersion: apps/v1
    kind: Deployment
    name: account-service
```

Version matrix for Service Bindings

This topic provides you with a version matrix for the Service Bindings package (`servicebinding.tanzu.vmware.com`) and its open source components in Tanzu Application Platform v1.9 (commonly known as TAP).

To view this information for another Tanzu Application Platform version, select the version from the drop-down menu at the top of this page.

Service Binding Runtime is the community reference implementation of Service Binding.

The following table has the component versions for the Service Bindings package.

Component	Version
Service Bindings package version	0.12.0
Service Bindings runtime version	0.8.0

Note

Tanzu Application Platform patch releases are only added to the table when there is a change to one or more of the other versions in the table. Otherwise, the corresponding versions remain the same for each Tanzu Application Platform patch release.

Overview of Service Registry

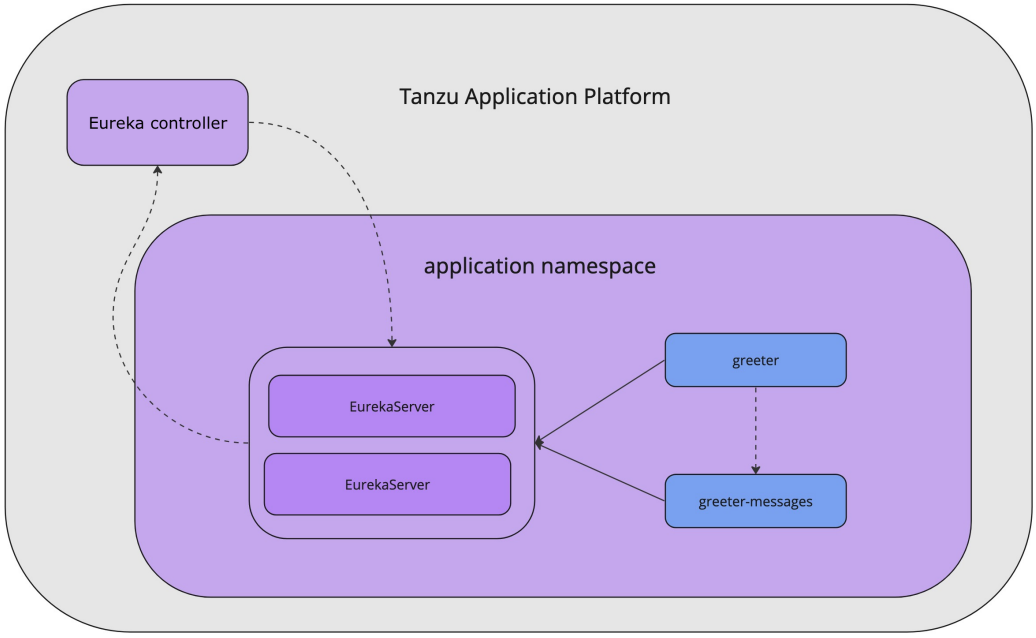
Service Registry for VMware Tanzu provides on-demand Eureka servers for your Tanzu Application Platform (commonly known as TAP) clusters. With Service Registry, you can create Eureka servers

in your namespaces and bind Spring Boot workloads to them. This provides a smooth transition from the past practice of using Spring Cloud Services for Tanzu Application Service to create Eureka servers.

On-demand namespaced Eureka server instances can be provisioned for microservices Spring Boot applications that rely on service discovery.

Architecture

Workloads communicate with the EurekaServer resources in the application namespace. From within Tanzu Application Platform, the Eureka controller communicates with the EurekaServer resources in the application namespace.



Capacity Requirements

Each node of the Eureka controller requires:

- 64 MiB of memory
- 10m of vCPU

with limits of:

- 128 MiB of memory
- 500m of vCPU

You can scale the controller horizontally for higher availability.

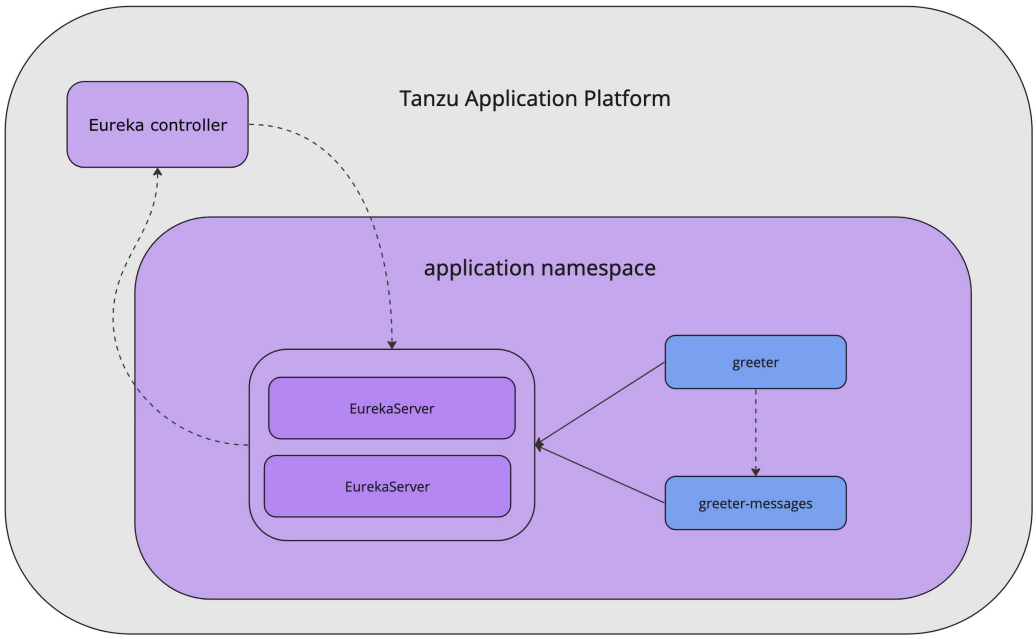
Overview of Service Registry

Service Registry for VMware Tanzu provides on-demand Eureka servers for your Tanzu Application Platform (commonly known as TAP) clusters. With Service Registry, you can create Eureka servers in your namespaces and bind Spring Boot workloads to them. This provides a smooth transition from the past practice of using Spring Cloud Services for Tanzu Application Service to create Eureka servers.

On-demand namespaced Eureka server instances can be provisioned for microservices Spring Boot applications that rely on service discovery.

Architecture

Workloads communicate with the EurekaServer resources in the application namespace. From within Tanzu Application Platform, the Eureka controller communicates with the EurekaServer resources in the application namespace.



Capacity Requirements

Each node of the Eureka controller requires:

- 64 MiB of memory
- 10m of vCPU

with limits of:

- 128 MiB of memory
- 500m of vCPU

You can scale the controller horizontally for higher availability.

Install Service Registry

This topic tells you how to install Service Registry from the Tanzu Application Platform (commonly known as TAP) package repository.

Prerequisites

Before installing Service Registry, you must:

- Fulfill all prerequisites for installing Tanzu Application Platform
- Install cert-manager

Install

To install Service Registry on a compliant Kubernetes cluster:

1. List version information for the package by running:

   ```
   tanzu package available list service-registry.spring.apps.tanzu.vmware.com --na
   mespace tap-install
   ```

   For example:

   ```
   $ tanzu package available list service-registry.spring.apps.tanzu.vmware.com --
   namespace tap-install
   NAME                                                VERSION  RELEASED-AT
   service-registry.spring.apps.tanzu.vmware.com       1.2.0    2023-09-12 19:00:0
   0 -0500 EST
   ```

2. Install the package by running:

   ```
   tanzu package install service-registry \
   --package service-registry.spring.apps.tanzu.vmware.com \
   --version VERSION -n tap-install
   ```

   Where `VERSION` is the version you want, such as `1.2.0`.

   For example:

   ```
   $ tanzu package install service-registry \
   --package service-registry.spring.apps.tanzu.vmware.com \
   --version 1.2.0 -n tap-install

   / Installing package 'service-registry.spring.apps.tanzu.vmware.com'
   | Getting namespace 'eureka-system'
   | Getting package metadata for 'service-registry.spring.apps.tanzu.vmware.com'
   | Creating service account 'service-registry-sa'
   | Creating cluster admin role 'service-registry-cluster-role'
   | Creating cluster role binding 'service-registry-cluster-rolebinding'
   / Creating package resource
   - Package install status: Reconciling


   Added installed package 'service-registry' in namespace 'tap-install'
   ```

   Note

   Because there are no customization options at this time, there is no need to include a `--values-file` option.

3. Verify that you installed the package by running:

   ```
   tanzu package installed get service-registry -n tap-install
   ```

   For example:

   ```
   $ tanzu package installed get service-registry -n tap-install

   NAMESPACE:          tap-install
   NAME:               service-registry
   PACKAGE-NAME:       service-registry.spring.apps.tanzu.vmware.com
   PACKAGE-VERSION:    1.2.0
   ```

```
STATUS:             Reconcile succeeded
CONDITIONS:         - type: ReconcileSucceeded
  status: "True"
  reason: ""
  message: ""
```

Create EurekaServer resources

This topic tells you about options when creating a `EurekaServer` resource.

Discover available parameters

Examine the available parameters when creating a `EurekaServer` resource by running:

```
kubectl explain eurekaservers.service-registry.spring.apps.tanzu.vmware.com.spec
```

For example:

```
$ kubectl explain eurekaservers.service-registry.spring.apps.tanzu.vmware.com.spec

GROUP:      service-registry.spring.apps.tanzu.vmware.com
KIND:       EurekaServer
VERSION:    v1alpha1

FIELD: spec <Object>

DESCRIPTION:
    EurekaServerSpec defines the desired state of EurekaServer

FIELDS:
  imagePullSecretName   <string>
    Name of secret to use for pulling the Eureka image

  replicas      <integer>
    Replica count for the EurekaServer StatefulSet
```

Create a EurekaServer resource

To create a `EurekaServer` resource:

1. Create a `EurekaServer` resource with two replicas by using the following YAML definition:

   ```
   ---
   apiVersion: service-registry.spring.apps.tanzu.vmware.com/v1alpha1
   kind: EurekaServer
   metadata:
     name: eurekaserver-sample
     namespace: my-apps
   spec:
     replicas: 2
   ```

2. Save the YAML definition as `eurekaserver.yaml`.

3. Apply the YAML definition by running:

   ```
   kubectl apply -f eurekaserver.yaml
   ```

4. Check the status of the `EurekaServer` resource by running:

   ```
   kubectl describe YOUR-ADDRESS RESOURCE-NAME
   ```

For example:

```
$ kubectl describe eurekaservers.service-registry.spring.apps.tanzu.vmware.com
eurekaserver

Name:         eurekaserver
Namespace:    my-apps
Labels:       <none>
Annotations:  <none>
API Version:  service-registry.spring.apps.tanzu.vmware.com/v1alpha1
Kind:         EurekaServer
Metadata:
 Creation Timestamp:  2023-08-30T14:51:04Z
 Generation:          1
 Resource Version:    4698719
 UID:                 4dd60698-6332-43bf-a27d-cef610579c98
Spec:
 Replicas:  2
Status:
 Binding:
   Name:  eureka-eurekaserver-client-binding-mvvlx
 Conditions:
   Last Transition Time:  2023-08-30T14:51:04Z
   Message:               EurekaServer reconciled
   Observed Generation:   1
   Reason:                EurekaServerReconciled
   Status:                True
   Type:                  Ready
 Observed Generation:     1
 Server Binding:
   Name:  eureka-eurekaserver-server-binding-2jq76
Events:    <none>
```

A successful `EurekaServer` resource has a `Ready` condition set to `true` and a `status.binding.name` field pointing to a secret containing connection information.

Configure workloads in Tanzu Application Platform by using Service Registry

This topic tells you how to configure Tanzu Application Platform (commonly known as TAP) workloads running Spring Boot applications to connect to Service Registry `EurekaServer` resources.

Prerequisite

Create a `EurekaServer` resource. For instructions, see Create Eureka Servers.

Claim credentials

Claim credentials from Tanzu CLI or by creating a `ResourceClaim` directly:

Using Tanzu CLI
Claim credentials by running:

```
tanzu service resource-claim create CLAIM-NAME \
    --resource-kind EurekaServer \
    --resource-api-version service-registry.spring.apps.tanzu.vmware.com/v1alpha1 \
    --resource-name RESOURCE-NAME \
    --resource-namespace RESOURCE-NAMESPACE \
    --namespace WORKLOAD-NAMESPACE
```

Where:

- `CLAIM-NAME` is a name you choose for your claim.
- `RESOURCE-NAME` is the name of the `EurekaServer` resource you want to claim.
- `RESOURCE-NAMESPACE` is the namespace that your `EurekaServer` resource is in.
- `WORKLOAD-NAMESPACE` is the namespace that your workload is in.

For example:

```
$ tanzu service resource-claim create eurekaserver-sample \
    --resource-kind EurekaServer \
    --resource-api-version service-registry.spring.apps.tanzu.vmware.com/v1alpha1 \
    --resource-name eurekaserver-sample \
    --resource-namespace my-apps \
    --namespace my-apps
```

Using a ResourceClaim

Create a YAML file, similar to the following example, and name it `resource-claim.yaml`:

```
---
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ResourceClaim
metadata:
  name: eurekaserver-sample
  namespace: my-apps
spec:
  ref:
    apiVersion: service-registry.spring.apps.tanzu.vmware.com/v1alpha1
    kind: EurekaServer
    name: eurekaserver-sample
    namespace: my-apps
```

In kubectl, create the `ResourceClaim` by running:

```
kubectl apply -f resource-claim.yaml
```

Inspect the progress of your claim

Inspect the progress of your claim creation by running:

```
tanzu service resource-claim get MY-CLAIM-NAME --namespace MY-NAMESPACE
```

or by running:

```
kubectl get resourceclaim MY-CLAIM-NAME --namespace MY-NAMESPACE --output yaml
```

Use Eureka for service discovery in workloads

To use Eureka for service discovery in workloads:

1. If you have an existing application already configured to use Spring Cloud Service Discovery, claim `EurekaServer` credentials to access the running Eureka servers by adding the following to the `spec.serviceClaims` section of a workload:

   ```
   serviceClaims:
     - name: eureka
       ref:
   ```

```
        apiVersion: services.apps.tanzu.vmware.com/v1alpha1
        kind: ResourceClaim
        name: eurekaserver-sample
```

By claiming the credentials, a workload has its Eureka client configured to interact with the referenced Eureka server.

For example, you can use the following workloads to deploy the greeting application.

`greeter-messages.yaml` example:

```
# greeter-messages.yaml
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: greeter-messages
  namespace: my-apps
  labels:
    apps.tanzu.vmware.com/workload-type: server
    apps.tanzu.vmware.com/has-tests: "true"
    app.kubernetes.io/part-of: greeter
spec:
  build:
    env:
      - name: BP_JVM_VERSION
        value: "17"
      - name: BP_GRADLE_BUILT_MODULE
        value: "greeter-messages"
      - name: BP_GRADLE_BUILD_ARGUMENTS
        value: "--no-daemon clean bootJar"
  env:
    - name: SPRING_PROFILES_ACTIVE
      value: "development"
  serviceClaims:
    - name: eureka
      ref:
        apiVersion: services.apps.tanzu.vmware.com/v1alpha1
        kind: ResourceClaim
        name: eurekaserver-sample
  source:
    git:
      url: https://github.com/spring-cloud-services-samples/greeting
      ref:
        branch: main
```

`greeter.yaml` example:

```
# greeter.yaml
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: greeter
  namespace: my-apps
  labels:
    apps.tanzu.vmware.com/workload-type: web
    apps.tanzu.vmware.com/has-tests: "true"
    app.kubernetes.io/part-of: greeter
spec:
  build:
    env:
      - name: BP_JVM_VERSION
        value: "17"
      - name: BP_GRADLE_BUILT_MODULE
```

```
        value: "greeter"
      - name: BP_GRADLE_BUILD_ARGUMENTS
        value: "--no-daemon clean bootJar"
  env:
    - name: SPRING_PROFILES_ACTIVE
      value: "development"
  serviceClaims:
    - name: eureka
      ref:
        apiVersion: services.apps.tanzu.vmware.com/v1alpha1
        kind: ResourceClaim
        name: eurekaserver-sample
  source:
    git:
      url: https://github.com/spring-cloud-services-samples/greeting
      ref:
        branch: main
```

2. Create the workloads. For example, for the greeter application you run:

```
tanzu apps workload create -f greeter-messages.yaml --yes
tanzu apps workload create -f greeter.yaml --yes
```

3. From the Tanzu CLI, retrieve the ingress route associated with the greeter application by running:

```
tanzu apps workload get greeter
```

   For example:

```
$ tanzu apps workload get greeter

# other output...

  Knative Services
  NAME      READY   URL
  greeter   Ready   https://greeter.my-apps.tap

To see logs: "tanzu apps workload tail greeter --timestamp --since 1h"
```

   Where `https://greeter.my-apps.tap` is the accessible ingress route to the greeter application

4. Visit `ROUTE/hello`, where `ROUTE` is the ingress route you just retrieved. The greeter application uses Service Registry to look up the Message Generation application and get a greeting message. This message initially is `Hello, Bob!`.

5. See what the Message Generation application is sending back from viewing its logs by running:

```
tanzu apps workload tail greeter-messages
```

   Example:

```
$ tanzu apps workload tail greeter-messages

greeter-messages-579d67c498-bf6zl[workload] 2023-10-20T17:52:17.001Z  INFO 1 --
- [nio-8080-exec-3] messages.MessagesController              : Now saying "Hi"
to John
```

6. Create a different greeting message by providing the `salutation` and `name` parameters. Example:

```
ROUTE/hello?salutation=Hi&name=John
```

The greeter application sends those parameters to the Message Generation application, and the resulting greeting is customized to match.

(Optional) Use Service Registry with an executable JAR file application

In the greeting application example, `BP_GRADLE_BUILD_ARGUMENTS` is set to include the `bootJar` task in addition to the default Gradle build arguments. This setting is necessary for this example base because the `build.gradle` file contains a `jar` section, and the Spring Boot buildpack will not inject the `spring-cloud-bindings` library into the application if it is an executable JAR file.

`spring-cloud-bindings` is required to process the `serviceClaim` into properties that tell the discovery client how to find the Eureka server.

To use Service Registry with an executable JAR file application, you must explicitly include spring-cloud-bindings v1.13.0 or later and set the `org.springframework.cloud.bindings.boot.enable=true` system property as described in the library README file in GitHub.

Overview of Services Toolkit

Services Toolkit is responsible for backing many of the most powerful service capabilities in Tanzu Application Platform (commonly known as TAP).

From the integration of an extensive list of cloud-based and on-prem services, through to the offering and discovery of those services, and finally to the claiming and binding of service instances to application workloads, Services Toolkit has the tools you need to make working with services on Tanzu Application Platform simple, easy, and effective.

Note

These docs apply to Services Toolkit v0.10 and later. To view the Services Toolkit documentation for v0.9 and earlier, see the previous Services Toolkit site.

Capabilities

The main capabilities on offer in Tanzu Application Platform through Services Toolkit are:

1. The classes and claims abstraction: provides a simple, but powerful, user experience to apps teams, while promoting a strong separation of concerns between apps teams and ops teams.
2. Dynamic provisioning of service instances: enables apps teams to create service instances that adhere to company policy. Apps teams can create instances on-demand as needed.
3. Seamless integration of almost any service, cloud-based or on-prem, into Tanzu Application Platform with minimal configuration overhead: provides a near-limitless range of services to help boost developer productivity.

Getting started

If this is your first time working with services on Tanzu Application Platform, you might want to start with Claim services on Tanzu Application Platform and Consume services on Tanzu Application Platform in the getting started guide. These guides run through the basics, after which you can

return here to the Services Toolkit component documentation to continue your services journey on Tanzu Application Platform.

How this documentation is organized

The Services Toolkit component documentation consists of the following sections that relate to what you are want to achieve:

- Concepts: To gain a deeper understanding of Services Toolkit.
- Tutorials: To learn through following examples.
- How-to guides: To find a set of steps to solve a specific problem.
- Reference material: To find specific information such as Services Toolkit's APIs, the Tanzu Service CLI plug-in, and troubleshooting information.

Tutorials and concepts are of most relevance when studying, while how-to guides and reference material are of most use while working.

The following is a selection of useful topics on offer:

For apps teams:

- Tutorial: Working with Bitnami Services

For ops teams:

- Tutorial: Setup Dynamic Provisioning of Service Instances
- Tutorial: Integrating Cloud Services (AWS, Azure, GCP, etc.) into Tanzu Application Platform
- How-to guide: Configure Dynamic Provisioning of AWS RDS Service Instances

For everyone:

- Concept: Four Levels of Service Consumption in Tanzu Application Platform

Overview of Services Toolkit

Services Toolkit is responsible for backing many of the most powerful service capabilities in Tanzu Application Platform (commonly known as TAP).

From the integration of an extensive list of cloud-based and on-prem services, through to the offering and discovery of those services, and finally to the claiming and binding of service instances to application workloads, Services Toolkit has the tools you need to make working with services on Tanzu Application Platform simple, easy, and effective.

Note

These docs apply to Services Toolkit v0.10 and later. To view the Services Toolkit documentation for v0.9 and earlier, see the previous Services Toolkit site.

Capabilities

The main capabilities on offer in Tanzu Application Platform through Services Toolkit are:

1. The classes and claims abstraction: provides a simple, but powerful, user experience to apps teams, while promoting a strong separation of concerns between apps teams and ops teams.

2. Dynamic provisioning of service instances: enables apps teams to create service instances that adhere to company policy. Apps teams can create instances on-demand as needed.
3. Seamless integration of almost any service, cloud-based or on-prem, into Tanzu Application Platform with minimal configuration overhead: provides a near-limitless range of services to help boost developer productivity.

Getting started

If this is your first time working with services on Tanzu Application Platform, you might want to start with Claim services on Tanzu Application Platform and Consume services on Tanzu Application Platform in the getting started guide. These guides run through the basics, after which you can return here to the Services Toolkit component documentation to continue your services journey on Tanzu Application Platform.

How this documentation is organized

The Services Toolkit component documentation consists of the following sections that relate to what you are want to achieve:

- Concepts: To gain a deeper understanding of Services Toolkit.
- Tutorials: To learn through following examples.
- How-to guides: To find a set of steps to solve a specific problem.
- Reference material: To find specific information such as Services Toolkit's APIs, the Tanzu Service CLI plug-in, and troubleshooting information.

Tutorials and concepts are of most relevance when studying, while how-to guides and reference material are of most use while working.

The following is a selection of useful topics on offer:

For apps teams:

- Tutorial: Working with Bitnami Services

For ops teams:

- Tutorial: Setup Dynamic Provisioning of Service Instances
- Tutorial: Integrating Cloud Services (AWS, Azure, GCP, etc.) into Tanzu Application Platform
- How-to guide: Configure Dynamic Provisioning of AWS RDS Service Instances

For everyone:

- Concept: Four Levels of Service Consumption in Tanzu Application Platform

Install Services Toolkit

This topic tells you how to install Services Toolkit from the Tanzu Application Platform (commonly known as TAP) package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Services Toolkit. For more information about profiles, see Components and installation profiles.

Prerequisites

Before installing Services Toolkit:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- Install cert-manager. For more information, see Install cert-manager.

Install Services Toolkit

To install Services Toolkit:

1. See what versions of Services Toolkit are available to install by running:

   ```
   tanzu package available list -n tap-install services-toolkit.tanzu.vmware.com
   ```

   For example:

   ```
   $ tanzu package available list -n tap-install services-toolkit.tanzu.vmware.com
   - Retrieving package versions for services-toolkit.tanzu.vmware.com...
     NAME                              VERSION           RELEASED-AT
     services-toolkit.tanzu.vmware.com  0.9.0             2022-09-08T00:00:00Z
   ```

2. Install Services Toolkit by running:

   ```
   tanzu package install services-toolkit \
     --package services-toolkit.tanzu.vmware.com \
     --version VERSION-NUMBER \
     --namespace tap-install
   ```

   Where `VERSION-NUMBER` is the Services Toolkit version you want to install. For example, `0.9.0`.

3. Verify that the package installed by running:

   ```
   tanzu package installed get services-toolkit -n tap-install
   ```

   In the output, confirm that the `STATUS` value is `Reconcile succeeded`.

   For example:

   ```
   $ tanzu package installed get services-toolkit -n tap-install
   | Retrieving installation details for services-toolkit...
   NAME:                    services-toolkit
   PACKAGE-NAME:            services-toolkit.tanzu.vmware.com
   PACKAGE-VERSION:         0.9.0
   STATUS:                  Reconcile succeeded
   CONDITIONS:              [{ReconcileSucceeded True  }]
   USEFUL-ERROR-MESSAGE:
   ```

Services Toolkit concepts

This section introduces you to Services Toolkit concepts.

In this section:

- The four levels of service consumption in Tanzu Application Platform
- Class claims vs resource claims

The four levels of service consumption in Tanzu Application Platform

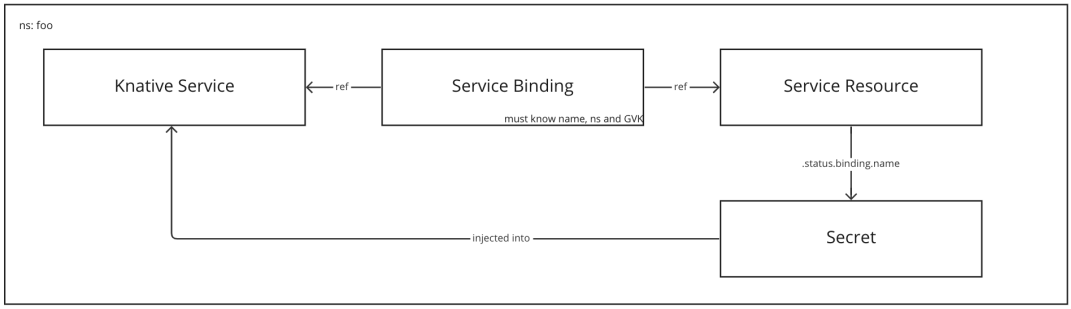
This topic describes the different levels of abstraction when using Services Toolkit and explains when and why you might choose to use one level over the other.

As Tanzu Application Platform has evolved, so has the way to offer and consume services on the platform. In this topic, the progress of this evolution is charted in terms of four levels of service consumption.

The introduction of a higher level does not automatically mean that all lower levels are made obsolete. In most cases, the higher levels build upon the foundations laid by the lower levels, and represent an abstraction that is higher-level and more opinionated.

Level 1 - direct bindings

Tanzu Application Platform v1.0 included support for service bindings, which back the level 1 concept of direct bindings.



For example, if you deploy an application workload to Tanzu Application Platform, this results in a Knative service in a namespace. An API resource in the same namespace that represents a service, such as a database or a cache, is called a service resource.

Using a service binding you can bind that service resource with the Knative service. This injects the credentials for the service resource into the Knative service, so that the application workload can consume it.

In this relatively straightforward scenario, there are only a few resources involved and they directly reference each other. In fact, users are not even directly exposed to the service binding. The service binding is created automatically as part of the Out of the Box Supply Chains whenever an application workload is configured to refer to a service.

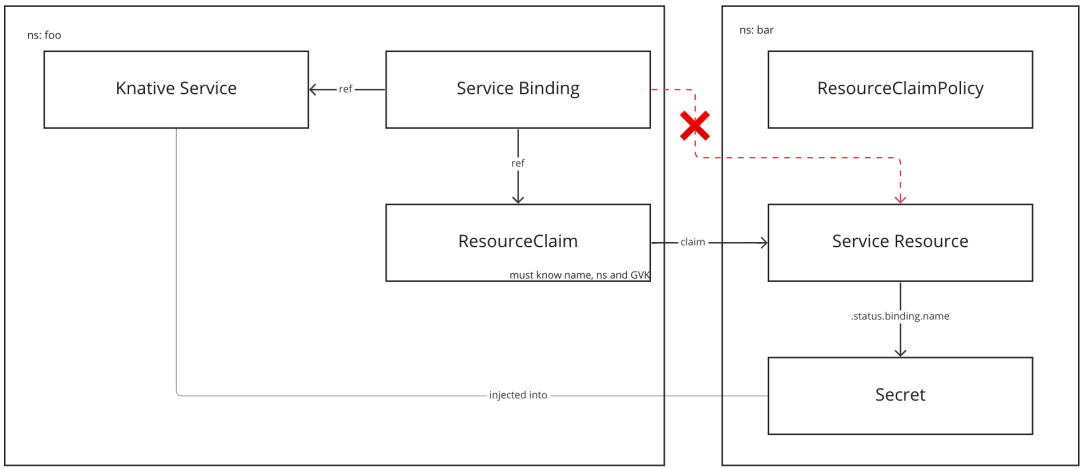
However, there are a number of limitations with this setup. The first is that the service resource must be bindable and which means it must adhere to the provisioned services definition in the service binding specification for Kubernetes. For more information, see Provisioned Service. There are some resources that adhere to this specification, primarily resources offered by VMware Tanzu's data services, but the overwhelming majority of resources don't.

The second limitation is that all resources have to be in the same namespace.

The third limitation is that the service binding must have detailed and specific information about the service resource, including its name, namespace, and API group, version, and kind. This is not a clear separation of concerns as it introduces tight coupling between app teams, who create the application workloads, and ops teams, who create the service resources.

Level 2 - resource claims

Level 2 addresses some limitations of direct bindings through the Services Toolkit feature resource claims. This feature coincided with the release of Tanzu Application Platform v1.0.



For example, if you have the same setup as in Level 1 - direct bindings, but you want the service resource to be in a separate namespace from the Knative service you cannot use direct bindings. This is because the schema for ServiceBinding provides no option to configure a `namespace` for the service.

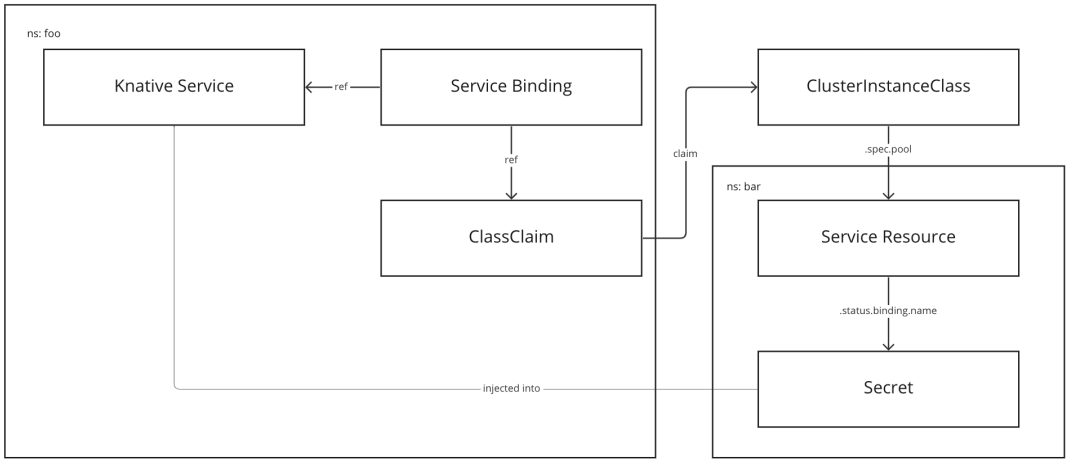
Resource claims allow you to claim a bindable service resource that exists in another namespace, and to then bind the application workload to the resource claim instead of to the service resource directly. Because you are now crossing namespace tenancy boundaries, you are only permitted to claim the service resource if you create a corresponding resource claim policy.

The advantage of level 2 over level 1 is that now the application workload and the service resource do not have to exist in the same namespace. This helps to promote a better separation of concerns. It is now possible for apps teams and ops teams to manage the life cycles of apps and services independently.

However, it's still not an ideal solution, and some limitations from level 1 still exist in level 2. The service resource still must be bindable, and apps teams still must know the name, namespace, and API group, version, and kind of the service resource. In addition, ops teams must ensure that the service resources exist. These resources must be manually provisioned and permitted to be claimed through policy, otherwise resource claims created by the apps teams remain in a pending state indefinitely.

Level 3 - class claims and pool-based classes

Level 3 introduces class claims and pool-based classes. Originally released with Tanzu Application Platform v1.4, class claims and pool-based classes help to alleviate the issue of apps teams having to know detailed information about service resources.



Level 3 builds on the example in level 2, which has an application workload in a namespace `foo` and a service resource in a namespace `bar`. Rather than relying on resource claim and resource claim policy, level 3 introduces a class claim and a pool-based class.

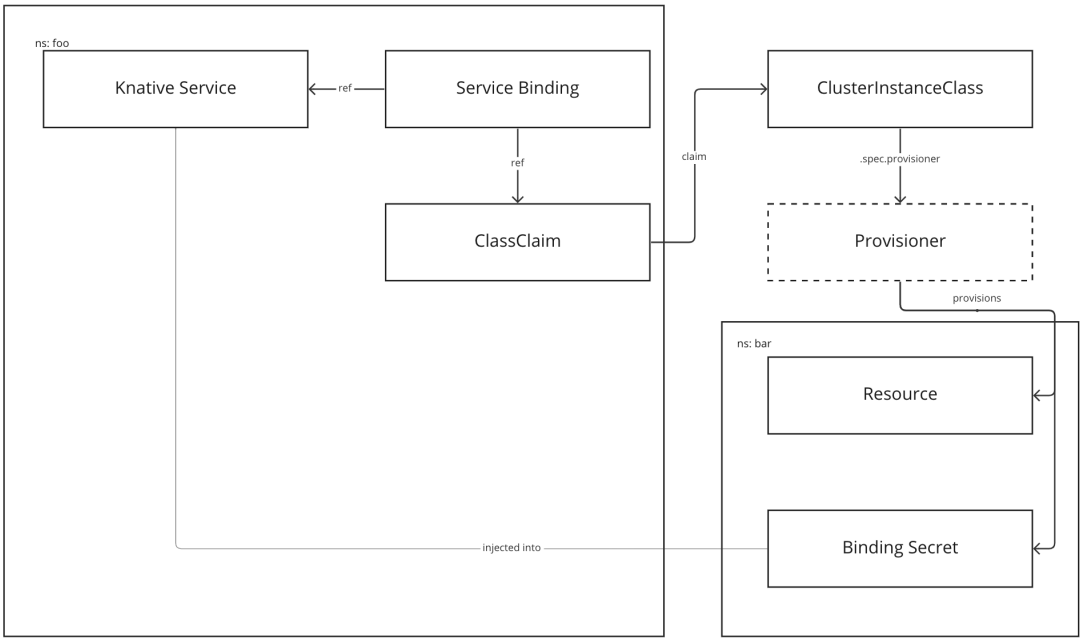
The pool-based class can pool service resources using label or field selectors from across all namespaces on the cluster. Ops teams create the service resources and then create a class to gather them all together into one logical group that apps teams can discover and claim from.

Apps teams can discover the available classes using the `tanzu service class list` command. Rather than creating a resource claim, they instead create a higher-level abstraction - a class claim. The class claim refers to the name of a class. You only need to create a class claim referring to a class and then bind your application workload to the class claim. There is no longer a need to provide detailed information such as the API group, version, and kind for the service resource behind the class.

Level 3 is much simpler for apps teams to consume services and the separation of concerns is much neater. A few limitations still remain. Service resources must still be bindable and ops teams still must manually provision the service resources to fill the pool.

Level 4 - class claims and provisioner-based classes (aka “Dynamic Provisioning”)

Level 4 is the current highest level of service consumption in Tanzu Application Platform. Released in Tanzu Application Platform v1.5, it introduces provisioner-based classes, which, together with class claims, power Tanzu Application Platform’s dynamic provisioning capability.



Level 4 builds on the example in level 3, but the class now defines a provisioner rather than a pool. Services Toolkit in Tanzu Application Platform v1.5 supports one provisioner type - Crossplane. Support for new provisioners might be added in the future.

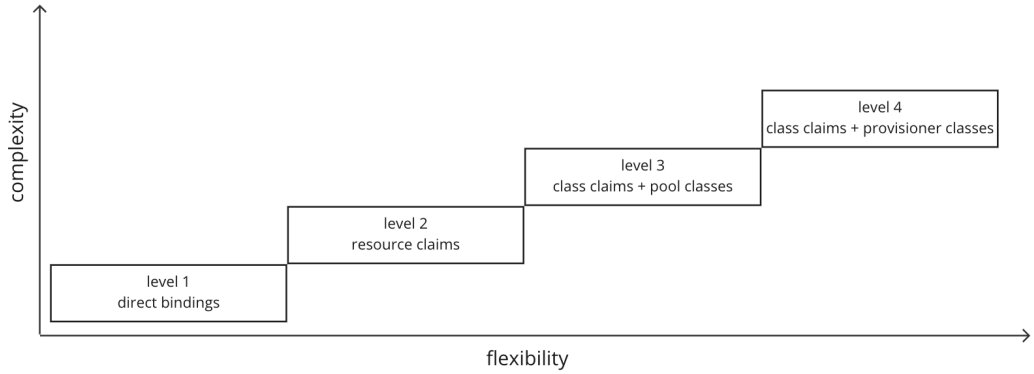
When you create a class claim that refers to a provisioner-based class, the Services Toolkit controller requests the provisioner to provision the resources necessary to create a service instance that can then be bound to application workloads. In this example, the provisioner creates a namespace, a service resource, and a secret that conforms to the binding specification. Then, that secret is wired all the way back through to the application workload.

Two big advantages are realized at level 4. Firstly, ops teams no longer need to manually provision service resources. They are now created on-demand as and when needed. This not only helps to remove unnecessary burden from ops teams, but also helps to provide better use of resources because service resources no longer need to remain in a pool waiting to be claimed. The second advantage is that service resources no longer need to be bindable. The provisioner can act almost like an adapter to bring pretty much any service you can think of into Tanzu Application Platform. The only requirement is that the provisioner create a binding-conforming secret that holds credentials for the provisioned service resources. You can configure this once during the dynamic provisioning setup.

While level 4 is very powerful and seemingly solves the problems mentioned so far, it's not entirely without its drawbacks. The main one being that all this flexibility comes at the cost of added complexity. There are many, many more moving parts involved at level 4 when compared to level 1. However, it might be a price worth paying to benefit from all that is on offer.

Summary

By now you have seen how each new level builds and improves upon the last. All levels are also valid use cases in their own right.



Tanzu Application Platform users can decide which level to operate at. Finding the level that is right for your situation depends on a number of factors, such as, the size of the organization you work for and the layout of apps teams and ops teams within the org.

If you work at a small startup in which there are no strict divides between apps teams and ops teams, level 1 might be suitable for your needs. However, if you work for a large organization with distinct and dedicated apps and ops teams, choosing one of the higher levels in which that separation is better catered to might make more sense. If you are not sure, it's probably best to start with level 4. Level 4 provides the ultimate services experience on Tanzu Application Platform, and as such will hopefully meet all your services needs.

Class claims compared to resource claims

There are two types of claim you can choose from when working with services on Tanzu Application Platform (commonly known as TAP). These are ClassClaim and ResourceClaim. This Services Toolkit topic explains the similarities and differences between the two and when using one is preferable over the other.

It is usually advisable to work with a ClassClaim where possible as they are easier to create and are more portable across multiple clusters. They are also used as the trigger mechanism for dynamic provisioning of service instances.

Similarities

- Both APIs express that you want to access to a service instance.
- Both APIs adhere to the ProvisionedService duck type. They both have the field `.status.binding.name` in their API. This means that you can target them using a ServiceBinding and, therefore, you can feed them into Cartographer's Workload API.
- Both APIs ensure that mutual exclusivity of claims on service instances. After using either a ClassClaim or a ResourceClaim to claim a service instance, no other ClassClaim or a ResourceClaim can claim that same service instance.

Using a ResourceClaim

A ResourceClaim targets a specific resource in the Kubernetes cluster. To target that resource, the ResourceClaim needs the name, namespace, kind, and API version of the resource.

The specificity of the ResourceClaim means it is most useful when you must guarantee which service instance the application workload uses. For example, if the application must connect to the exact same database instance while it advances through development, test, and production

environments. If you do not need this guarantee VMware recommends that you use the ClassClaim API instead.

Using a ClassClaim

A ClassClaim targets a ClusterInstanceClass in the Kubernetes cluster. To target this class, the ClassClaim only requires the name of the ClusterInstanceClass.

The ClusterInstanceClass can represent any set of service instances and therefore each time you create a new ClassClaim, you can claim any of the service instances represented by that ClusterInstanceClass. After a ClassClaim has claimed a service instance, it never looks for another. This is true even if the ClassClaim's `spec` is updated, or the ClusterInstanceClass is updated. Therefore, the ClassClaim is performing a **point-in-time** lookup at its creation, using the ClusterInstanceClass for that lookup.

The loose coupling between the ClassClaim and the service instances means that a ClassClaim is best in situations where:

- You must inject different service instances into the application workload at different points in its advancement from development to production environments. For more information, see Abstracting Service Implementations Behind A Class Across Clusters.
- The ClassClaim, and also any workload referencing it, must be promoted from one environment to the next without changing their specification.

The ClassClaim is the only type of claim that you can use to dynamically provision service instances.

Tutorials

In this section:

For apps teams:

- Working with Bitnami Services

For ops teams:

- Set up dynamic provisioning of service instances
- Integrating cloud services into Tanzu Application Platform
- Using direct secret references
- Abstracting service implementations behind a class across clusters

Set up dynamic provisioning of service instances

In this Services Toolkit tutorial you learn how service operators can set up a new, self-serve, and customized service for Tanzu Application Platform (commonly known as TAP). The example uses VMware RabbitMQ for Kubernetes, but the steps and learnings can apply to almost any other service.

About this tutorial

Target user role: Service Operator
Complexity: Advanced
Estimated time: 60 minutes
Topics covered: Dynamic Provisioning, Crossplane, VMware RabbitMQ for Kubernetes operator
Learning outcomes: Ability to offer new, on-demand, and customized services in your Tanzu Application Platform clusters

Prerequisites

- Access to a Tanzu Application Platform cluster v1.5.0 or later.
- Basic familiarity with Crossplane, particularly the concepts of Composition and CompositeResourceDefinitions.

Scenario

The tutorial is centered around the following hypothetical, but somewhat realistic, real-world scenario.

You work at BigCorp and are tasked to provide an on-demand, self-serve RabbitMQ service for BigCorp's development teams who are working with Tanzu Application Platform. You have already reviewed the RabbitMQ offering that is available with Bitnami Services, but have discovered that while it is an excellent service for testing and for quickly getting started, it is not quite suitable for BigCorp's stringent and specific needs.

In particular, you must comply with BigCorp's auditing and logging policy, and want to enforce that every RabbitMQ cluster in use on the platform adheres to that policy. At the same time, you don't want to be a blocker for the application teams and want to offer them self-serve access to RabbitMQ whenever they need it, without incurring any untoward delays. You have heard great things about Tanzu Application Platform's dynamic provisioning capability, and are now looking to make use of it to help you complete your task.

In this tutorial you will learn how to:

- Install the RabbitMQ Cluster Kubernetes operator
- Create a `CompositeResourceDefinition`
- Create a `Composition`
- Create a provisioner-based class
- Understand and create the necessary RBAC permissions
- Create a claim for the class to test it all out
- Understand how all the pieces fit together to power the dynamic provisioning capability in Tanzu Application Platform

Concepts

The following diagram provides an overview of the elements of dynamic provisioning and how they fit together.

The following is a high-level overview of how the system works:

1. The service operator creates a `CompositeResourceDefinition` and a `Composition`, which together define the configuration of the service instances that will be dynamically provisioned.
2. The service operator creates a class pointing to the `CompositeResourceDefinition`. This informs application development teams that the service is available.
3. The service operator applies necessary Role-Based Access Control (RBAC) to permit the system to create the necessary resources, and to authorize application development teams to create claims for the class.
4. The application developer creates a claim referring to the class, optionally passing through parameters to override any default configuration where permissible.
5. The system creates a `CompositeResource`, merging information provided in the claim with default configuration specified by the system and configuration defined in the `Composition`.
6. Crossplane reconciles the `CompositeResource` into a service instance and writes credentials for the instance into a `Secret`.
7. The `Secret` is written back to the application developer's namespace, so that application workloads can use it.

As you follow this tutorial, it will address the parts of this diagram in more detail.

Procedure

The following steps show how to configure dynamic provisioning for a service.

Step 1: Install the operator

When adding any new service to Tanzu Application Platform, ensure that there are a suitable set of APIs available in the cluster from which to construct the service instances. Usually, this involves installing one or more Kubernetes Operators into the cluster.

Given the aim of this tutorial is to set up a new RabbitMQ service, install the RabbitMQ Cluster Operator for Kubernetes.

Note

> The steps in this tutorial use the open source version of the operator. For most real-world deployments, VMware recommends using the official, supported version provided by VMware. For more information, see VMware RabbitMQ for Kubernetes.

Use `kapp` to install the operator by running:

```
kapp -y deploy --app rmq-operator --file https://github.com/rabbitmq/cluster-operator/releases/latest/download/cluster-operator.yml
```

This causes a new API Group/Version of `rabbitmq.com/v1beta1` and Kind named `RabbitmqCluster` to become available in the cluster. You can now use this API to create RabbitMQ cluster instances as part of the dynamic provisioning setup.

Step 2: Creating a `CompositeResourceDefinition`

Tanzu Application Platform's dynamic provisioning capability relies on Crossplane. You can find the specific integration point at `.spec.provisioner.crossplane.compositeResourceDefinition` in Tanzu Application Platform's `ClusterInstanceClass` API.

As the name suggests, this field is looking for a `CompositeResourceDefinition`, which you create in this step of the procedure. The `CompositeResourceDefinition` (XRD) defines the shape of a new, custom API type that encompasses the specific set of requirements laid out by the scenario in this tutorial.

Create a file named `xrabbitmqclusters.messaging.bigcorp.org.xrd.yaml` and copy in the following contents.

```
# xrabbitmqclusters.messaging.bigcorp.org.xrd.yaml

---
apiVersion: apiextensions.crossplane.io/v1
kind: CompositeResourceDefinition
metadata:
  name: xrabbitmqclusters.messaging.bigcorp.org
spec:
  connectionSecretKeys:
  - host
  - password
  - port
  - provider
  - type
  - username
  group: messaging.bigcorp.org
  names:
    kind: XRabbitmqCluster
    plural: xrabbitmqclusters
  versions:
  - name: v1alpha1
    referenceable: true
    schema:
      openAPIV3Schema:
        properties:
          spec:
            description: The OpenAPIV3Schema of this Composite Resource Definition.
            properties:
              replicas:
                description: The desired number of replicas forming the cluster
                type: integer
              storageGB:
                description: The desired storage capacity of a single replica, in GB.
```

```
                type: integer
            type: object
        type: object
    served: true
```

Then use kubectl to apply the file to the Tanzu Application Platform cluster.

```
kubectl apply -f xrabbitmqclusters.messaging.bigcorp.org.xrd.yaml
```

For a detailed explanation of `CompositeResourceDefinition` see, the Crossplane documentation.

The following is a condensed explanation of the most relevant pieces of the `CompositeResourceDefinition` configuration, provided in this section, as it relates to dynamic provisioning in Tanzu Application Platform.

The example in this tutorial does **not** specify `.spec.claimNames` in the XRD. Tanzu Application Platform's dynamic provisioning capability makes use of Crossplane's cluster-scoped Composite Resources, rather than the namespace-scoped Claims ("Claims" here not to be confused with Tanzu Application Platform's own concept of claims). As such, this configuration is not required, although it does not cause any adverse effects if you add it.

Next, see the `.spec.connectionKeys` field. This field detects the keys that will exist in the `Secret` resulting from the dynamic provisioning request. You likely want this `Secret` to conform with the Service Binding Specification for Kubernetes, as this, in part, is what allows for automatic configuration of the service instance by Tanzu Application Platform's application workloads. This is assuming that the application is using a binding-aware library such as Spring Cloud Bindings. Specific key name requirements vary by service type, however all must provide the `type` key.

Finally, see the `.spec.properties` section in the schema for `v1alpha1`. This is where you, as the service operator, can set which configuration options you want to expose to application development teams. In the example in this section, there are two configuration options: `replicas` and `storageGB`. By adding these properties to the specification, you are handing over control of these specific configuration options to the development teams. For example, you might want to add `storageGB` if the development teams have more knowledge about how much storage their apps require than you do. By adding `storageGB` you can allow them to decide for themselves how much storage they require.

You can choose to add as many or as few configuration options here as you like. You can also choose to set default values. In highly regulated environments, you might not want to allow for any configuration by developers at all.

In the scenario at the beginning of this tutorial, it says that you must comply with the auditing and logging policy. You do not specify any configuration related to auditing or logging in the XRD in this step. This is intentional as in this scenario there are strict auditing and logging requirements and cannot permit developers to override those. In the next step you learn how to ensure that those requirements get enforced on the resulting RabbitMQ clusters.

To verify the status of the XRD you created, run:

```
kubectl get xrds
```

If successful, the `xrabbitmqclusters.messaging.bigcorp.org` is listed with `ESTABLISHED=True`.

You might see some other XRDs listed as well. These are the `*.bitnami.*.tanzu.vmware.com` XRDs. These are part of the `bitnami.services.tanzu.vmware.com` package with Tanzu Application Platform and serve as the basis of the Bitnami Services. You can ignore these other XRDs for now, but if you want to see how they are used in practice, see Claim services on Tanzu Application Platform and Consume services on Tanzu Application Platform in the Tanzu Application Platform getting started guide.

As a result of creating the XRD, a new API Group/Version of `messaging.bigcorp.org/v1alpha1` and Kind named `XRabbitmqCluster` become available in the cluster. If you inspect this API further, notice that the `replicas` and `storageGB` properties configured in the XRD are present in the specification of `XRabbitmqCluster`.

```
kubectl explain --api-version=messaging.bigcorp.org/v1alpha1 xrabbitmqclusters.spec
```

You will also notice that Crossplane has injected some other fields into the specification as well, but you can mostly ignore these for now.

Step 3: Creating a Crossplane `Composition`

You do most of the configuration for dynamic provisioning during the creation of the `Composition`.

For a more detailed explanation about the `Composition`, see the Crossplane documentation.

The following are the basics you must know to start to create a `Composition` for use in Tanzu Application Platform.

Create a file named `xrabbitmqclusters.messaging.bigcorp.org.composition.yaml` and copy in the following contents.

```
# xrabbitmqclusters.messaging.bigcorp.org.composition.yaml

---
apiVersion: apiextensions.crossplane.io/v1
kind: Composition
metadata:
  name: xrabbitmqclusters.messaging.bigcorp.org
spec:
  compositeTypeRef:
    apiVersion: messaging.bigcorp.org/v1alpha1
    kind: XRabbitmqCluster
  resources:
  - base:
      apiVersion: kubernetes.crossplane.io/v1alpha2
      kind: Object
      spec:
        forProvider:
          manifest:
            apiVersion: rabbitmq.com/v1beta1
            kind: RabbitmqCluster
            metadata:
              namespace: rmq-clusters
            spec:
              terminationGracePeriodSeconds: 0
              replicas: 1
              persistence:
                storage: 1Gi
              resources:
                requests:
                  cpu: 200m
                  memory: 1Gi
                limits:
                  cpu: 300m
                  memory: 1Gi
              rabbitmq:
                envConfig: |
                  RABBITMQ_LOGS=""
                additionalConfig: |
                  log.console = true
                  log.console.level = debug
                  log.console.formatter = json
                  log.console.formatter.json.field_map = verbosity:v time msg domain f
```

```
ile line pid level:-
                  log.console.formatter.json.verbosity_map = debug:7 info:6 notice:5 w
arning:4 error:3 critical:2 alert:1 emergency:0
                  log.console.formatter.time_format = epoch_usecs
        connectionDetails:
        - apiVersion: v1
          kind: Secret
          namespace: rmq-clusters
          fieldPath: data.provider
          toConnectionSecretKey: provider
        - apiVersion: v1
          kind: Secret
          namespace: rmq-clusters
          fieldPath: data.type
          toConnectionSecretKey: type
        - apiVersion: v1
          kind: Secret
          namespace: rmq-clusters
          fieldPath: data.host
          toConnectionSecretKey: host
        - apiVersion: v1
          kind: Secret
          namespace: rmq-clusters
          fieldPath: data.port
          toConnectionSecretKey: port
        - apiVersion: v1
          kind: Secret
          namespace: rmq-clusters
          fieldPath: data.username
          toConnectionSecretKey: username
        - apiVersion: v1
          kind: Secret
          namespace: rmq-clusters
          fieldPath: data.password
          toConnectionSecretKey: password
        writeConnectionSecretToRef:
          namespace: rmq-clusters
    connectionDetails:
    - fromConnectionSecretKey: provider
    - fromConnectionSecretKey: type
    - fromConnectionSecretKey: host
    - fromConnectionSecretKey: port
    - fromConnectionSecretKey: username
    - fromConnectionSecretKey: password
    patches:
      - fromFieldPath: metadata.name
        toFieldPath: spec.forProvider.manifest.metadata.name
        type: FromCompositeFieldPath
      - fromFieldPath: spec.replicas
        toFieldPath: spec.forProvider.manifest.spec.replicas
        type: FromCompositeFieldPath
      - fromFieldPath: spec.storageGB
        toFieldPath: spec.forProvider.manifest.spec.persistence.storage
        transforms:
        - string:
            fmt: '%dGi'
            type: Format
          type: string
        type: FromCompositeFieldPath
      - fromFieldPath: metadata.name
        toFieldPath: spec.writeConnectionSecretToRef.name
        transforms:
        - string:
            fmt: '%s-rmq'
            type: Format
          type: string
```

```
        type: FromCompositeFieldPath
      - fromFieldPath: metadata.name
        toFieldPath: spec.connectionDetails[0].name
        transforms:
        - string:
            fmt: '%s-default-user'
            type: Format
          type: string
        type: FromCompositeFieldPath
      - fromFieldPath: metadata.name
        toFieldPath: spec.connectionDetails[1].name
        transforms:
        - string:
            fmt: '%s-default-user'
            type: Format
          type: string
        type: FromCompositeFieldPath
      - fromFieldPath: metadata.name
        toFieldPath: spec.connectionDetails[2].name
        transforms:
        - string:
            fmt: '%s-default-user'
            type: Format
          type: string
        type: FromCompositeFieldPath
      - fromFieldPath: metadata.name
        toFieldPath: spec.connectionDetails[3].name
        transforms:
        - string:
            fmt: '%s-default-user'
            type: Format
          type: string
        type: FromCompositeFieldPath
      - fromFieldPath: metadata.name
        toFieldPath: spec.connectionDetails[4].name
        transforms:
        - string:
            fmt: '%s-default-user'
            type: Format
          type: string
        type: FromCompositeFieldPath
      - fromFieldPath: metadata.name
        toFieldPath: spec.connectionDetails[5].name
        transforms:
        - string:
            fmt: '%s-default-user'
            type: Format
          type: string
        type: FromCompositeFieldPath
    readinessChecks:
      - type: MatchString
        fieldPath: status.atProvider.manifest.status.conditions[1].status # ClusterAva
ilable
        matchString: "True"
```

Use kubectl to apply the file to the Tanzu Application Platform cluster.

```
kubectl apply -f xrabbitmqclusters.messaging.bigcorp.org.composition.yaml
```

About `.spec.compositeTypeRef`

The `.spec.compositeTypeRef` is configured to refer to `XRabbitmqCluster` on the `messaging.bigcorp.org/v1alpha1` API group and version.

```
...
spec:
  compositeTypeRef:
    apiVersion: messaging.bigcorp.org/v1alpha1
    kind: XRabbitmqCluster
...
```

This is the API that was created when you applied the XRD in Step 2: Creating a `CompositeResourceDefinition`. By configuring `.spec.compositeTypeRef` to refer to this API, you are instructing Crossplane to use the configuration contained within this `Composition` to compose subsequent managed resources whenever it observes that a new `XRabbitmqCluster` resource is created in the cluster. Tanzu Application Platform's dynamic provisioning system creates the `XRabbitmqCluster` resources automatically. To visualize how these pieces fit together, see the diagram in the Concepts section.

About `.spec.resources`

The `.spec.resources` section is where you specify the managed resources to be created. Managed resources are tied to Crossplane's `Providers`, with each `Provider` defining a set of managed resources which can then be used in compositions. Tanzu Application Platform includes two `Providers` with the Crossplane package: provider-helm and provider-kubernetes. This makes a `Release` managed resource available, which is used to manage Helm releases, and makes an `Object` managed resource available, which used to manage arbitrary Kubernetes resources. You can install and use any other `Provider`. To find the latest providers, see the Upbound Marketplace. The more providers you install, the more managed resources you can choose from in your compositions.

The `Object` managed resource

The overarching goal is to compose whatever resources are necessary to create functioning, usable service instances and to surface the credentials and connectivity information required to connect to those instances in a known and repeatable way. This tutorial uses the `RabbitmqCluster` resource, which presents one single API to use to create fully functioning RabbitMQ clusters, credentials for which get stored in `Secrets` in the cluster.

However, `RabbitmqCluster` is not a Crossplane managed resource so you cannot refer to this resource directly under `.spec.resources`. To work around this, use `provider-kubernetes` and its corresponding `Object` managed resource. `Object` enables you to wrap any arbitrary Kubernetes resource, such as `RabbitmqCluster`, into a Crossplane managed resource and to then use them like any other managed resource inside `Compositions`.

```
...
spec:
  resources:
  - base:
      apiVersion: kubernetes.crossplane.io/v1alpha2
      kind: Object
      spec:
        forProvider:
          manifest:
            apiVersion: rabbitmq.com/v1beta1
            kind: RabbitmqCluster
            metadata:
              namespace: rmq-clusters
            spec:
              terminationGracePeriodSeconds: 0
              replicas: 1
              persistence:
                storage: 1Gi
```

```
              resources:
                requests:
                  cpu: 200m
                  memory: 1Gi
                limits:
                  cpu: 300m
                  memory: 1Gi
              rabbitmq:
                envConfig: |
                  RABBITMQ_LOGS=""
                additionalConfig: |
                  log.console = true
                  log.console.level = debug
                  log.console.formatter = json
                  log.console.formatter.json.field_map = verbosity:v time msg domain f
ile line pid level:-
                  log.console.formatter.json.verbosity_map = debug:7 info:6 notice:5 w
arning:4 error:3 critical:2 alert:1 emergency:0
                  log.console.formatter.time_format = epoch_usecs
...
```

The `Object` managed resource is where you configure `RabbitmqCluster` resources. This is the place in which you can now fine-tune the configuration of the RabbitMQ Clusters to your needs.

Recall from the hypothetical scenario that you are particularly concerned about your company's logging policy. The configuration in the `Object` translates that hypothetical policy into default configuration on the `RabbitmqCluster` resource by specifying `.spec.rabbitmq.additionalConfig` for the resource. This was taken from one of the examples in the RabbitMQ Cluster Operator GitHub repository. You can configure the resource however you want and to whatever requirements necessary.

The `patches` section

The `Object` also sets default values for the number of replicas and the amount of persistent storage for new `RabbitmqClusters` to one replica and 1 Gi. However, you want to allow these two values to be configurable by the application development teams as specified in Step 2: Creating a `CompositeResourceDefinition`. You can configure this using patches.

```
...
patches:
  - fromFieldPath: metadata.name
    toFieldPath: spec.forProvider.manifest.metadata.name
    type: FromCompositeFieldPath
...
```

The first thing to note is that all the patches are of type `FromCompositeFieldPath`, which allows you to take values defined on the composite resource (`XRabbitmqCluster` in this case) and to pass them through to the underlying managed resource (an `Object` wrapping `RabbitmqCluster` in this case). The first patch sets the name of the `RabbitmqCluster` to the same name as the name of the composite resource `XRabbitmqCluster`, which were created using `generateName`, thereby ensuring a unique name for each dynamically provisioned `RabbitmqCluster` instance.

```
...
patches:
...
  - fromFieldPath: spec.replicas
    toFieldPath: spec.forProvider.manifest.spec.replicas
    type: FromCompositeFieldPath
  - fromFieldPath: spec.storageGB
    toFieldPath: spec.forProvider.manifest.spec.persistence.storage
    transforms:
```

```
    - string:
        fmt: '%dGi'
        type: Format
      type: string
    type: FromCompositeFieldPath
...
```

The second and third patches pass through configuration for the number of replicas and amount of persistent storage, which overrides the default values already configured.

The remaining patches all do the same thing, which is to patch in the name of the `Secret` for the fields in the `connectionDetails` section.

```
...
- fromFieldPath: metadata.name
  toFieldPath: spec.connectionDetails[0].name
  transforms:
  - string:
      fmt: '%s-default-user'
      type: Format
    type: string
  type: FromCompositeFieldPath
...
```

When creating a `RabbitmqCluster` resource using the RabbitMQ Cluster Kubernetes operator, the operator creates a `Secret` containing credentials and connectivity information used to connect to the cluster. That `Secret` is named `x-default-user`, where `x` is the name of the `RabbitmqCluster` resource. Because the name of the `RabbitmqCluster` cannot be known upfront, you must use patches to ensure that the `connectionDetails` section refers to the correctly-named `Secret`.

The `connectionDetails` sections are where you configure which keys and values to expose in the resulting `Secret`. You must specify the same set of keys as defined in the original XRD.

The `readinessChecks` section

Configuring readiness checks helps to keep consumers of dynamic provisioning, that is, the application teams, informed about when the resulting service instances are ready for application workloads to use.

```
...
readinessChecks:
  - type: MatchString
    fieldPath: status.atProvider.manifest.status.conditions[1].status # ClusterAvailab
le
    matchString: "True"
```

Where possible it is simplest to use the `Ready` condition to verify readiness. However, the `RabbitmqCluster` API doesn't expose a simple `Ready` condition, so you must configure the ready check on `ClusterAvailable` instead.

Check the namespace

One final important decision is the name of the namespace in which to create the dynamically provisioned `RabbitmqCluster` resources. This tutorial uses the `rmq-clusters` namespace.

```
...
spec:
  resources:
  - base:
      apiVersion: kubernetes.crossplane.io/v1alpha2
```

```
      kind: Object
      spec:
        forProvider:
          manifest:
            apiVersion: rabbitmq.com/v1beta1
            kind: RabbitmqCluster
            metadata:
              namespace: rmq-clusters
...
```

To make sure that the `rmq-clusters` namespace exists.

```
kubectl create namespace rmq-clusters
```

This configuration says that all dynamically provisioned `RabbitmqCluster` resources must be placed in the same `rmq-clusters` namespace. If you want to place each new cluster into a separate namespace, you must create an additional `Object` managed resource to wrap the creation of a `Namespace` and to apply patches to the resources accordingly. For this tutorial you only require one namespace.

Step 4: Creating a provisioner-based class

The creation of the XRD and the Composition brings to an end the Crossplane-centric part of this tutorial. What remains is to integrate all that you configured into Tanzu Application Platform's classes and claims model so that application teams can more easily make use of it. The first step here is to create a provisioner-based class and to point it at the XRD you created.

Create a file named `bigcorp-rabbitmq.class.yaml` and copy in the following contents.

```
# bigcorp-rabbitmq.class.yaml

---
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ClusterInstanceClass
metadata:
  name: bigcorp-rabbitmq
spec:
  description:
    short: On-demand RabbitMQ clusters precision engineered to meet the needs of BigCo
rp!
  provisioner:
    crossplane:
      compositeResourceDefinition: xrabbitmqclusters.messaging.bigcorp.org
```

Then use kubectl to apply the file to the Tanzu Application Platform cluster.

```
kubectl apply -f bigcorp-rabbitmq.class.yaml
```

This is referred to as a provisioner-based class due to the configuration of `.spec.provisioner`. For more information, see ClusterInstanceClass.

By creating this class you are informing application teams that the service is available. Application teams can discover it by using the `tanzu service class list` command. They can also use `tanzu service class get bigcorp-rabbitmq`, which provides detailed information about the class, including details of the `replicas` and `storageGB` parameters that you configured earlier.

Step 5: Configure supporting RBAC

There are two parts of RBAC to consider when you set up a new service for dynamic provisioning in Tanzu Application Platform. The first relates to granting permissions to the providers used in the compositions. The `Composition` created earlier uses `Object` managed resources ultimately to create

`RabbitmqCluster` resources. Therefore, you must grant `provider-kubernetes` permission to create `RabbitmqCluster` resources. You can do this by using an aggregating `ClusterRole` as follows.

Create a file named `provider-kubernetes-rmqcluster-read-writer.rbac.yaml` and copy in the following contents.

```
# provider-kubernetes-rmqcluster-read-writer.rbac.yaml

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: rmqcluster-read-writer
  labels:
    services.tanzu.vmware.com/aggregate-to-provider-kubernetes: "true"
rules:
- apiGroups:
  - rabbitmq.com
  resources:
  - rabbitmqclusters
  verbs:
  - "*"
```

Then use kubectl to apply the file to the Tanzu Application Platform cluster.

```
kubectl apply -f provider-kubernetes-rmqcluster-read-writer.rbac.yaml
```

While not necessary here, a corresponding label `services.tanzu.vmware.com/aggregate-to-provider-helm: "true"` exists for aggregating RBAC permissions to `provider-helm` as well.

The second element of RBAC detects who is authorized to use the new service. This is an important piece of configuration. You are configuring an on-demand service and making it available to application teams. Without any other supporting policy in place, application teams can create as many `RabbitmqClusters` as they like. This is of course the whole point of an on-demand service, but you must be conscious of resource use, and might want to control who can create new service instances on-demand.

You can grant authorization by using standard Kubernetes RBAC resources. Dynamic provisioning uses a custom RBAC verb, `claim`, which you can apply to classes to permit claiming from classes.

Create a file named `app-operator-claim-class-bigcorp-rabbitmq.rbac.yaml` and copy in the following contents.

```
# app-operator-claim-class-bigcorp-rabbitmq.rbac.yaml

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: app-operator-claim-class-bigcorp-rabbitmq
  labels:
    apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access: "true"
rules:
- apiGroups:
  - services.apps.tanzu.vmware.com
  resources:
  - clusterinstanceclasses
  resourceNames:
  - bigcorp-rabbitmq
  verbs:
  - claim
```

Then use kubectl to apply the file to the Tanzu Application Platform cluster.

```
kubectl apply -f app-operator-claim-class-bigcorp-rabbitmq.rbac.yaml
```

This `ClusterRole` grants anyone holding the `app-operator` Tanzu Application Platform user role the ability to claim from the `bigcorp-rabbitmq` class.

Step 6: Verify your configuration

To test your configuration, create a claim for the class and thereby trigger the dynamic provisioning of a new RabbitMQ cluster. This step is typically performed by the application operator, rather than the service operator, but it is important that you to confirm that everything is configured correctly.

Create a file named `bigcorp-rmq-1.claim.yaml` and copy in the following contents.

```
# bigcorp-rmq-1.claim.yaml

---
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ClassClaim
metadata:
  name: bigcorp-rmq-1
spec:
  classRef:
    name: bigcorp-rabbitmq
  parameters:
    storageGB: 2
    replicas: 3
```

Then use kubectl to apply the file to the Tanzu Application Platform cluster.

```
kubectl apply -f bigcorp-rmq-1.claim.yaml
```

After the RabbitMQ service is provisioned, the claim status reports `Ready=True`.

```
kubectl get classclaim bigcorp-rmq-1
```

Working with Bitnami Services

For the tutorial about working with Bitnami Services, see Working with Bitnami Services.

Integrating cloud services into Tanzu Application Platform

In this Services Toolkit tutorial you learn how service operators can integrate the cloud services of their choice into Tanzu Application Platform (commonly known as TAP).

There are a multitude of cloud-based services available on the market for consumers today. AWS, Azure, and GCP all provide support for a wide range of fully-managed, performant and on-demand services ranging from databases, to message queues, to storage solutions and beyond. In this tutorial you will learn how to integrate any one of these services into Tanzu Application Platform, so that you can offer it for apps teams to consume in a simple and effective way.

This tutorial is written at a slightly higher level than the other tutorials in this documentation. This is because it is not feasible to write detailed, step-by-step documentation for integrating every cloud-based service into Tanzu Application Platform. Each service brings a different set of considerations and concerns.

Instead, this tutorial guides you through the general approach to integrating cloud-based services into Tanzu Application Platform. While specific configurations change between services, the overall process remains the same through a consistent set of steps. The aim is to give you enough

understanding so that you can integrate any cloud-based service you want into Tanzu Application Platform.

For a more specific and low-level procedure, see Configure dynamic provisioning of AWS RDS service instances, which provides each step in detail for AWS RDS integration. It might be useful to read through that guide even if you want to integrate with one of the other cloud providers.

In addition, for Tanzu Application Platform v1.7 or later, you can instead use the AWS Services package, which provides a more streamlined approach for integrating services from AWS into Tanzu Application Platform.

About this tutorial

Target user role: Service Operator
Complexity: Advanced
Estimated time: 30 minutes
Topics covered: Dynamic Provisioning, Cloud-based Services, AWS, Azure, GCP, Crossplane
Learning outcomes: An understanding of the steps involved in integrating cloud-based services into Tanzu Application Platform

Concepts

The following is a high-level workflow outlining what is required to integrate a cloud-based service into Tanzu Application Platform.

1. **Install Provider and create ProviderConfig:**
    - Follow the official Upbound documentation to install the Provider and create a ProviderConfig.
2. **Create CompositeResourceDefinition:**
    - Create a CompositeResourceDefinition to define the shape of a new API type representing the service.
    - Choose which (if any) configuration parameters to expose to apps teams.
3. **Create Composition:**
    - Create a Composition using managed resources supplied by the Provider.
    - You can compose as many or as few managed resources as required to generate a service instance that application workloads can connect to and use over the network.
    - (Optional but recommended) Configure the connection secret to adhere to the Service Binding Specification for Kubernetes.
4. **Create provisioner-based ClusterInstanceClass:**
    - Create a provisioner-based ClusterInstanceClass pointing to the CompositeResourceDefinition created earlier.
5. **Create required RBAC:**
    - Create RBAC using the `claim` verb pointing to the provisioner-based ClusterInstanceClass to permit claiming from the class.
6. **Create ClassClaim:**
    - Create a ClassClaim pointing to the provisioner-based ClusterInstanceClass to begin a dynamic provisioning request.
    - Wait for the ClassClaim to report `READY=True`.

Procedure

This tutorial provides the steps required to integrate cloud services, and includes tips and references to example configurations where appropriate.

Step 1: Install a Provider

Install a suitable Crossplane `Provider` for your cloud of choice. Upbound provides support for the three main cloud providers:

- provider-aws
- provider-azure
- provider-gcp

Note

These cloud-based Providers often install many hundreds of additional CRDs onto the cluster, which can have a negative impact on cluster performance. For more information, see Cluster performance degradation due to large number of CRDs.

Choose the Provider you want, and then follow Upbound's official documentation to install the `Provider` and to create a corresponding `ProviderConfig`.

Important

The official documentation for the `Provider` includes a step to "Install Universal Crossplane". You can skip this step because Crossplane is already installed as part of Tanzu Application Platform.

The documentation also assumes Crossplane is installed in the `upbound-system` namespace. However, when working with Crossplane on Tanzu Application Platform, it is installed to the `crossplane-system` namespace by default. Ensure that you use the correct namespace when you create the `Secret` and the `ProviderConfig` with credentials for the `Provider`.

Step 2: Create a CompositeResourceDefinition

Create a `CompositeResourceDefinition`, which defines the shape of a new API type which is used to create the cloud-based resources.

For help creating the `CompositeResourceDefinition`, see the Crossplane documentation, or see Create a CompositeResourceDefinition in *Configure dynamic provisioning of AWS RDS service instances*.

Step 3: Create a Composition

This step is likely to be the most time-consuming. The `Composition` is where you define the configuration for the resources that make up the service instances for app teams to claim. Configure the necessary resources for usable service instances that users can connect to and use over the network.

To get started with creating a `Composition`, first read through Configuring Composition in the Upbound documentation.

You can also see the following `Composition` examples:

- For AWS RDS, see Define composite resource types (AWS).
- For Azure Flexible Server, see Define Composite Resource Types (Azure).
- For GCP Cloud SQL, see Define Composite Resource Types (GCP).

Step 4: Create a provisioner-based ClusterInstanceClass

Create a provisioner-based `ClusterInstanceClass` which is configured to refer to the `CompositeResourceDefinition` created earlier. For example:

```
---
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ClusterInstanceClass
metadata:
  name: cloud-service-foo
spec:
  description:
    short: FooDB by cloud provider Foo!
  provisioner:
    crossplane:
      compositeResourceDefinition: NAME-OF-THE-COMPOSITE-RESOURCE-DEFINITION
```

For a real-world example, see Make the service discoverable in *Configure dynamic provisioning of AWS RDS service instances*.

Step 5: Configure RBAC

Create an Role-Based Access Control (RBAC) rule using the `claim` verb pointing to the `ClusterInstanceClass` you created. For example:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: app-operator-claim-foo-db
  labels:
    apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access: "true"
rules:
- apiGroups:
  - "services.apps.tanzu.vmware.com"
  resources:
  - clusterinstanceclasses
  resourceNames:
  - cloud-service-foo
  verbs:
  - claim
```

For a real-world example, see Configure RBAC in *Configure dynamic provisioning of AWS RDS service instances*.

Step 6: Verify your integration

To test your integration, create a `ClassClaim` that points to the `ClusterInstanceClass` you created. For example:

```
---
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ClassClaim
metadata:
  name: claim-1
spec:
  classRef:
    name: cloud-service-foo
```

```
  parameters:
    key: value
```

Verify that the `ClassClaim` eventually transitions into a `READY=True` state. If it doesn't, debug the `ClassClaim` using kubectl. For how to do this, see Troubleshoot Services Toolkit.

Abstracting service implementations behind a class across clusters

In this Services Toolkit tutorial you learn how service operators can configure a class that allows for claims to resolve to different backing implementations of a service, such as PostgreSQL, depending on which cluster the class is claimed in.

This sort of setup allows the configurations of workloads and class claims to remain unchanged as they are promoted through environments, whilst also enabling service operators to change the implementations of the backing services without further configuration.

About this tutorial

Target user role: Service Operator
Complexity: Medium
Estimated time: 60 minutes
Topics covered: Classes, Claims, Claim-by-Class, Multi-Cluster
Learning outcomes: Ability to abstract the implementation (for example, helm, tanzu data service, cloud) of a service (for example, RabbitMQ) across multiple clusters

Prerequisites

- Access to three separate Tanzu Application Platform clusters v1.5.0 or later. This tutorial refers to them as `iterate`, `run-test`, and `run-production`, but you can use different names if required.

Scenario

The tutorial is centered around the following hypothetical, but somewhat realistic, real-world scenario.

You work at BigCorp as a service operator. BigCorp uses three separate Tanzu Application Platform clusters: `iterate`, `run-test`, and `run-production`. Application workloads begin on the `iterate` cluster, before being promoted to the `run-test` cluster, and then finally to the `run-production` cluster. The application development team have asked you for a PostgreSQL service they can use with their workloads, which must be available on all three clusters.

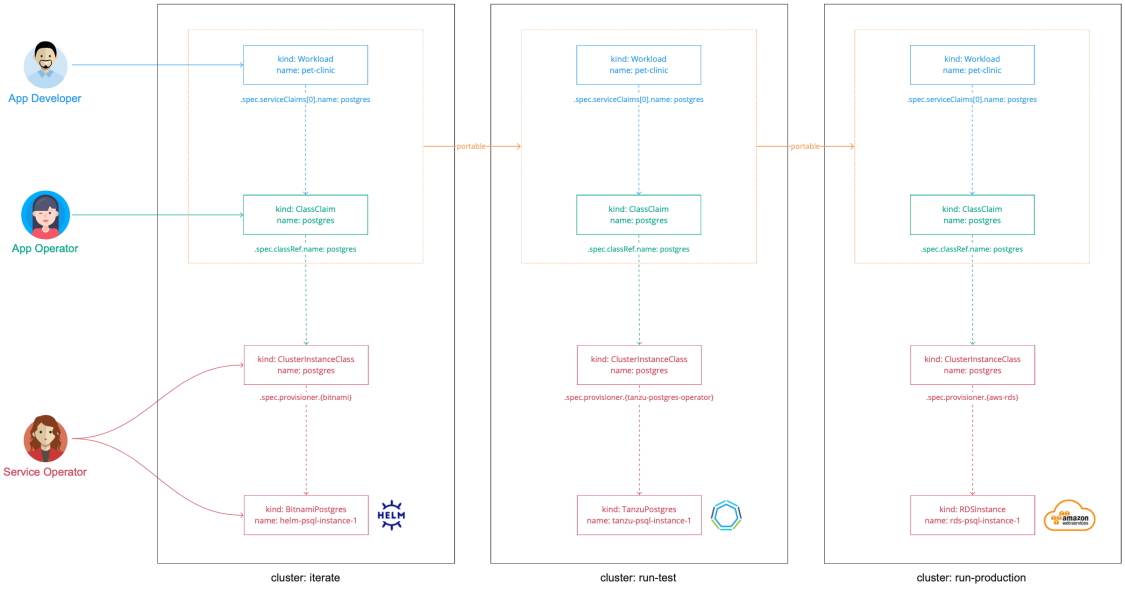
You are aware that the service level objectives (SLOs) for each cluster are different and want to tailor the implementation of the PostgreSQL service to each of the clusters accordingly. The `iterate` cluster has low level SLOs, so you want to offer an unmanaged PostgreSQL service backed by simple Helm chart. The `run-test` cluster has more robust requirements, so want to offer a PostgreSQL service backed by VMware SQL with Postgres for Kubernetes. The `run-production` cluster is critically important, so you want to use a fully managed, cloud-based PostgreSQL implementation there.

You want to ensure that the differing implementations are completely opaque to development teams. They do not need to know about the inner workings of the services, and must be able to keep their workloads and class claims the same as they are promoted across clusters. You have

heard great things about Tanzu Application Platform's claims and classes abstractions and want to make use of them to help you complete your task.

Concepts

This section provides a high-level overview of the elements you will use during this tutorial and how they all fit together.



In this diagram:

- There are three clusters: `iterate`, `run-test`, and `run-production`.
- In each cluster, the service operator creates a `ClusterInstanceClass` called postgres.
  - In the `iterate` cluster, this is a provisioner-based class that uses Bitnami Services to provision Helm instances of PostgreSQL.
  - In the `run-test` cluster, this is a provisioner-based class that uses VMware SQL with Postgres for Kubernetes to provision instances of PostgreSQL.
  - In the `run-production` cluster, this is a provisioner-based class that uses Amazon RDS to provision instances running in Amazon AWS RDS.
- The app operator creates a `ClassClaim`. This is applied with a consuming workload.
  - When it is applied in `iterate` it resolves to a Helm chart instance.
  - When it is promoted to `run-test` it resolves to a VMware PostgreSQL instance.
  - When it is promoted to `run-production` it resolves to an Amazon AWS RDS instance.
- The definition of the `ClassClaim` remains identical across the clusters, which is easier for the application development team.

Important

The backing service implementations and environment layouts used in this scenario are arbitrary. They are not recommendations or requirements.

Although this tutorial uses provisioner-based classes on all three clusters, you can also use a combination of provisioner-based and pool-based classes across the clusters. You might want to do this in cases where, for example, you want to allow for dynamic provisioning of service instances in

the `iterate` cluster, but want to be more considered about the approach in the `run-production` cluster where you might want to ensure that workloads only ever connect to one specific service instance. You can achieve this by using a provisioner-based class on the `iterate` cluster, and an identically named pool-based class on the `run-production` cluster that is configured to only ever select from a pool that consists of one service instance.

Procedure

The following steps explain how to set up a class that allows for claims to resolve to differing implementations of PostgreSQL depending on the cluster it is in.

Step 1: Set up the run-test cluster

Configure the `run-test` cluster for dynamic provisioning of VMware PostgreSQL service instances. To do that, see Configure dynamic provisioning of VMware SQL with Postgres for Kubernetes service instances and complete the steps in the following sections only:

1. Install the Tanzu VMware Postgres Operator
2. Set up the namespace
3. Create a CompositeResourceDefinition
4. Create a Composition
5. Configure RBAC

You do not have to do any other sections in that topic.

Step 2: Set up the run-production cluster

Configure the `run-production` cluster for dynamic provisioning of AWS RDS PostgreSQL service instances. To do that, see Configure Dynamic Provisioning of AWS RDS Service Instances and complete the steps in the following sections only:

1. Install the AWS Provider for Crossplane
2. Create a CompositeResourceDefinition
3. Create a Composition
4. Configure RBAC

You do not have to do any other sections in that topic.

Step 3: Create the class

The `ClusterInstanceClass` acts as the abstraction fronting the differing service implementations across the different clusters. You must create a class with the same name on all three of the clusters, but the configuration of the class varies slightly on each. The `ClassClaim` refers to classes by name. The fact that the class name remains consistent is what allows for the `ClassClaim`, which the application development teams create, to remain unchanged as they are promoted across the clusters.

Create a file named `postgres.class.iterate-cluster.yaml` and copy in the following contents.

```
# postgres.class.iterate-cluster.yaml

---
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ClusterInstanceClass
metadata:
  name: bigcorp-postgresql
```

```
spec:
  description:
    short: PostgreSQL by BigCorp
  provisioner:
    crossplane:
      compositeResourceDefinition: xpostgresqlinstances.bitnami.database.tanzu.vmware.
com
```

This class refers to the `xpostgresqlinstances.bitnami.database.tanzu.vmware.com` CompositeResourceDefinition. This is installed as part of the Bitnami Services package and powers the PostgreSQL service.

You are reusing the underlying CompositeResourceDefinition here from a different class using the class name you want.

Use kubectl to apply the file to the `iterate` cluster.

```
kubectl apply -f postgres.class.iterate-cluster.yaml
```

Create a file named `postgres.class.run-test-cluster.yaml` and copy in the following contents.

```
# postgres.class.run-test-cluster.yaml

---
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ClusterInstanceClass
metadata:
  name: bigcorp-postgresql
spec:
  description:
    short: PostgreSQL by BigCorp
  provisioner:
    crossplane:
      compositeResourceDefinition: xpostgresqlinstances.database.tanzu.example.org
```

This class is almost identical to the previous one, however this one refers instead to the `xpostgresqlinstances.database.tanzu.example.org` CompositeResourceDefinition.

Use kubectl to apply the file to the `run-test` cluster.

```
kubectl apply -f postgres.class.run-test-cluster.yaml
```

Create a file named `postgres.class.run-production-cluster.yaml` and copy in the following contents.

```
# postgres.class.run-production-cluster.yaml

---
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ClusterInstanceClass
metadata:
  name: bigcorp-postgresql
spec:
  description:
    short: PostgreSQL by BigCorp
  provisioner:
    crossplane:
      compositeResourceDefinition: xpostgresqlinstances.database.rds.example.org
```

Again, this class is almost identical to the previous two, but this time refers to the `xpostgresqlinstances.database.rds.example.org` CompositeResourceDefinition.

Use kubectl to apply the file to the `run-production` cluster.

```
kubectl apply -f postgres.class.run-production-cluster.yaml
```

Step 4: Create and promote the workload and class claim

After configuring the clusters and classes, switch roles from service operator to application operator and developer to create the workload and class claim YAML and promote it through the three clusters.

Create a file named `app-with-postgres.yaml` and copy in the following contents.

```
# app-with-postgres.yaml

---
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ClassClaim
metadata:
  name: postgres
  namespace: default
spec:
  classRef:
    name: bigcorp-postgresql

---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: pet-clinic
  namespace: default
  labels:
    apps.tanzu.vmware.com/workload-type: web
    app.kubernetes.io/part-of: pet-clinic
spec:
  params:
  - name: annotations
    value:
      autoscaling.knative.dev/minScale: "1"
  env:
  - name: SPRING_PROFILES_ACTIVE
    value: postgres
  serviceClaims:
  - name: db
    ref:
      apiVersion: services.apps.tanzu.vmware.com/v1alpha1
      kind: ClassClaim
      name: postgres
  source:
    git:
      url: https://github.com/sample-accelerators/spring-petclinic
      ref:
        branch: main
        tag: tap-1.2
```

Then use kubectl to apply the file to the `iterate` cluster.

```
kubectl apply -f app-with-postgres.yaml
```

Wait for the workload to become ready and then inspect the cluster to see that the workload is bound to a Helm-based PostgreSQL service instance. Target the `iterate` cluster then run `helm list -A` to confirm.

Next, apply the exact same `app-with-postgres.yaml` to the `run-test` cluster. When it is ready, confirm that the workload is bound to a Tanzu-based PostgreSQL service instance. Target the `run-test` cluster then run `kubectl get postgres -n tanzu-psql-service-instances` to confirm.

Finally, apply the exact same `app-with-postgres.yaml` to the `run-production` cluster. When it is ready, confirm that the workload is bound to a RDS-based PostgreSQL service instance. Target the `run-production` cluster then run `kubectl get RDSInstance -A` to confirm.

Using direct secret references

In this Services Toolkit tutorial you learn how developers can use direct references to Kubernetes `Secret` resources to connect their application workloads to almost any backing service.

This includes backing services that:

- Run external to Tanzu Application Platform
- Do not adhere to ProvisionedService in the Service Binding Specification for Kubernetes in GitHub.

If you are familiar with Cloud Foundry and Tanzu Application Service, this capability is similar to the concept of user-provided service instances. For more information about user-provided service instances in Cloud Foundry, see the Cloud Foundry documentation.

This tutorial demonstrates a procedure to bind a new application on Tanzu Application Platform to an existing PostgreSQL database that exists in Azure. However, the steps are applicable to any backing service that you want to connect to.

About this tutorial

Target user role: Service Operator and Application Operator
Complexity: Easy
Estimated time: 10 minutes
Topics covered: Service Binding, Direct Secret References
Learning outcomes: Ability to bind workloads to almost any backing service using direct secret references

Prerequisites

Before you can follow this tutorial, you must have:

- Access to a Tanzu Application Platform cluster v1.5.0 or later.
- An Azure PostgreSQL database to connect to.
- Configured networking between the workload and the service endpoint and you must have the credentials for the backing service. Whether this requires extra steps depends on your Kubernetes distribution and the backing service you want to connect your Tanzu Application Platform workloads to.

Create a binding-compatible secret

1. Create a file named `external-azure-db-binding-compatible.yaml` and enter a Kubernetes secret resource similar to the following example:

```
# external-azure-db-binding-compatible.yaml
---
apiVersion: v1
kind: Secret
metadata:
  name: external-azure-db-binding-compatible
type: Opaque
stringData:
```

```
  type: postgresql
  provider: azure
  host: EXAMPLE.DATABASE.AZURE.COM
  port: "5432"
  database: "EXAMPLE-DB-NAME"
  username: "USER@EXAMPLE"
  password: "PASSWORD"
```

Substitute in the values as required.

When using direct secret references, the `Secret` values must abide by the Well-known Secret Entries specifications as defined by the Service Binding Specification for Kubernetes. If you plan to bind this secret to a Spring-based application workload and want to take advantage of the auto-wiring feature, this secret must also contain the properties required by Spring Cloud Bindings.

2. Apply the YAML file by running:

```
kubectl apply -f external-azure-db-binding-compatible.yaml
```

If you are using a multicluster Tanzu Application Platform topology, apply the YAML file to all Run clusters.

3. In a file named `stk-secret-reader.yaml`, grant sufficient Role-Based Access Control (RBAC) permissions to permit Services Toolkit to read the secrets specified by the class:

```
# stk-secret-reader.yaml
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: stk-secret-reader
  labels:
    servicebinding.io/controller: "true"
rules:
- apiGroups:
  - ""
  resources:
  - secrets
  verbs:
  - get
  - list
  - watch
```

4. Apply your changes by running:

```
kubectl apply -f stk-secret-reader.yaml
```

If you are using a multicluster Tanzu Application Platform topology, apply the YAML file to all Run clusters.

5. Create a claim for the newly created secret by running:

```
tanzu service resource-claim create external-azure-db-claim \
  --resource-name external-azure-db-binding-compatible \
  --resource-kind Secret \
  --resource-api-version v1
```

If you are using a multicluster Tanzu Application Platform topology, create the claim on the Build cluster.

6. Obtain the claim reference of the claim by running:

```
tanzu service resource-claim list -o wide
```

If you are using a multicluster Tanzu Application Platform topology, obtain the claim reference on the Build cluster.

Expected output:

```
NAME                     READY  REASON  CLAIM REF
external-azure-db-claim  True           services.apps.tanzu.vmware.com/v1alpha
1:ResourceClaim:external-azure-db-claim
```

From the output, record the value of `CLAIM REF`.

7. Create an application workload by running a command similar to the following example:

```
tanzu apps workload create WORKLOAD-NAME \
  --git-repo https://github.com/sample-accelerators/spring-petclinic \
  --git-branch main \
  --git-tag tap-1.2 \
  --type web \
  --label app.kubernetes.io/part-of=spring-petclinic \
  --annotation autoscaling.knative.dev/minScale=1 \
  --env SPRING_PROFILES_ACTIVE=postgres \
  --service-ref db=REFERENCE
```

Where:

- `WORKLOAD-NAME` is the name of the application workload. For example, `pet-clinic`.
- `REFERENCE` is the value of the `CLAIM REF` for the newly created claim in the output of the last step.

If you are using a multicluster Tanzu Application Platform topology, create the application workload on the Build cluster.

Services Toolkit how-to guides

This section contains how-to guides for Services Toolkit.

In this section:

- Authorize users and groups to claim from provisioner-based classes
- Configure dynamic provisioning of AWS RDS service instances
- Configure dynamic provisioning of VMware SQL with Postgres for Kubernetes service instances
- Configure private registry and VMware Tanzu Application Catalog integration for Bitnami Services
- Troubleshoot Services Toolkit

Authorize users and groups to claim from provisioner-based classes

This Services Toolkit topic for service operators explains how you configure access control so that the required users and groups have authorization to claim from provisioner-based classes.

By default, only users with `cluster-admin` privileges are authorized to create claims for provisioner-based classes. This is because creating claims for provisioner-based classes creates new service

instances, all of which consume resources and might incur monetary cost. As such, you might want to configure some form of access control.

There is one exception to this rule, which is that by default, users with the `app-operator` user role are authorized to create claims for the provisioner-based classes that are part of the Bitnami Services package. For how-to deactivate this default behavior, see Revoke default authorization for claiming from the Bitnami Services classes later in this topic.

Access control is implemented through standard Kubernetes Role-Based Access Control (RBAC) with the use of the custom verb `claim`. You must create a rule in a `ClusterRole` which specifies the `claim` verb for one or more `clusterinstanceclasses`, and then bind the `ClusterRole` to the roles that you want to authorize to create claims for classes with a `ClusterRoleBinding`. This approach is particularly effective when paired with Tanzu Application Platform's aggregated user roles. For more information about user roles in Tanzu Application Platform, see Role descriptions.

Authorize all users with the app-operator user role to claim from any namespace

Create a `ClusterRole` with a rule that specifies the `claim` verb for one or more `ClusterInstanceClass` resources and apply the relevant label.

For example:

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: app-operator-claim-class-bigcorp-rabbitmq
  labels:
    apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access: "true"
rules:
- apiGroups:
  - services.apps.tanzu.vmware.com
  resources:
  - clusterinstanceclasses
  resourceNames:
  - bigcorp-rabbitmq
  verbs:
  - claim
```

This example specifies a `ClusterRole` that permits claiming from a class named `bigcorp-rabbitmq`. The example also includes the `apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access: "true"` label, which causes this `ClusterRole` to aggregate to Tanzu Application Platform's app-operator user role at the cluster scope.

The result is that any user who has the `app-operator` role is now authorized to claim from the `bigcorp-rabbitmq` class. By default, the `app-operator` user role is authorized to create claims for the provisioner-based class.

Authorize a user to claim from a specific namespace

Create a `ClusterRole` with a rule that specifies the `claim` verb for one or more `ClusterInstanceClass` resource and a corresponding `RoleBinding` to bind it to a user.

For example:

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
```

```
  name: claim-class-bigcorp-rabbitmq
rules:
- apiGroups:
  - services.apps.tanzu.vmware.com
  resources:
  - clusterinstanceclasses
  resourceNames:
  - bigcorp-rabbitmq
  verbs:
  - claim

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: alice-claim-class-bigcorp-rabbitmq
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: claim-class-bigcorp-rabbitmq
subjects:
- kind: User
  name: "alice@example.com"
  apiGroup: rbac.authorization.k8s.io
```

This example specifies a `ClusterRole` that permits claiming from a class named `bigcorp-rabbitmq`. The YAML also creates a `ClusterRoleBinding`that binds the user `alice@example.com` to the `ClusterRole`.

The result is that `alice@example.com` is now authorized to claim from `bigcorp-rabbitmq` class.

The user `alice@example.com` still needs permission to create `ClassClaims` in namespaces that they want to consume the services from.

The following example gives `alice@example.com` permission to get, create, update, or delete `ClassClaims` in the `apps` namespace:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
   name: create-class-claim-example
   namespace: apps
rules:
  - apiGroups:
    - services.apps.tanzu.vmware.com
    resources:
    - classclaims
    verbs:
    - get
    - create
    - update
    - delete
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: rbac-role-binding-role-binding
  namespace: apps
subjects:
  - kind: User
    name: "alice@example.com"
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
```

```
  name: create-class-claim-example
  apiGroup: rbac.authorization.k8s.io
```

Revoke default authorization for claiming from the Bitnami Services classes

By default, users with the `app-operator` user role are authorized to create claims for the provisioner-based classes which are part of the Bitnami Services package.

To revoke this authorization:

1. Add the following to your `tap-values.yaml` file:

   ```
   bitnami_services:
     globals:
       create_clusterroles: false
   ```

2. Update Tanzu Application Platform by running:

   ```
   tanzu package installed update tap -p tap.tanzu.vmware.com --values-file tap-va
   lues.yaml -n tap-install
   ```

The result is that any user who has the `app-operator` role is now not authorized to create claims for any of the Bitnami services in any namespace on the cluster.

Configure dynamic provisioning of AWS RDS service instances

This Services Toolkit topic for service operators explains how you set up dynamic provisioning. This enables app development teams to self-serve AWS RDS service instances that are customized.

If you are not already familiar with dynamic provisioning in Tanzu Application Platform, following the tutorial Set up dynamic provisioning of service instances might help you to understand the steps presented in this topic.

In Tanzu Application Platform v1.7 or later, consider using the AWS Services package, rather than using this topic. Both provide the same outcome, which is the availability of an AWS RDS service in the platform. However, the AWS Services package provides a more streamlined and user-friendly approach for integrating AWS RDS into Tanzu Application Platform.

Prerequisites

Before you configure dynamic provisioning, you must have:

- Access to a Tanzu Application Platform cluster v1.6.1 or later.
- The Tanzu services CLI plug-in v0.7.0 or later.
- Access to AWS.

Configure dynamic provisioning

To configure dynamic provisioning for AWS RDS service instances, you must:

1. Install the AWS Provider for Crossplane
2. Create a CompositeResourceDefinition
3. Create a Composition

4. Make the service discoverable
5. Configure RBAC
6. Verify your configuration

Install the AWS Provider for Crossplane

The first step is to install the AWS `Provider` for Crossplane.

The following variants of AWS `Provider` are available:

- crossplane-contrib/provider-aws
- upbound/provider-aws
- upbound/provider-family-aws

VMware recommends that you use the upbound/provider-family-aws. This variant is both fully supported by Upbound and also gives you more control when installing APIs. This helps to improve performance issues often associated with the more monolithic upbound/provider-aws. For more information about how Crossplane is resolving the Provider CRD scaling problem, see the Crossplane Blog.

You must install both the top-level family Provider and the `provider-aws-rds` Provider. To do so:

1. Create a file named `provider-family-aws.yaml` and copy in the following contents, which configures both Providers:

```
# provider-family-aws.yaml

---
# The AWS "family" Provider - manages the ProviderConfig for all other Provider
s in the same family.
# Does not have to be created explicitly, if not created explicitly it will be
installed by the first Provider created
# in the family.
apiVersion: pkg.crossplane.io/v1
kind: Provider
metadata:
  name: upbound-provider-family-aws
spec:
  package: xpkg.upbound.io/upbound/provider-family-aws:v0.36.0
  controllerConfigRef:
    name: upbound-provider-family-aws
---
# The AWS RDS Provider - just one of the many Providers in the AWS family.
# You can add as few or as many additional Providers in the same family as you
wish.
apiVersion: pkg.crossplane.io/v1
kind: Provider
metadata:
  name: upbound-provider-aws-rds
spec:
  package: xpkg.upbound.io/upbound/provider-aws-rds:v0.36.0
  controllerConfigRef:
    name: upbound-provider-family-aws
---
# The ControllerConfig applies settings to a Provider Pod.
# With family Providers each Provider is a unique Pod running in the cluster.
apiVersion: pkg.crossplane.io/v1alpha1
kind: ControllerConfig
metadata:
  name: upbound-provider-family-aws
```

2. Apply the file to the Tanzu Application Platform cluster by running:

```
kubectl apply -f provider-family-aws.yaml
```

3. Verify that both Providers are installed by running:

   ```
   kubectl get providers
   ```

   From the output, confirm that `INSTALLED=True` and `HEALTHY=TRUE`.

4. Create a `ProviderConfig` for the Providers. For instructions, see the sections *Create a Kubernetes secret for AWS* and *Create a ProviderConfig* in the Upbound documentation.

Important

The Upbound documentation assumes Crossplane is installed in the `upbound-system` namespace. However, when working with Crossplane on Tanzu Application Platform, it is installed to the `crossplane-system` namespace. Ensure that you use the correct namespace when you create the `Secret` with credentials for the `Provider`. **Note** Depending on the setup of your AWS account, you might also need to include `aws_session_token` in the `Secret`.

Create a CompositeResourceDefinition

To create the CompositeResourceDefinition (XRD):

1. Create a file named `xpostgresqlinstances.database.rds.example.org.xrd.yaml` and copy in the following contents:

```
# xpostgresqlinstances.database.rds.example.org.xrd.yaml

---
apiVersion: apiextensions.crossplane.io/v1
kind: CompositeResourceDefinition
metadata:
  name: xpostgresqlinstances.database.rds.example.org
spec:
  claimNames:
    kind: PostgreSQLInstance
    plural: postgresqlinstances
  connectionSecretKeys:
  - type
  - provider
  - host
  - port
  - database
  - username
  - password
  group: database.rds.example.org
  names:
    kind: XPostgreSQLInstance
    plural: xpostgresqlinstances
  versions:
  - name: v1alpha1
    referenceable: true
    schema:
      openAPIV3Schema:
        properties:
          spec:
            properties:
              storageGB:
                type: integer
```

```
                default: 20
            type: object
        type: object
    served: true
```

This XRD configures the parameter `storageGB`. This gives application teams the option to choose a suitable amount of storage for the AWS RDS service instance when they create a claim. You can choose to expose as many or as few parameters to application teams as you like.

2. Apply the file to the Tanzu Application Platform cluster by running:

```
kubectl apply -f xpostgresqlinstances.database.rds.example.org.xrd.yaml
```

Create a Composition

To create the composition:

1. Create a file named `xpostgresqlinstances.database.rds.example.org.composition.yaml` and copy in the following contents:

```
# xpostgresqlinstances.database.rds.example.org.composition.yaml

---
apiVersion: apiextensions.crossplane.io/v1
kind: Composition
metadata:
  labels:
    provider: "aws"
    vpc: "default"
  name: xpostgresqlinstances.database.rds.example.org
spec:
  compositeTypeRef:
    apiVersion: database.rds.example.org/v1alpha1
    kind: XPostgreSQLInstance
  publishConnectionDetailsWithStoreConfigRef:
    name: default
  resources:
  - base:
      apiVersion: rds.aws.upbound.io/v1beta1
      kind: Instance
      spec:
        forProvider:
          # NOTE: configure this section to your specific requirements
          instanceClass: db.t3.micro
          autoGeneratePassword: true
          passwordSecretRef:
            key: password
            namespace: crossplane-system
          engine: postgres
          engineVersion: "13.8"                    # <---- Refer to https://doc
s.aws.amazon.com/AmazonRDS/latest/PostgreSQLReleaseNotes/postgresql-release-cal
endar.html for latest
          name: postgres
          username: masteruser
          publiclyAccessible: true                 # <---- DANGER
          region: us-east-1
          skipFinalSnapshot: true
        writeConnectionSecretToRef:
          namespace: crossplane-system
    connectionDetails:
    - name: type
      value: postgresql
    - name: provider
```

```
      value: aws
    - name: database
      value: postgres
    - fromConnectionSecretKey: username
    - fromConnectionSecretKey: password
    - name: host
      fromConnectionSecretKey: endpoint
    - fromConnectionSecretKey: port
    name: instance
    patches:
    - fromFieldPath: metadata.uid
      toFieldPath: spec.forProvider.passwordSecretRef.name
      transforms:
      - string:
          fmt: '%s-postgresql-pw'
          type: Format
        type: string
      type: FromCompositeFieldPath
    - fromFieldPath: metadata.uid
      toFieldPath: spec.writeConnectionSecretToRef.name
      transforms:
      - string:
          fmt: '%s-postgresql'
          type: Format
        type: string
      type: FromCompositeFieldPath
    - fromFieldPath: spec.storageGB
      toFieldPath: spec.forProvider.allocatedStorage
      type: FromCompositeFieldPath
```

This Composition configures all RDS PostgreSQL instances as follows:

- All instances are placed in the `us-east-1` region.
- All instances use the default Virtual Private Cloud (VPC) for the respective AWS account.
- All instances are publicly accessible over the Internet.

2. If you want to keep the instances publicly accessible over the Internet and use the default VPC, add an inbound rule for TCP on port 5432 to the security group of the default VPC to allow connection to the instances.
3. If you do not want the instances to be publicly accessible over the Internet, edit the Composition as required. Specific requirements vary, but this might include composing a combination of VPCs, Subnets, SubnetGroups, Routes, SecurityGroups, and SecurityGroupRules. Refer to Compositions that are available online for inspiration and guidance.

   For example, see `getting-started-with-aws-with-vpc` in the Upbound documentation. This example defines a Composition that creates a separate VPC for each RDS PostgreSQL instance and automatically configures inbound rules. If you want to follow this example, you might need to install additional Providers from the AWS Provider Family, such as upbound/provider-aws-ec2.
4. Make any other configuration changes to the Composition so that it meets your specific requirements.
5. Apply the file to the Tanzu Application Platform cluster by running:

   ```
   kubectl apply -f xpostgresqlinstances.database.rds.example.org.composition.yaml
   ```

Make the service discoverable

To make the service discoverable to application teams:

1. Create a file named `rds.class.yaml` and copy in the following contents:

```
# rds.class.yaml

---
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ClusterInstanceClass
metadata:
  name: aws-rds-psql
spec:
  description:
    short: Amazon AWS RDS PostgreSQL
  provisioner:
    crossplane:
      compositeResourceDefinition: xpostgresqlinstances.database.rds.example.or
g
```

2. Apply the file to the Tanzu Application Platform cluster by running:

```
kubectl apply -f rds.class.yaml
```

Configure RBAC

To configure Role-Based Access Control (RBAC) to authorize users with the app-operator role to claim from the class:

1. Create a file named `app-operator-claim-aws-rds-psql.rbac.yaml` and copy in the following contents:

```
# app-operator-claim-aws-rds-psql.rbac.yaml

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: app-operator-claim-aws-rds-psql
  labels:
    apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access: "true"
rules:
- apiGroups:
  - "services.apps.tanzu.vmware.com"
  resources:
  - clusterinstanceclasses
  resourceNames:
  - aws-rds-psql
  verbs:
  - claim
```

2. Apply the file to the Tanzu Application Platform cluster by running:

```
kubectl apply -f app-operator-claim-aws-rds-psql.rbac.yaml
```

Verify your configuration

To verify your configuration, create a claim for an AWS RDS service instance by running:

```
tanzu service class-claim create rds-psql-1 --class aws-rds-psql -p storageGB=30
```

Note

Whether application workloads can establish network connectivity to the resulting RDS database depends on a number of factors. This includes specifics about the environment you're working in and the configuration in the `Composition` file. At a minimum, you can configure a `securityGroup` to permit inbound traffic. There might be other requirements as well.

Configure dynamic provisioning of VMware SQL with Postgres for Kubernetes service instances

This Services Toolkit topic for service operators explains how you set up dynamic provisioning. This enables app development teams to create self-serve VMware SQL with Postgres for Kubernetes service instances that are customized to meet their needs.

If you are not already familiar with dynamic provisioning in Tanzu Application Platform, following the tutorial Set up dynamic provisioning of service instances. might be help you understand the steps presented in this topic.

Prerequisites

Before you configure dynamic provisioning, you must have:

- Access to a Tanzu Application Platform cluster v1.5.0 or later.
- The Tanzu services CLI plug-in v0.6.0 or later.

Configure dynamic provisioning

To configure dynamic provisioning for VMware SQL with Postgres for Kubernetes services instances, you must:

1. Install the VMware Postgres Operator
2. Set up the namespace
3. Create a CompositeResourceDefinition
4. Create a Composition
5. Make the service discoverable
6. Configure RBAC
7. Verify your configuration

Install the VMware Postgres Operator

Install the VMware Postgres Operator by following the steps in Installing a VMware Postgres Operator.

Set up the namespace

This topic configures dynamic provisioning to provision all PostgreSQL service instances into the same namespace. This namespace is named `tanzu-psql-service-instances`.

To set up the namespace:

1. Ensure that the namespace exists by running the following:

```
kubectl create namespace tanzu-psql-service-instances
```

2. The VMware Postgres Operator also requires that a secret holding registry credentials exists in the same namespace that the service instances will be created in. Ensure that the secret exists in the namespace by running:

```
kubectl create secret --namespace=tanzu-psql-service-instances docker-registry
regsecret \
  --docker-server=https://registry.tanzu.vmware.com \
  --docker-username=`USERNAME` \
  --docker-password=`PASSWORD`
```

   Where:

   - `USERNAME` is your registry username.
   - `PASSWORD` is your registry password.

Note

You must update the `--docker-server` value if you relocated images as part of the installation of the operator.

Create a CompositeResourceDefinition

To create the CompositeResourceDefinition (XRD):

1. Create a file named `xpostgresqlinstances.database.tanzu.example.org.xrd.yaml` and copy in the following contents:

```
# xpostgresqlinstances.database.tanzu.example.org.xrd.yaml

---
apiVersion: apiextensions.crossplane.io/v1
kind: CompositeResourceDefinition
metadata:
  name: xpostgresqlinstances.database.tanzu.example.org
spec:
  connectionSecretKeys:
  - provider
  - type
  - database
  - host
  - password
  - port
  - uri
  - username
  group: database.tanzu.example.org
  names:
    kind: XPostgreSQLInstance
    plural: xpostgresqlinstances
  versions:
  - name: v1alpha1
    referenceable: true
    schema:
      openAPIV3Schema:
        properties:
          spec:
            properties:
              storageGB:
                type: integer
                default: 20
            type: object
```

```
        type: object
    served: true
```

This XRD configures the parameter `storageGB`. This gives application teams the option to choose a suitable amount of storage for the Tanzu Postgres service instance when they create a claim. You can choose to expose as many or as few parameters to application teams as you like.

2. Apply the file to the Tanzu Application Platform cluster by running:

```
kubectl apply -f xpostgresqlinstances.database.tanzu.example.org.xrd.yaml
```

Create a Composition

To create the Composition:

1. Create a file named `xpostgresqlinstances.database.tanzu.example.org.composition.yaml` and copy in the following contents:

```
# xpostgresqlinstances.database.tanzu.example.org.composition.yaml

---
apiVersion: apiextensions.crossplane.io/v1
kind: Composition
metadata:
  name: xpostgresqlinstances.database.tanzu.example.org
spec:
  compositeTypeRef:
    apiVersion: database.tanzu.example.org/v1alpha1
    kind: XPostgreSQLInstance
  publishConnectionDetailsWithStoreConfigRef:
    name: default
  resources:
  - base:
      apiVersion: kubernetes.crossplane.io/v1alpha2
      kind: Object
      spec:
        forProvider:
          manifest:
            apiVersion: sql.tanzu.vmware.com/v1
            kind: Postgres
            metadata:
              name: PATCHED
              namespace: tanzu-psql-service-instances
            spec:
              storageSize: 2G
        connectionDetails:
        - apiVersion: v1
          kind: Secret
          namespace: tanzu-psql-service-instances
          fieldPath: data.provider
          toConnectionSecretKey: provider
        - apiVersion: v1
          kind: Secret
          namespace: tanzu-psql-service-instances
          fieldPath: data.type
          toConnectionSecretKey: type
        - apiVersion: v1
          kind: Secret
          namespace: tanzu-psql-service-instances
          fieldPath: data.host
          toConnectionSecretKey: host
        - apiVersion: v1
          kind: Secret
```

```
        namespace: tanzu-psql-service-instances
        fieldPath: data.port
        toConnectionSecretKey: port
      - apiVersion: v1
        kind: Secret
        namespace: tanzu-psql-service-instances
        fieldPath: data.username
        toConnectionSecretKey: username
      - apiVersion: v1
        kind: Secret
        namespace: tanzu-psql-service-instances
        fieldPath: data.password
        toConnectionSecretKey: password
      - apiVersion: v1
        kind: Secret
        namespace: tanzu-psql-service-instances
        fieldPath: data.database
        toConnectionSecretKey: database
      - apiVersion: v1
        kind: Secret
        namespace: tanzu-psql-service-instances
        fieldPath: data.uri
        toConnectionSecretKey: uri
      writeConnectionSecretToRef:
        namespace: tanzu-psql-service-instances
  connectionDetails:
  - fromConnectionSecretKey: provider
  - fromConnectionSecretKey: type
  - fromConnectionSecretKey: host
  - fromConnectionSecretKey: port
  - fromConnectionSecretKey: username
  - fromConnectionSecretKey: password
  - fromConnectionSecretKey: database
  - fromConnectionSecretKey: uri
  patches:
    - fromFieldPath: metadata.name
      toFieldPath: spec.forProvider.manifest.metadata.name
      type: FromCompositeFieldPath
    - fromFieldPath: spec.storageSize
      toFieldPath: spec.forProvider.manifest.spec.persistence.storage
      transforms:
      - string:
          fmt: '%dG'
          type: Format
        type: string
      type: FromCompositeFieldPath
    - fromFieldPath: metadata.name
      toFieldPath: spec.writeConnectionSecretToRef.name
      transforms:
      - string:
          fmt: '%s-psql'
          type: Format
        type: string
      type: FromCompositeFieldPath
    - fromFieldPath: metadata.name
      toFieldPath: spec.connectionDetails[0].name
      transforms:
      - string:
          fmt: '%s-app-user-db-secret'
          type: Format
        type: string
      type: FromCompositeFieldPath
    - fromFieldPath: metadata.name
      toFieldPath: spec.connectionDetails[1].name
      transforms:
      - string:
```

```
            fmt: '%s-app-user-db-secret'
            type: Format
          type: string
        type: FromCompositeFieldPath
      - fromFieldPath: metadata.name
        toFieldPath: spec.connectionDetails[2].name
        transforms:
        - string:
            fmt: '%s-app-user-db-secret'
            type: Format
          type: string
        type: FromCompositeFieldPath
      - fromFieldPath: metadata.name
        toFieldPath: spec.connectionDetails[3].name
        transforms:
        - string:
            fmt: '%s-app-user-db-secret'
            type: Format
          type: string
        type: FromCompositeFieldPath
      - fromFieldPath: metadata.name
        toFieldPath: spec.connectionDetails[4].name
        transforms:
        - string:
            fmt: '%s-app-user-db-secret'
            type: Format
          type: string
        type: FromCompositeFieldPath
      - fromFieldPath: metadata.name
        toFieldPath: spec.connectionDetails[5].name
        transforms:
        - string:
            fmt: '%s-app-user-db-secret'
            type: Format
          type: string
        type: FromCompositeFieldPath
      - fromFieldPath: metadata.name
        toFieldPath: spec.connectionDetails[6].name
        transforms:
        - string:
            fmt: '%s-app-user-db-secret'
            type: Format
          type: string
        type: FromCompositeFieldPath
      - fromFieldPath: metadata.name
        toFieldPath: spec.connectionDetails[7].name
        transforms:
        - string:
            fmt: '%s-app-user-db-secret'
            type: Format
          type: string
        type: FromCompositeFieldPath
    readinessChecks:
      - type: MatchString
        fieldPath: status.atProvider.manifest.status.currentState
        matchString: "Running"
```

2. Configure the Composition you just copied to your specific requirements.
3. Apply the file to the Tanzu Application Platform cluster by running:

```
kubectl apply -f xpostgresqlinstances.database.tanzu.example.org.composition.ya
ml
```

Make the service discoverable

To make the service discoverable to application teams:

1. Create a file named `tanzu-psql.class.yaml` and copy in the following contents:

```
# tanzu-psql.class.yaml

---
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ClusterInstanceClass
metadata:
  name: tanzu-psql
spec:
  description:
    short: VMware SQL with Postgres
  provisioner:
    crossplane:
      compositeResourceDefinition: xpostgresqlinstances.database.tanzu.example.
org
```

2. Apply the file to the Tanzu Application Platform cluster by running:

```
kubectl apply -f tanzu-psql.class.yaml
```

Configure RBAC

To configure access control with RBAC:

1. Create a file named `provider-kubernetes-tanzu-postgres-read-writer.rbac.yaml` and copy in the following contents:

```
# provider-kubernetes-tanzu-postgres-read-writer.rbac.yaml

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: tanzu-postgres-read-writer
  labels:
    services.tanzu.vmware.com/aggregate-to-provider-kubernetes: "true"
rules:
- apiGroups:
  - sql.tanzu.vmware.com
  resources:
  - postgres
  verbs:
  - "*"
```

2. Apply the file to the Tanzu Application Platform cluster by running:

```
kubectl apply -f provider-kubernetes-tanzu-postgres-read-writer.rbac.yaml
```

3. Create a file named `app-operator-claim-tanzu-psql.rbac.yaml` and copy in the following contents:

```
# app-operator-claim-tanzu-psql.rbac.yaml

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: app-operator-claim-tanzu-psql
  labels:
    apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access: "true"
rules:
```

```
- apiGroups:
  - "services.apps.tanzu.vmware.com"
  resources:
  - clusterinstanceclasses
  resourceNames:
  - tanzu-psql
  verbs:
  - claim
```

4. Apply the file to the Tanzu Application Platform cluster by running:

```
kubectl apply -f app-operator-claim-tanzu-psql.rbac.yaml
```

Verify your configuration

To verify your configuration, create a claim for a PostgreSQL service instance by running:

```
tanzu service class-claim create tanzu-psql-1 --class tanzu-psql -p storageGB=5
```

Troubleshoot Services Toolkit

This topic explains how you can troubleshoot issues related to working with services on Tanzu Application Platform (commonly known as TAP).

For the limitations of services on Tanzu Application Platform, see Services Toolkit limitations.

Debug `ClassClaim` and provisioner-based `ClusterInstanceClass`

This section provides guidance on how to debug issues related to using `ClassClaim` and provisioner-based `ClusterInstanceClass`. The approach starts by inspecting a `ClassClaim` and tracing back through the chain of resources that are created when fulfilling the `ClassClaim`.

Prerequisites

To follow the steps in this section, you must have kubectl access to the cluster.

Step 1: Inspect the `ClassClaim`, `ClusterInstanceClass`, and `CompositeResourceDefinition`

1. Inspect the status of `ClassClaim` by running:

   ```
   kubectl describe classclaim claim-name -n NAMESPACE
   ```

   Where `NAMESPACE` is your namespace.

   From the output, check the following:

   - Check the status conditions for information that can lead you to the cause of the issue.
   - Check `.spec.classRef.name` and record the value.

2. Inspect the status of the `ClusterInstanceClass` by running:

   ```
   kubectl describe clusterinstanceclass CLASS-NAME
   ```

   Where `CLASS-NAME` is the value of `.spec.classRef.name` you retrieved in the previous step.

From the output, check the following:

- Check the status conditions for information that can lead you to the cause of the issue.
- Check that the `Ready` condition has status `"True"`.
- Check `.spec.provisioner.crossplane` and record the value.

3. Inspect the status of the `CompositeResourceDefinition` by running:

```
kubectl describe xrd XRD-NAME
```

Where `XRD-NAME` is the value of `.spec.provisioner.crossplane` you retrieved in the previous step.

From the output, check the following:

- Check the status conditions for information that can lead you to the cause of the issue.
- Check that the `Established` condition has status `"True"`.
- Check events for any errors or warnings that can lead you to the cause of the issue.
- If both the `ClusterInstanceClass` reports `Ready="True"` and the `CompositeResourceDefinition` reports `Established="True"`, move on to the next section.

Step 2: Inspect the Composite Resource, the Managed Resources and the underlying resources

1. Check `.status.provisionedResourceRef` by running:

```
kubectl describe classclaim claim-name -n NAMESPACE
```

Where `NAMESPACE` is your namespace.

From the output, check the following:

- Check `.status.provisionedResourceRef`, and record the values of `kind`, `apiVersion`, and `name`.

2. Inspect the status of the Composite Resource by running:

```
kubectl describe KIND.API-GROUP NAME
```

Where:

- `KIND` is the value of `kind` you retrieved in the previous step.
- `API-GROUP` is the value of `apiVersion` you retrieved in the previous step without the `/<version>` part.
- `NAME` is the value of `name` you retrieved in the previous step.

From the output, check the following:

- Check the status conditions for information that can lead you to the cause of the issue.
- Check if there was an issue creating the Managed Resources from which this Composite Resource is composed. Refer to `.spec.resourceRefs` in the output and for each:

        - Use the values of `kind`, `apiVersion`, and `name` to inspect the status of the Managed Resource.
        - Check the status conditions for information that can lead you to the cause of the issue.
    - Check events for any errors or warnings that can lead you to the cause of the issue.
    - If all Managed Resources appear healthy, move on to the next section.

Step 3: Inspect the events log

Inspect the events log by running:

```
kubectl get events -A
```

From the output, check the following:

- Check for any errors or warnings that can lead you to the cause of the issue.
- If there are no errors or warnings, move on to the next section.

Step 4: Inspect the secret

1. Check `.status.resourceRef` by running:

   ```
   kubectl get classclaim claim-name -n NAMESPACE -o yaml
   ```

   Where `NAMESPACE` is your namespace.

   From the output, check the following:

   - Check `.status.resourceRef` and record the values `kind`, `apiVersion`, `name`, and `namespace`

2. Inspect the claimed resource, which is likely a secret, by running:

   ```
   kubectl get secret NAME -n NAMESPACE -o yaml
   ```

   Where:

   - `NAME` is the `name` you retrieved in the previous step.
   - `NAMESPACE` is the `namespace` you retrieved in the previous step.

   If the secret is there and has data, then something else must be causing the issue.

Step 5: Contact support

If you have followed the steps in this section and are still unable to discover the cause of the issue, contact VMware Support for further guidance and help to resolve the issue.

Unexpected error if `additionalProperties` is `true` in a CompositeResourceDefinition

Symptom:

When creating a `CompositeResourceDefinition`, if you set `additionalProperties: true` in the `openAPIV3Schema` section, an error occurs during the validation step of the creation of any `ClassClaim` that refers to a class that refers to the `CompositeResourceDefinitions`.

The error appears as follows:

```
json: cannot unmarshal bool into Go struct field JSONSchemaProps.AdditionalProperties
of type apiextensions.JSONSchemaPropsOrBool
```

Solution:

Rather than setting `additionalProperties: true`, you can set `additionalProperties: {}`. This has the same effect, but does not cause unexpected errors.

Cannot claim from clusterinstanceclass when creating a `ClassClaim`

Symptom:

Users who were previously able to create a `ClassClaim` now get an admission error similar to:

```
user 'alice@example.com' cannot 'claim' from clusterinstanceclass 'bigcorp-rabbitmq'
```

This occurs even if users were granted the `claim` permission on `ClusterInstanceClasses` through either:

- A `Role` and a `RoleBinding`
- A `ClusterRole` and a `RoleBinding`

Explanation:

You now need the cluster-level `claim` permission, granted through a `ClusterRole` and `ClusterRoleBinding`. Namespace-scoped permissions are no longer enough. This is to protect against unexpected access to resources in other namespaces.

This change was introduced with Services Toolkit v0.12.0 in Tanzu Application Platform v1.7.0. For more information about this change, see The claim verb for ClusterInstanceClass.

Solution:

To allow users to create `ClassClaims` again, you must:

1. Move from a `Role` to a `ClusterRole` for granting users permission to claim a `ClusterInstanceClass`.
2. Move from a `RoleBinding` to a `ClusterRoleBinding` for binding this permission to a user.

For more information, see Authorize users and groups to claim from provisioner-based classes.

Services Toolkit reference

This section provides reference documentation for Services Toolkit.

In this section:

- API documentation
- Tanzu Service CLI plug-in reference
- Services Toolkit terminology and user roles
- Services Toolkit limitations

Services Toolkit API documentation

This section of the documentation provides detailed information about Services Toolkit's APIs.

- ClusterInstanceClass and ClassClaim

- ResourceClaim and ResourceClaimPolicy
- InstanceQuery
- RBAC

ClusterInstanceClass and ClassClaim

This topic provides Services Toolkit API documentation for `ClusterInstanceClass` and `ClassClaim`.

ClusterInstanceClass

You can configure `ClusterInstanceClass` to one of two variants - either pool-based or provisioner-based.

- Claims for pool-based classes are fulfilled by identifying service instances using the configuration in `.spec.pool`.
- Claims for provisioner-based classes are fulfilled by provisioning new service instances using the configuration in `.spec.provisioner`.

A class can either be a pool-based class or a provisioner-based class, but never both.

The following snippet outlines the `ClusterInstanceClass` YAML:

```
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ClusterInstanceClass

metadata:
  # A name for the class. The class name is used by consumers (application operators
  # and developers) when creating claims.
  name: mysql-unmanaged

spec:
  # Provide information about the class in the description.
  description:
    # A short description for the class. Aim to provide just enough information
    # to help consumers (application operators and developers) to understand
    # what's on offer by the class.
    short: MySQL by Bitnami

  # (Optional) Configure a provisioner-based class.
  # Must specify one of either `provisioner` or `pool`.
  provisioner:
    # Configure provisioning using Crossplane (https://www.crossplane.io/).
    crossplane:
      # CompositeResourceDefinition refers to the name of a Composite Resource Definit
ion (XRD).
      # For example, "xpostgresqlinstances.database.example.org".
      compositeResourceDefinition: xmysqlinstances.bitnami.database.tanzu.vmware.com

      # (Optional) The compositionSelector allows you to match a Composition by
      # labels rather than naming one explicitly. It is used to set the compositionRef
      # if none is specified explicitly.
      compositionSelector:
        matchLabels:
          provider: bitnami
          type: mysql

      # (Optional) CompositionRef specifies which Composition this XR will use to
      # compose resources when it is created, updated, or deleted.
      # This can be omitted and is set automatically if the XRD has a default or
      # enforced composition reference, or if the below composition selector is set.
      compositionRef:
```

```
        name: composition-name

      # (Optional) CompositionUpdatePolicy specifies how existing XRs should be
      # updated to new revisions of their underlying composition.
      # One of either 'Automatic' or 'Manual'; default=Automatic.
      compositionUpdatePolicy: Manual

  # (Optional) Configure a pool-based class.
  # Must specify one of either `provisioner` or `pool`.
  pool:
    # (Optional) Group specifies the API group for the resources belonging to this cla
ss.
    group:

    # Kind specifies the API Kind for the resources belonging to this class.
    kind:

    # (Optional) FieldSelector specifies a set of fields that MUST match certain condi
tions.
    # See https://kubernetes.io/docs/concepts/overview/working-with-objects/field-sele
ctors/.
    fieldSelector:

    # (Optional) LabelSelector specifies a set of labels that MUST match.
    # See https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/#la
bel-selectors.
    labelSelector:

# status is populated by the controller
status:
  # Conditions for the class.
  conditions:
    # The condition type. Currently only 'Ready'.
    - type: Ready
      # status can be either 'True' or 'False'.
      status: "True"
      # reason provides a reason for status: "False" for additional context.
      # One of 'PooledResourceNotFound', 'CompositeResourceDefinitionNotFound',
      # 'CompositeResourceDefinitionNotValid', or 'CompositeResourceDefinitionNotRead
y'.
      # Not set if status: "True".
      reason:

  # (Optional) claimParameters contains the OpenAPIV3Schema used to configure
  # claims for this class.
  # Not set on pool-based classes.
  claimParameters:
    openAPIV3Schema:
      description: The OpenAPIV3Schema of this Composite Resource Definition.
      properties:
        storageGB:
          default: 1
          description: The desired storage capacity of the database, in Gigabytes.
          type: integer
      type: object

  # instanceType holds information about the resource selected by this class.
  # If using the Crossplane provisioner, this refers to the CompositeResource (XR)
  # defined by the CompositeResourceDefinition (XRD) referred to in the class.
  instanceType:
    # (Optional) Group specifies the API group.
    group: bitnami.database.tanzu.vmware.com
    # Kind specifies the API Kind.
    kind: XMySQLInstance
    # Version specifies the API version.
    version: v1alpha1
```

```
  # Populated based on metadata.generation when controller observes a change to
  # the resource. If this value is out of date, other status fields do not
  # reflect latest state.
  observedGeneration: 1
```

ClassClaim

`ClassClaim` refers to a `ClusterInstanceClass` from which service instances are then either selected (for pool-based classes) or provisioned (for provisioner-based classes) to fulfill the claim. `ClassClaim` adheres to Provisioned Service as defined by the Service Binding Specification for Kubernetes.You can bind a `ClassClaim` to an application workload by using a reference in the workload's `.spec.serviceClaims` configuration.

The following snippet outlines the `ClassClaims` YAML:

```
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ClassClaim

metadata:
  # The name for the claim.
  name: mysql-claim-1
  # The namespace in which to create the claim.
  namespace: my-apps

spec:
  # classRef holds a reference to a ClusterInstanceClass.
  classRef:
    # The name of the class from which to claim a service instance.
    # For information about the permissions users must have to claim from the class,
    # see the note below this snippet.
    name: mysql-unmanaged

  # (Optional) parameters are key-value pairs that are configuration inputs to the
  # instance obtained from the referenced ClusterInstanceClass. These parameters
  # only take effect when referring to a provisioner-based class.
  parameters:
    key: value

status:
  # Conditions for the claim.
  conditions:
    # The condition type. Can be one of 'Ready', 'ClassMatched', 'Validated', or
    # 'ResourceClaimCreated'. All condition types are initialized for all claims.
    # The Ready condition reports status: "True" once all other condition types are he
althy.
    - type: Ready
      # status can be either 'True' or 'False'.
      status: "True"
      # reason provides a reason for status: "False" for additional context.
      # One of 'UnableToFetchClass', 'ClassDoesNotExist', 'ClassNotReady',
      # 'UnableToQueryClaimableInstances', 'NoClaimableInstances',
      # 'UnableToCreateResourceClaim', 'UnableToCreateClaimableInstance', 'ResourceRea
dy',
      # 'ResourceNotReady', 'ResourceReadyUnsupported', or 'ReasonParametersInvalid'.
      # Not set if status: "True".
      reason:

  # binding holds a reference to a secret, in the same namespace, which contains
  # credentials for accessing the claimed service instance.
  binding:
    # The name of the `Secret`. The presence of the .status.binding.name field
    # marks this resource as a Provisioned Service.
    name: 770845b6-02f0-4c1b-8d0c-3dae81bad35c
```

```
  # provisionedResourceRef contains a reference to the provisioned resource.
  # Only set if the claim refers to a provisioner-based class.
  provisionedResourceRef:
    # The API Group/Version of the provisioned resource in the GROUP/VERSION format.
    apiVersion: bitnami.database.tanzu.vmware.com/v1alpha1
    # The API kind of the provisioned resource.
    kind: XMysqlInstance
    # The name of the provisioned resource.
    name: mysql-1-57dr7

  # resourceRef contains a reference to the claimed resource.
  resourceRef:
    # The API Group/Version of the claimed resource in the GROUP/VERSION format.
    apiVersion: v1
    # The API kind of the claimed resource.
    kind: Secret
    # The name of the claimed resource.
    name: 770845b6-02f0-4c1b-8d0c-3dae81bad35c
    # The namespace of the claimed resource.
    namespace: my-apps

  # Populated based on metadata.generation when controller observes a change to
  # the resource. If this value is out of date, other status fields do not reflect
  # latest state.
  observedGeneration: 1
```

Note

If you refer to a provisioner-based class in `spec.classref.name`, you must have sufficient RBAC permission to claim from the class. For more information, see Authorize users and groups to claim from provisioner-based classes.

ResourceClaim and ResourceClaimPolicy

This topic provides Services Toolkit API documentation for `ResourceClaim` and `ResourceClaimPolicy`.

ResourceClaim

`ResourceClaim` is used to claim a specific Kubernetes resource by using a reference. `ResourceClaim` adheres to Provisioned Service as defined by the Service Binding Specification for Kubernetes. you can bind a `ResourceClaim` to an application workload by using a reference in the workload's `.spec.serviceClaims` configuration.

A `ResourceClaim` is exclusive by nature. This means that after a `ResourceClaim` has claimed a resource, no other `ResourceClaim` can claim that same resource.

```
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ResourceClaim

metadata:
  # The name for the claim.
  name: claim-1
  # The namespace in which to create the claim.
  namespace: my-apps
  # Internal finalizers applied by the resource claim controller to ensure
  # resources are cleaned up.
  finalizers:
  - resourceclaims.services.apps.tanzu.vmware.com/finalizer
```

```
  - resourceclaims.services.apps.tanzu.vmware.com/lease-finalizer

spec:
  # ref is a reference to the resource to be claimed.
  ref:
    # The API Group/Version of the resource to claim in the GROUP/VERSION format.
    apiVersion: v1
    # The API Kind of the resource to claim.
    kind: Secret
    # The name of the resource to claim.
    name: 770845b6-02f0-4c1b-8d0c-3dae81bad35c
    # (Optional) The namespace of the resource to claim. If the resource exists
    # in a different namespace to the namespace of the claim, then you must configure
    # a corresponding ResourceClaimPolicy to permit claiming of the resource.
    namespace: service-instances

status:
  # Conditions for the claim.
  conditions:
    # The condition type. Can be one of 'Ready', 'ResourceMatched' or 'ResourceMatche
d'.
    # All condition types are initialized for all claims.
    # The Ready condition reports status: "True" once all other condition types are he
althy.
    - type: Ready
      # status can be either 'True' or 'False'.
      status: "True"
      # reason provides a reason for status: "False" for additional context.
      # One of 'ResourceNotFound', 'BindingNotCopyable', 'UnableToSetExclusiveClaim',
      # 'ResourceNonBindable', 'NoMatchingResourceClaimPolicy',
      # 'UnableToTrackReferencedResource', 'ResourceAlreadyClaimed',
      # 'UpdatedResourceReference', or 'ClaimMarkedForDeletion'.
      # Not set if status: "True".
      reason:

  # binding holds a reference to a secret in the same namespace which contains
  # credentials for accessing the claimed service instance.
  binding:
    # The name of the secret. The presence of the .status.binding.name field marks
    # this resource as a Provisioned Service.
    name: 770845b6-02f0-4c1b-8d0c-3dae81bad35c

  # claimedResourceRef holds a reference to the claimed resource.
  claimedResourceRef:
    # The API Group/Version of the claimed resource in the GROUP/VERSION format.
    apiVersion: v1
    # The API kind of the claimed resource.
    kind: Secret
    # The name of the claimed resource.
    name: 770845b6-02f0-4c1b-8d0c-3dae81bad35c
    # The namespace of the claimed resource.
    namespace: service-instances

  # Populated based on metadata.generation when controller observes a change to
  # the resource. If this value is out of date, other status fields do not
  # reflect latest state.
  observedGeneration: 1
```

ResourceClaimPolicy

`ResourceClaimPolicy` provides a mechanism to either permit or deny the claiming of resources across namespaces.

```
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ResourceClaimPolicy

metadata:
  # The name for the policy.
  name: default-ns-can-claim-secret-1
  # The namespace for the policy.
  # ResourceClaimPolicy resources must exist in the same namespace as the resources
  # they are permitting to be claimed.
  namespace: x-namespace-1

spec:
  # consumingNamespaces specifies the source namespace(s) to permit the claiming
  # of the resources from.
  # Use '*' to configure all namespaces.
  consumingNamespaces:
  - default

  # The API group of the resource to permit the claiming of.
  group: rabbitmq.com
  # The API kind of the resource to permit the claiming of.
  kind: RabbitmqCluster
  # (Optional) selector is a labelSelector to match resources to permit the claiming o
f.
  selector:
    matchLabels:
      "key": "value"
```

InstanceQuery

This topic provides Services Toolkit API documentation for `InstanceQuery`.

InstanceQuery

`InstanceQuery` is a create-only API that, given a pool-based `ClusterInstanceClass`, returns the intersection of the set of service instances represented by that class and the claimable service instances for the namespace of the `InstanceQuery`.

```
apiVersion: claimable.services.apps.tanzu.vmware.com/v1alpha1
kind: InstanceQuery

metadata:
  # An arbitrary name for the query.
  name: test
  # The namespace from which to run the query. The resulting list of instances is
  # specific to the namespace of the query itself.
  namespace: my-apps

spec:
  # The name of the class to query for claimable instances. Must refer to a
  # pool-based class and not a provisioner-based class.
  class: pooled-class-1
  # (Optional) A limit on the maximum number of instances to return.
  # The default is 50.
  limit: 1

status:
  # A list of service instances that you can claim by using ResourceClaims created
  # in the same namespace as the query.
  instances:
    # The API group/version of the claimable instance in the format GROUP/VERSION.
  - apiVersion: v1
```

```
    # The API kind of the claimable instance.
    kind: Secret
    # The name of the claimable instance.
    name: my-secret-two
    # The namespace of the claimable instance.
    namespace: default
```

RBAC

This topic provides API documentation for Role-Based Access Control (RBAC) relating to Services Toolkit's APIs.

Aggregation labels

This section describes the following Aggregation labels:

- servicebinding.io/controller: "true"
- services.tanzu.vmware.com/aggregate-to-provider-kubernetes: "true"
- services.tanzu.vmware.com/aggregate-to-provider-helm: "true"

servicebinding.io/controller: "true"

Use this label to grant the Services Toolkit and service bindings controllers permission to get, list, and watch resources to be claimed and bound in the cluster.

For example, the following `ClusterRole` grants the controllers permission to get, list, and watch `RabbitmqCluster` resources. You cannot create `ClassClaims` or `ResourceClaims` unless the controllers have at least these permissions for each resource type being claimed.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: resource-claims-rmq-role
  labels:
    servicebinding.io/controller: "true"
rules:
- apiGroups:
  - rabbitmq.com
  resources:
  - rabbitmqclusters
  verbs:
  - get
  - list
  - watch
```

services.tanzu.vmware.com/aggregate-to-provider-kubernetes: "true"

Use this label to aggregate RBAC rules to `provider-kubernetes`, which is a Crossplane `Provider` installed by default as part of the Crossplane package in Tanzu Application Platform. You must grant relevant RBAC permissions for each API Group/Kind used during the creation of `Compositions` as part of setting up dynamic provisioning.

For example, the following `ClusterRole` grants `provider-kubernetes` full control over `rabbitmqclusters` on the `rabbitmq.com` API Group. This allows you to compose `rabbitmqclusters` in `Compositions`. For a full example, see Setup Dynamic Provisioning of Service Instances.

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: rmqcluster-read-writer
  labels:
    services.tanzu.vmware.com/aggregate-to-provider-kubernetes: "true"
rules:
- apiGroups:
  - rabbitmq.com
  resources:
  - rabbitmqclusters
  verbs:
  - "*"
```

services.tanzu.vmware.com/aggregate-to-provider-helm: “true”

Use this label to aggregate RBAC rules to `provider-helm`, which is a Crossplane `Provider` installed by default as part of the Crossplane package in Tanzu Application Platform. You must grant relevant RBAC permissions for each API Group/Kind used during the creation of Helm releases when using the `Release` managed resource as part of `Compositions`.

For example, the following `ClusterRole` grants `provider-helm` full control over `rabbitmqclusters` on the `rabbitmq.com` API Group. This allows you to compose Helm `Releases` which themselves eventually deploy `rabbitmqclusters` in your `Compositions`.

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: rmqcluster-read-writer
  labels:
    services.tanzu.vmware.com/aggregate-to-provider-helm: "true"
rules:
- apiGroups:
  - rabbitmq.com
  resources:
  - rabbitmqclusters
  verbs:
  - "*"
```

The claim verb for ClusterInstanceClass

Services Toolkit supports using the `claim` verb for RBAC rules that apply to a `ClusterInstanceClass`. You can use this with relevant aggregating labels or `ClusterRoleBindings` as a form of access control to specify who can claim from which `ClusterInstanceClass`.

For example:

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: app-operator-claim-class-bigcorp-rabbitmq
  labels:
    # (Optional) Aggregates this ClusterRole to Tanzu Application Platform's
    # app-operator user role at the cluster scope. You can choose to aggregate
    # this to any of the other standard user roles as well.
    apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access: "true"
rules:
# Permits claiming from the 'bigcorp-rabbitmq' class
```

```
- apiGroups:
  - services.apps.tanzu.vmware.com
  resources:
  - clusterinstanceclasses
  resourceNames:
  - bigcorp-rabbitmq
  verbs:
  - claim
```

As of Services Toolkit v0.12.0 in Tanzu Application Platform v1.7.0, you must grant the permission to `claim` from a `ClusterInstanceClass` at the cluster level. You now must use a `ClusterRole` and `ClusterRoleBinding`. Namespace-scoped permissions, such as using a `Role` and `RoleBinding` or `ClusterRole` and `RoleBinding`, are not sufficient. If you used `Roles` and `RoleBindings`, or `ClusterRoles` and `RoleBindings` to grant `claim` permissions in specific namespaces only, this change might affect you. For more information, see Authorize users and groups to claim from provisioner-based classes.

Previously, Services Toolkit allowed you to `claim` from a `ClusterInstanceClass` with only namespace-level permissions. However, this allowed users with only namespace-level permissions to obtain or indirectly deploy resources into namespaces that they do not have access to according to the RBAC permissions.

Tanzu Service CLI plug-in command reference

The Tanzu Service CLI plug-in command reference has moved to the Tanzu CLI Command Reference documentation.

Services Toolkit terminology and user roles

This topic provides descriptions of the terms and user roles used in the Services Toolkit documentation.

Terminology

The following terms are used in the Services Toolkit documentation.

Service

Service is broad, high-level term that describes something used in either the development of, or running of application workloads. Often, but not exclusively, synonymous with the concept of a backing service as defined by the Twelve Factor App:

“… any service the app consumes over the network as part of its normal operation”

For example:

- A PostgreSQL service (implemented as a Kubernetes Operator provided by Tanzu Data Services)
- A PostgreSQL service (implemented as a process running on an Application Developer’s laptop)
- Object storage (implemented as SaaS running on AWS)
- AppSSO

Service resource

A service resource is a Kubernetes resource that provides some of the functions related to a Service.

For example:

- A Kubernetes resource with API Kind `PostgreSQL`
- A Kubernetes resource with API Kind `FirewallRule`
- A Kubernetes resource with API Kind `RabbitmqUser`
- A Kubernetes resource with API Kind `ClientRegistration` that provides access to an App SSO service
- A Kubernetes resource with API Kind `Secret` containing credentials and connectivity information for a Service that may or may not be running on the cluster itself.

Provisioned service

A provisioned service is any service resource that defines a `.status.binding.name` which points to a secret in the same namespace that contains credentials and connectivity information for the resource.

This term is defined in the Service Binding Specification for Kubernetes. For the full definition, see the Service Binding Specification in GitHub.

Service binding

A service binding is a mechanism in which service instance credentials and other related connectivity information are automatically communicated to application workloads.

For example:

- The Service binding concept implemented through the `ServiceBinding` service resource provided by servicebinding in GitHub.

Service instance

A service instance is an abstraction over one or a group of interrelated service resources that together provide the functions for a particular service.

One of the service resources that make up an instance must either adhere to the definition of provisioned service, or be a secret conforming to the service binding specification for Kubernetes. This guarantees that you can claim a service and subsequently bind service instances to application workloads.

You make service instances discoverable through service instance classes.

For example:

- The `RabbitmqCluster` service resource provided by the RabbitMQ Cluster Kubernetes operator. This service resource adheres to provisioned service. Therefore, you can consider any `RabbitmqCluster` resource on a Kubernetes cluster to be a service instance.
- A logical grouping of the following service resources form a single AWS RDS service instance:
    - An AWS RDS `DBInstance`
    - An AWS RDS `DBSubnetGroup`
    - A Carvel `SecretTemplate` configured to produce a secret conforming to the Service Binding Specification for Kubernetes
    - A `Role`, `RoleBinding`, and `ServiceAccount`
- A Kubernetes `Secret` conforming to the Service Binding Specification for Kubernetes containing credentials for a Service running external to the cluster.

Service instance class

A service instance class is more commonly called a class. Service instance classes provide a way to describe categories of service instances.

A service instance class enables service instances belonging to the class to be discovered. They come in one of two varieties: pool-based or provisioner-based.

- Claims for pool-based classes are fulfilled by selecting a service instance from a pool.
- Claims for provisioner-based classes are fulfilled by provisioning new service instances.

Different classes might map to different services or to different configurations of the same service.

For example:

- A `ClusterInstanceClass` named “rabbitmq-dev” pointing to all `RabbitmqCluster` service resources configured with `.spec.replicas=1` identified by label `class: rmq-dev`.
- A `ClusterInstanceClass` named “rabbitmq-prod” pointing to all `RabbitmqCluster` service resources configured with `.spec.replicas=3` identified by label `class: rmq-prod`.
- A `ClusterInstanceClass` named “aws-rds-postgresql” pointing to secrets that conform with the Binding Specification and identified by label `class: aws-rds`.
- A `ClusterInstanceClass` named “mysql-on-demand” which provisions MySQL service instances.

Claim

A claim is a mechanism in which requests for service instances can be declared and fulfilled without requiring detailed knowledge of the service instances themselves.

Claims come in one of two varieties - resource claim and class claim:

- Resource claims refer to a specific service instance.
- Class claims refer to a class from which a service instance is then either selected (pool-based) or provisioned (provisioner-based).

For example:

- A resource claim pointing to a `RabbitmqCluster` service instance named `rmq-1` in the namespace `service-instances`.
- A class claim pointing to a class named `on-demand-rabbitmq`.

Claimable service instance

A claimable service instance is any service instance that you are permitted to claim using a resource claim from a namespace, taking into consideration:

- Location (namespace) of the service instance in relation to the location of the resource claim.
- Any matching resource claim policies.
- Exclusivity of resource claims, that is, you can only claim an instance once.

For example:

- A `RabbitmqCluster` service resource located in the same namespace as a resource claim and that has not already been claimed by another resource claim is a claimable service instance.
- A `RabbitmqCluster` service resource located in a different namespace to a resource claim, for which a matching resource claim policy exists, and has not already been claimed by

another resource claim is a claimable service instance.

- A `RabbitmqCluster` service resource located in the same namespace as a resource claim that has already been claimed is not a claimable service instance due to the exclusive nature of Resource Claims.

Dynamic provisioning

Dynamic provisioning is a capability of Services Toolkit in which class claims that refer to provisioner-based classes are fulfilled automatically through the provisioning of new service instances.

Service resource life cycle API

A service resource life cycle API is any Kubernetes API that you can use to manage the life cycle—create, read, update and delete (CRUD)—of a service resource.

For example:

- `rabbitmqclusters.rabbitmq.com/v1beta1`

Service cluster

A service cluster is applicable within the context of Service API Projection and Service Resource Replication. It is a Kubernetes cluster that has Service Resource Lifecycle APIs installed and a corresponding controller managing their life cycle.

Workload cluster

A workload cluster is applicable within the context of Service API Projection and Service Resource Replication. It is a Kubernetes cluster that has developer-created applications running on it.

User roles

Services Toolkit caters to the following user roles.

These user roles are not user personas. It is possible, and even expected, that one person can be associated with many user roles at any given time. The user roles align to Tanzu Application Platform's user roles. Services Toolkit is primarily responsible for defining the service operator role. For more information about the user roles, see Role descriptions.

The user roles listed in this section consist of a short description and the tasks required. For detailed information about the corresponding Role-Based Access Control (RBAC) associated with each role, see Detailed role permissions breakdown.

Application developer (AD)

The application developer role encompasses both app-editor and app-viewer roles as defined by Tanzu Application Platform. For more information about the app-editor and app-viewer roles, see Role descriptions.

Application developers do the following:

- Bind and unbind application workloads to and from resource claims.
- Get, list, and watch ResourceClaims.
- get, list, and watch ClusterInstanceClasses associated with ResourceClaims.

Application operator (AO)

Encompasses the app-operator role as defined by Tanzu Application Platform. For more information about the app-operator role, see Role descriptions.

Application operators do the following:

- Discover and learn about service instance classes available on a cluster.
- Discover claimable service instances associated with service instance classes.
- Life cycle management (CRUD) of resource claims.

Service operator (SO)

Service operators do the following:

- Life cycle management (CRUD) of service instances.
- Life cycle management (CRUD) of service instance classes.
- Life cycle management (CRUD) of resource claim policies.
- Identify pending resource claims and, if appropriate, help to fulfil such claims through a combination of the previous tasks.
- Setup and configure dynamic provisioning.

Services Toolkit limitations

This topic tells you about the limitations related to working with services on Tanzu Application Platform (commonly known as TAP).

Cannot claim and bind to the same service instance from across multiple namespaces

Two or more workloads located in two or more distinct namespaces cannot claim and bind to the same service instance. This is due to the mutually exclusive nature of claims. After a claim has claimed a service instance, no other claim can then claim that same service instance.

This limitation does not exist for two or more workloads located in the same namespace. In this case, the workloads can all still all bind to one claim. This is not possible across the namespace divide.

Overview of Source Controller

Tanzu Source Controller provides a standard interface for artifact acquisition and extends the function of Flux CD Source Controller.

Tanzu Source Controller supports the following two resource types:

- ImageRepository [Deprecated]
- MavenArtifact

An `ImageRepository` resource can resolve the source from the contents of an image in an image registry. This enables app developers to create and update workloads from local source code or a code repository.

A `MavenArtifact` resource can resolve a binary artifact from a Maven repository. This functionality enables the supply chain to support artifacts produced externally.

Note

Fetching `RELEASE` version from GitHub packages is not currently supported. The `metadata.xml` in GitHub packages does not have the `release` tag that contains the released version number. For more information, see Maven-metadata.xml is corrupted on upload to registry on GitHub.

Overview of Source Controller

Tanzu Source Controller provides a standard interface for artifact acquisition and extends the function of Flux CD Source Controller.

Tanzu Source Controller supports the following two resource types:

- ImageRepository [Deprecated]
- MavenArtifact

An `ImageRepository` resource can resolve the source from the contents of an image in an image registry. This enables app developers to create and update workloads from local source code or a code repository.

A `MavenArtifact` resource can resolve a binary artifact from a Maven repository. This functionality enables the supply chain to support artifacts produced externally.

Note

Fetching `RELEASE` version from GitHub packages is not currently supported. The `metadata.xml` in GitHub packages does not have the `release` tag that contains the released version number. For more information, see Maven-metadata.xml is corrupted on upload to registry on GitHub.

Install Source Controller

This topic tells you how to install Source Controller from the Tanzu Application Platform (commonly known as TAP) package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Source Controller. For more information about profiles, see Components and installation profiles.

Prerequisites

Before installing Source Controller:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- Install cert-manager on the cluster. For more information, see Install cert-manager.

Install

To install Source Controller:

1. List version information for the package by running:

```
tanzu package available list controller.source.apps.tanzu.vmware.com --namespac
e tap-install
```

For example:

```
$ tanzu package available list controller.source.apps.tanzu.vmware.com --namesp
ace tap-install
- Retrieving package versions for controller.source.apps.tanzu.vmware.com...
  NAME                                     VERSION  RELEASED-AT
  controller.source.apps.tanzu.vmware.com  0.3.1    2022-01-23 19:00:00 -0500 -
05
  controller.source.apps.tanzu.vmware.com  0.3.2    2022-02-21 19:00:00 -0500 -
05
  controller.source.apps.tanzu.vmware.com  0.3.3    2022-03-03 19:00:00 -0500 -
05
  controller.source.apps.tanzu.vmware.com  0.4.1    2022-06-09 19:00:00 -0500 -
05
  controller.source.apps.tanzu.vmware.com  0.9.0    2024-03-19 00:00:00 -0500 -
05
```

2. (Optional) Gather the values schema:

```
tanzu package available get controller.source.apps.tanzu.vmware.com/VERSION-NUM
BER --values-schema --namespace tap-install
```

Where `VERSION-NUMBER` is the version of the package listed in step 1 above.

For example:

```
tanzu package available get controller.source.apps.tanzu.vmware.com/0.9.0-buil
d.2 --values-schema --namespace tap-install

  KEY               DEFAULT  TYPE    DESCRIPTION
  resources                           Optional: Tanzu Source Controller resource
limit configuration
  aws_iam_role_arn  ""       string  Optional: Arn role that has access to pull
images from ECR container registry
  ca_cert_data      ""       string  Optional: PEM Encoded certificate data for
image registries with private CA.
```

3. (Optional) Create a file named `source-controller-values.yaml` to override the default installation settings. You can configure the following fields:

    - `ca_cert_data`: Enables Source Controller to connect to image registries that use self-signed or private certificate authorities. If a certificate error `x509: certificate signed by unknown authority` occurs, use this option to trust additional certificate authorities.

      To provide a custom certificate, add the PEM-encoded CA certificate data to `source-controller-values.yaml`. For example:

      ```
      ca_cert_data: |
          -----BEGIN CERTIFICATE-----
          MIICpTCCAYUCBgkqhkiG9w0BBQ0wMzAbBgkqhkiG9w0BBQwwDgQIYg9x6gkCAggA
          ...
          9TlA7A4FFpQqbhAuAVH6KQ8WMZIrVxJSQ03c9lKVkI62wQ==
          -----END CERTIFICATE-----
      ```

    - `aws_iam_role_arn`: Annotates the Source Controller service with an AWS Identity and Access Management (IAM) role. This allows Source Controller to pull images from Amazon Elastic Container Registry (ECR).

To add the AWS IAM role Amazon Resource Name (ARN) to the Source Controller service, add the ARN to `source-controller-values.yaml`. For example:

```
aws_iam_role_arn: "eks.amazonaws.com/role-arn: arn:aws:iam::112233445566:
role/source-controller-manager"
```

- `resources`: Allows you to update the default resource configuration for the Source Controller. By default, the Source Controller resource configuration is set as follows:

  ```
  resources:
    limits:
      cpu: 100m
      memory: 512Mi
    requests:
      cpu: 100m
      memory: 20Mi
  ```

  To update the default resource configuration, add the configuration you require to `source-controller-values.yaml`. For example:

  ```
  resources:
    limits:
      cpu: 100m
      memory: 1Gi
  ```

4. Install the package by running:

   ```
   tanzu package install source-controller \
     --package controller.source.apps.tanzu.vmware.com \
     --version VERSION-NUMBER \
     --namespace tap-install \
     --values-file VALUES-FILE
   ```

   Where:

   - `VERSION-NUMBER` is the version of the package listed in step 1 above.
   - `VALUES-FILE` is the path to the file created in step 3.

   For example:

   ```
   $ tanzu package install source-controller
       --package controller.source.apps.tanzu.vmware.com
       --version 0.9.0-build.2
       --namespace tap-install
       --values-file source-controller-values.yaml

   4:33:18PM: Creating service account 'source-controller-tap-install-sa'
   4:33:18PM: Creating cluster admin role 'source-controller-tap-install-cluster-r
   ole'
   4:33:18PM: Creating cluster role binding 'source-controller-tap-install-cluster
   -rolebinding'
   4:33:18PM: Creating secret 'source-controller-tap-install-values'
   4:33:18PM: Creating overlay secrets
   4:33:18PM: Creating package install resource
   4:33:18PM: Waiting for PackageInstall reconciliation for 'source-controller'
   4:33:18PM: Fetch started (2s ago)
   4:33:20PM: Fetching
         | apiVersion: vendir.k14s.io/v1alpha1
         | directories:
         | - contents:
         |   - imgpkgBundle:
         |       image: dev.registry.tanzu.vmware.com/tanzu-application-platform/c
   ```

```
onstellation/controller.source.apps.tanzu.vmware.com@sha256:ed0925a9533aae03491
07ada38c2a508a6ae4a855b89c3c1c5b4019b706fe1b4
      |     path: .
      |   path: "0"
      | kind: LockConfig
      |
4:33:20PM: Fetch succeeded
4:33:20PM: Template succeeded
4:33:20PM: Deploy started (2s ago)
4:33:22PM: Deploying
      | Target cluster 'https://10.96.0.1:443' (nodes: tap-local-control-plane)
      | Changes
      | Namespace      Name                                                Kind
Age  Op      Op st.  Wait to    Rs  Ri
      | (cluster)      imagerepositories.source.apps.tanzu.vmware.com  CustomRe
sourceDefinition        -    create  -       reconcile  -   -
      | ^              mavenartifacts.source.apps.tanzu.vmware.com     CustomRe
sourceDefinition        -    create  -       reconcile  -   -
      | ^              source-app-viewer                               ClusterR
ole                     -    create  -       reconcile  -   -
      | ^              source-manager-role                             ClusterR
ole                     -    create  -       reconcile  -   -
      | ^              source-manager-rolebinding                      ClusterR
oleBinding              -    create  -       reconcile  -   -
      | ^              source-metrics-reader                           ClusterR
ole                     -    create  -       reconcile  -   -
      | ^              source-proxy-role                               ClusterR
ole                     -    create  -       reconcile  -   -
      | ^              source-proxy-rolebinding                        ClusterR
oleBinding              -    create  -       reconcile  -   -
      | ^              source-system                                   Namespac
e                       -    create  -       reconcile  -   -
      | ^              source-validating-webhook-configuration         Validati
ngWebhookConfiguration  -    create  -       reconcile  -   -
      | source-system  reg-creds                                       Secret
-    create  -       reconcile  -   -
      | ^              source-ca-certificates                          Secret
-    create  -       reconcile  -   -
      | ^              source-controller-manager                       Deployme
nt                      -    create  -       reconcile  -   -
      | ^              source-controller-manager                       ServiceA
ccount                  -    create  -       reconcile  -   -
      | ^              source-controller-manager-artifact-service      Service
-    create  -       reconcile  -   -
      | ^              source-controller-manager-metrics-service       Service
-    create  -       reconcile  -   -
      | ^              source-leader-election-role                     Role
-    create  -       reconcile  -   -
      | ^              source-leader-election-rolebinding              RoleBind
ing                     -    create  -       reconcile  -   -
      | ^              source-manager-config                           ConfigMa
p                       -    create  -       reconcile  -   -
      | ^              source-selfsigned-issuer                        Issuer
-    create  -       reconcile  -   -
      | ^              source-serving-cert                             Certific
ate                     -    create  -       reconcile  -   -
      | ^              source-webhook-service                          Service
-    create  -       reconcile  -   -
      | Op:      22 create, 0 delete, 0 update, 0 noop, 0 exists
      | Wait to: 22 reconcile, 0 delete, 0 noop
      | 8:33:20PM: ---- applying 8 changes [0/22 done] ----
      | ...
      | 8:33:45PM: ok: reconcile deployment/source-controller-manager (apps/v1)
namespace: source-system
      | 8:33:45PM: ---- applying complete [22/22 done] ----
      | 8:33:45PM: ---- waiting complete [22/22 done] ----
```

```
      | Succeeded
4:33:45PM: Deploy succeeded
```

5. Verify the package installation by running:

   ```
   tanzu package installed get source-controller -n tap-install
   ```

   For example:

   ```
   tanzu package installed get source-controller -n tap-install
   NAMESPACE:          tap-install
   NAME:               source-controller
   PACKAGE-NAME:       controller.source.apps.tanzu.vmware.com
   PACKAGE-VERSION:    0.9.0-build.2
   STATUS:             Reconcile succeeded
   CONDITIONS:         - status: "True"
     type: ReconcileSucceeded
   ```

   Verify that `STATUS` is `Reconcile succeeded`:

   ```
   kubectl get pods -n source-system
   ```

   For example:

   ```
   $ kubectl get pods -n source-system
   NAME                                        READY   STATUS    RESTARTS   AGE
   source-controller-manager-f68dc7bb6-4lrn6   1/1     Running   0          100s
   ```

   Verify that `STATUS` is `Running`.

Troubleshoot Source Controller

This topic gives you guidance about how to troubleshoot issues with Source Controller.

Collecting Logs from Source Controller Manager

To retrieve Pod logs from the `controller-manager`, run the following command in the `source-system` namespace:

```
kubectl logs -n source-system -l control-plane=controller-manager
```

For example:

```
kubectl logs -n source-system -l control-plane=controller-manager
2021-11-18T17:59:43.152Z        INFO    controller.imagerepository      Starting Event
Source  {"reconciler group": "source.apps.tanzu.vmware.com", "reconciler kind": "Image
Repository", "source": "kind source: /, Kind="}
2021-11-18T17:59:43.152Z        INFO    controller.metarepository       Starting Event
Source  {"reconciler group": "source.apps.tanzu.vmware.com", "reconciler kind": "MetaR
epository", "source": "kind source: /, Kind="}
2021-11-18T17:59:43.152Z        INFO    controller.metarepository       Starting Event
Source  {"reconciler group": "source.apps.tanzu.vmware.com", "reconciler kind": "MetaR
epository", "source": "kind source: /, Kind="}
2021-11-18T17:59:43.152Z        INFO    controller.metarepository       Starting Event
Source  {"reconciler group": "source.apps.tanzu.vmware.com", "reconciler kind": "MetaR
epository", "source": "kind source: /, Kind="}
2021-11-18T17:59:43.152Z        INFO    controller.metarepository       Starting Contr
oller   {"reconciler group": "source.apps.tanzu.vmware.com", "reconciler kind": "MetaR
epository"}
2021-11-18T17:59:43.152Z        INFO    controller.imagerepository      Starting Event
Source  {"reconciler group": "source.apps.tanzu.vmware.com", "reconciler kind": "Image
```

```
Repository", "source": "kind source: /, Kind="}
2021-11-18T17:59:43.152Z        INFO    controller.imagerepository      Starting Event
Source  {"reconciler group": "source.apps.tanzu.vmware.com", "reconciler kind": "Image
Repository", "source": "kind source: /, Kind="}
2021-11-18T17:59:43.152Z        INFO    controller.imagerepository      Starting Contr
oller   {"reconciler group": "source.apps.tanzu.vmware.com", "reconciler kind": "Image
Repository"}
2021-11-18T17:59:43.389Z        INFO    controller.metarepository       Starting worke
rs      {"reconciler group": "source.apps.tanzu.vmware.com", "reconciler kind": "MetaR
epository", "worker count": 1}
2021-11-18T17:59:43.391Z        INFO    controller.imagerepository      Starting worke
rs      {"reconciler group": "source.apps.tanzu.vmware.com", "reconciler kind": "Image
Repository", "worker count": 1}
```

Source Controller reference

This topic provides reference documentation for Source Controller.

ImageRepository [Deprecated]

```
---
apiVersion: source.apps.tanzu.vmware.com/v1alpha1
kind: ImageRepository
spec:
  image: registry.example/image/repository:tag
  # optional fields
  interval: 5m
  imagePullSecrets: []
  serviceAccountName: default
```

`ImageRepository` resolves source code defined in an Open Container Initiative (OCI) image repository, exposing the resulting source artifact at a URL defined by `.status.artifact.url`.

The interval determines how often to check tagged images for changes. Setting this value too high will result in delays in discovering new sources, while setting it too low may trigger a registry's rate limits.

Repository credentials can be defined as image pull secrets. You can reference them either directly from the resources at `.spec.imagePullSecrets` or attach them to a service account referenced at `.spec.serviceAccountName`. The default service account name `"default"` is used if not otherwise specified. The default credential helpers for the registry are also used, for example, pulling from Google Container Registry (GCR) on a Google Kubernetes Engine (GKE) cluster.

MavenArtifact

```
---
apiVersion: source.apps.tanzu.vmware.com/v1alpha1
kind: MavenArtifact
metadata:
  name: mavenartifact-sample
spec:
  artifact:
    groupId: org.springframework.boot
    artifactId: spring-boot
    version: "2.7.0"
  repository:
    url: https://repo1.maven.org/maven2
  interval: 5m0s
  timeout: 1m0s
```

`MavenArtifact` resolves artifact from a Maven repository, exposing the resulting artifact at a URL defined by `.status.artifact.url`.

The `interval` determines how often to check artifact for changes. Setting this value too high results in delays in discovering new sources, while setting it too low may trigger a repository's rate limits.

Repository credentials may be defined as secrets referenced from the resources at `.spec.repository.secretRef`. Secrets referenced by `spec.repository.secretRef` is parsed as follows:

```
---
apiVersion: v1
kind: Secret
metadata:
  name: auth-secret
type: Opaque
data:
  username: <BASE64>
  password: <BASE64>
  caFile:   <BASE64>   // PEM Encoded certificate data for Custom CA
  certFile: <BASE64>   // PEM-encoded client certificate
  keyFile:  <BASE64>   // Private Key
```

Maven supports a broad set of `version` syntax. Source Controller supports a strict subset of Maven's version syntax in order to ensure compatibility and avoid user confusion. The subset of supported syntax may grow over time, but will never expand past the syntax defined directly by Maven. This behavior means that we can use `mvn` as a reference implementation for artifact resolution.

Version support implemented in the following order:

1. Pinned version - an exact match of a version in2 `maven-metadata.xml (versioning/versions/version)`.
2. `RELEASE` - metaversion defined in `maven-metadata.xml (versioning/release)`.
3. `*-SNAPSHOT` - the newest artifact for a snapshot version.
4. `LATEST` - metaversion defined in `maven-metadata.xml (versioning/latest)`.
5. Version ranges - https://maven.apache.org/enforcer/enforcer-rules/versionRanges.html. Support is planned for a future release.

Note

Pinned versions should be immutable, all other versions are dynamic and can change at any time. The `.spec.interval` defines how frequently to check for updated artifacts.

Overview of Spring Boot conventions

This topic tells you about the Spring Boot conventions component.

Spring Boot conventions enriches Spring Boot application deployments in a Tanzu Application Platform supply chain by mutating the `PodSpec` and adding or editing some security checks, probes, and other features to provide a production-ready application.

Spring Boot conventions also sets the relevant Application Live View labels automatically on the `PodSpec` so that Application Live View is enabled by default for a Spring Boot application.

Spring Boot conventions comprises many conventions applied to any Spring Boot application that is submitted to the supply chain in which the convention controller is configured. The conventions include:

- Setting liveness/readiness actuator probes
- Enabling the automatic configuration of actuators on the platform and workload levels
- Setting Prometheus annotations when `micrometer-registry-prometheus` dependency exists on the classpath
- Setting `JAVA_TOOL_OPTIONS` and Application Live View labels for a Spring Boot application

Important

Spring Boot conventions supports Spring Boot and Spring Cloud Gateway applications. Application Live View conventions supports only Steeltoe applications. For more information about Application Live View conventions, see Application Live View convention server.

Overview of Spring Boot conventions

This topic tells you about the Spring Boot conventions component.

Spring Boot conventions enriches Spring Boot application deployments in a Tanzu Application Platform supply chain by mutating the `PodSpec` and adding or editing some security checks, probes, and other features to provide a production-ready application.

Spring Boot conventions also sets the relevant Application Live View labels automatically on the `PodSpec` so that Application Live View is enabled by default for a Spring Boot application.

Spring Boot conventions comprises many conventions applied to any Spring Boot application that is submitted to the supply chain in which the convention controller is configured. The conventions include:

- Setting liveness/readiness actuator probes
- Enabling the automatic configuration of actuators on the platform and workload levels
- Setting Prometheus annotations when `micrometer-registry-prometheus` dependency exists on the classpath
- Setting `JAVA_TOOL_OPTIONS` and Application Live View labels for a Spring Boot application

Important

Spring Boot conventions supports Spring Boot and Spring Cloud Gateway applications. Application Live View conventions supports only Steeltoe applications. For more information about Application Live View conventions, see Application Live View convention server.

Install Spring Boot conventions

This topic tells you how to install Spring Boot conventions from the Tanzu Application Platform package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Spring Boot conventions. For more information about profiles, see Components and installation profiles.

Prerequisites

Before installing Spring Boot conventions:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- Install Supply Chain Choreographer.

Install Spring Boot conventions

To install Spring Boot conventions:

1. Get the exact name and version information for the Spring Boot conventions package to install by running:

   ```
   tanzu package available list spring-boot-conventions.tanzu.vmware.com --namespa
   ce tap-install
   ```

   For example:

   ```
   $ tanzu package available list spring-boot-conventions.tanzu.vmware.com --names
   pace tap-install
   / Retrieving package versions for spring-boot-conventions.tanzu.vmware.com...
   NAME                                        VERSION           RELEASED-AT
   ...
   spring-boot-conventions.tanzu.vmware.com    1.9.0     2024-03-22T00:00:00Z
   ...
   ```

2. (Optional) Change the default installation settings by running:

   ```
   tanzu package available get spring-boot-conventions.tanzu.vmware.com/VERSION-NU
   MBER \
   --values-schema --namespace tap-install
   ```

   Where `VERSION-NUMBER` is the version of the package listed. For example: `1.9.1`.

   For example:

   ```
   $ tanzu package available get spring-boot-conventions.tanzu.vmware.com/1.8.0-bu
   ild.1 --values-schema --namespace tap-install
        KEY                                          DEFAULT             TYPE
   DESCRIPTION
        autoConfigureActuators                       false               boolean
   Enable or disable the automatic configuration of actuators on the TAP platform
   level
        kubernetes_distribution                                          string
   Kubernetes distribution that this package is being installed on. Accepted

   values: [''',''openshift'']
        kubernetes_version                                               string
   Optional: The Kubernetes Version. Valid values are '1.24.*', or ''
        livenessProbe.terminationGracePeriodSeconds                      number
   configure a grace period for the kubelet to wait  between triggering a shut dow
   n

   of the failed container, and then forcing the container runtime to stop that
   ```

```
container
     livenessProbe.timeoutSeconds                1                   number
Number of seconds after which the probe times out
     livenessProbe.failureThreshold                                  number
After a probe fails failureThreshold times in a row, Kubernetes considers that

the overall check has failed
     livenessProbe.initialDelaySeconds           0                   number
Number of seconds after the container has started before liveness probes are

initiated
     livenessProbe.periodSeconds                 10                  number
How often (in seconds) to perform the probe
     livenessProbe.successThreshold              1                   number
Minimum consecutive successes for the probe to be considered successful after

having failed
     readinessProbe.initialDelaySeconds            0                 number
Number of seconds after the container has started before readiness probes are

initiated
     readinessProbe.periodSeconds                  10                number
How often (in seconds) to perform the probe
     readinessProbe.successThreshold               1                 number
Minimum consecutive successes for the probe to be considered successful after

having failed
     readinessProbe.terminationGracePeriodSeconds                    number
configure a grace period for the kubelet to wait between triggering a shut down

of the failed container, and then forcing the container runtime to stop that

container
     readinessProbe.timeoutSeconds                 1                 number
Number of seconds after which the probe times out
     readinessProbe.failureThreshold                                 number
After a probe fails failureThreshold times in a row, Kubernetes considers that

the overall check has failed
     startupProbe.periodSeconds                    10                number
How often (in seconds) to perform the probe
     startupProbe.successThreshold                 1                 number
Minimum consecutive successes for the probe to be considered successful after

having failed
     startupProbe.terminationGracePeriodSeconds                      number
configure a grace period for the kubelet to wait between triggering a shut down

of the failed container, and then forcing the container runtime to stop that

container
     startupProbe.timeoutSeconds                   1                 number
Number of seconds after which the probe times out
     startupProbe.failureThreshold                                   number
After a probe fails failureThreshold times in a row, Kubernetes considers that

the overall check has failed
     startupProbe.initialDelaySeconds              0                 number
Number of seconds after the container has started before probes are initiated
```

For more information about configuring probes in Spring Boot conventions, see Configure liveness, readiness, and startup probes for Spring Boot applications (alpha)

3. Install the package by running:

```
tanzu package install spring-boot-conventions \
 --package spring-boot-conventions.tanzu.vmware.com \
 --version VERSION-NUMBER \
 --namespace tap-install
```

Where `VERSION-NUMBER` is the version of the package you listed earlier.

4. Verify you installed the package by running:

```
tanzu package installed get spring-boot-conventions --namespace tap-install
```

For example:

```
tanzu package installed get spring-boot-conventions -n tap-install
| Retrieving installation details for spring-boot-conventions...
NAME:                    spring-boot-conventions
PACKAGE-NAME:            spring-boot-conventions.tanzu.vmware.com
PACKAGE-VERSION:         1.9.1
STATUS:                  Reconcile succeeded
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`

Configure and access Spring Boot actuators in Tanzu Application Platform

This topic tells you how the Spring Boot conventions in Tanzu Application Platform configure Spring Boot actuators automatically. With this feature, users can activate or deactivate the automatic configuration of actuators on Tanzu Application Platform and on individual workloads.

Workload-level configuration

Developers can add a label to their workloads to activate or deactivate the automatic configuration of actuators. By default, all existing and future accelerator projects are configured to activate automatic configuration on the workload level.

To activate or deactivate the automatic configuration of actuators at the workload level:

- To activate automatic configuration of actuators, set the following label to `true` in your workload YAML:

  `apps.tanzu.vmware.com/auto-configure-actuators: "true"`

  If the preceding label is set to `true`, the Spring Boot actuator convention sets the following actuator configuration:

  - The `JAVA_TOOL_OPTIONS` property is set as `-Dmanagement.server.port="8081"`.
  - The `JAVA_TOOL_OPTIONS` property is set as `-Dmanagement.endpoints.web.base-path="/actuator"`.
  - Annotation on the PodIntent is set as `boot.spring.io/actuator: http://:8081/actuator`.

  In addition to these settings, Application Live View is activated with the following actuator configuration:

  - Label on the PodIntent is set as `tanzu.app.live.view.application.actuator: actuator`.

    - Label on the PodIntent is set as `tanzu.app.live.view.application.actuator.port: 8081`.

- To deactivate automatic configuration of actuators, set the following label to `false` in your workload YAML:

  `apps.tanzu.vmware.com/auto-configure-actuators: "false"`

  If the preceding label is set to `false`, the Spring Boot actuator convention does not set any `JAVA_TOOL_OPTIONS` and does not set the annotation `boot.spring.io/actuator`.

  Application Live View is activated and configured with default values for Spring Boot web applications, assuming that some actuators are activated on the default port. On activating Application Live View, the following actuator settings are set:

    - The `JAVA_TOOL_OPTIONS` property is set as `-Dserver.port="8080"`.
    - Label on the PodIntent is set as `tanzu.app.live.view.application.actuator: actuator`.
    - Label on the PodIntent is set as `tanzu.app.live.view.application.actuator.port: 8080`.

The Application Live View GUI renders the pages with accessible information based on whether the actuator endpoints are accessible for an application.

By default, as an additional security measure, Spring Boot conventions does not expose all the actuator data over HTTP by exposing all the actuator endpoints. In addition, the information exposed by the health endpoint is not set to `always` by default.

If the automatic configuration of actuators is set to `true` either at the workload level or platform level, the Spring Boot convention then sets the runtime environment properties `management.endpoints.web.exposure.include="*"` and `management.endpoint.health.show-details=true` on to the PodSpec to expose all the actuator endpoints and detailed health information. You do not need to add these properties manually in `application.properties` or `application.yml`.

Platform-level configuration

In contrast to activating or deactivating the automatic configuration of actuators on the level of individual workloads, you can set a global setting for the platform instead. This setting is taken into account ONLY when there is no specific `auto-configure-actuators` setting on the individual workload.

To activate or deactivate the automatic configuration of actuators at a global level:

- To activate the automatic configuration, when you install Spring Boot conventions, provide an entry in the `values.yaml` file. For example:

  ```
  springboot_conventions:
    autoConfigureActuators: true
  ```

- To deactivate the automatic configuration, you can provide the following entry:

  ```
  springboot_conventions:
    autoConfigureActuators: false
  ```

Note

The default values for both platform level and workload level configuration is false.

To run Application Live View with Spring Boot apps, the Spring Boot convention recognizes PodIntents and adds the following metadata labels:

- `tanzu.app.live.view: "true"`: Activates the connector to observe application pod
- `tanzu.app.live.view.application.name: APPLICATION-NAME`: Identifies the app name to be used internally by Application Live View
- `tanzu.app.live.view.application.actuator.port: ACTUATOR-PORT`: Identifies the port on the pod at which the actuators are available for Application Live View
- `tanzu.app.live.view.application.flavours: spring-boot`: Exposes the framework flavor of the app

To run Application Live View with Spring Cloud Gateway apps, Spring Boot conventions recognizes PodIntents and adds the following metadata labels:

- `tanzu.app.live.view: "true"`: Activates the connector to observe application pod
- `tanzu.app.live.view.application.name: APPLICATION-NAME`: Identifies the app name to be used internally by Application Live View
- `tanzu.app.live.view.application.actuator.port: ACTUATOR-PORT`: Identifies the port on the pod at which the actuators are available for Application Live View
- `tanzu.app.live.view.application.flavours: spring-boot,spring-cloud-gateway`: Exposes the framework flavors of the app

Configure liveness, readiness, and startup probes for Spring Boot applications (alpha)

This topic tells you how to override the liveness, readiness, and startup probes' settings for Spring Boot apps in Tanzu Application Platform. You can override the Kubernetes default probe settings in `spring-boot-conventions` for containers on Tanzu Application Platform.

Caution

This feature is in alpha testing. It is intended for evaluation and test purposes only.

Overview of the probes

Kubernetes provides built-in mechanisms for checking the health and readiness of your Spring Boot apps. By default, Kubernetes uses specific endpoints provided by Spring Boot Actuator for liveness and readiness checks. However, you might need to customize these probes to align them with your app's specific requirements.

Probe	Function
Liveness	Indicates whether the app is running and responsive. If the probe fails, Kubernetes might restart the container.
Readiness	Indicates whether the app is ready to serve requests. If the probe fails, Kubernetes won't route traffic to the container.
Startup	Indicates when the app has completed its initialization and is ready to handle traffic. This is especially useful for apps with a slow startup process.

For more information about the probes, see the Kubernetes documentation.

Override the default configurations of the probes

To override the default settings and customize the probes at a cluster level:

1. List the schema for the `spring-boot-conventions` package by running:

```
tanzu package available get spring-boot-conventions.tanzu.vmware.com/VERSION-NU
MBER \
--values-schema --namespace tap-install
```

Where `VERSION-NUMBER` is the version of the package listed. For example: `1.8.0-build.1`.

For example:

```
$ tanzu package available get spring-boot-conventions.tanzu.vmware.com/1.8.0-bu
ild.1 --values-schema --namespace tap-install
    KEY                                          DEFAULT             TYPE     D
ESCRIPTION
    autoConfigureActuators                       false               boolean  E
nable or disable the automatic configuration of actuators on the TAP platform l
evel
    kubernetes_distribution                                          string   K
ubernetes distribution that this package is being installed on. Accepted

values: [''''],''openshift'']
    kubernetes_version                                               string   O
ptional: The Kubernetes Version. Valid values are '1.24.*', or ''
    livenessProbe.terminationGracePeriodSeconds                      number   c
onfigure a grace period for the kubelet to wait  between triggering a shut down

of the failed container, and then forcing the container runtime to stop that

container
    livenessProbe.timeoutSeconds                 1                   number   N
umber of seconds after which the probe times out
    livenessProbe.failureThreshold                                   number   A
fter a probe fails failureThreshold times in a row, Kubernetes considers that

the overall check has failed
    livenessProbe.initialDelaySeconds            0                   number   N
umber of seconds after the container has started before liveness probes are

initiated
    livenessProbe.periodSeconds                  10                  number   H
ow often (in seconds) to perform the probe
    livenessProbe.successThreshold               1                   number   M
inimum consecutive successes for the probe to be considered successful after

having failed
    readinessProbe.initialDelaySeconds             0                 number   N
umber of seconds after the container has started before readiness probes are

initiated
    readinessProbe.periodSeconds                   10                number   H
ow often (in seconds) to perform the probe
    readinessProbe.successThreshold                1                 number   M
inimum consecutive successes for the probe to be considered successful after

having failed
    readinessProbe.terminationGracePeriodSeconds                     number   c
onfigure a grace period for the kubelet to wait between triggering a shut down

of the failed container, and then forcing the container runtime to stop that

container
    readinessProbe.timeoutSeconds                  1                 number   N
umber of seconds after which the probe times out
    readinessProbe.failureThreshold                                  number   A
```

```
fter a probe fails failureThreshold times in a row, Kubernetes considers that

the overall check has failed
    startupProbe.periodSeconds                      10                number   H
ow often (in seconds) to perform the probe
    startupProbe.successThreshold                   1                 number   M
inimum consecutive successes for the probe to be considered successful after

having failed
    startupProbe.terminationGracePeriodSeconds                        number   c
onfigure a grace period for the kubelet to wait between triggering a shut down

of the failed container, and then forcing the container runtime to stop that

container
    startupProbe.timeoutSeconds                     1                 number   N
umber of seconds after which the probe times out
    startupProbe.failureThreshold                                     number   A
fter a probe fails failureThreshold times in a row, Kubernetes considers that

the overall check has failed
    startupProbe.initialDelaySeconds                0                 number   N
umber of seconds after the container has started before probes are initiated
```

2. When you install Spring Boot conventions, provide an entry in the `values.yaml` file to override the configuration.

   For example:

   ```
   springboot_conventions:
     autoConfigureActuators: true
     livenessProbe:
       initialDelaySeconds: 11
       periodSeconds: 12
       timeoutSeconds: 13
       terminationGracePeriodSeconds: 14
     readinessProbe:
       initialDelaySeconds: 15
       periodSeconds: 12
       successThreshold: 3
       failureThreshold: 4
   ```

3. Install the package by running:

   ```
   tanzu package install spring-boot-conventions \
    --package spring-boot-conventions.tanzu.vmware.com \
    --version VERSION-NUMBER \
    --namespace tap-install \
    --values-file values.yaml
   ```

   Where `VERSION-NUMBER` is the version of the package to install. For example: `1.8.0-build.1`.

4. Verify that you installed the package by running:

   ```
   tanzu package installed get spring-boot-conventions --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package installed get spring-boot-conventions -n tap-install
   | Retrieving installation details for spring-boot-conventions...
   NAME:                    spring-boot-conventions
   PACKAGE-NAME:            spring-boot-conventions.tanzu.vmware.com
   PACKAGE-VERSION:         1.8.0-build.1
   STATUS:                  Reconcile succeeded
   ```

```
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`

The Spring Boot convention applies the probe and timeout settings at the cluster level to all Spring Boot and Spring Native workloads.

5. Verify the `PodIntent` of your workload by ensuring that `spec.containers.livenessProbe` shows the overridden configuration values propagated through the Spring Boot conventions:

```
kubectl get podintents.conventions.carto.run tanzu-java-web-app -o yaml

spec:
 containers:
 - env:
   - name: JAVA_TOOL_OPTIONS
     value: -Dmanagement.endpoint.health.probes.add-additional-paths="true" -Dm
anagement.endpoint.health.show-details="always"
       -Dmanagement.endpoints.web.base-path="/actuator" -Dmanagement.endpoints.
web.exposure.include="*"
       -Dmanagement.health.probes.enabled="true" -Dmanagement.server.port="808
1"
       -Dserver.port="8080" -Dserver.shutdown.grace-period="24s"
   image: dev.registry.tanzu.vmware.com/app-live-view/test/tanzu-java-web-app-d
efault@sha256:fa822a6585eb1287a817a956f16d77dd391624462a626bf37bbf0f9e89ff7562
   livenessProbe:
     httpGet:
       path: /livez
       port: 8080
       scheme: HTTP
     initialDelaySeconds: 11
     periodSeconds: 12
     terminationGracePeriodSeconds: 14
     timeoutSeconds: 13
   name: workload
   ports:
   - containerPort: 8080
     protocol: TCP
   readinessProbe:
     failureThreshold: 4
     httpGet:
       path: /readyz
       port: 8080
       scheme: HTTP
     initialDelaySeconds: 15
     periodSeconds: 12
     successThreshold: 3
   resources: {}
   securityContext:
     runAsUser: 1000
   startupProbe:
     httpGet:
       path: /startupz
       port: 8080
       scheme: HTTP
```

Setting additional paths for the probes

The Spring Boot convention automatically sets `management.endpoint.health.probes.add-additional-paths` to `true` if the Spring Boot application is v2.6.0 or later. For any versions earlier than v2.6.0, `management.endpoint.health.probes.add-additional-paths` is not supported.

The `spring-boot-actuator-probes` convention in Spring Boot conventions sets the liveness, readiness, and startup probe endpoints to `/livez`, `/readyz`, and `readyz` respectively. These endpoints act as additional endpoint paths on the actuator probes.

For example, when you run `https://tanzu-java-web-app.default.apps.20.22.106.199.nip.io/actuator/health/livez` to monitor the liveness probe, you can view the health of the application. This helps you evaluate whether an application that is running in a container is in a healthy state.

Similarly, when running `https://tanzu-java-web-app.default.apps.20.22.106.199.nip.io/actuator/health/readyz` to view the readiness probe and the startup probe, you can evaluate whether the application is ready to receive traffic and whether the application has started up properly.

Enable Application Live View for Spring Boot applications

To run Application Live View for Spring Boot apps, Spring Boot conventions recognizes PodIntents and automatically adds the following metadata labels:

- `tanzu.app.live.view: "true"`: Enables the connector to observe application pod
- `tanzu.app.live.view.application.name: APPLICATION-NAME`: Identifies the app name to be used internally by Application Live View
- `tanzu.app.live.view.application.actuator.port: ACTUATOR-PORT`: Identifies the port on the pod at which the actuators are available for Application Live View
- `tanzu.app.live.view.application.flavours: spring-boot`: Exposes the framework flavor of the app

To run Application Live View for Spring Cloud Gateway apps, Spring Boot conventions recognizes PodIntents and adds the following metadata labels:

- `tanzu.app.live.view: "true"`: Enables the connector to observe application pod
- `tanzu.app.live.view.application.name: APPLICATION-NAME`: Identifies the app name to be used internally by Application Live View
- `tanzu.app.live.view.application.actuator.port: ACTUATOR-PORT`: Identifies the port on the pod at which the actuators are available for Application Live View
- `tanzu.app.live.view.application.flavours: spring-boot,spring-cloud-gateway`: Exposes the framework flavors of the app

These metadata labels allow Application Live View to identify pods that are enabled for Application Live View. The metadata labels also tell the Application Live View connector what kind of app it is and on which port the actuators are accessible for Application Live View. For more information, see Configuring and accessing Spring Boot actuators in Tanzu Application Platform.

Verify the applied labels and annotations

To verify the applied labels and annotations, run:

```
kubectl get podintents.conventions.carto.run WORKLOAD-NAME -o yaml
```

Where `WORKLOAD-NAME` is the name of the deployed workload. For example: `tanzu-java-web-app`.

Expected output of Spring Boot workload:

```
apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
```

```
  creationTimestamp: "2022-11-14T10:07:55Z"
  generation: 1
  labels:
    app.kubernetes.io/component: intent
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/auto-configure-actuators: "true"
    apps.tanzu.vmware.com/workload-type: web
    carto.run/cluster-template-name: convention-template
    carto.run/resource-name: config-provider
    carto.run/supply-chain-name: source-to-url
    carto.run/template-kind: ClusterConfigTemplate
    carto.run/workload-name: tanzu-java-web-app
    carto.run/workload-namespace: default
  name: tanzu-java-web-app
  namespace: default
  ownerReferences:
  - apiVersion: carto.run/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: Workload
    name: tanzu-java-web-app
    uid: dfd3c0c2-9d1f-4231-9390-3e16f23bb62d
  resourceVersion: "444497"
  uid: 224de2aa-307a-48e3-a826-2c474c435bb2
spec:
  serviceAccountName: default
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/min-scale: "1"
        developer.conventions/target-containers: workload
      labels:
        app.kubernetes.io/component: run
        app.kubernetes.io/part-of: tanzu-java-web-app
        apps.tanzu.vmware.com/auto-configure-actuators: "true"
        apps.tanzu.vmware.com/workload-type: web
        carto.run/workload-name: tanzu-java-web-app
    spec:
      containers:
      - image: dev.registry.tanzu.vmware.com/app-live-view/test/tanzu-java-web-app-def
ault@sha256:444686bb8bfbaba5552676140619b00f43c8f85b6823b87676c0ccdcdead65ac
        name: workload
        resources: {}
        securityContext:
          runAsUser: 1000
      serviceAccountName: default
status:
  conditions:
  - lastTransitionTime: "2022-11-14T10:07:59Z"
    message: ""
    reason: Applied
    status: "True"
    type: ConventionsApplied
  - lastTransitionTime: "2022-11-14T10:07:59Z"
    message: ""
    reason: ConventionsApplied
    status: "True"
    type: Ready
  observedGeneration: 1
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/min-scale: "1"
        boot.spring.io/actuator: http://:8081/actuator
        boot.spring.io/version: 2.7.3
        conventions.carto.run/applied-conventions: |-
```

```
          spring-boot-convention/auto-configure-actuators-check
          spring-boot-convention/spring-boot
          spring-boot-convention/spring-boot-graceful-shutdown
          spring-boot-convention/spring-boot-web
          spring-boot-convention/spring-boot-actuator
          spring-boot-convention/spring-boot-actuator-probes
          spring-boot-convention/app-live-view-appflavour-check
          spring-boot-convention/app-live-view-connector-boot
          spring-boot-convention/app-live-view-appflavours-boot
        developer.conventions/target-containers: workload
      labels:
        app.kubernetes.io/component: run
        app.kubernetes.io/part-of: tanzu-java-web-app
        apps.tanzu.vmware.com/auto-configure-actuators: "true"
        apps.tanzu.vmware.com/workload-type: web
        carto.run/workload-name: tanzu-java-web-app
        conventions.carto.run/framework: spring-boot
        tanzu.app.live.view: "true"
        tanzu.app.live.view.application.actuator.path: actuator
        tanzu.app.live.view.application.actuator.port: "8081"
        tanzu.app.live.view.application.flavours: spring-boot
        tanzu.app.live.view.application.name: tanzu-java-web-app
    spec:
      containers:
      - env:
        - name: JAVA_TOOL_OPTIONS
          value: -Dmanagement.endpoint.health.probes.add-additional-paths="true" -Dman
agement.endpoint.health.show-details="always"
            -Dmanagement.endpoints.web.base-path="/actuator" -Dmanagement.endpoints.we
b.exposure.include="*"
            -Dmanagement.health.probes.enabled="true" -Dmanagement.server.port="8081"
            -Dserver.port="8080" -Dserver.shutdown.grace-period="24s"
        image: dev.registry.tanzu.vmware.com/app-live-view/test/tanzu-java-web-app-def
ault@sha256:444686bb8bfbaba5552676140619b00f43c8f85b6823b87676c0ccdcdead65ac
        livenessProbe:
          httpGet:
            path: /livez
            port: 8080
            scheme: HTTP
        name: workload
        ports:
        - containerPort: 8080
          protocol: TCP
        readinessProbe:
          httpGet:
            path: /readyz
            port: 8080
            scheme: HTTP
        resources: {}
        securityContext:
          runAsUser: 1000
      serviceAccountName: default
```

Expected output of Spring Cloud Gateway workload:

```
apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
  creationTimestamp: "2022-11-14T10:29:51Z"
  generation: 1
  labels:
    app.kubernetes.io/component: intent
    app.kubernetes.io/part-of: tanzu-scg-web-app
    apps.tanzu.vmware.com/auto-configure-actuators: "true"
    apps.tanzu.vmware.com/workload-type: web
    carto.run/cluster-template-name: convention-template
```

```
    carto.run/resource-name: config-provider
    carto.run/supply-chain-name: source-to-url
    carto.run/template-kind: ClusterConfigTemplate
    carto.run/workload-name: tanzu-scg-web-app
    carto.run/workload-namespace: default
  name: tanzu-scg-web-app
  namespace: default
  ownerReferences:
  - apiVersion: carto.run/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: Workload
    name: tanzu-scg-web-app
    uid: 5d8cdc5b-0236-471d-8c1e-335e659f1ae6
  resourceVersion: "475756"
  uid: d086f02c-6ff0-47f8-8dee-4da8748d8adc
spec:
  serviceAccountName: default
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/min-scale: "1"
        developer.conventions/target-containers: workload
      labels:
        app.kubernetes.io/component: run
        app.kubernetes.io/part-of: tanzu-scg-web-app
        apps.tanzu.vmware.com/auto-configure-actuators: "true"
        apps.tanzu.vmware.com/workload-type: web
        carto.run/workload-name: tanzu-scg-web-app
    spec:
      containers:
      - image: dev.registry.tanzu.vmware.com/app-live-view/test/tanzu-scg-web-app-defa
ult@sha256:7656f4ca56b7d0d6376b374643d6ac09c8cdcdbcc13d065f9224651b12724d0b
        name: workload
        resources: {}
        securityContext:
          runAsUser: 1000
      serviceAccountName: default
status:
  conditions:
  - lastTransitionTime: "2022-11-14T10:29:58Z"
    message: ""
    reason: Applied
    status: "True"
    type: ConventionsApplied
  - lastTransitionTime: "2022-11-14T10:29:58Z"
    message: ""
    reason: ConventionsApplied
    status: "True"
    type: Ready
  observedGeneration: 1
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/min-scale: "1"
        boot.spring.io/actuator: http://:8081/actuator
        boot.spring.io/version: 2.6.3
        conventions.carto.run/applied-conventions: |-
          spring-boot-convention/auto-configure-actuators-check
          spring-boot-convention/spring-boot
          spring-boot-convention/spring-boot-web
          spring-boot-convention/spring-boot-actuator
          spring-boot-convention/spring-boot-actuator-probes
          spring-boot-convention/app-live-view-appflavour-check
          spring-boot-convention/app-live-view-connector-boot
          spring-boot-convention/app-live-view-appflavours-boot
```

```
          spring-boot-convention/app-live-view-connector-scg
          spring-boot-convention/app-live-view-appflavours-scg
        developer.conventions/target-containers: workload
      labels:
        app.kubernetes.io/component: run
        app.kubernetes.io/part-of: tanzu-scg-web-app
        apps.tanzu.vmware.com/auto-configure-actuators: "true"
        apps.tanzu.vmware.com/workload-type: web
        carto.run/workload-name: tanzu-scg-web-app
        conventions.carto.run/framework: spring-boot
        tanzu.app.live.view: "true"
        tanzu.app.live.view.application.actuator.path: actuator
        tanzu.app.live.view.application.actuator.port: "8081"
        tanzu.app.live.view.application.flavours: spring-boot_spring-cloud-gateway
        tanzu.app.live.view.application.name: tanzu-scg-web-app
    spec:
      containers:
      - env:
        - name: JAVA_TOOL_OPTIONS
          value: -Dmanagement.endpoint.health.probes.add-additional-paths="true" -Dman
agement.endpoint.health.show-details="always"
            -Dmanagement.endpoints.web.base-path="/actuator" -Dmanagement.endpoints.we
b.exposure.include="*"
            -Dmanagement.health.probes.enabled="true" -Dmanagement.server.port="8081"
            -Dserver.port="8080"
        image: dev.registry.tanzu.vmware.com/app-live-view/test/tanzu-scg-web-app-defa
ult@sha256:7656f4ca56b7d0d6376b374643d6ac09c8cdcdbcc13d065f9224651b12724d0b
        livenessProbe:
          httpGet:
            path: /livez
            port: 8080
            scheme: HTTP
        name: workload
        ports:
        - containerPort: 8080
          protocol: TCP
        readinessProbe:
          httpGet:
            path: /readyz
            port: 8080
            scheme: HTTP
        resources: {}
        securityContext:
          runAsUser: 1000
      serviceAccountName: default
```

Enable Spring Native apps for Application Live View

This topic tells you how to run Spring Native workloads within Tanzu Application Platform (commonly known as TAP).

This topic describes how the Spring Boot convention server enhances Tanzu `PodIntents` with metadata. This metadata can include labels, annotations, or properties required to run native workloads in Tanzu Application Platform.

The metadata enables Application Live View to discover and register the app instances so that Application Live View can access the actuator data from those workloads.

Configure a Spring Native application

To make Application Live View interact with a Spring Native app within Tanzu Application Platform:

1. Add the `spring-boot-starter-actuator` module dependency for Maven in `pom.xml` as follows:

```
<dependency>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

2. Add a native profile that includes the `native-maven-plugin` for the build phase in `pom.xml`, as follows:

```
<profiles>
 <profile>
     <id>native</id>
     <build>
       <plugins>
           <plugin>
             <groupId>org.graalvm.buildtools</groupId>
               <artifactId>native-maven-plugin</artifactId>
             </plugin>
         </plugins>
     </build>
 </profile>
</profiles>
```

Important

For Maven-based projects, you must add the `org.graalvm.buildtools` build plug-in and configure it to run when the native profile is active.

3. Add the following configuration in `pom.xml` to generate `build-info.properties` in your Spring Boot application. Application Live View uses this information to display the Spring Boot version that the app uses.

```
<plugin>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-maven-plugin</artifactId>
<executions>
 <execution>
   <goals>
     <goal>build-info</goal>
   </goals>
   <configuration>
     <additionalProperties>
       <spring.boot.version>${project.parent.version}</spring.boot.version>
     </additionalProperties>
   </configuration>
 </execution>
</executions>
</plugin>
```

Create a native workload

Create a workload for the Spring Native application similar to the following example:

```
$ tanzu apps workload create spring-cloud-serverless \
--git-repo https://github.com/vudayani-vmw/spring-cloud-serverless --git-branch main \
--type web --label apps.tanzu.vmware.com/auto-configure-actuators=true \
--label app.kubernetes.io/part-of=spring-cloud-serverless --yes \
```

```
--annotation autoscaling.knative.dev/min-scale=1 \
--build-env "BP_JVM_VERSION=17" --build-env "BP_NATIVE_IMAGE=true" \
--build-env "BP_MAVEN_BUILD_ARGUMENTS= -Pnative -Dmaven.test.skip=true \
--no-transfer-progress package \
-Dspring-boot.aot.jvmArguments='-Dmanagement.endpoint.health.probes.add-additional-pat
hs='true' \
-Dmanagement.endpoint.health.show-details='always' \
-Dmanagement.endpoints.web.base-path='/actuator' \
-Dmanagement.endpoints.web.exposure.include='*' \
-Dmanagement.health.probes.enabled='true' \
-Dmanagement.server.port=8081 -Dserver.port=8080' "
```

Where:

- The required build arguments, such as `BP_JVM_VERSION`, `BP_NATIVE_IMAGE`, and `BP_MAVEN_BUILD_ARGUMENTS`, are configured to build a native executable file as shown in the example. The build environment variables are required to enable a native build profile and native image-building.
- The override configuration options at runtime provide AOT (Ahead-Of-Time) configuration to the build process by using the `spring-boot.aot.jvmArguments` option. For Gradle builds, the `spring-boot.aot.jvmArguments` is set as part of the `BP_GRADLE_ADDITIONAL_BUILD_ARGUMENTS` build environment variable.

These build environment parameters do not directly set the runtime values. The build environment parameters, provided as part of `spring-boot.aot.jvmArguments`, are only used as build-time signals. These signals indicate to generate code that allows configuration options to be set at runtime.

Configure at runtime

There are two methods to override the configuration at runtime. You can, by default, let Spring Boot conventions configure automatically, or you can use environment variables to configure manually.

Configure automatically (default)
Automatic configuration by Spring Boot conventions is the default method. Spring Boot conventions add the necessary environment variables and labels to the workload `PodSpec` based on the `apps.tanzu.vmware.com/auto-configure-actuators` flag. When the `apps.tanzu.vmware.com/auto-configure-actuators` is set to `true`, Spring Boot conventions adds the following environment variables to the native workload `PodSpec`:

```
Spec:
  Containers:
    Env:
      Name:   MANAGEMENT_SERVER_PORT
      Value:  8081
      Name:   MANAGEMENT_ENDPOINT_HEALTH_PROBES_ADD_ADDITIONAL_PATHS
      Value:  true
      Name:   MANAGEMENT_ENDPOINT_HEALTH_SHOW_DETAILS
      Value:  always
      Name:   MANAGEMENT_ENDPOINTS_WEB_BASE_PATH
      Value:  /actuator
      Name:   MANAGEMENT_ENDPOINTS_WEB_EXPOSURE_INCLUDE
      Value:  *
      Name:   MANAGEMENT_HEALTH_PROBES_ENABLED
      Value:  true
      Name:   SERVER_PORT
      Value:  8080
      Name:   SERVER_SHUTDOWN_GRACE_PERIOD
      Value:  24s
```

Configure manually

You can manually override runtime configuration settings by providing environment variables. To customize the configuration, set the relevant environment variables in accordance with your requirements. The following example overrides `MANAGEMENT_SERVER_PORT` as `8088`, and it overrides `SERVER_PORT` as `8082`.

```
tanzu apps workload create spring-cloud-serverless \
--git-repo https://github.com/vudayani-vmw/spring-cloud-serverless --git-branch main
--type web \
--label tanzu.app.live.view.application.flavours=spring-boot_spring-native \
--label apps.tanzu.vmware.com/auto-configure-actuators=true --label \
app.kubernetes.io/part-of=spring-cloud-serverless --label apps.tanzu.vmware.com/has-
tests=true \
--yes --annotation autoscaling.knative.dev/min-scale=1 --build-env "BP_JVM_VERSION=1
7" \
--build-env "BP_NATIVE_IMAGE=true" --build-env "BP_MAVEN_BUILD_ARGUMENTS= -Pnative \
-Dmaven.test.skip=true --no-transfer-progress package \
-Dspring-boot.aot.jvmArguments='-Dmanagement.endpoint.health.probes.add-additional-p
aths='true' \
-Dmanagement.endpoint.health.show-details='always' -Dmanagement.endpoints.web.base-p
ath='/actuator' \
-Dmanagement.endpoints.web.exposure.include='*' -Dmanagement.health.probes.enabled
='true' \
-Dmanagement.server.port=8081 -Dserver.port=8080' " --env MANAGEMENT_SERVER_PORT=808
8 \
--env SERVER_PORT=8082
```

The resulting `PodSpec` is:

```
Spec:
  Containers:
    Env:
      Name:   MANAGEMENT_SERVER_PORT
      Value:  8088
      Name:   SERVER_PORT
      Value:  8082
      Name:   MANAGEMENT_ENDPOINT_HEALTH_PROBES_ADD_ADDITIONAL_PATHS
      Value:  true
      Name:   MANAGEMENT_ENDPOINT_HEALTH_SHOW_DETAILS
      Value:  always
      Name:   MANAGEMENT_ENDPOINTS_WEB_BASE_PATH
      Value:  /actuator
      Name:   MANAGEMENT_ENDPOINTS_WEB_EXPOSURE_INCLUDE
      Value:  *
      Name:   MANAGEMENT_HEALTH_PROBES_ENABLED
      Value:  true
      Name:   SERVER_SHUTDOWN_GRACE_PERIOD
      Value:  24s
```

Spring Boot conventions also set the Application Live View labels on the `PodSpec` to enable Application Live View.

Verify the applied labels and annotations

Verify the applied labels and annotations by running:

```
kubectl get podintents.conventions.carto.run WORKLOAD-NAME -o yaml
```

Where `WORKLOAD-NAME` is the name of the deployed workload. For example, `spring-cloud-serverless`.

Expected output of the Spring Boot workload:

```
...
Labels:       app.kubernetes.io/component=intent
              app.kubernetes.io/part-of=spring-cloud-serverless
              apps.tanzu.vmware.com/auto-configure-actuators=true
              apps.tanzu.vmware.com/has-tests=true
              apps.tanzu.vmware.com/workload-type=web
              carto.run/cluster-template-name=convention-template
              carto.run/resource-name=config-provider
              carto.run/supply-chain-name=source-to-url
              carto.run/template-kind=ClusterConfigTemplate
              carto.run/template-lifecycle=mutable
              carto.run/workload-name=spring-cloud-serverless
              carto.run/workload-namespace=default
              tanzu.app.live.view.application.flavours=spring-boot_spring-native
...
```

Enable Prometheus scraping for Spring Boot applications on workloads

This topic tells you how to enable Prometheus to get metrics for your Spring Boot applications on Tanzu Application Platform (commonly known as TAP) workloads.

About Prometheus scraping for Spring Boot applications

Spring Boot conventions recognizes PodIntents and automatically adds Prometheus annotations on the Tanzu Application Platform workload PodSpec. This means that you do not need to manually add the annotations in your workload YAML. The pods with these Prometheus annotations are considered for scraping of metrics.

The following annotations are added:

- `prometheus.io/scrape: "true"`: Enables the Prometheus scrape feature. This identifies which pods to scrape for metrics.
- `prometheus.io/path: "/actuator/prometheus"`: Sets the Prometheus scrape path to the endpoint where Prometheus exposes metrics.
- `prometheus.io/port: "8080"`: Sets the Prometheus port to the default server port of a Spring Boot application. This ensures that the right port is used for the scrape job for each pod.

Note

If any of these annotations are added manually, the manual annotation setting takes precedence over the automatic setting.

Spring Boot conventions allows you to activate or deactivate the automatic configuration of actuators on Tanzu Application Platform and on individual workloads. If the label `apps.tanzu.vmware.com/auto-configure-actuators` on the workload is set to `true`, Spring Boot conventions activates automatic configuration of actuators. The default label setting on the workload is `apps.tanzu.vmware.com/auto-configure-actuators: "false"`.

If `apps.tanzu.vmware.com/auto-configure-actuators` label is set to `true` on the workload YAML, then the management server port is also switched to 8081 and Spring Boot conventions uses that port to access the actuators instead of the default app port. Therefore, the Prometheus port is also

set to 8081 automatically on the PodSpec so that the metrics can be scraped using that port. All the other native Prometheus annotations remain the same as the `false` case.

For more information, see Configuring and accessing Spring Boot actuators in Tanzu Application Platform.

Enable Prometheus to scrape metrics

To enable Prometheus to scrape metrics from HTTP endpoints of a Spring Boot application, add the Micrometer registry dependency for Prometheus support in the POM dependencies section:

```
<dependency>
   <groupId>io.micrometer</groupId>
   <artifactId>micrometer-registry-prometheus</artifactId>
</dependency>
```

Verify the applied annotations

To verify the applied labels and annotations, run:

```
kubectl get podintents.conventions.carto.run WORKLOAD-NAME -o yaml
```

Where `WORKLOAD-NAME` is the name of the deployed workload. For example: `tanzu-java-web-app`.

Example output

Example output for a Spring Boot workload PodIntent:

```
apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
  creationTimestamp: "2024-03-01T11:32:31Z"
  generation: 1
  labels:
    app.kubernetes.io/component: intent
    app.kubernetes.io/part-of: tanzu-java-web-app-prometheus-without-annotations
    apps.tanzu.vmware.com/workload-type: web
    carto.run/cluster-template-name: convention-template
    carto.run/resource-name: config-provider
    carto.run/supply-chain-name: source-to-url
    carto.run/template-kind: ClusterConfigTemplate
    carto.run/template-lifecycle: mutable
    carto.run/workload-name: tanzu-java-web-app-prometheus-without-annotations
    carto.run/workload-namespace: default
  name: tanzu-java-web-app-prometheus-without-annotations
  namespace: default
  ownerReferences:
  - apiVersion: carto.run/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: Workload
    name: tanzu-java-web-app-prometheus-without-annotations
    uid: 8531983d-338d-45e9-8398-53bf0f53b3a4
  resourceVersion: "4410028"
  uid: 1fe59b2b-72ac-4756-9123-3cc1db007c0d
spec:
  serviceAccountName: default
  template:
    metadata:
      annotations:
        apps.tanzu.vmware.com/correlationid: https://github.com/ksankaranara-vmw/tanzu
```

```
-java-web-app-prometheus?sub_path=/
        autoscaling.knative.dev/min-scale: "1"
        developer.conventions/target-containers: workload
      labels:
        app.kubernetes.io/component: run
        app.kubernetes.io/part-of: tanzu-java-web-app-prometheus-without-annotations
        apps.tanzu.vmware.com/workload-type: web
        carto.run/workload-name: tanzu-java-web-app-prometheus-without-annotations
    spec:
      containers:
      - image: dev.registry.tanzu.vmware.com/app-live-view/test/tanzu-java-web-app-pro
metheus-without-annotations-default@sha256:552c7f86b410b700716c709742434e0ea9e022934bc
4fe66a5bb12a5241ea644
        name: workload
        resources: {}
        securityContext:
          allowPrivilegeEscalation: false
          capabilities:
            drop:
            - ALL
          runAsNonRoot: true
          runAsUser: 1000
          seccompProfile:
            type: RuntimeDefault
      serviceAccountName: default
status:
  conditions:
  - lastTransitionTime: "2024-03-01T11:32:31Z"
    message: ""
    reason: Applied
    status: "True"
    type: ConventionsApplied
  - lastTransitionTime: "2024-03-01T11:32:31Z"
    message: ""
    reason: ConventionsApplied
    status: "True"
    type: Ready
  observedGeneration: 1
  template:
    metadata:
      annotations:
        apps.tanzu.vmware.com/correlationid: https://github.com/ksankaranara-vmw/tanzu
-java-web-app-prometheus?sub_path=/
        autoscaling.knative.dev/min-scale: "1"
        boot.spring.io/version: 3.1.3
        conventions.carto.run/applied-conventions: |-
          spring-boot-convention/auto-configure-actuators-check
          spring-boot-convention/is-native-app-check
          spring-boot-convention/is-prometheus-enabled-check
          spring-boot-convention/spring-boot
          spring-boot-convention/spring-boot-graceful-shutdown
          spring-boot-convention/spring-boot-web
          spring-boot-convention/spring-boot-actuator
          spring-boot-convention/spring-boot-prometheus-enabled
          spring-boot-convention/spring-boot-actuator-probes
          spring-boot-convention/app-live-view-appflavour-check
          spring-boot-convention/app-live-view-connector-boot
          spring-boot-convention/app-live-view-appflavours-boot
        developer.conventions/target-containers: workload
        prometheus.io/path: /actuator/prometheus
        prometheus.io/port: "8080"
        prometheus.io/scrape: "true"
      labels:
        app.kubernetes.io/component: run
        app.kubernetes.io/part-of: tanzu-java-web-app-prometheus-without-annotations
        apps.tanzu.vmware.com/auto-configure-actuators: "false"
```

```
        apps.tanzu.vmware.com/workload-type: web
        carto.run/workload-name: tanzu-java-web-app-prometheus-without-annotations
        conventions.carto.run/framework: spring-boot
        tanzu.app.live.view: "true"
        tanzu.app.live.view.application.actuator.path: actuator
        tanzu.app.live.view.application.actuator.port: "8080"
        tanzu.app.live.view.application.flavours: spring-boot
        tanzu.app.live.view.application.name: tanzu-java-web-app-prometheus-without-an
notations
    spec:
      containers:
      - env:
        - name: JAVA_TOOL_OPTIONS
          value: -Dmanagement.endpoint.health.probes.add-additional-paths="true" -Dman
agement.health.probes.enabled="true"
            -Dserver.port="8080" -Dserver.shutdown.grace-period="24s"
        image: dev.registry.tanzu.vmware.com/app-live-view/test/tanzu-java-web-app-pro
metheus-without-annotations-default@sha256:552c7f86b410b700716c709742434e0ea9e022934bc
4fe66a5bb12a5241ea644
        livenessProbe:
          httpGet:
            path: /livez
            port: 8080
            scheme: HTTP
        name: workload
        ports:
        - containerPort: 8080
          protocol: TCP
        readinessProbe:
          httpGet:
            path: /readyz
            port: 8080
            scheme: HTTP
        resources: {}
        securityContext:
          allowPrivilegeEscalation: false
          capabilities:
            drop:
            - ALL
          runAsNonRoot: true
          runAsUser: 1000
          seccompProfile:
            type: RuntimeDefault
        startupProbe:
          httpGet:
            path: /readyz
            port: 8080
            scheme: HTTP
      serviceAccountName: default
```

Example output if automatic configuration of actuators is `true`

Example output for a Spring Boot workload PodIntent if the `apps.tanzu.vmware.com/auto-configure-actuators` label is set to `true` on the workload YAML:

```
apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
  creationTimestamp: "2024-03-01T16:33:48Z"
  generation: 1
  labels:
    app.kubernetes.io/component: intent
    app.kubernetes.io/part-of: tanzu-java-web-app-prometheus-without-annotations
    apps.tanzu.vmware.com/auto-configure-actuators: "true"
    apps.tanzu.vmware.com/workload-type: web
```

```
    carto.run/cluster-template-name: convention-template
    carto.run/resource-name: config-provider
    carto.run/supply-chain-name: source-to-url
    carto.run/template-kind: ClusterConfigTemplate
    carto.run/template-lifecycle: mutable
    carto.run/workload-name: tanzu-java-web-app-prometheus-without-annotations-auto-co
nfigure-actuators
    carto.run/workload-namespace: default
  name: tanzu-java-web-app-prometheus-without-annotations-auto-configure-actuators
  namespace: default
  ownerReferences:
  - apiVersion: carto.run/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: Workload
    name: tanzu-java-web-app-prometheus-without-annotations-auto-configure-actuators
    uid: 2a52e88e-a819-4f96-a49b-ca976225bb9c
  resourceVersion: "42809"
  uid: 2eb9b397-b423-45f2-b618-45bf6503e81a
spec:
  serviceAccountName: default
  template:
    metadata:
      annotations:
        apps.tanzu.vmware.com/correlationid: https://github.com/ksankaranara-vmw/tanzu
-java-web-app-prometheus?sub_path=/
        autoscaling.knative.dev/min-scale: "1"
        developer.conventions/target-containers: workload
      labels:
        app.kubernetes.io/component: run
        app.kubernetes.io/part-of: tanzu-java-web-app-prometheus-without-annotations
        apps.tanzu.vmware.com/auto-configure-actuators: "true"
        apps.tanzu.vmware.com/workload-type: web
        carto.run/workload-name: tanzu-java-web-app-prometheus-without-annotations-aut
o-configure-actuators
    spec:
      containers:
      - image: dev.registry.tanzu.vmware.com/app-live-view/test/tanzu-java-web-app-pro
metheus-without-annotations-auto-configure-actuators-default@sha256:25bac27dd331267a48
eda30dfb7d9f96294ec88ce225a4294c7d2377f6765ee0
        name: workload
        resources: {}
        securityContext:
          allowPrivilegeEscalation: false
          capabilities:
            drop:
            - ALL
          runAsNonRoot: true
          runAsUser: 1000
          seccompProfile:
            type: RuntimeDefault
      serviceAccountName: default
status:
  conditions:
  - lastTransitionTime: "2024-03-01T16:33:48Z"
    message: ""
    reason: Applied
    status: "True"
    type: ConventionsApplied
  - lastTransitionTime: "2024-03-01T16:33:48Z"
    message: ""
    reason: ConventionsApplied
    status: "True"
    type: Ready
  observedGeneration: 1
  template:
```

```
    metadata:
      annotations:
        apps.tanzu.vmware.com/correlationid: https://github.com/ksankaranara-vmw/tanzu
-java-web-app-prometheus?sub_path=/
        autoscaling.knative.dev/min-scale: "1"
        boot.spring.io/actuator: http://:8081/actuator
        boot.spring.io/version: 3.1.3
        conventions.carto.run/applied-conventions: |-
          spring-boot-convention/auto-configure-actuators-check
          spring-boot-convention/is-native-app-check
          spring-boot-convention/is-prometheus-enabled-check
          spring-boot-convention/spring-boot
          spring-boot-convention/spring-boot-graceful-shutdown
          spring-boot-convention/spring-boot-web
          spring-boot-convention/spring-boot-actuator
          spring-boot-convention/spring-boot-prometheus-enabled
          spring-boot-convention/spring-boot-actuator-probes
          spring-boot-convention/app-live-view-appflavour-check
          spring-boot-convention/app-live-view-connector-boot
          spring-boot-convention/app-live-view-appflavours-boot
        developer.conventions/target-containers: workload
        prometheus.io/path: /actuator/prometheus
        prometheus.io/port: "8081"
        prometheus.io/scrape: "true"
      labels:
        app.kubernetes.io/component: run
        app.kubernetes.io/part-of: tanzu-java-web-app-prometheus-without-annotations
        apps.tanzu.vmware.com/auto-configure-actuators: "true"
        apps.tanzu.vmware.com/workload-type: web
        carto.run/workload-name: tanzu-java-web-app-prometheus-without-annotations-aut
o-configure-actuators
        conventions.carto.run/framework: spring-boot
        tanzu.app.live.view: "true"
        tanzu.app.live.view.application.actuator.path: actuator
        tanzu.app.live.view.application.actuator.port: "8081"
        tanzu.app.live.view.application.flavours: spring-boot
        tanzu.app.live.view.application.name: tanzu-java-web-app-prometheus-without-an
notations-auto-configure-actuators
    spec:
      containers:
      - env:
        - name: JAVA_TOOL_OPTIONS
          value: -Dmanagement.endpoint.health.probes.add-additional-paths="true" -Dman
agement.endpoint.health.show-details="always"
            -Dmanagement.endpoints.web.base-path="/actuator" -Dmanagement.endpoints.we
b.exposure.include="*"
            -Dmanagement.health.probes.enabled="true" -Dmanagement.server.port="8081"
            -Dserver.port="8080" -Dserver.shutdown.grace-period="24s"
        image: dev.registry.tanzu.vmware.com/app-live-view/test/tanzu-java-web-app-pro
metheus-without-annotations-auto-configure-actuators-default@sha256:25bac27dd331267a48
eda30dfb7d9f96294ec88ce225a4294c7d2377f6765ee0
        livenessProbe:
          httpGet:
            path: /livez
            port: 8080
            scheme: HTTP
        name: workload
        ports:
        - containerPort: 8080
          protocol: TCP
        readinessProbe:
          httpGet:
            path: /readyz
            port: 8080
            scheme: HTTP
        resources: {}
```

```
        securityContext:
          allowPrivilegeEscalation: false
          capabilities:
            drop:
            - ALL
          runAsNonRoot: true
          runAsUser: 1000
          seccompProfile:
            type: RuntimeDefault
        startupProbe:
          httpGet:
            path: /readyz
            port: 8080
            scheme: HTTP
      serviceAccountName: default
```

List of Spring Boot conventions

This topic tells you about what the conventions do and how to apply them.

When submitting the following pod `Pod Intent` on each convention, the output can change depending on the applied convention.

Before any Spring Boot conventions are applied, the pod intent looks similar to this YAML:

```
apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
  name: spring-sample
spec:
  template:
    spec:
      containers:
      - name: workload
        image: springio/petclinic
```

Most of the Spring Boot conventions either edit or add properties to the environment variable `JAVA_TOOL_OPTIONS`. You can override those conventions by providing the `JAVA_TOOL_OPTIONS` value you want through the Tanzu CLI or `workload.yaml`.

When a `JAVA_TOOL_OPTIONS` property already exists for a workload, the convention uses the existing value rather than the value that the convention applies by default. The property value that you provide is used for the pod specification mutation.

Set a `JAVA_TOOL_OPTIONS` property for a workload

Do one of the following actions to set `JAVA_TOOL_OPTIONS` property and values:

Use the Tanzu CLI apps plug-in
When creating or updating a workload, set a `JAVA_TOOL_OPTIONS` property using the `--env` flag by running:

```
tanzu apps workload create APP-NAME --env JAVA_TOOL_OPTIONS="-DPROPERTY-NAME=VALUE"
```

For example, to set the management port to `8080` rather than the spring-boot-actuator-convention default port `8081`, run:

```
tanzu apps workload create APP-NAME --env JAVA_TOOL_OPTIONS="-Dmanagement.server.por
t=8080"
```

Use workload.yaml

Follow these steps:

1. Provide one or more values for the `JAVA_TOOL_OPTIONS` property in the `workload.yaml`. For example:

```
apiVersion: carto.run/v1alpha1
kind: Workload
...
spec:
env:
- name: JAVA_TOOL_OPTIONS
  value: -Dmanagement.server.port=8082
source:
...
```

2. Apply the `workload.yaml` file by running the command:

```
tanzu apps workload create -f workload.yaml
```

Spring Boot convention

If the `spring-boot` dependency is in the metadata within the `SBOM` file under `dependencies`, the Spring Boot convention is applied to the `PodTemplateSpec` object.

The Spring Boot convention adds a label (`conventions.carto.run/framework: spring-boot`) to the `PodTemplateSpec` that describes the framework associated with the workload, and adds an annotation (`boot.spring.io/version: VERSION-NO`) that describes the Spring Boot version of the dependency.

The label and annotation are added for informational purposes only.

Example of PodIntent after applying the convention:

```
apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
 annotations:
   kubectl.kubernetes.io/last-applied-configuration: |
     {"apiVersion":"conventions.carto.run/v1alpha1","kind":"PodIntent","metadata":{"an
notations":{},"name":"spring-sample","namespace":"default"},"spec":{"template":{"spe
c":{"containers":[{"image":"springio/petclinic","name":"workload"}]}}}}

...

status:
 conditions:
 - lastTransitionTime: "..." # This status indicates that all worked as expected
   status: "True"
   type: ConventionsApplied
 - lastTransitionTime: "..."
   status: "True"
   type: Ready
 observedGeneration: 1
 template:
   metadata:
     annotations:
       boot.spring.io/version: 2.3.3.RELEASE
       conventions.carto.run/applied-conventions: |-
         spring-boot-convention/spring-boot
     labels:
       conventions.carto.run/framework: spring-boot
```

```
  spec:
    containers:
    - image: index.docker.io/springio/petclinic@sha256:...
      name: workload
      resources: {}
```

Spring boot graceful shut down convention

If any of the following dependencies are in the metadata within the `SBOM` file under `dependencies`, the Spring Boot graceful shut down convention is applied to the `PodTemplateSpec` object:

- `spring-boot-starter-tomcat`
- `spring-boot-starter-jetty`
- `spring-boot-starter-reactor-netty`
- `spring-boot-starter-undertow`
- `tomcat-embed-core`

The Graceful Shutdown convention `spring-boot-graceful-shutdown` adds a property in the environment variable `JAVA_TOOL_OPTIONS` with the key `server.shutdown.grace-period`. The key value is calculated to be 80% of the value set in `.target.Spec.TerminationGracePeriodSeconds`. The default value for `.target.Spec.TerminationGracePeriodSeconds` is 30 seconds.

Example of PodIntent after applying the convention:

```
apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"conventions.carto.run/v1alpha1","kind":"PodIntent","metadata":{"a
nnotations":{},"name":"spring-sample","namespace":"default"},"spec":{"template":{"spe
c":{"containers":[{"image":"springio/petclinic","name":"workload"}]}}}}

...

status:
  conditions:
  - lastTransitionTime: "..." # This status indicates that all worked as expected
    status: "True"
    type: ConventionsApplied
  - lastTransitionTime: "..."
    status: "True"
    type: Ready
  observedGeneration: 1
  template:
    metadata:
      annotations:
        boot.spring.io/version: 2.3.3.RELEASE
        conventions.carto.run/applied-conventions: |-
          spring-boot-convention/spring-boot
          spring-boot-convention/spring-boot-graceful-shutdown
      labels:
        conventions.carto.run/framework: spring-boot
    spec:
      containers:
      - env:
        - name: JAVA_TOOL_OPTIONS
          value: -Dserver.shutdown.grace-period="24s"
        image: index.docker.io/springio/petclinic@sha256:...
        name: workload
        resources: {}
```

Spring Boot web convention

If any of the following dependencies are in the metadata within the `SBOM` file under `dependencies`, the Spring Boot web convention is applied to the `PodTemplateSpec` object:

- spring-boot
- spring-boot-web

The web convention `spring-boot-web` obtains the `server.port` property from the `JAVA_TOOL_OPTIONS` environment variable and sets it as a port in the `PodTemplateSpec`. If `JAVA_TOOL_OPTIONS` environment variable does not contain a `server.port` property or value, the convention adds the property and sets the value to `8080`, which is the Spring Boot default.

Example of PodIntent after applying the convention:

```
apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"conventions.carto.run/v1alpha1","kind":"PodIntent","metadata":{"a
nnotations":{},"name":"spring-sample","namespace":"default"},"spec":{"template":{"spe
c":{"containers":[{"image":"springio/petclinic","name":"workload"}]}}}}

...

status:
  conditions:
  - lastTransitionTime: "..." # This status indicates that all worked as expected
    status: "True"
    type: ConventionsApplied
  - lastTransitionTime: "..."
    status: "True"
    type: Ready
  observedGeneration: 1
  template:
    metadata:
      annotations:
        boot.spring.io/version: 2.3.3.RELEASE
        conventions.carto.run/applied-conventions: |-
          spring-boot-convention/spring-boot
          spring-boot-convention/spring-boot-web
      labels:
        conventions.carto.run/framework: spring-boot
    spec:
      containers:
      - env:
        - name: JAVA_TOOL_OPTIONS
          value: -Dserver.port="8080"
        image: index.docker.io/springio/petclinic@sha256:...
        name: workload
        ports:
        - containerPort: 8080
          protocol: TCP
        resources: {}
```

Spring Boot Actuator convention

If the `spring-boot-actuator` dependency is in the metadata within the `SBOM` file under `dependencies`, the Spring Boot actuator convention is applied to the `PodTemplateSpec` object.

The Spring Boot Actuator convention does the following actions:

If the workload-level or platform-level automatic configuration of actuators is enabled:

1. Sets the management port in the `JAVA_TOOL_OPTIONS` environment variable to `8081`.
2. Sets the base path in the `JAVA_TOOL_OPTIONS` environment variable to `/actuator`.
3. Adds an annotation, `boot.spring.io/actuator`, to where the actuator is accessed.

The management port is set to port `8081` for security reasons. Although you can prevent public access to the actuator endpoints that are exposed on the management port when it is set to the default `8080`, the threat of exposure through internal access remains. The best practice for security is to set the management port to something other than `8080`.

However, if a management port number value is provided using the `-Dmanagement.server.port` property in `JAVA_TOOL_OPTIONS`, the Spring Boot actuator convention uses that value rather than its default `8081` as the management port.

You can access the management context of a Spring Boot application by creating a service pointing to port `8081` and base path `/actuator`.

Important

To override the management port setting applied by this convention, see How to set a JAVA_TOOL_OPTIONS property for a workload earlier in this topic. Any alternative methods for setting the management port are overwritten. For example, if you configure the management port using `application.properties/yml` or `config server`, the Spring Boot Actuator convention overrides your configuration.

If the workload-level or platform-level automatic configuration of actuators is deactivated, the Spring Boot actuator convention does not set any `JAVA_TOOLS_OPTIONS` and does not set the annotation `boot.spring.io/actuator`.

Example of PodIntent after applying the convention:

```
apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
 annotations:
   kubectl.kubernetes.io/last-applied-configuration: |
     {"apiVersion":"conventions.carto.run/v1alpha1","kind":"PodIntent","metadata":{"an
notations":{},"name":"spring-sample","namespace":"default"},"spec":{"template":{"spe
c":{"containers":[{"image":"springio/petclinic","name":"workload"}]}}}}

...

status:
 conditions:
 - lastTransitionTime: "..." # This status indicates that all worked as expected
   status: "True"
   type: ConventionsApplied
 - lastTransitionTime: "..."
   status: "True"
   type: Ready
 observedGeneration: 1
 template:
   metadata:
     annotations:
       boot.spring.io/actuator: http://:8081/actuator
       boot.spring.io/version: 2.3.3.RELEASE
       conventions.carto.run/applied-conventions: |-
         spring-boot-convention/spring-boot
         spring-boot-convention/spring-boot-web
```

```
        spring-boot-convention/spring-boot-actuator
    labels:
      conventions.carto.run/framework: spring-boot
  spec:
    containers:
    - env:
      - name: JAVA_TOOL_OPTIONS
        value: Dmanagement.endpoints.web.base-path="/actuator" -Dmanagement.server.po
rt="8081" -Dserver.port="8080"
      image: index.docker.io/springio/petclinic@sha256:...
      name: workload
      ports:
      - containerPort: 8080
        protocol: TCP
      resources: {}
```

Spring Boot Actuator Probes convention

The Spring Boot Actuator Probes convention is applied only if all of the following conditions are met:

- The `spring-boot-actuator` dependency exists and is **>= 2.6**
- The `JAVA_TOOL_OPTIONS` environment variable does not include the following properties or, if either of the properties is included, it is set to a value of `true`:
  - `-Dmanagement.health.probes.enabled`
  - `-Dmanagement.endpoint.health.probes.add-additional-paths`

The Spring Boot Actuator Probes convention does the following actions:

1. Uses the main server port, which is the `server.port` value on `JAVA_TOOL_OPTIONS`, to set the liveness and readiness probes. For more information see the Kubernetes documentation
2. Adds the following properties and values to the `JAVA_TOOL_OPTIONS` environment variable:
   - `-Dmanagement.health.probes.enabled="true"`
   - `-Dmanagement.endpoint.health.probes.add-additional-paths="true"`

   When this convention is applied, the probes are exposed as follows:
   - Liveness probe: `/livez`
   - Readiness probe: `/readyz`

Example of PodIntent after applying the convention:

```
apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"conventions.carto.run/v1alpha1","kind":"PodIntent","metadata":{"a
nnotations":{},"name":"spring-sample","namespace":"default"},"spec":{"template":{"spe
c":{"containers":[{"image":"springio/petclinic","name":"workload"}]}}}}

...

status:
  conditions:
  - lastTransitionTime: "..." # This status indicates that all worked as expected
    status: "True"
    type: ConventionsApplied
  - lastTransitionTime: "..."
    status: "True"
```

```
    type: Ready
  observedGeneration: 1
  template:
    metadata:
      annotations:
        boot.spring.io/actuator: http://:8080/actuator
        boot.spring.io/version: 2.6.0
        conventions.carto.run/applied-conventions: |-
          spring-boot-convention/spring-boot
          spring-boot-convention/spring-boot-web
          spring-boot-convention/spring-boot-actuator
      labels:
        conventions.carto.run/framework: spring-boot
    spec:
      containers:
      - env:
        - name: JAVA_TOOL_OPTIONS
          value: -Dmanagement.endpoint.health.probes.add-additional-paths="true" -Dman
agement.endpoints.web.base-path="/actuator" -Dmanagement.health.probes.enabled="true"
-Dmanagement.server.port="8081" -Dserver.port="8080"
        image: index.docker.io/springio/petclinic@sha256:...
        name: workload
        livenessProbe:
          httpGet:
            path: /livez
            port: 8080
            scheme: HTTP
        ports:
        - containerPort: 8080
          protocol: TCP
        readinessProbe:
          httpGet:
            path: /readyz
            port: 8080
            scheme: HTTP
        resources: {}
```

Service intent conventions

The Service intent conventions do not change the behavior of the final deployment, but you can use them as added information to process in the supply chain. For example, when an app requires to be bound to database service. This convention adds an annotation and a label to the `PodTemplateSpec` for each detected dependency. It also adds an annotation and a label to the `conventions.carto.run/applied-conventions`.

The list of the supported intents are:

MySQL

- **Name**: `service-intent-mysql`
- **Label**: `services.conventions.apps.tanzu.vmware.com/mysql`
- **Dependencies**: `mysql-connector-java`, `r2dbc-mysql`

PostgreSQL

- **Name**: `service-intent-postgres`
- **Label**: `services.conventions.apps.tanzu.vmware.com/postgres`
- **Dependencies**: `postgresql`, `r2dbc-postgresql`

MongoDB

- **Name**: `service-intent-mongodb`

- **Label**: `services.conventions.apps.tanzu.vmware.com/mongodb`
- **Dependencies**: `mongodb-driver-core`

RabbitMQ

- **Name**: `service-intent-rabbitmq`
- **Label**: `services.conventions.apps.tanzu.vmware.com/rabbitmq`
- **Dependencies**: `amqp-client`

Redis

- **Name**: `service-intent-redis`
- **Label**: `services.conventions.apps.tanzu.vmware.com/redis`
- **Dependencies**: `jedis`

Kafka

- **Name**: `service-intent-kafka`
- **Label**: `services.conventions.apps.tanzu.vmware.com/kafka`
- **Dependencies**: `kafka-clients`

Kafka-streams

- **Name**: `service-intent-kafka-streams`
- **Label**: `services.conventions.apps.tanzu.vmware.com/kafka-streams`
- **Dependencies**: `kafka-streams`

Example

When you apply the `Pod Intent` and the image contains a dependency, for example, of MySQL, then the output of the convention is:

```
  apiVersion: conventions.carto.run/v1alpha1
  kind: PodIntent
  metadata:
    annotations:
      kubectl.kubernetes.io/last-applied-configuration: |
        {"apiVersion":"conventions.carto.run/v1alpha1","kind":"PodIntent","metadata":
{"annotations":{},"name":"spring-sample","namespace":"default"},"spec":{"template":{"s
pec":{"containers":[{"image":"springio/petclinic","name":"workload"}]}}}}
    creationTimestamp: "..."
    generation: 1
    name: spring-sample
    namespace: default
    resourceVersion: "..."
    uid: ...
  spec:
    serviceAccountName: default
    template:
      metadata: {}
      spec:
        containers:
        - image: springio/petclinic
          name: workload
          resources: {}
  status:
    conditions:
    - lastTransitionTime: "..." # This status indicates that all worked as expected
      status: "True"
      type: ConventionsApplied
```

```
    - lastTransitionTime: "..."
      status: "True"
      type: Ready
    observedGeneration: 1
    template:
      metadata:
        annotations:
          boot.spring.io/actuator: http://:8080/actuator
          boot.spring.io/version: 2.3.3.RELEASE
          conventions.carto.run/applied-conventions: |-
            spring-boot-convention/spring-boot
            spring-boot-convention/spring-boot-web
            spring-boot-convention/spring-boot-actuator
            spring-boot-convention/service-intent-mysql
          services.conventions.apps.tanzu.vmware.com/mysql: mysql-connector-java/8.0.2
1
        labels:
          conventions.apps.tanzu.vmware.com/framework: spring-boot
          services.conventions.apps.tanzu.vmware.com/mysql: workload
      spec:
        containers:
        - env:
          - name: JAVA_TOOL_OPTIONS
            value: Dmanagement.endpoints.web.base-path="/actuator" -Dmanagement.serve
r.port="8081" -Dserver.port="8080"
          image: index.docker.io/springio/petclinic@sha256:...
          name: workload
          ports:
          - containerPort: 8080
            protocol: TCP
          resources: {}
```

Troubleshoot Spring Boot conventions

This topic tells you how to troubleshoot Spring Boot conventions.

Collect logs

If you have trouble, you can retrieve and examine logs from the Spring Boot convention server as follows:

1. The Spring Boot convention server creates a namespace to contain all of the associated resources. By default the namespace is `spring-boot-convention`. To inspect the logs, run:

   ```
   kubectl logs -l app=spring-boot-webhook -n spring-boot-convention
   ```

   For example:

   ```
   $ kubectl logs -l app=spring-boot-webhook -n spring-boot-convention

   {"level":"info","timestamp":"2021-11-11T16:00:26.597Z","caller":"spring-boot-co
   nventions/server.go:83","msg":"Successfully applied convention: spring-boot","c
   omponent":"spring-boot-conventions"}
   {"level":"info","timestamp":"2021-11-11T16:00:26.597Z","caller":"spring-boot-co
   nventions/server.go:83","msg":"Successfully applied convention: spring-boot-gra
   ceful-shutdown","component":"spring-boot-conventions"}
   {"level":"info","timestamp":"2021-11-11T16:00:26.597Z","caller":"spring-boot-co
   nventions/server.go:83","msg":"Successfully applied convention: spring-boot-we
   b","component":"spring-boot-conventions"}
   {"level":"info","timestamp":"2021-11-11T16:00:26.597Z","caller":"spring-boot-co
   nventions/server.go:83","msg":"Successfully applied convention: spring-boot-act
   uator","component":"spring-boot-conventions"}
   {"level":"info","timestamp":"2021-11-11T16:00:26.597Z","caller":"spring-boot-co
   ```

```
nventions/server.go:83","msg":"Successfully applied convention: service-intent-
mysql","component":"spring-boot-conventions"}
```

2. For all of the conventions that were applied successfully, a log entry is added. If an error occurs, a log entry is added with a description.

Overview of Spring Cloud Gateway for Kubernetes

Spring Cloud Gateway for Kubernetes is an API gateway solution based on the open-source Spring Cloud Gateway project. It provides a simple means to route internal or external API requests to application services that expose APIs.

Spring Cloud Gateway for Kubernetes handles cross-cutting concerns on behalf of API development teams, including single sign-on (SSO), access control, rate limiting, resiliency, security, and more.

It enables you to accelerate API delivery using modern cloud-native patterns with any programming language you choose for API development. It also integrates with your existing CI/CD pipeline strategy.

For more information about Spring Cloud Gateway for Kubernetes, see the Spring Cloud Gateway for Kubernetes documentation.

Overview of Spring Cloud Gateway for Kubernetes

Spring Cloud Gateway for Kubernetes is an API gateway solution based on the open-source Spring Cloud Gateway project. It provides a simple means to route internal or external API requests to application services that expose APIs.

Spring Cloud Gateway for Kubernetes handles cross-cutting concerns on behalf of API development teams, including single sign-on (SSO), access control, rate limiting, resiliency, security, and more.

It enables you to accelerate API delivery using modern cloud-native patterns with any programming language you choose for API development. It also integrates with your existing CI/CD pipeline strategy.

For more information about Spring Cloud Gateway for Kubernetes, see the Spring Cloud Gateway for Kubernetes documentation.

Install Spring Cloud Gateway for Kubernetes

This topic describes how to install Spring Cloud Gateway for Kubernetes from the Tanzu Application Platform package repository.

Prerequisites

Before installing Spring Cloud Gateway, complete all prerequisites for installing Tanzu Application Platform. For more information, see Prerequisites.

Install

To install Spring Cloud Gateway:

1. See which versions of Spring Cloud Gateway are available to install from the Tanzu Application Platform repository by running:

```
tanzu package available list spring-cloud-gateway.tanzu.vmware.com --namespace
tap-install
```

For example:

```
$ tanzu package available list spring-cloud-gateway.tanzu.vmware.com --namespac
e tap-install

NAME                                    VERSION  RELEASED-AT
spring-cloud-gateway.tanzu.vmware.com  2.0.0    2022-02-01T00:00:00Z
```

2. (Optional) View the changes you can make to the default installation settings by running:

```
tanzu package available get spring-cloud-gateway.tanzu.vmware.com/VERSION-NUMBE
R \
  --namespace tap-install --values-schema
```

Where `VERSION-NUMBER` is the version of the package listed earlier.

For example:

```
tanzu package available get spring-cloud-gateway.tanzu.vmware.com/2.0.0 \
  --namespace tap-install --values-schema
```

You can use the information to generate a values override file for use in the following installation step.

For more information about values schema options, see the Spring Cloud Gateway for Kubernetes documentation.

Important

The value of `deployment.namespace` must always be set to the same value as the `--namespace` flag.

3. Install Spring Cloud Gateway by running:

Default values

Run this command to install Spring Cloud Gateway with the default values

```
tanzu package install spring-cloud-gateway \
  --package spring-cloud-gateway.tanzu.vmware.com \
  --version VERSION-NUMBER \
  --namespace tap-install
```

For example:

```
$ tanzu package install spring-cloud-gateway \
    --package spring-cloud-gateway.tanzu.vmware.com \
    --version 2.0.0 \
    --namespace tap-install

Installing package 'spring-cloud-gateway.tanzu.vmware.com'
Getting package metadata for 'spring-cloud-gateway.tanzu.vmware.com'
Creating service account 'spring-cloud-gateway-tap-install-sa'
Creating cluster admin role 'spring-cloud-gateway-tap-install-cluster-role'
Creating cluster role binding 'spring-cloud-gateway-tap-install-cluster-roleb
inding'
Creating package resource
Waiting for 'PackageInstall' reconciliation for 'spring-cloud-gateway'
```

```
'PackageInstall' resource install status: Reconciling
'PackageInstall' resource install status: ReconcileSucceeded

Added installed package 'spring-cloud-gateway'
```

Overriding values

Run this command to install Spring Cloud Gateway while overriding the default values

```
tanzu package install spring-cloud-gateway \
  --package spring-cloud-gateway.tanzu.vmware.com \
  --version VERSION-NUMBER \
  --namespace tap-install \
  --values-file values.yml
```

Overview of Supply Chain Choreographer for Tanzu

This topic introduces you to Supply Chain Choreographer.

Overview

Supply Chain Choreographer is based on open source Cartographer. It allows App Operators to create pre-approved paths to production by integrating Kubernetes resources with the elements of their existing toolchains, for example, Jenkins.

Each pre-approved supply chain creates a path to production. Orchestrating supply chain resources including, test, build, scan, and deploy allows developers to focus on delivering value to their users and provides App Operators the assurance that all code in production has passed through all the steps of an approved workflow.

Out of the Box Supply Chains

Out of the box supply chains are provided with Tanzu Application Platform.

The following three supply chains are included:

- Out of the Box Supply Chain Basic
- Out of the Box Supply Chain with Testing
- Out of the Box Supply Chain with Testing and Scanning

As auxiliary components, Tanzu Application Platform also includes:

- Out of the Box Templates, for providing templates used by the supply chains to perform common tasks such as fetching source code, running tests, and building container images.
- Out of the Box Delivery Basic, for delivering to a Kubernetes cluster the configuration built throughout a supply chain

Both Templates and Delivery Basic are requirements for the Supply Chains.

Supply Chain Choreographer supports the following pipeline types:

- Tekton pipelines
- Jenkins pipelines

Overview of Supply Chain Choreographer for Tanzu

This topic introduces you to Supply Chain Choreographer.

Overview

Supply Chain Choreographer is based on open source Cartographer. It allows App Operators to create pre-approved paths to production by integrating Kubernetes resources with the elements of their existing toolchains, for example, Jenkins.

Each pre-approved supply chain creates a path to production. Orchestrating supply chain resources including, test, build, scan, and deploy allows developers to focus on delivering value to their users and provides App Operators the assurance that all code in production has passed through all the steps of an approved workflow.

Out of the Box Supply Chains

Out of the box supply chains are provided with Tanzu Application Platform.

The following three supply chains are included:

- Out of the Box Supply Chain Basic
- Out of the Box Supply Chain with Testing
- Out of the Box Supply Chain with Testing and Scanning

As auxiliary components, Tanzu Application Platform also includes:

- Out of the Box Templates, for providing templates used by the supply chains to perform common tasks such as fetching source code, running tests, and building container images.
- Out of the Box Delivery Basic, for delivering to a Kubernetes cluster the configuration built throughout a supply chain

Both Templates and Delivery Basic are requirements for the Supply Chains.

Supply Chain Choreographer supports the following pipeline types:

- Tekton pipelines
- Jenkins pipelines

Install Supply Chain Choreographer

This topic describes how you can install Supply Chain Choreographer from the Tanzu Application Platform package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Supply Chain Choreographer. For more information about profiles, see Components and installation profiles..

Supply Chain Choreographer provides the custom resource definitions the supply chain uses. Each pre-approved supply chain creates a clear road to production and orchestrates supply chain resources. You can test, build, scan, and deploy. Developers can focus on delivering value to users. Application operators can rest assured that all code in production has passed through an approved workflow.

For example, Supply Chain Choreographer passes the results of fetching source code to the component that builds a container image of it, and then passes the container image to a component that deploys the image.

Prerequisites

Before installing Supply Chain Choreographer:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.

Install

To install Supply Chain Choreographer:

1. Get the values schema to see what properties can be configured during installation. Run:

```
tanzu package available get cartographer.tanzu.vmware.com/0.7.1+tap.1 --values-
schema --namespace tap-install

KEY                                     DEFAULT  TYPE     DESCRIPTION
ca_cert_data                            ""       string   Optional: PEM Encode
d certificate data for image registries with private CA.
cartographer.concurrency.max_deliveries  2        integer  Optional: maximum nu
mber of Deliverables to process concurrently.
cartographer.concurrency.max_runnables   2        integer  Optional: maximum nu
mber of Runnables to process concurrently.
cartographer.concurrency.max_workloads   2        integer  Optional: maximum nu
mber of Workloads to process concurrently.
cartographer.resources.limits.cpu        1000m            Optional: maximum am
ount of cpu resources to allow the controller to use
cartographer.resources.limits.memory     128Mi            Optional: maximum am
ount of memory to allow the controller to use
cartographer.resources.requests.cpu      250m             Optional: minimum am
ount of cpu to reserve
cartographer.resources.requests.memory   128Mi            Optional: minimum am
ount of memory to reserve
```

2. Install v0.4.0 of the `cartographer.tanzu.vmware.com` package, naming the installation `cartographer`. Run:

```
tanzu package install cartographer \
  --namespace tap-install \
  --package cartographer.tanzu.vmware.com \
  --version 0.7.1+tap.1
```

Example output:

```
| Installing package 'cartographer.tanzu.vmware.com'
| Getting namespace 'tap-install'
| Getting package metadata for 'cartographer.tanzu.vmware.com'
| Creating service account 'cartographer-tap-install-sa'
| Creating cluster admin role 'cartographer-tap-install-cluster-role'
| Creating cluster role binding 'cartographer-tap-install-cluster-rolebinding'
- Creating package resource
\ Package install status: Reconciling

Added installed package 'cartographer' in namespace 'tap-install'
```

Out of the Box Supply Chain Basic for Supply Chain Choreographer

This topic provides an overview of Out of the Box Supply Chain Basic for Supply Chain Choreographer.

This package contains Cartographer supply chains that tie together a series of Kubernetes resources that drive a developer-provided workload from source code to a Kubernetes configuration ready to be deployed to a cluster. It contains the most basic supply chains that focus on providing a quick path to deployment making no use of testing or scanning resources.

The supply chains in this package perform the following:

- Building from source code:
    1. Watching a Git repository, Maven repository, or local directory for changes
    2. Building a container image out of the source code with Buildpacks
    3. Applying operator-defined conventions to the container definition
    4. Creating a deliverable object for deploying the application to a cluster
    5. (Beta) Alternatively, outputting a Carvel package containing the application to a Git repository. See Carvel Package Supply Chains.
- Using a prebuilt application image:
    1. Applying operator-defined conventions to the container definition
    2. Creating a deliverable object for deploying the application to a cluster
    3. (Beta) Alternatively, outputting a Carvel package containing the application to a Git repository. See Carvel Package Supply Chains.

Prerequisites

To use this package, you must:

- Install Out of the Box Templates.
- Configure the Developer namespace with auxiliary objects that are used by the supply chain as described in the following section.
- (Optionally) install Out of the Box Delivery Basic, if you are willing to deploy the application to the same cluster as the workload and supply chains.

Developer Namespace

The supply chains provide definitions of many of the objects that they create to transform the source code to a container image and make it available as an application in a cluster.

The developer must provide or configure particular objects in the developer namespace so that the supply chain can provide credentials and use permissions granted to a specific development team.

The objects that the developer must provide or configure include:

- **registries secrets**: Kubernetes secrets of type `kubernetes.io/dockerconfigjson` that contain credentials for pushing and pulling the container images built by the supply chain and the installation of Tanzu Application Platform.
- **service account**: The identity to be used for any interaction with the Kubernetes API made by the supply chain.
- **rolebinding**: Grant to the identity the necessary roles for creating the resources prescribed by the supply chain.

Registries Secrets

Regardless of the supply chain that a workload goes through, there must be Kubernetes secrets in the developer namespace containing credentials for both pushing and pulling the container image

that the supply chain builds when source code is provided. The developer namespace must also contain registry credentials for Kubernetes to run pods using images from the installation of Tanzu Application Platform.

1. Add read/write registry credentials for pushing and pulling application images:

   ```
   tanzu secret registry add registry-credentials \
     --server REGISTRY-SERVER \
     --username REGISTRY-USERNAME \
     --password REGISTRY-PASSWORD \
     --namespace YOUR-NAMESPACE
   ```

   Where:

   - `YOUR-NAMESPACE` is the name you want to use for the developer namespace. For example, use `default` for the default namespace.
   - `REGISTRY-SERVER` is the URL of the registry. For Docker Hub, this must be `https://index.docker.io/v1/`. Specifically, it must have the leading `https://`, the `v1` path, and the trailing `/`. For Google Container Registry (GCR), this is `gcr.io`. Based on the information used in Installing the Tanzu Application Platform package and profiles, you can use the same registry server as in `ootb_supply_chain_basic` - `registry` - `server`.

2. Add a placeholder secret for gathering the credentials used for pulling container images from the installation of Tanzu Application Platform:

   ```
   cat <<EOF | kubectl -n YOUR-NAMESPACE apply -f -
   apiVersion: v1
   kind: Secret
   metadata:
     name: tap-registry
     annotations:
       secretgen.carvel.dev/image-pull-secret: ""
   type: kubernetes.io/dockerconfigjson
   data:
     .dockerconfigjson: e30K
   EOF
   ```

With the two secrets created:

- `tap-registry` is a placeholder secret filled indirectly by `secretgen-controller` Tanzu Application Platform credentials set up during the installation of Tanzu Application Platform.
- `registry-credentials` is a secret providing credentials for the registry where application container images are pushed to.

The following section discusses setting up the identity required for the workload.

ServiceAccount

In a Kubernetes cluster, a ServiceAccount provides a way of representing an actor within the Kubernetes role-based access control (RBAC) system. In the case of a developer namespace, this represents a developer or development team.

You can directly attach secrets to the ServiceAccount through both the `secrets` and `imagePullSecets` fields. This allows you to provide indirect ways for resources to find credentials without knowing the exact name of the secrets.

```
apiVersion: v1
kind: ServiceAccount
metadata:
```

```
  name: default
secrets:
  - name: registry-credentials
  - name: tap-registry
imagePullSecrets:
  - name: registry-credentials
  - name: tap-registry
```

Important

The ServiceAccount must have the secrets created earlier linked to it. If it does not, services like Tanzu Build Service (used in the supply chain) lack the necessary credentials for pushing the images it builds for that workload.

RoleBinding

As the supply chain takes action in the cluster on behalf of the users who created the workload, it needs permissions within Kubernetes' RBAC system to do so.

Tanzu Application Platform v1.2 includes two ClusterRoles that describe all of the necessary permissions to grant to the service account:

- `workload` clusterrole, providing the necessary roles for the supply chains to manage the resources prescribed by them.
- `deliverable` clusterrole, providing the roles for deliveries to deploy to the cluster the application Kubernetes objects produced by the supply chain.

To provide those permissions to the identity you created for this workload, bind the `workload` ClusterRole to the ServiceAccount you created above:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: default-permit-workload
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: workload
subjects:
  - kind: ServiceAccount
    name: default
```

If this is a Build cluster, and you do not intend to run the application in it, this single RoleBinding is all that's necessary.

If you intend to also deploy the application that's been built, bind to the same ServiceAccount the `deliverable` ClusterRole too:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: default-permit-deliverable
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: deliverable
subjects:
  - kind: ServiceAccount
    name: default
```

For more information about authentication and authorization in Tanzu Application Platform, see Overview of Default roles for Tanzu Application Platform.

Developer workload

With the developer namespace set up with the preceding objects, such as secret, serviceaccount, and rolebinding, you can create the workload object.

To do so, use the `apps` plug-in from the Tanzu CLI:

```
tanzu apps workload create FLAGS WORKLOAD-NAME
```

Where:

- `FLAGS` are the one or more flags you want to include.
- `WORKLOAD-NAME` is the name of the workload you want to target.

Depending on what you are aiming to achieve, you can set different flags. To know more about those (including details about different features of the supply chains), see the following sections:

- Building from source, for more information about different ways of creating a workload where the application is built from source code.
- Using an existing image, for more information about how to use prebuilt images in the supply chains.
- GitOps vs RegistryOps, for a description of the different ways of propagating the deployment configuration through external systems (Git repositories and image registries).
- Carvel Package Workflow, for information about how to configure workloads to output Carvel Packages for delivery through Git repositories.

Out of the Box Supply Chain Basic for Supply Chain Choreographer

This topic provides an overview of Out of the Box Supply Chain Basic for Supply Chain Choreographer.

This package contains Cartographer supply chains that tie together a series of Kubernetes resources that drive a developer-provided workload from source code to a Kubernetes configuration ready to be deployed to a cluster. It contains the most basic supply chains that focus on providing a quick path to deployment making no use of testing or scanning resources.

The supply chains in this package perform the following:

- Building from source code:
    1. Watching a Git repository, Maven repository, or local directory for changes
    2. Building a container image out of the source code with Buildpacks
    3. Applying operator-defined conventions to the container definition
    4. Creating a deliverable object for deploying the application to a cluster
    5. (Beta) Alternatively, outputting a Carvel package containing the application to a Git repository. See Carvel Package Supply Chains.
- Using a prebuilt application image:
    1. Applying operator-defined conventions to the container definition
    2. Creating a deliverable object for deploying the application to a cluster

3. (Beta) Alternatively, outputting a Carvel package containing the application to a Git repository. See Carvel Package Supply Chains.

Prerequisites

To use this package, you must:

- Install Out of the Box Templates.
- Configure the Developer namespace with auxiliary objects that are used by the supply chain as described in the following section.
- (Optionally) install Out of the Box Delivery Basic, if you are willing to deploy the application to the same cluster as the workload and supply chains.

Developer Namespace

The supply chains provide definitions of many of the objects that they create to transform the source code to a container image and make it available as an application in a cluster.

The developer must provide or configure particular objects in the developer namespace so that the supply chain can provide credentials and use permissions granted to a specific development team.

The objects that the developer must provide or configure include:

- **registries secrets**: Kubernetes secrets of type `kubernetes.io/dockerconfigjson` that contain credentials for pushing and pulling the container images built by the supply chain and the installation of Tanzu Application Platform.
- **service account**: The identity to be used for any interaction with the Kubernetes API made by the supply chain.
- **rolebinding**: Grant to the identity the necessary roles for creating the resources prescribed by the supply chain.

Registries Secrets

Regardless of the supply chain that a workload goes through, there must be Kubernetes secrets in the developer namespace containing credentials for both pushing and pulling the container image that the supply chain builds when source code is provided. The developer namespace must also contain registry credentials for Kubernetes to run pods using images from the installation of Tanzu Application Platform.

1. Add read/write registry credentials for pushing and pulling application images:

   ```
   tanzu secret registry add registry-credentials \
     --server REGISTRY-SERVER \
     --username REGISTRY-USERNAME \
     --password REGISTRY-PASSWORD \
     --namespace YOUR-NAMESPACE
   ```

   Where:

   - `YOUR-NAMESPACE` is the name you want to use for the developer namespace. For example, use `default` for the default namespace.
   - `REGISTRY-SERVER` is the URL of the registry. For Docker Hub, this must be `https://index.docker.io/v1/`. Specifically, it must have the leading `https://`, the `v1` path, and the trailing `/`. For Google Container Registry (GCR), this is `gcr.io`. Based on the information used in Installing the Tanzu Application Platform package

and profiles, you can use the same registry server as in `ootb_supply_chain_basic` - `registry` - `server`.

2. Add a placeholder secret for gathering the credentials used for pulling container images from the installation of Tanzu Application Platform:

```
cat <<EOF | kubectl -n YOUR-NAMESPACE apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: tap-registry
  annotations:
    secretgen.carvel.dev/image-pull-secret: ""
type: kubernetes.io/dockerconfigjson
data:
  .dockerconfigjson: e30K
EOF
```

With the two secrets created:

- `tap-registry` is a placeholder secret filled indirectly by `secretgen-controller` Tanzu Application Platform credentials set up during the installation of Tanzu Application Platform.
- `registry-credentials` is a secret providing credentials for the registry where application container images are pushed to.

The following section discusses setting up the identity required for the workload.

ServiceAccount

In a Kubernetes cluster, a ServiceAccount provides a way of representing an actor within the Kubernetes role-based access control (RBAC) system. In the case of a developer namespace, this represents a developer or development team.

You can directly attach secrets to the ServiceAccount through both the `secrets` and `imagePullSecets` fields. This allows you to provide indirect ways for resources to find credentials without knowing the exact name of the secrets.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: default
secrets:
  - name: registry-credentials
  - name: tap-registry
imagePullSecrets:
  - name: registry-credentials
  - name: tap-registry
```

Important

The ServiceAccount must have the secrets created earlier linked to it. If it does not, services like Tanzu Build Service (used in the supply chain) lack the necessary credentials for pushing the images it builds for that workload.

RoleBinding

As the supply chain takes action in the cluster on behalf of the users who created the workload, it needs permissions within Kubernetes' RBAC system to do so.

Tanzu Application Platform v1.2 includes two ClusterRoles that describe all of the necessary permissions to grant to the service account:

- `workload` clusterrole, providing the necessary roles for the supply chains to manage the resources prescribed by them.
- `deliverable` clusterrole, providing the roles for deliveries to deploy to the cluster the application Kubernetes objects produced by the supply chain.

To provide those permissions to the identity you created for this workload, bind the `workload` ClusterRole to the ServiceAccount you created above:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: default-permit-workload
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: workload
subjects:
  - kind: ServiceAccount
    name: default
```

If this is a Build cluster, and you do not intend to run the application in it, this single RoleBinding is all that's necessary.

If you intend to also deploy the application that's been built, bind to the same ServiceAccount the `deliverable` ClusterRole too:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: default-permit-deliverable
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: deliverable
subjects:
  - kind: ServiceAccount
    name: default
```

For more information about authentication and authorization in Tanzu Application Platform, see Overview of Default roles for Tanzu Application Platform.

Developer workload

With the developer namespace set up with the preceding objects, such as secret, serviceaccount, and rolebinding, you can create the workload object.

To do so, use the `apps` plug-in from the Tanzu CLI:

```
tanzu apps workload create FLAGS WORKLOAD-NAME
```

Where:

- `FLAGS` are the one or more flags you want to include.
- `WORKLOAD-NAME` is the name of the workload you want to target.

Depending on what you are aiming to achieve, you can set different flags. To know more about those (including details about different features of the supply chains), see the following sections:

- Building from source, for more information about different ways of creating a workload where the application is built from source code.
- Using an existing image, for more information about how to use prebuilt images in the supply chains.
- GitOps vs RegistryOps, for a description of the different ways of propagating the deployment configuration through external systems (Git repositories and image registries).
- Carvel Package Workflow, for information about how to configure workloads to output Carvel Packages for delivery through Git repositories.

Install Out of the Box Supply Chain Basic for Supply Chain Choreographer

This topic shows you how to install the Out of the Box Supply Chain Basic package for Supply Chain Choreographer from the Tanzu Application Platform package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Out of the Box Supply Chain Basic. For more information about profiles, see Components and installation profiles.

The Out of the Box Supply Chain Basic package provides the most basic ClusterSupplyChain that brings an application from source code to a deployed instance of it running in a Kubernetes environment.

Prerequisites

Fulfill the following prerequisites:

- Fulfill the prerequisites for installing Tanzu Application Platform.
- Install Supply Chain Choreographer.

Install

To install Out of the Box Supply Chain Basic:

1. Familiarize yourself with the set of values of the package that can be configured by running:

   ```
   tanzu package available get ootb-supply-chain-basic.tanzu.vmware.com/0.15.6 \
     --values-schema \
     -n tap-install
   ```

   For example:

   ```
   KEY                                  DESCRIPTION

   registry.repository                   Name of the repository in the image regi
   stry server where the application
                                         images from the workload should be pushe
   d (required).

   registry.server                       Name of the registry server where applic
   ation images should be pushed to
                                         (required).
   ```

```
source.credentials_secret              Name of a Secret in the developer namesp
ace which provides the credentials to the
                                       source code repository.

gitops.server_address                  Default server address to be used for fo
rming Git URLs for pushing
                                       Kubernetes configuration produced by the
supply chain. This must
                                       include the scheme/protocol (e.g. http
s:// or ssh://)

gitops.repository_owner                Default project or user of the repositor
y. Used to create URLs for pushing
                                       Kubernetes configuration produced by the
supply chain.

gitops.repository_name                 Default repository name used for forming
Git URLs for pushing Kubernetes
                                       configuration produced by the supply cha
in.

gitops.username                        Default user name to be used for the com
mits produced by the supply chain.

gitops.branch                          Default branch to use for pushing Kubern
etes configuration files produced
                                       by the supply chain.

gitops.commit_message                  Default git commit message to write when
publishing Kubernetes
                                       configuration files produces by the supp
ly chain to git.

gitops.email                           Default user email to be used for the co
mmits produced by the supply chain.

gitops.credentials_secret              Name of a Secret in the developer namesp
ace which provides the credentials to the
                                       GitOps repository.

gitops.ssh_secret                      DEPRECATED: Use gitops.credentials_secre
t and source.credentials_secret instead.
                                       Name of the default Secret containing SS
H credentials to lookup in the
                                       developer namespace for the supply chain
to fetch source code from and
                                       push configuration to.

gitops.commit_strategy                 Specification of how commits are made to
the branch; directly or through a
                                       pull request.

gitops.repository_prefix               DEPRECATED: Use server_address and repos
itory_owner instead.
                                       Default prefix to be used for forming Gi
t SSH URLs for pushing Kubernetes
                                       configuration produced by the supply cha
in.

gitops.pull_request.server_kind        The git source control platform used

gitops.pull_request.commit_branch      The branch to which commits will be mad
e, before opening a pull request
                                       to the branch specified in .gitops.branc
h If the string "" is specified,
                                       an essentially random string will be use
```

```
d for the branch name, in order
                                        to prevent collisions.

gitops.pull_request.pull_request_title The title for the pull request

gitops.pull_request.pull_request_body  Any further information to add to the pu
ll request

cluster_builder                        Name of the Tanzu Build Service ClusterB
uilder to
                                        use by default on image objects managed
by the supply chain.

service_account                        Name of the service account in the names
pace where the Workload
                                        is submitted to utilize for providing re
gistry credentials to
                                        Tanzu Build Service Image objects as wel
l as deploying the
                                        application.

maven.repository.url                   The URL of the Maven repository to be us
ed when pulling Maven
                                        artifacts.  HTTP is not supported.  e.
g.: "https://repo.maven.apache.org/maven"

maven.repository.secret_name           The name of the Secret resource that con
tains the credentials used
                                        to access the Maven repository.
```

2. Create a file named `ootb-supply-chain-basic-values.yaml` that specifies the corresponding values to the properties you want to change. For example:

```
registry:
  server: REGISTRY-SERVER
  repository: REGISTRY-REPOSITORY

source:
  credentials_secret: source-creds

gitops:
  server_address: https://github.com/
  repository_owner: vmware-tanzu
  branch: main
  username: supplychain
  email: supplychain
  commit_message: supplychain@cluster.local
  credentials_secret: gitops-creds
  commit_strategy: direct

maven:
  repository:
    url: https://my-maven-repository/releases
    secret_name: my-maven-repository-credentials

cluster_builder: default
service_account: default
```

3. With the configuration ready, install the package by running:

```
tanzu package install ootb-supply-chain-basic \
  --package ootb-supply-chain-basic.tanzu.vmware.com \
  --version 0.15.6 \
```

```
  --namespace tap-install \
  --values-file ootb-supply-chain-basic-values.yaml
```

Example output:

```
\ Installing package 'ootb-supply-chain-basic.tanzu.vmware.com'
| Getting package metadata for 'ootb-supply-chain-basic.tanzu.vmware.com'
| Creating service account 'ootb-supply-chain-basic-tap-install-sa'
| Creating cluster admin role 'ootb-supply-chain-basic-tap-install-cluster-rol
e'
| Creating cluster role binding 'ootb-supply-chain-basic-tap-install-cluster-ro
lebinding'
| Creating secret 'ootb-supply-chain-basic-tap-install-values'
| Creating package resource
- Waiting for 'PackageInstall' reconciliation for 'ootb-supply-chain-basic'
/ 'PackageInstall' resource install status: Reconciling


 Added installed package 'ootb-supply-chain-basic' in namespace 'tap-install'
```

Out of the Box Supply Chain with Testing for Supply Chain Choreographer

This topic provides an overview of Out of the Box Supply Chain with Testing for Supply Chain Choreographer.

This package contains Cartographer Supply Chains that tie together a series of Kubernetes resources that drive a developer-provided workload from source code to a Kubernetes configuration ready to be deployed to a cluster. It passes the source code forward to image building only if the testing pipeline supplied by the developers runs successfully.

This package includes all the capabilities of the Out of the Box Supply Chain Basic, but adds testing with Tekton.

For workloads that use either source code or prebuilt images, it performs the following:

- Building from source code:
    1. Watching a Git Repository or local directory for changes
    2. Running tests from a developer-provided Tekton pipeline
    3. Building a container image out of the source code with Buildpacks
    4. Applying operator-defined conventions to the container definition
    5. Deploying the application to the same cluster
    6. (Beta) Alternatively, outputting a Carvel Package containing the application to a Git Repository. See Carvel Package Supply Chains.
- Using a prebuilt application image:
    1. Applying operator-defined conventions to the container definition
    2. Creating a deliverable object for deploying the application to a cluster
    3. (Beta) Alternatively, outputting a Carvel Package containing the application to a Git Repository. See Carvel Package Supply Chains.

Prerequisites

To use this supply chain, ensure:

- Out of the Box Templates is installed.
- Out of the Box Supply Chain With Testing **is installed**.
- Out of the Box Supply Chain With Testing and Scanning **is NOT installed**.
- Developer namespace is configured with the objects per Out of the Box Supply Chain Basic guidance. This supply chain is in addition to the basic one.
- (optionally) Install Out of the Box Delivery Basic, if you are willing to deploy the application to the same cluster as the workload and supply chains.

To verify that you have the right set of supply chains installed (that is, the one with Scanning and *not* the one with testing), run:

```
tanzu apps cluster-supply-chain list
```

```
NAME                        LABEL SELECTOR
source-test-to-url          apps.tanzu.vmware.com/has-tests=true,apps.tanzu.vmware.com/w
orkload-type=web
source-to-url               apps.tanzu.vmware.com/workload-type=web
```

If you see `source-test-scan-to-url` in the list, the setup is wrong: you **must not have the *source-test-scan-to-url* installed** at the same time as *source-test-to-url*.

Developer namespace

As mentioned in the prerequisites section, this supply chain builds on the previous Out of the Box Supply Chain, so only additions are included here.

To ensure that you have configured the namespace correctly, the namespace must have the following objects in it (including the ones marked with '*new*' whose explanation and details are provided below):

- **registries secrets**: Kubernetes secrets of type `kubernetes.io/dockerconfigjson` that contain credentials for pushing and pulling the container images built by the supply chain and the installation of Tanzu Application Platform.

  For more information, see Out of the Box Supply Chain Basic.
- **service account**: The identity to be used for any interaction with the Kubernetes API made by the supply chain

  For more information, see Out of the Box Supply Chain Basic.
- **rolebinding**: Grant to the identity the necessary roles for creating the resources prescribed by the supply chain.

  For more information, see Out of the Box Supply Chain Basic.
- **Tekton pipeline** (*new*): A pipeline runs whenever the supply chain hits the stage of testing the source code.

The following sections provide details about the new objects compared to Out of the Box Supply Chain Basic.

Updates to the developer Namespace

For source code testing to be present in the supply chain, a Tekton Pipeline must exist in the same namespace as the Workload so that, at the right moment, the Tekton PipelineRun object that gets created to run the tests can reference such developer-provided Pipeline.

Aside from the objects previously defined in the Out of the Box Supply Chain Basic section, you must include one more:

- `tekton/Pipeline`: the definition of a series of tasks to run against the source code that was found by earlier resources in the Supply Chain.

Tekton/Pipeline

By default, the workload is matched to the corresponding pipeline to run using labels. Pipelines must have the label `apps.tanzu.vmware.com/pipeline: test` at a minimum, but you can add additional labels for granularity. This provides a default match if no other labels are provided. The pipeline expects two parameters:

- `source-url`, an HTTP address where a `.tar.gz` file containing all the source code to be tested is found
- `source-revision`, the revision of the commit or image reference (in case of `workload.spec.source.image` being set instead of `workload.spec.source.git`)

For example:

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: developer-defined-tekton-pipeline
  labels:
    apps.tanzu.vmware.com/pipeline: test      # (!) required
spec:
  params:
    - name: source-url                        # (!) required
    - name: source-revision                   # (!) required
  tasks:
    - name: test
      params:
        - name: source-url
          value: $(params.source-url)
        - name: source-revision
          value: $(params.source-revision)
      taskSpec:
        params:
          - name: source-url
          - name: source-revision
        # Remove this step template for Tanzu Application Platform v1.9.1 when running
on OpenShift.
        stepTemplate:
          securityContext:
            allowPrivilegeEscalation: false
            runAsUser: 1000
            capabilities:
              drop:
                - ALL
            seccompProfile:
              type: "RuntimeDefault"
            runAsNonRoot: true
        steps:
          - name: test
            image: gradle
            script: |-
              cd `mktemp -d`
              wget -qO- $(params.source-url) | tar xvz -m
              ./mvnw test
```

Currently, changes to the developer-provided Tekton Pipeline do not automatically trigger a re-run of the pipeline. That is, a new Tekton PipelineRun is not automatically created if a field in the

Pipeline object is changed. As a workaround, delete the latest PipelineRun to trigger a re-run.

Note

If your cluster has Pod Security Admission enabled, you must update all pipeline tasks to adhere to the admission policy. For more information, see Tekton Tasks on a cluster with Pod Security Admission.

Allow multiple Tekton pipelines in a namespace

You can configure your developer namespace to include more than one pipeline using either of the following methods:

- Use a single pipeline running on a container image that includes testing tools and runs a common script to execute tests. This allows you to accommodate multiple workloads based in different languages in the same namespace that use a common make test script, as shown in the following example:

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: developer-defined-tekton-pipeline
  labels:
    apps.tanzu.vmware.com/pipeline: test
spec:
  #...
        steps:
          - name: test
            image: <image_that_has_JDK_and_Go>
            script: |-
              cd `mktemp -d`
              wget -qO- $(params.source-url) | tar xvz -m
              make test
```

- Update the pipeline resources to include labels that differentiate between the pipelines. For example, differentiate between Java and Go pipelines by adding labels for Java and Go:

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: java-tests
  labels:
    apps.tanzu.vmware.com/pipeline: test
    apps.tanzu.vmware.com/language: java
spec:
  #...
        steps:
          - name: test
            image: gradle
            script: |-
              # ...
              ./mvnw test
---
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: go-tests
  labels:
    apps.tanzu.vmware.com/pipeline: test
    apps.tanzu.vmware.com/language: go
spec:
```

```
  #...
        steps:
          - name: test
            image: golang
            script: |-
              # ...
              go test -v ./...
```

To match the correct pipeline, you add a `testing_pipeline_matching_labels` parameter to the workload. For example, if you want to match to the Java pipeline, you have the following `workload.yaml`:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: sample-java-app
  labels:
    apps.tanzu.vmware.com/has-tests: true
    apps.tanzu.vmware.com/workload-type: web
    app.kubernetes.io/part-of: sample-java-app
spec:
  params:
    - name: testing_pipeline_matching_labels
      value:
        apps.tanzu.vmware.com/pipeline: test
        apps.tanzu.vmware.com/language: java
  ...
```

This matches the workload to the pipeline with the `apps.tanzu.vmware.com/language: java` label.

Developer Workload

With the Tekton Pipeline object submitted to the same namespace as the one where the Workload will be submitted to, you can submit your Workload.

Regardless of the workflow being targeted (local development or gitops), the Workload configuration details are the same as in Out of the Box Supply Chain Basic, except that you mark the workload with tests enabled using the `has-tests` label.

For example:

```
tanzu apps workload create tanzu-java-web-app \
  --git-branch main \
  --git-repo https://github.com/vmware-tanzu/application-accelerator-samples \
  --sub-path tanzu-java-web-app \
  --label apps.tanzu.vmware.com/has-tests=true \
  --label app.kubernetes.io/part-of=tanzu-java-web-app \
  --type web
```

```
Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    apps.tanzu.vmware.com/workload-type: web
      7 + |    apps.tanzu.vmware.com/has-tests: "true"
      8 + |    app.kubernetes.io/part-of: tanzu-java-web-app
      9 + |  name: tanzu-java-web-app
     10 + |  namespace: default
     11 + |spec:
     12 + |  source:
     13 + |    git:
```

```
14 + |      ref:
15 + |        branch: main
16 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
17 + |    subPath: tanzu-java-web-app
```

Out of the Box Supply Chain with Testing for Supply Chain Choreographer

This topic provides an overview of Out of the Box Supply Chain with Testing for Supply Chain Choreographer.

This package contains Cartographer Supply Chains that tie together a series of Kubernetes resources that drive a developer-provided workload from source code to a Kubernetes configuration ready to be deployed to a cluster. It passes the source code forward to image building only if the testing pipeline supplied by the developers runs successfully.

This package includes all the capabilities of the Out of the Box Supply Chain Basic, but adds testing with Tekton.

For workloads that use either source code or prebuilt images, it performs the following:

- Building from source code:
    1. Watching a Git Repository or local directory for changes
    2. Running tests from a developer-provided Tekton pipeline
    3. Building a container image out of the source code with Buildpacks
    4. Applying operator-defined conventions to the container definition
    5. Deploying the application to the same cluster
    6. (Beta) Alternatively, outputting a Carvel Package containing the application to a Git Repository. See Carvel Package Supply Chains.
- Using a prebuilt application image:
    1. Applying operator-defined conventions to the container definition
    2. Creating a deliverable object for deploying the application to a cluster
    3. (Beta) Alternatively, outputting a Carvel Package containing the application to a Git Repository. See Carvel Package Supply Chains.

Prerequisites

To use this supply chain, ensure:

- Out of the Box Templates is installed.
- Out of the Box Supply Chain With Testing **is installed**.
- Out of the Box Supply Chain With Testing and Scanning **is NOT installed**.
- Developer namespace is configured with the objects per Out of the Box Supply Chain Basic guidance. This supply chain is in addition to the basic one.
- (optionally) Install Out of the Box Delivery Basic, if you are willing to deploy the application to the same cluster as the workload and supply chains.

To verify that you have the right set of supply chains installed (that is, the one with Scanning and *not* the one with testing), run:

```
tanzu apps cluster-supply-chain list
```

```
NAME                        LABEL SELECTOR
source-test-to-url          apps.tanzu.vmware.com/has-tests=true,apps.tanzu.vmware.com/w
orkload-type=web
source-to-url               apps.tanzu.vmware.com/workload-type=web
```

If you see `source-test-scan-to-url` in the list, the setup is wrong: you **must not have the *source-test-scan-to-url* installed** at the same time as *source-test-to-url*.

Developer namespace

As mentioned in the prerequisites section, this supply chain builds on the previous Out of the Box Supply Chain, so only additions are included here.

To ensure that you have configured the namespace correctly, the namespace must have the following objects in it (including the ones marked with '*new*' whose explanation and details are provided below):

- **registries secrets**: Kubernetes secrets of type `kubernetes.io/dockerconfigjson` that contain credentials for pushing and pulling the container images built by the supply chain and the installation of Tanzu Application Platform.

  For more information, see Out of the Box Supply Chain Basic.

- **service account**: The identity to be used for any interaction with the Kubernetes API made by the supply chain

  For more information, see Out of the Box Supply Chain Basic.

- **rolebinding**: Grant to the identity the necessary roles for creating the resources prescribed by the supply chain.

  For more information, see Out of the Box Supply Chain Basic.

- **Tekton pipeline** (*new*): A pipeline runs whenever the supply chain hits the stage of testing the source code.

The following sections provide details about the new objects compared to Out of the Box Supply Chain Basic.

Updates to the developer Namespace

For source code testing to be present in the supply chain, a Tekton Pipeline must exist in the same namespace as the Workload so that, at the right moment, the Tekton PipelineRun object that gets created to run the tests can reference such developer-provided Pipeline.

Aside from the objects previously defined in the Out of the Box Supply Chain Basic section, you must include one more:

- `tekton/Pipeline`: the definition of a series of tasks to run against the source code that was found by earlier resources in the Supply Chain.

Tekton/Pipeline

By default, the workload is matched to the corresponding pipeline to run using labels. Pipelines must have the label `apps.tanzu.vmware.com/pipeline: test` at a minimum, but you can add additional labels for granularity. This provides a default match if no other labels are provided. The pipeline expects two parameters:

- `source-url`, an HTTP address where a `.tar.gz` file containing all the source code to be tested is found

- `source-revision`, the revision of the commit or image reference (in case of `workload.spec.source.image` being set instead of `workload.spec.source.git`)

For example:

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: developer-defined-tekton-pipeline
  labels:
    apps.tanzu.vmware.com/pipeline: test      # (!) required
spec:
  params:
    - name: source-url                        # (!) required
    - name: source-revision                   # (!) required
  tasks:
    - name: test
      params:
        - name: source-url
          value: $(params.source-url)
        - name: source-revision
          value: $(params.source-revision)
      taskSpec:
        params:
          - name: source-url
          - name: source-revision
        # Remove this step template for Tanzu Application Platform v1.9.1 when running
on OpenShift.
        stepTemplate:
          securityContext:
            allowPrivilegeEscalation: false
            runAsUser: 1000
            capabilities:
              drop:
                - ALL
            seccompProfile:
              type: "RuntimeDefault"
            runAsNonRoot: true
        steps:
          - name: test
            image: gradle
            script: |-
              cd `mktemp -d`
              wget -qO- $(params.source-url) | tar xvz -m
              ./mvnw test
```

Currently, changes to the developer-provided Tekton Pipeline do not automatically trigger a re-run of the pipeline. That is, a new Tekton PipelineRun is not automatically created if a field in the Pipeline object is changed. As a workaround, delete the latest PipelineRun to trigger a re-run.

Note

If your cluster has Pod Security Admission enabled, you must update all pipeline tasks to adhere to the admission policy. For more information, see Tekton Tasks on a cluster with Pod Security Admission.

Allow multiple Tekton pipelines in a namespace

You can configure your developer namespace to include more than one pipeline using either of the following methods:

- Use a single pipeline running on a container image that includes testing tools and runs a common script to execute tests. This allows you to accommodate multiple workloads based in different languages in the same namespace that use a common make test script, as shown in the following example:

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: developer-defined-tekton-pipeline
  labels:
    apps.tanzu.vmware.com/pipeline: test
spec:
  #...
        steps:
          - name: test
            image: <image_that_has_JDK_and_Go>
            script: |-
              cd `mktemp -d`
              wget -qO- $(params.source-url) | tar xvz -m
              make test
```

- Update the pipeline resources to include labels that differentiate between the pipelines. For example, differentiate between Java and Go pipelines by adding labels for Java and Go:

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: java-tests
  labels:
    apps.tanzu.vmware.com/pipeline: test
    apps.tanzu.vmware.com/language: java
spec:
  #...
        steps:
          - name: test
            image: gradle
            script: |-
              # ...
              ./mvnw test
---
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: go-tests
  labels:
    apps.tanzu.vmware.com/pipeline: test
    apps.tanzu.vmware.com/language: go
spec:
  #...
        steps:
          - name: test
            image: golang
            script: |-
              # ...
              go test -v ./...
```

To match the correct pipeline, you add a `testing_pipeline_matching_labels` parameter to the workload. For example, if you want to match to the Java pipeline, you have the following `workload.yaml`:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
```

```
  name: sample-java-app
  labels:
    apps.tanzu.vmware.com/has-tests: true
    apps.tanzu.vmware.com/workload-type: web
    app.kubernetes.io/part-of: sample-java-app
spec:
  params:
    - name: testing_pipeline_matching_labels
      value:
        apps.tanzu.vmware.com/pipeline: test
        apps.tanzu.vmware.com/language: java
  ...
```

This matches the workload to the pipeline with the `apps.tanzu.vmware.com/language: java` label.

Developer Workload

With the Tekton Pipeline object submitted to the same namespace as the one where the Workload will be submitted to, you can submit your Workload.

Regardless of the workflow being targeted (local development or gitops), the Workload configuration details are the same as in Out of the Box Supply Chain Basic, except that you mark the workload with tests enabled using the `has-tests` label.

For example:

```
tanzu apps workload create tanzu-java-web-app \
  --git-branch main \
  --git-repo https://github.com/vmware-tanzu/application-accelerator-samples \
  --sub-path tanzu-java-web-app \
  --label apps.tanzu.vmware.com/has-tests=true \
  --label app.kubernetes.io/part-of=tanzu-java-web-app \
  --type web
```

```
Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    apps.tanzu.vmware.com/workload-type: web
      7 + |    apps.tanzu.vmware.com/has-tests: "true"
      8 + |    app.kubernetes.io/part-of: tanzu-java-web-app
      9 + |  name: tanzu-java-web-app
     10 + |  namespace: default
     11 + |spec:
     12 + |  source:
     13 + |    git:
     14 + |      ref:
     15 + |        branch: main
     16 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
     17 + |    subPath: tanzu-java-web-app
```

Install Out of the Box Supply Chain with Testing for Supply Chain Choreographer

This topic describes how you can install Out of the Box Supply Chain with Testing for Supply Chain Choreographer from the Tanzu Application Platform package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Out of the Box Supply Chain with Testing. For more information about profiles, see Components and installation profiles.

The Out of the Box Supply Chain with Testing package provides a ClusterSupplyChain that brings an application from source code to a deployed instance that:

- Runs in a Kubernetes environment.
- Runs developer-provided tests in the form of Tekton/Pipeline objects to validate the source code before building container images.

Prerequisites

Before installing Out of the Box Supply Chain with Testing:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- Install cartographer. For more information, see Install Supply Chain Choreographer.
- Install Out of the Box Delivery Basic
- Install Out of the Box Templates

Install

Install by following these steps:

1. Ensure you do not have Out of the Box Supply Chain With Testing and Scanning (`ootb-supply-chain-testing-scanning.tanzu.vmware.com`) installed:
    1. Run the following command:

        ```
        tanzu package installed list --namespace tap-install
        ```

    2. Verify `ootb-supply-chain-testing-scanning` is in the output:

        ```
        NAME                                PACKAGE-NAME
        ootb-delivery-basic                 ootb-delivery-basic.tanzu.vmware.com
        ootb-supply-chain-basic             ootb-supply-chain-basic.tanzu.vmware.
        com
        ootb-templates                      ootb-templates.tanzu.vmware.com
        ```

    3. If you see `ootb-supply-chain-testing-scanning` in the list, uninstall it by running:

        ```
        tanzu package installed delete ootb-supply-chain-testing-scanning --names
        pace tap-install
        ```

        Example output:

        ```
        Deleting installed package 'ootb-supply-chain-testing-scanning' in namesp
        ace 'tap-install'.
        Are you sure? [y/N]: y

        | Uninstalling package 'ootb-supply-chain-testing-scanning' from namespac
        e 'tap-install'
        \ Getting package install for 'ootb-supply-chain-testing-scanning'
        - Deleting package install 'ootb-supply-chain-testing-scanning' from name
        space 'tap-install'
        | Deleting admin role 'ootb-supply-chain-testing-scanning-tap-install-clu
        ```

```
ster-role'
| Deleting role binding 'ootb-supply-chain-testing-scanning-tap-install-c
luster-rolebinding'
| Deleting secret 'ootb-supply-chain-testing-scanning-tap-install-values'
| Deleting service account 'ootb-supply-chain-testing-scanning-tap-instal
l-sa'

 Uninstalled package 'ootb-supply-chain-testing-scanning' from namespace
'tap-install'
```

2. Verify that the values of the package can be configured by referencing the values below:

```
KEY                                DESCRIPTION

registry.repository                 Name of the repository in the image regi
stry server where the application
                                    images from the workload should be pushe
d (required).

registry.server                     Name of the registry server where applic
ation images should be pushed to
                                    (required).

source.credentials_secret           Name of a Secret in the developer namesp
ace which provides the credentials to the
                                    source code repository.

gitops.server_address               Default server address to be used for fo
rming Git URLs for pushing
                                    Kubernetes configuration produced by the
supply chain. This must
                                    include the scheme/protocol (e.g. http
s:// or ssh://)

gitops.repository_owner             Default project or user of the repositor
y. Used to create URLs for pushing
                                    Kubernetes configuration produced by the
supply chain.

gitops.repository_name              Default repository name used for forming
Git URLs for pushing Kubernetes
                                    configuration produced by the supply cha
in.

gitops.username                     Default user name to be used for the com
mits produced by the supply chain.

gitops.branch                       Default branch to use for pushing Kubern
etes configuration files produced
                                    by the supply chain.

gitops.commit_message               Default git commit message to write when
publishing Kubernetes
                                    configuration files produces by the supp
ly chain to git.

gitops.email                        Default user email to be used for the co
mmits produced by the supply chain.

gitops.credentials_secret           Name of a Secret in the developer namesp
ace which provides the credentials to the
                                    GitOps repository.

gitops.ssh_secret                   DEPRECATED: Use gitops.credentials_secre
t and source.credentials_secret instead.
                                    Name of the default Secret containing SS
```

```
H credentials to lookup in the
                                        developer namespace for the supply chain
to fetch source code from and
                                        push configuration to.

gitops.commit_strategy                  Specification of how commits are made to
the branch; directly or through a
                                        pull request.

gitops.repository_prefix                DEPRECATED: Use server_address and repos
itory_owner instead.
                                        Default prefix to be used for forming Gi
t SSH URLs for pushing Kubernetes
                                        configuration produced by the supply cha
in.

gitops.pull_request.server_kind          The git source control platform used

gitops.pull_request.commit_branch        The branch to which commits will be mad
e, before opening a pull request
                                        to the branch specified in .gitops.branc
h If the string "" is specified,
                                        an essentially random string will be use
d for the branch name, in order
                                        to prevent collisions.

gitops.pull_request.pull_request_title  The title for the pull request

gitops.pull_request.pull_request_body   Any further information to add to the p
ull request

cluster_builder           Name of the Tanzu Build Service ClusterBuilder to
                          use by default on image objects managed by the supply
chain.

service_account           Name of the service account in the namespace where th
e Workload
                          is submitted to utilize for providing registry creden
tials to
                          Tanzu Build Service Image objects as well as deployin
g the
                          application.
```

3. Create a file named `ootb-supply-chain-testing-values.yaml` that specifies the corresponding values to the properties you want to change. For example:

```
registry:
  server: REGISTRY-SERVER
  repository: REGISTRY-REPOSITORY

source:
  credentials_secret: source-creds

gitops:
  server_address: https://github.com/
  repository_owner: vmware-tanzu
  branch: main
  username: supplychain
  email: supplychain
  commit_message: supplychain@cluster.local
  credentials_secret: gitops-creds
  commit_strategy: direct

cluster_builder: default
service_account: default
```

4. With the configuration ready, install the package by running:

```
tanzu package install ootb-supply-chain-testing \
  --package ootb-supply-chain-testing.tanzu.vmware.com \
  --version 0.15.6 \
  --namespace tap-install \
  --values-file ootb-supply-chain-testing-values.yaml
```

Example output:

```
\ Installing package 'ootb-supply-chain-testing.tanzu.vmware.com'
| Getting package metadata for 'ootb-supply-chain-testing.tanzu.vmware.com'
| Creating service account 'ootb-supply-chain-testing-tap-install-sa'
| Creating cluster admin role 'ootb-supply-chain-testing-tap-install-cluster-ro
le'
| Creating cluster role binding 'ootb-supply-chain-testing-tap-install-cluster-
rolebinding'
| Creating secret 'ootb-supply-chain-testing-tap-install-values'
| Creating package resource
- Waiting for 'PackageInstall' reconciliation for 'ootb-supply-chain-testing'
\ 'PackageInstall' resource install status: Reconciling

Added installed package 'ootb-supply-chain-testing' in namespace 'tap-install'
```

Out of the Box Supply Chain with Testing and Scanning for Supply Chain Choreographer

This topic provides an overview of Out of the Box Supply Chain with Testing and Scanning for Supply Chain Choreographer.

This package contains Cartographer Supply Chains that tie together a series of Kubernetes resources that drive a developer-provided workload from source code to a Kubernetes configuration ready to be deployed to a cluster. It contains supply chains that pass the source code through testing and vulnerability scanning, and also the container image.

This package includes all the capabilities of the Out of the Box Supply Chain With Testing, but adds source and image scanning using Grype.

Workloads that use source code or prebuilt images perform the following:

- Building from source code:
    1. Watching a Git Repository or local directory for changes
    2. Running tests from a developer-provided Tekton pipeline
    3. Scanning the source code for known vulnerabilities using Grype
    4. Building a container image out of the source code with Buildpacks
    5. Scanning the image for known vulnerabilities
    6. Applying operator-defined conventions to the container definition
    7. Deploying the application to the same cluster
    8. (Beta) Alternatively, outputting a Carvel Package containing the application to a Git repository. See Carvel Package Supply Chains.
- Using a prebuilt application image:
    1. Scanning the image for known vulnerabilities
    2. Applying operator-defined conventions to the container definition

3. Creating a deliverable object for deploying the application to a cluster
4. (Beta) Alternatively, outputting a Carvel package containing the application to a Git repository. See Carvel Package Supply Chains.

Prerequisites

To use this supply chain, verify that:

- Out of the Box Templates is installed.
- Out of the Box Supply Chain With Testing **is NOT installed**.
- Out of the Box Supply Chain With Testing and Scanning **is installed**.
- The developer namespace is configured with the objects according to Out of the Box Supply Chain With Testing guidance. This supply chain exists in addition to the Supply Chain with testing.
- (Optional) Out of the Box Delivery Basic is installed if you want to deploy the application to the same cluster as the workload and supply chains.

Verify that you have the supply chains with scanning, not with testing, installed. Run:

```
tanzu apps cluster-supply-chain list
```

```
NAME                       LABEL SELECTOR
source-test-scan-to-url    apps.tanzu.vmware.com/has-tests=true,apps.tanzu.vmware.com/w
orkload-type=web
source-to-url              apps.tanzu.vmware.com/workload-type=web
```

If you see `source-test-to-url` in the list, the setup is wrong. You **must not have the *source-test-to-url* installed** at the same time as *source-test-scan-to-url*.

Developer namespace

This example builds on the previous Out of the Box Supply Chain examples, so only additions are included here.

To ensure that you configured the namespace correctly, it is important that the namespace has the objects that you configured in the other supply chain setups:

- **registries secrets**: Kubernetes secrets of type `kubernetes.io/dockerconfigjson` that contain credentials for pushing and pulling the container images built by the supply chain and the installation of Tanzu Application Platform.
- **service account**: The identity to be used for any interaction with the Kubernetes API made by the supply chain.
- **rolebinding**: Grant to the identity the necessary roles for creating the resources prescribed by the supply chain.

  For more information about the preceding objects, see Out of the Box Supply Chain Basic.
- **Tekton pipeline**: A pipeline runs whenever the supply chain hits the stage of testing the source code.

  For more information, see Out of the Box Supply Chain Testing.

And the new objects, that you create here:

- **scan policy**: Defines what to do with the results taken from scanning the source code and image produced. For more information, see ScanPolicy section.

- **source scan template**: A template of how TaskRuns are created for scanning the source code. See ScanTemplate section.
- **image scan template**: A template of how TaskRuns are created for scanning the image produced by the supply chain. See ScanTemplate section.

The following section includes details about the new objects, compared to Out of the Box Supply Chain With Testing.

Updates to the developer namespace

For source and image scans, scan templates and scan policies must exist in the same namespace as the workload. These define:

- `ScanTemplate`: how to run a scan, allowing one to change details about the execution of the scan (either for images or source code)
- `ScanPolicy`: how to evaluate whether the artifacts scanned are compliant. For example, allowing one to be either very strict, or restrictive about particular vulnerabilities found.

The names of the objects **must** match the names in the example with default installation configurations. This is overridden either by using the `ootb_supply_chain_testing_scanning` package configuration in the `tap-values.yaml` file or by using workload parameters:

- To override by using the `ootb_supply_chain_testing_scanning` package configuration, make the following modification to your `tap-values.yaml` file and perform a Tanzu Application Platform update.

  ```
  ootb_supply_chain_testing_scanning:
    scanning:
      source:
        policy: SCAN-POLICY
        template: SCAN-TEMPLATE
      image:
        policy: SCAN-POLICY
        template: SCAN-TEMPLATE
  ```

  Where `SCAN-POLICY` and `SCAN-TEMPLATE` are the names of the `ScanPolicy` and `ScanTemplate`.

- To override through workload parameters, use the these commands.

  ```
  tanzu apps workload apply WORKLOAD --param "scanning_source_policy=SCAN-POLICY"
  -n DEV-NAMESPACE
  tanzu apps workload apply WORKLOAD --param "scanning_source_template=SCAN-TEMPL
  ATE" -n DEV-NAMESPACE
  ```

  Where:

  - `WORKLOAD` is the name of the workload.
  - `SCAN-POLICY` and `SCAN-TEMPLATE` are the names of the `ScanPolicy` and `ScanTemplate`.
  - `DEV-NAMESPACE` is the developer namespace.

  For more information, see Create or update a workload.

ScanPolicy

The ScanPolicy defines a set of rules to evaluate for a particular scan to consider the artifacts (image or source code) either compliant or not.

When a ImageScan or SourceScan is created to run a scan, those reference a policy whose name **must** match the following sample `scan-policy`:

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: scan-policy
  labels:
    'app.kubernetes.io/part-of': 'enable-in-gui'
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "UnknownSeve
rity"
    notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]
    ignoreCves := []

    contains(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
      severities := { e | e := match.ratings.rating.severity } | { e | e := match.rati
ngs.rating[_].severity }
      some i
      fails := contains(notAllowedSeverities, severities[i])
      not fails
    }

    isSafe(match) {
      ignore := contains(ignoreCves, match.id)
      ignore
    }

    deny[msg] {
      comps := { e | e := input.bom.components.component } | { e | e := input.bom.comp
onents.component[_] }
      some i
      comp := comps[i]
      vulns := { e | e := comp.vulnerabilities.vulnerability } | { e | e := comp.vulne
rabilities.vulnerability[_] }
      some j
      vuln := vulns[j]
      ratings := { e | e := vuln.ratings.rating.severity } | { e | e := vuln.ratings.r
ating[_].severity }
      not isSafe(vuln)
      msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
    }
```

See Writing Policy Templates.

ScanTemplate

A ScanTemplate defines the PodTemplateSpec used by a TaskRun to run a particular scan (image or source). When the supply chain initiates an ImageScan or SourceScan, they reference these templates which must live in the same namespace as the workload with the names matching the following:

- source scanning (`blob-source-scan-template`)
- image scanning (`private-image-scan-template`)

You can install the Grype ScanTemplates in the namespace that you are writing the workload to with Namespace Provisioner. See Provision developer namespaces in Namespace Provisioner.

1. Label the namespace that you are writing the workload to with the default *namespace_selector* `apps.tanzu.vmware.com/tap-ns=""`.

```
kubectl label namespaces YOUR-DEV-NAMESPACE apps.tanzu.vmware.com/tap-ns=""
```

Note

Although you can customize the templates, if you are following the Getting Started guide, VMware recommends that you follow what is provided in the installation of `grype.scanning.apps.tanzu.vmware.com`. For more information, see About Source and Image Scans.

Enable storing scan results

To enable SCST - Scan to store scan results by using SCST - Store, see Developer namespace setup for exporting the SCST - Store CA certificate and authentication token to the developer namespace.

Allow multiple Tekton pipelines in a namespace

You can configure your developer namespace to include more than one pipeline using either of the following methods:

- Use a single pipeline running on a container image that includes testing tools and runs a common script to execute tests. This allows you to accommodate multiple workloads based in different languages in the same namespace that use a common make test script, as shown in the following example:

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: developer-defined-tekton-pipeline
  labels:
    apps.tanzu.vmware.com/pipeline: test
spec:
  #...
        steps:
          - name: test
            image: <image_that_has_JDK_and_Go>
            script: |-
              cd `mktemp -d`
              wget -qO- $(params.source-url) | tar xvz -m
              make test
```

- Update the pipeline resources to include labels that differentiate between the pipelines. For example, differentiate between Java and Go pipelines by adding labels for Java and Go:

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: java-tests
  labels:
    apps.tanzu.vmware.com/pipeline: test
    apps.tanzu.vmware.com/language: java
spec:
  #...
        steps:
          - name: test
```

```
            image: gradle
            script: |-
              # ...
              ./mvnw test
---
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: go-tests
  labels:
    apps.tanzu.vmware.com/pipeline: test
    apps.tanzu.vmware.com/language: go
spec:
  #...
        steps:
          - name: test
            image: golang
            script: |-
              # ...
              go test -v ./...
```

To match the correct pipeline, you add a `testing_pipeline_matching_labels` parameter to the workload. For example, if you want to match to the Java pipeline, you have the following `workload.yaml`:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: sample-java-app
  labels:
    apps.tanzu.vmware.com/has-tests: true
    apps.tanzu.vmware.com/workload-type: web
    app.kubernetes.io/part-of: sample-java-app
spec:
  params:
    - name: testing_pipeline_matching_labels
      value:
        apps.tanzu.vmware.com/pipeline: test
        apps.tanzu.vmware.com/language: java
  ...
```

This matches the workload to the pipeline with the `apps.tanzu.vmware.com/language: java` label.

Developer workload

With the ScanPolicy and ScanTemplate objects, with the required names set, submitted to the same namespace where the workload is submitted, you are ready to submit your workload.

Regardless of the workflow being targeted, such as local development or GitOps, the workload configuration details are the same as in Out of the Box Supply Chain Basic, except that you mark the workload as having tests enabled.

For example:

```
tanzu apps workload create tanzu-java-web-app \
  --git-branch main \
  --git-repo https://github.com/vmware-tanzu/application-accelerator-samples \
  --sub-path tanzu-java-web-app \
  --label apps.tanzu.vmware.com/has-tests=true \
  --label app.kubernetes.io/part-of=tanzu-java-web-app \
  --type web
```

```
Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    apps.tanzu.vmware.com/workload-type: web
      7 + |    apps.tanzu.vmware.com/has-tests: "true"
      8 + |    app.kubernetes.io/part-of: tanzu-java-web-app
      9 + |  name: tanzu-java-web-app
     10 + |  namespace: default
     11 + |spec:
     12 + |  source:
     13 + |    git:
     14 + |      ref:
     15 + |        branch: main
     16 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
     17 + |    subPath: tanzu-java-web-app
```

CVE triage workflow

The Supply Chain halts progression if either a SourceScan (`sourcescans.scanning.apps.tanzu.vmware.com`) or an ImageScan (`imagescans.scanning.apps.tanzu.vmware.com`) fails policy enforcement through the ScanPolicy (`scanpolicies.scanning.apps.tanzu.vmware.com`). This can prevent source code from building or images deploying that contain vulnerabilities that are in violation of the user-defined scan policy. For information about learning how to handle these vulnerabilities and unblock your Supply Chain, see Triaging and Remediating CVEs.

Scan Images using a different scanner

Supply Chain Security Tools - Scan includes additional integrations for running an image scan using Snyk and VMware Carbon Black.

Out of the Box Supply Chain with Testing and Scanning for Supply Chain Choreographer

This topic provides an overview of Out of the Box Supply Chain with Testing and Scanning for Supply Chain Choreographer.

This package contains Cartographer Supply Chains that tie together a series of Kubernetes resources that drive a developer-provided workload from source code to a Kubernetes configuration ready to be deployed to a cluster. It contains supply chains that pass the source code through testing and vulnerability scanning, and also the container image.

This package includes all the capabilities of the Out of the Box Supply Chain With Testing, but adds source and image scanning using Grype.

Workloads that use source code or prebuilt images perform the following:

- Building from source code:
    1. Watching a Git Repository or local directory for changes
    2. Running tests from a developer-provided Tekton pipeline
    3. Scanning the source code for known vulnerabilities using Grype
    4. Building a container image out of the source code with Buildpacks
    5. Scanning the image for known vulnerabilities

6. Applying operator-defined conventions to the container definition
7. Deploying the application to the same cluster
8. (Beta) Alternatively, outputting a Carvel Package containing the application to a Git repository. See Carvel Package Supply Chains.

- Using a prebuilt application image:
    1. Scanning the image for known vulnerabilities
    2. Applying operator-defined conventions to the container definition
    3. Creating a deliverable object for deploying the application to a cluster
    4. (Beta) Alternatively, outputting a Carvel package containing the application to a Git repository. See Carvel Package Supply Chains.

Prerequisites

To use this supply chain, verify that:

- Out of the Box Templates is installed.
- Out of the Box Supply Chain With Testing **is NOT installed**.
- Out of the Box Supply Chain With Testing and Scanning **is installed**.
- The developer namespace is configured with the objects according to Out of the Box Supply Chain With Testing guidance. This supply chain exists in addition to the Supply Chain with testing.
- (Optional) Out of the Box Delivery Basic is installed if you want to deploy the application to the same cluster as the workload and supply chains.

Verify that you have the supply chains with scanning, not with testing, installed. Run:

```
tanzu apps cluster-supply-chain list
```

```
NAME                       LABEL SELECTOR
source-test-scan-to-url    apps.tanzu.vmware.com/has-tests=true,apps.tanzu.vmware.com/w
orkload-type=web
source-to-url              apps.tanzu.vmware.com/workload-type=web
```

If you see `source-test-to-url` in the list, the setup is wrong. You **must not have the *source-test-to-url* installed** at the same time as *source-test-scan-to-url*.

Developer namespace

This example builds on the previous Out of the Box Supply Chain examples, so only additions are included here.

To ensure that you configured the namespace correctly, it is important that the namespace has the objects that you configured in the other supply chain setups:

- **registries secrets**: Kubernetes secrets of type `kubernetes.io/dockerconfigjson` that contain credentials for pushing and pulling the container images built by the supply chain and the installation of Tanzu Application Platform.
- **service account**: The identity to be used for any interaction with the Kubernetes API made by the supply chain.
- **rolebinding**: Grant to the identity the necessary roles for creating the resources prescribed by the supply chain.

For more information about the preceding objects, see Out of the Box Supply Chain Basic.

- **Tekton pipeline**: A pipeline runs whenever the supply chain hits the stage of testing the source code.

  For more information, see Out of the Box Supply Chain Testing.

And the new objects, that you create here:

- **scan policy**: Defines what to do with the results taken from scanning the source code and image produced. For more information, see ScanPolicy section.
- **source scan template**: A template of how TaskRuns are created for scanning the source code. See ScanTemplate section.
- **image scan template**: A template of how TaskRuns are created for scanning the image produced by the supply chain. See ScanTemplate section.

The following section includes details about the new objects, compared to Out of the Box Supply Chain With Testing.

Updates to the developer namespace

For source and image scans, scan templates and scan policies must exist in the same namespace as the workload. These define:

- `ScanTemplate`: how to run a scan, allowing one to change details about the execution of the scan (either for images or source code)
- `ScanPolicy`: how to evaluate whether the artifacts scanned are compliant. For example, allowing one to be either very strict, or restrictive about particular vulnerabilities found.

The names of the objects **must** match the names in the example with default installation configurations. This is overridden either by using the `ootb_supply_chain_testing_scanning` package configuration in the `tap-values.yaml` file or by using workload parameters:

- To override by using the `ootb_supply_chain_testing_scanning` package configuration, make the following modification to your `tap-values.yaml` file and perform a Tanzu Application Platform update.

  ```
  ootb_supply_chain_testing_scanning:
    scanning:
      source:
        policy: SCAN-POLICY
        template: SCAN-TEMPLATE
      image:
        policy: SCAN-POLICY
        template: SCAN-TEMPLATE
  ```

  Where `SCAN-POLICY` and `SCAN-TEMPLATE` are the names of the `ScanPolicy` and `ScanTemplate`.

- To override through workload parameters, use the these commands.

  ```
  tanzu apps workload apply WORKLOAD --param "scanning_source_policy=SCAN-POLICY"
  -n DEV-NAMESPACE
  tanzu apps workload apply WORKLOAD --param "scanning_source_template=SCAN-TEMPL
  ATE" -n DEV-NAMESPACE
  ```

  Where:

  - `WORKLOAD` is the name of the workload.
  - `SCAN-POLICY` and `SCAN-TEMPLATE` are the names of the `ScanPolicy` and `ScanTemplate`.
  - `DEV-NAMESPACE` is the developer namespace.

For more information, see Create or update a workload.

ScanPolicy

The ScanPolicy defines a set of rules to evaluate for a particular scan to consider the artifacts (image or source code) either compliant or not.

When a ImageScan or SourceScan is created to run a scan, those reference a policy whose name **must** match the following sample `scan-policy`:

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: scan-policy
  labels:
    'app.kubernetes.io/part-of': 'enable-in-gui'
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "UnknownSeve
rity"
    notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]
    ignoreCves := []

    contains(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
      severities := { e | e := match.ratings.rating.severity } | { e | e := match.rati
ngs.rating[_].severity }
      some i
      fails := contains(notAllowedSeverities, severities[i])
      not fails
    }

    isSafe(match) {
      ignore := contains(ignoreCves, match.id)
      ignore
    }

    deny[msg] {
      comps := { e | e := input.bom.components.component } | { e | e := input.bom.comp
onents.component[_] }
      some i
      comp := comps[i]
      vulns := { e | e := comp.vulnerabilities.vulnerability } | { e | e := comp.vulne
rabilities.vulnerability[_] }
      some j
      vuln := vulns[j]
      ratings := { e | e := vuln.ratings.rating.severity } | { e | e := vuln.ratings.r
ating[_].severity }
      not isSafe(vuln)
      msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
    }
```

See Writing Policy Templates.

ScanTemplate

A ScanTemplate defines the PodTemplateSpec used by a TaskRun to run a particular scan (image or source). When the supply chain initiates an ImageScan or SourceScan, they reference these templates which must live in the same namespace as the workload with the names matching the following:

- source scanning (`blob-source-scan-template`)
- image scanning (`private-image-scan-template`)

You can install the Grype ScanTemplates in the namespace that you are writing the workload to with Namespace Provisioner. See Provision developer namespaces in Namespace Provisioner.

1. Label the namespace that you are writing the workload to with the default *namespace_selector* `apps.tanzu.vmware.com/tap-ns=""`.

   ```
   kubectl label namespaces YOUR-DEV-NAMESPACE apps.tanzu.vmware.com/tap-ns=""
   ```

Note

Although you can customize the templates, if you are following the Getting Started guide, VMware recommends that you follow what is provided in the installation of `grype.scanning.apps.tanzu.vmware.com`. For more information, see About Source and Image Scans.

Enable storing scan results

To enable SCST - Scan to store scan results by using SCST - Store, see Developer namespace setup for exporting the SCST - Store CA certificate and authentication token to the developer namespace.

Allow multiple Tekton pipelines in a namespace

You can configure your developer namespace to include more than one pipeline using either of the following methods:

- Use a single pipeline running on a container image that includes testing tools and runs a common script to execute tests. This allows you to accommodate multiple workloads based in different languages in the same namespace that use a common make test script, as shown in the following example:

  ```
  apiVersion: tekton.dev/v1beta1
  kind: Pipeline
  metadata:
    name: developer-defined-tekton-pipeline
    labels:
      apps.tanzu.vmware.com/pipeline: test
  spec:
    #...
          steps:
            - name: test
              image: <image_that_has_JDK_and_Go>
              script: |-
                cd `mktemp -d`
                wget -qO- $(params.source-url) | tar xvz -m
                make test
  ```

- Update the pipeline resources to include labels that differentiate between the pipelines. For example, differentiate between Java and Go pipelines by adding labels for Java and Go:

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: java-tests
  labels:
    apps.tanzu.vmware.com/pipeline: test
    apps.tanzu.vmware.com/language: java
spec:
  #...
        steps:
          - name: test
            image: gradle
            script: |-
              # ...
              ./mvnw test
---
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: go-tests
  labels:
    apps.tanzu.vmware.com/pipeline: test
    apps.tanzu.vmware.com/language: go
spec:
  #...
        steps:
          - name: test
            image: golang
            script: |-
              # ...
              go test -v ./...
```

To match the correct pipeline, you add a `testing_pipeline_matching_labels` parameter to the workload. For example, if you want to match to the Java pipeline, you have the following `workload.yaml`:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: sample-java-app
  labels:
    apps.tanzu.vmware.com/has-tests: true
    apps.tanzu.vmware.com/workload-type: web
    app.kubernetes.io/part-of: sample-java-app
spec:
  params:
    - name: testing_pipeline_matching_labels
      value:
        apps.tanzu.vmware.com/pipeline: test
        apps.tanzu.vmware.com/language: java
  ...
```

This matches the workload to the pipeline with the `apps.tanzu.vmware.com/language: java` label.

Developer workload

With the ScanPolicy and ScanTemplate objects, with the required names set, submitted to the same namespace where the workload is submitted, you are ready to submit your workload.

Regardless of the workflow being targeted, such as local development or GitOps, the workload configuration details are the same as in Out of the Box Supply Chain Basic, except that you mark the workload as having tests enabled.

For example:

```
tanzu apps workload create tanzu-java-web-app \
  --git-branch main \
  --git-repo https://github.com/vmware-tanzu/application-accelerator-samples \
  --sub-path tanzu-java-web-app \
  --label apps.tanzu.vmware.com/has-tests=true \
  --label app.kubernetes.io/part-of=tanzu-java-web-app \
  --type web
```

```
Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    apps.tanzu.vmware.com/workload-type: web
      7 + |    apps.tanzu.vmware.com/has-tests: "true"
      8 + |    app.kubernetes.io/part-of: tanzu-java-web-app
      9 + |  name: tanzu-java-web-app
     10 + |  namespace: default
     11 + |spec:
     12 + |  source:
     13 + |    git:
     14 + |      ref:
     15 + |        branch: main
     16 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
     17 + |    subPath: tanzu-java-web-app
```

CVE triage workflow

The Supply Chain halts progression if either a SourceScan (`sourcescans.scanning.apps.tanzu.vmware.com`) or an ImageScan (`imagescans.scanning.apps.tanzu.vmware.com`) fails policy enforcement through the ScanPolicy (`scanpolicies.scanning.apps.tanzu.vmware.com`). This can prevent source code from building or images deploying that contain vulnerabilities that are in violation of the user-defined scan policy. For information about learning how to handle these vulnerabilities and unblock your Supply Chain, see Triaging and Remediating CVEs.

Scan Images using a different scanner

Supply Chain Security Tools - Scan includes additional integrations for running an image scan using Snyk and VMware Carbon Black.

Install Out of the Box Supply Chain with Testing and Scanning for Supply Chain Choreographer

This topic describes how you can install Out of the Box Supply Chain with Testing and Scanning for Supply Chain Choreographer from the Tanzu Application Platform package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Out of the Box Supply Chain with Testing and Scanning. For more information about profiles, see Components and installation profiles.

The Out of the Box Supply Chain with Testing and Scanning package provides a ClusterSupplyChain that brings an application from source code to a deployed instance that:

- Runs in a Kubernetes environment.
- Performs validations in terms of running application tests.
- Scans the source code and image for vulnerabilities.

Install

To install Out of the Box Supply Chain with Testing and Scanning:

1. Complete all prerequisites to install Tanzu Application Platform.
2. Install Supply Chain Choreographer.
3. Install Out of the Box Delivery Basic.
4. Install Tekton Pipelines.
5. Install Out of the Box Templates.
6. Ensure you do not have Out of The Box Supply Chain With Testing (`ootb-supply-chain-testing.tanzu.vmware.com`) installed:
    1. Run the following command:

        ```
        tanzu package installed list --namespace tap-install
        ```

    2. Inspect to determine if `ootb-supply-chain-testing` is in the output:

        ```
        NAME                                PACKAGE-NAME
        ootb-delivery-basic                 ootb-delivery-basic.tanzu.vmware.com
        ootb-supply-chain-testing           ootb-supply-chain-testing.tanzu.vmwar
        e.com
        ootb-templates                      ootb-templates.tanzu.vmware.com
        ```

    3. If you see `ootb-supply-chain-testing` in the list, uninstall it by running:

        ```
        tanzu package installed delete ootb-supply-chain-testing --namespace tap-
        install
        ```

        Example output:

        ```
        Deleting installed package 'ootb-supply-chain-testing' in namespace 'tap-
        install'.
        Are you sure? [y/N]: y

        | Uninstalling package 'ootb-supply-chain-testing' from namespace 'tap-in
        stall'
        \ Getting package install for 'ootb-supply-chain-testing'
        - Deleting package install 'ootb-supply-chain-testing' from namespace 'ta
        p-install'
        | Deleting admin role 'ootb-supply-chain-testing-tap-install-cluster-rol
        e'
        | Deleting role binding 'ootb-supply-chain-testing-tap-install-cluster-ro
        lebinding'
        | Deleting secret 'ootb-supply-chain-testing-tap-install-values'
        | Deleting service account 'ootb-supply-chain-testing-tap-install-sa'

         Uninstalled package 'ootb-supply-chain-testing' from namespace 'tap-inst
        all'
        ```

7. Check the values of the package that can be configured by running:

```
tanzu package available get ootb-supply-chain-testing-scanning.tanzu.vmware.co
m/0.15.6 \
  --values-schema \
  -n tap-install
```

For example:

```
KEY                                  DESCRIPTION

registry.repository                   Name of the repository in the image regi
stry server where the application
                                      images from the workload should be pushe
d (required).

registry.server                       Name of the registry server where applic
ation images should be pushed to
                                      (required).

source.credentials_secret             Name of a Secret in the developer namesp
ace which provides the credentials to the
                                      source code repository.

gitops.server_address                 Default server address to be used for fo
rming Git URLs for pushing
                                      Kubernetes configuration produced by the
supply chain. This must
                                      include the scheme/protocol (e.g. http
s:// or ssh://)

gitops.repository_owner               Default project or user of the repositor
y. Used to create URLs for pushing
                                      Kubernetes configuration produced by the
supply chain.

gitops.repository_name                Default repository name used for forming
Git URLs for pushing Kubernetes
                                      configuration produced by the supply cha
in.

gitops.username                       Default user name to be used for the com
mits produced by the supply chain.

gitops.branch                         Default branch to use for pushing Kubern
etes configuration files produced
                                      by the supply chain.

gitops.commit_message                 Default git commit message to write when
publishing Kubernetes
                                      configuration files produces by the supp
ly chain to git.

gitops.email                          Default user email to be used for the co
mmits produced by the supply chain.

gitops.credentials_secret             Name of a Secret in the developer namesp
ace which provides the credentials to the
                                      GitOps repository.

gitops.ssh_secret                     DEPRECATED: Use gitops.credentials_secre
t and source.credentials_secret instead.
                                      Name of the default Secret containing SS
H credentials to lookup in the
                                      developer namespace for the supply chain
to fetch source code from and
                                      push configuration to.
```

```
gitops.commit_strategy                 Specification of how commits are made to
the branch; directly or through a
                                       pull request.

gitops.repository_prefix               DEPRECATED: Use server_address and repos
itory_owner instead.
                                       Default prefix to be used for forming Gi
t SSH URLs for pushing Kubernetes
                                       configuration produced by the supply cha
in.

gitops.pull_request.server_kind         The git source control platform used

gitops.pull_request.commit_branch       The branch to which commits will be mad
e, before opening a pull request
                                       to the branch specified in .gitops.branc
h If the string "" is specified,
                                       an essentially random string will be use
d for the branch name, in order
                                       to prevent collisions.

gitops.pull_request.pull_request_title  The title for the pull request

gitops.pull_request.pull_request_body   Any further information to add to the p
ull request

cluster_builder           Name of the Tanzu Build Service ClusterBuilder to
                          use by default on image objects managed by the supply
chain.

service_account           Name of the service account in the namespace where th
e Workload
                          is submitted to utilize for providing registry creden
tials to
                          Tanzu Build Service Image objects as well as deployin
g the
                          application.
```

8. Create a file named `ootb-supply-chain-testing-scanning-values.yaml` that specifies the corresponding values to the properties you want to change. For example:

```
registry:
  server: REGISTRY-SERVER
  repository: REGISTRY-REPOSITORY

source:
  credentials_secret: source-creds

gitops:
  server_address: https://github.com/
  repository_owner: vmware-tanzu
  branch: main
  username: supplychain
  email: supplychain
  commit_message: supplychain@cluster.local
  credentials_secret: gitops-creds
  commit_strategy: direct

cluster_builder: default
service_account: default
```

Important

The `gitops.repository_prefix` field must end with `/`.

9. With the configuration ready, install the package by running:

```
tanzu package install ootb-supply-chain-testing-scanning \
  --package ootb-supply-chain-testing-scanning.tanzu.vmware.com \
  --version 0.15.6 \
  --namespace tap-install \
  --values-file ootb-supply-chain-testing-scanning-values.yaml
```

Example output:

```
\ Installing package 'ootb-supply-chain-testing-scanning.tanzu.vmware.com'
| Getting package metadata for 'ootb-supply-chain-testing-scanning.tanzu.vmwar
e.com'
| Creating service account 'ootb-supply-chain-testing-scanning-tap-install-sa'
| Creating cluster admin role 'ootb-supply-chain-testing-scanning-tap-install-c
luster-role'
| Creating cluster role binding 'ootb-supply-chain-testing-scanning-tap-install
-cluster-rolebinding'
| Creating secret 'ootb-supply-chain-testing-scanning-tap-install-values'
| Creating package resource
- Waiting for 'PackageInstall' reconciliation for 'ootb-supply-chain-testing-sc
anning'
\ 'PackageInstall' resource install status: Reconciling

Added installed package 'ootb-supply-chain-testing-scanning' in namespace 'tap-
install'
```

Out of the Box Templates for Supply Chain Choreographer

This topic describes the templates you can use with Supply Chain Choreographer.

Templates define Kubernetes objects based on configuration in the workload, supply chain Tanzu Application Platform values, and results output from other templated objects. A supply chain organizes a set of templates into a directed acyclic graph. This package contains templates that are used by the Out of the Box Supply Chains and the Out of the Box Delivery. You must install this package to have Workloads delivered properly.

The OOTB Template package includes:

- Cartographer Templates: See reference
- Cartographer ClusterRunTemplates: See reference
- Tekton Tasks
- ClusterRoles
- openshift SecurityContextConstraints

For information about OOTB Supply Chains and Delivery, see:

- Out of the Box Supply Chain Basic
- Out of the Box Supply Chain with Testing
- Out of the Box Supply Chain with Testing and Scanning
- Out of the Box Delivery Basic

Out of the Box Templates for Supply Chain Choreographer

This topic describes the templates you can use with Supply Chain Choreographer.

Templates define Kubernetes objects based on configuration in the workload, supply chain Tanzu Application Platform values, and results output from other templated objects. A supply chain organizes a set of templates into a directed acyclic graph. This package contains templates that are used by the Out of the Box Supply Chains and the Out of the Box Delivery. You must install this package to have Workloads delivered properly.

The OOTB Template package includes:

- Cartographer Templates: See reference
- Cartographer ClusterRunTemplates: See reference
- Tekton Tasks
- ClusterRoles
- openshift SecurityContextConstraints

For information about OOTB Supply Chains and Delivery, see:

- Out of the Box Supply Chain Basic
- Out of the Box Supply Chain with Testing
- Out of the Box Supply Chain with Testing and Scanning
- Out of the Box Delivery Basic

Install Out of the Box Templates for Supply Chain Choreographer

This topic describes how you can install Out of the Box Templates for Supply Chain Choreographer from the Tanzu Application Platform package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Out of the Box Templates. For more information about profiles, see Components and installation profiles.

The Out of the Box Templates package is used by all the Out of the Box Supply Chains to provide the templates that are used by the Supply Chains to create the objects that drive source code all the way to a deployed application in a cluster.

Prerequisites

Before installing Out of the Box Templates:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- Install cartographer. For more information, see Install Supply Chain Choreographer.
- Install Tekton Pipelines.

Install

To install Out of the Box Templates:

1. View the configurable values of the package by running:

```
tanzu package available get ootb-templates.tanzu.vmware.com/0.7.0 \
  --values-schema \
  -n tap-install
```

For example:

```
KEY                  DEFAULT  TYPE    DESCRIPTION
excluded_templates   []       array   List of templates to exclude from the
                                       installation (e.g. ['git-writer'])
```

2. Create a file named `ootb-templates.yaml` that specifies the corresponding values to the properties you want to change.

   For example, the contents of the file might look like this:

   ```
   excluded_templates: []
   ```

3. After the configuration is ready, install the package by running:

   ```
   tanzu package install ootb-templates \
     --package ootb-templates.tanzu.vmware.com \
     --version 0.7.0 \
     --namespace tap-install \
     --values-file ootb-templates-values.yaml
   ```

   Example output:

   ```
   \ Installing package 'ootb-templates.tanzu.vmware.com'
   | Getting package metadata for 'ootb-templates.tanzu.vmware.com'
   | Creating service account 'ootb-templates-tap-install-sa'
   | Creating cluster admin role 'ootb-templates-tap-install-cluster-role'
   | Creating cluster role binding 'ootb-templates-tap-install-cluster-rolebindin
   g'
   | Creating secret 'ootb-templates-tap-install-values'
   | Creating package resource
   - Waiting for 'PackageInstall' reconciliation for 'ootb-templates'
   / 'PackageInstall' resource install status: Reconciling

    Added installed package 'ootb-templates' in namespace 'tap-install'
   ```

Out of the Box Delivery Basic for Supply Chain Choreographer

This topic is an overview of the Out of the Box Delivery Basic package for Supply Chain Choreographer.

This package provides a reusable ClusterDelivery object that delivers the Kubernetes configuration that the Out of the Box Supply Chain produces to an environment, including Basic, Testing, and Testing With Scanning supply chains.

Prerequisites

To make use of this package you must have installed:

- Supply Chain Cartographer
- Out of the Box Templates

Using Out of the Box Delivery Basic

Out of the Box Delivery Basic support both GitOps and local development workflows:

```
GITOPS

    Deliverable:
      points at a git repository where source code is found and
      kubernetes configuration is pushed to


LOCAL DEVELOPMENT

    Deliverable:

      points at a container image registry where the supplychain
      pushes source code and configuration to


---

DELIVERY

    takes a Deliverable (local or gitops) and passes is through
    a series of resources:


           config-provider  <---[config]--- deployer
                 .                             .
                 .                             .
    GitRepository/ImageRepository         kapp-ctrl/App
                                                - knative/Service
                                                - ResourceClaim
                                                - ServiceBinding
                                                ...
```

You must install this package to have Workloads delivered properly with the Basic, Testing, and Testing With Scanning Out of the Box Supply Chains.

Consumers do not interact directly with this package. Instead, this package is used after the supply chains create a carto.run/Deliverable object to express the intention of having the Workloads that go through them delivered to an environment. The environment is the same Kubernetes cluster as the Supply Chains.

More information

- Reference
- Installation

Out of the Box Delivery Basic for Supply Chain Choreographer

This topic is an overview of the Out of the Box Delivery Basic package for Supply Chain Choreographer.

This package provides a reusable ClusterDelivery object that delivers the Kubernetes configuration that the Out of the Box Supply Chain produces to an environment, including Basic, Testing, and Testing With Scanning supply chains.

Prerequisites

To make use of this package you must have installed:

- Supply Chain Cartographer
- Out of the Box Templates

Using Out of the Box Delivery Basic

Out of the Box Delivery Basic support both GitOps and local development workflows:

```
GITOPS

    Deliverable:
      points at a git repository where source code is found and
      kubernetes configuration is pushed to


LOCAL DEVELOPMENT

    Deliverable:

      points at a container image registry where the supplychain
      pushes source code and configuration to


---

DELIVERY

    takes a Deliverable (local or gitops) and passes is through
    a series of resources:


           config-provider  <---[config]--- deployer
                 .                             .
                 .                             .
    GitRepository/ImageRepository         kapp-ctrl/App
                                                - knative/Service
                                                - ResourceClaim
                                                - ServiceBinding
                                                ...
```

You must install this package to have Workloads delivered properly with the Basic, Testing, and Testing With Scanning Out of the Box Supply Chains.

Consumers do not interact directly with this package. Instead, this package is used after the supply chains create a carto.run/Deliverable object to express the intention of having the Workloads that go through them delivered to an environment. The environment is the same Kubernetes cluster as the Supply Chains.

More information

- Reference
- Installation

Install Out of the Box Delivery Basic for Supply Chain Choreographer

This topic shows you how to install the Out of the Box Delivery Basic package for Supply Chain Choreographer from the Tanzu Application Platform package repository.

Note

> Follow the steps in this topic if you do not want to use a profile to install Out of the Box Delivery Basic. For more information about profiles, see Components and installation profiles.

The Out of the Box Delivery Basic package is used by all the Out of the Box Supply Chains to deliver the objects that have been produced by them to a Kubernetes environment.

Prerequisites

Before installing Out of the Box Delivery Basic:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- Install cartographer. For more information, see Install Supply Chain Choreographer.

Install

To install Out of the Box Delivery Basic:

1. Familiarize yourself with the set of values of the package that can be configured by running:

   ```
   tanzu package available get ootb-delivery-basic.tanzu.vmware.com/0.7.0 \
     --values-schema \
     -n tap-install
   ```

   For example:

   ```
   KEY                  DEFAULT  TYPE    DESCRIPTION
   service_account      default  string  Name of the service account in the
                                         namespace where the Deliverable is
                                         submitted to.
   ```

2. Create a file named `ootb-delivery-basic-values.yaml` that specifies the corresponding values to the properties you want to change.

   For example, the contents of the file might look like this:

   ```
   service_account: default
   ```

3. With the configuration ready, install the package by running:

   ```
   tanzu package install ootb-delivery-basic \
     --package ootb-delivery-basic.tanzu.vmware.com \
     --version 0.7.0 \
     --namespace tap-install \
     --values-file ootb-delivery-basic-values.yaml
   ```

   Example output:

   ```
   \ Installing package 'ootb-delivery-basic.tanzu.vmware.com'
   | Getting package metadata for 'ootb-delivery-basic.tanzu.vmware.com'
   | Creating service account 'ootb-delivery-basic-tap-install-sa'
   | Creating cluster admin role 'ootb-delivery-basic-tap-install-cluster-role'
   | Creating cluster role binding 'ootb-delivery-basic-tap-install-cluster-rolebi
   nding'
   | Creating secret 'ootb-delivery-basic-tap-install-values'
   | Creating package resource
   - Waiting for 'PackageInstall' reconciliation for 'ootb-delivery-basic'
   / 'PackageInstall' resource install status: Reconciling
   ```

```
Added installed package 'ootb-delivery-basic' in namespace 'tap-install'
```

How-to guides for Supply Chain Choreographer for Tanzu

This topic describes the how-to guides you can use for Supply Chain Choreographer for Tanzu.

How-to guides

The following how-to guides apply to Supply Chain Choreographer for Tanzu:

- Install Supply Chain Choreographer
- Install Out of the Box Delivery Basic
- Install Out of the Box Supply Chain Basic
- Install Out of the Box Supply Chain with Testing
- Install Out of the Box Supply Chain with Testing and Scanning
- Install Out of the Box Templates
- Tanzu Build Service Integration
- Building from source
- Git authentication
- Output Carvel Packages from your Supply Chain
- Deploy Carvel Packages using Carvel App CR
- Deploy Carvel Packages using Flux CD Kustomization
- Deploy Carvel Packages using Argo CD
- Use Blue-green deployments with Contour and Carvel Packages
- Use Canary deployments with Contour and Carvel Packages
- Use Blue-green deployments with Flagger

Out of the Box Supply Chain with testing on Jenkins for Supply Chain Choreographer

This topic provides an overview of Out of the Box Supply Chain with testing on Jenkins for Supply Chain Choreographer.

The Out of the Box templates package includes a Tekton `Task` resource, which triggers a build for a specified Jenkins job.

You can configure the Jenkins task in both the Out of the Box Supply Chain with Testing and Out of the Box Supply Chain With Testing and Scanning to trigger a Jenkins job. The task is implemented as a Tekton `Task` and can now run from a Tekton `Pipeline`.

Prerequisites

Follow the instructions from either Out of the Box Supply Chain With Testing or Out of the Box Supply Chain With Testing and Scanning to install the required packages. You need to set up only one of these packages.

Either of these Supply Chains can use the Jenkins service during the `source-tester` phase of the pipeline.

Using the Out of the Box Jenkins Task

The intent of the Jenkins task provided using Out of the Box templates is to help Tanzu Application Platform users to integrate with and make use of the modern application deployment pipeline provided by our platform while maintaining their existing test suites on their Jenkins services.

The Out of the Box Jenkins task makes use of an existing Jenkins Job to run test suites on source code.

Configuring a Jenkins job in an existing Jenkins Pipeline

This section of the guide shows how to configure a Jenkins job that you can kick off by the Tanzu Application Platform Jenkins task.

Example Jenkins Job

Here is an example of a script that you can add to your pipeline that specifies source URL and source revision information for your source code target. This example uses a Jenkins instance that is deployed on a Kubernetes cluster although this is not the only possible configuration for a Jenkins instance.

```
#!/bin/env groovy

pipeline {
  agent {
    // Use an agent that is appropriate
    // for your Jenkins installation.
    // This is only an example
    kubernetes {
      label 'maven'
    }
  }

  stages {
    stage('Checkout code') {
      steps {
        script {
          sourceUrl = params.SOURCE_REVISION
          indexSlash = sourceUrl.indexOf("/")
          if (indexSlash == -1) {
            // TAP 1.6+
            revision = sourceUrl.substring(sourceUrl.indexOf(":") + 1)
          } else {
            // pre TAP 1.6
            revision = sourceUrl.substring(indexSlash + 1)
          }
        }
        sh "git clone ${params.GIT_URL} target"
        dir("target") {
          sh "git checkout ${revision}"
        }
      }
    }

    stage('Maven test') {
      steps {
        container('maven') {
          dir("target") {
            // Example tests with maven
            sh "mvn clean test --no-transfer-progress"
          }
        }
      }
```

```
    }
  }
}
```

Where:

- `SOURCE_URL` **string** The URL of the source code being tested. The `source-provider` resource in the supply chain provides this code and is only resolvable inside the Kubernetes cluster. This URL is only useful if your Jenkins service is running inside the cluster or if there is ingress set up and the Jenkins service can make requests to services inside the cluster.
- `SOURCE_REVISION` **string** The revision of the source code being tested. The format of this value can vary depending on the implementation of the `source_provider` resource. If the `source-provider` is the Flux CD `GitRepository` resource, then the value of the `SOURCE_REVISION` is the Git branch name followed by the commit SHA, both separated by a (`/`) slash character. For example, `main/2b1ed6c3c4f74f15b0e4de2732234eafd050eb1ca`. Your Jenkins pipeline script must extract the commit SHA from the `SOURCE_REVISION` to be useful.

Note

If you can't use the `SOURCE_URL` because your Jenkins service cannot make requests into the Kubernetes cluster, you can supply the source code URL to the Jenkins job with other parameters instead.

The following fields are also required in the Jenkins Job definition

- `SOURCE_REVISION` **string**
- `GIT_URL` **string**

To configure your `Workload` to pass the `GIT_URL` parameter into the Jenkins task:

```
tanzu apps workload create workload \
  --namespace your-test-namespace \
  --git-branch main \
  --git-repo https://your.git/repository.git \
  --label apps.tanzu.vmware.com/has-tests=true \
  --label app.kubernetes.io/part-of=test-workload \
  --param-yaml testing_pipeline_matching_labels='{"apps.tanzu.vmware.com/pipeline":"je
nkins-pipeline"}' \
  --param-yaml testing_pipeline_params='{"secret-name":"my-secret","job-name":"jenkins
-job-name","job-params":"[{\"name\":\"GIT_URL\",\"value\":\"https://your.git/repositor
y.git\"}]"}' \
  --type web \
  --yes
```

The `Workload` is described in the later Developer Workload section.

Create a secret with authentication credentials

A secret must be created in the developer namespace to contain the credentials required to authenticate and interact with your Jenkins instance's builds. The following properties are required:

- `url` **required**: URL of the Jenkins instance that hosts the job, including the scheme. For example: `https://my-jenkins.com`.
- `username` **required**: User name of the user that has access to trigger a build on Jenkins.
- `password` **required**: Password of the user that has access to trigger a build on Jenkins.

- `ca-cert` **optional**: The PEM-encoded CA certificate to verify the Jenkins instance identity.

Use the Kubernetes CLI tool (kubectl) to create the secret. You can provide the optional PEM-encoded CA certificate as a file using the `--from-file` flag:

```
kubectl create secret generic my-secret \
  --from-literal=url=https://jenkins.instance \
  --from-literal=username=literal-username \
  --from-file=password=/path/to/file/with/password.txt \
  --from-file=ca-cert=/path/to/ca-certificate.pem \
```

The expected format of the secret is as follows:

```
apiVersion: v1
kind: Secret
metadata:
  name: MY-SECRET # secret name that will be referenced by the workload
type: Opaque
stringData:
  url: JENKINS-URL # target jenkins instance url
  username: USERNAME # jenkins username
  password: PASSWORD # jenkins password or token
  ca-cert: PEM-CA-CERT # PEM encoded certificate
```

Where:

- `MY-SECRET` is the secret name that is referenced by the workload.
- `JENKINS-URL` is the target Jenkins instance URL.
- `USERNAME` is the Jenkins username.
- `PASSWORD` is the Jenkins password or token.
- `PEM-CA-CERT` is the PEM encoded certificate.

Create a Tekton pipeline

The developer must create a Tekton `Pipeline` object with the following parameters:

- `source-url`, **required**: An HTTP address where a `.tar.gz` file containing all the source code being tested is supplied.
- `source-revision`, **required**: The revision of the commit or image reference found by the `source-provider`.
- `secret-name`, **required**: The secret that contains the URL, user name, password, and certificate (optional) to the Jenkins instance that houses the job that is required to run.
- `job-name`, **required**: The name of the Jenkins job that is required to run.
- `job-params`, **required**: A list of key-value pairs, encoded as a JSON string, that passes in parameters needed for the Jenkins job.

Tasks:

- `jenkins-task`, **required**: This `Task` is one of the tasks that the pipeline runs to trigger the Jenkins job. The Out of the Box Templates package installs the `tap-tasks` namespace on the cluster.

Results:

- `jenkins-job-url`: A string result that outputs the URL of the Jenkins build that the Tekton task triggered. The `jenkins-task` `Task` populates the output.

Here is an example of how to create a tekton pipeline with the required parameters

```
cat <<EOF | kubectl apply -f -
---
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: developer-defined-jenkins-tekton-pipeline
  namespace: developer-namespace
  labels:
    #! This label should be provided to the Workload so that
    #! the supply chain can find this pipeline
    apps.tanzu.vmware.com/pipeline: jenkins-pipeline
spec:
  results:
  - name: jenkins-job-url   #! To show the job URL on the
    #! Tanzu Developer Portal
    value: $(tasks.jenkins-task.results.jenkins-job-url)
  params:
  - name: source-url        #! Required
  - name: source-revision   #! Required
  - name: secret-name       #! Required
  - name: job-name          #! Required
  - name: job-params        #! Required
  tasks:
  #! Required: Include the built-in task that triggers the
  #! given job in Jenkins
  - name: jenkins-task
    taskRef:
      resolver: cluster
      params:
        - name: kind
          value: task
        - name: namespace
          value: tap-tasks
        - name: name
          value: jenkins-task
    params:
      - name: source-url
        value: $(params.source-url)
      - name: source-revision
        value: $(params.source-revision)
      - name: secret-name
        value: $(params.secret-name)
      - name: job-name
        value: $(params.job-name)
      - name: job-params
        value: $(params.job-params)
EOF
```

Patching the default Service Account

Tanzu Application Platform includes a Namespace Provisioner which is not enabled by default. This section requires that you do not use the Namespace Provisioner.

The `jenkins-task` `Task` resource uses a container image with the Jenkins Adapter application to trigger the Jenkins job and wait for it to complete. This container image is distributed with Tanzu Application Platform on VMware Tanzu Network, but it is not installed at the same time as the other packages. It is pulled at the time that the supply chain executes the job. As a result, it does not implicitly have access to the `imagePullSecrets` with the required credentials.

Important

The `ServiceAccount` that a developer can configure with their `Workload` is *not* passed to the task and is not used to pull the Jenkins Adapter container image. If you followed the Tanzu Application Platform Install Guide, then you have a `Secret` named `tap-registry` in each of your cluster's namespaces. You can patch the default Service Account in your workload's namespace so that your supply chain can pull the Jenkins Adapter image. For example:

```
kubectl patch serviceaccount default \
  --patch '{"imagePullSecrets": [{"name": "tap-registry"}]}' \
  --namespace developer-namespace
```

Create a Developer Workload

Submit your `Workload` to the same namespace as the Tekton `Pipeline` defined earlier.

To enable the supply chain to run Jenkins tasks, the `Workload` must include the following parameters:

```
parameters:

  #! Required: selects the pipeline
  - name: testing_pipeline_matching_labels
    value:
      #! This label must match the label on the pipeline created earlier
      apps.tanzu.vmware.com/pipeline: jenkins-pipeline

  #! Required: Passes parameters to pipeline
  - name: testing_pipeline_params
    value:

      #! Required: Name of the Jenkins job
      job-name: my-jenkins-job

      #! Required: The secret created earlier to access Jenkins
      secret-name: my-secret

      #! Required: The `job-params` element is required, but the parameter string
      #! might be empty. If empty, then set this value to `[]`.  If non-empty then the
      #! value contains a JSON-encoded list of parameters to pass to the Jenkins job.
      #! Ensure that the quotation marks inside the JSON-encoded string are escaped.
      job-params: "[{\"name\":\"A\",\"value\":\"x\"},{\"name\":\"B\",\"value\":\"y
\"},...]"
```

You can create the workload by using the `apps` CLI plug-in. For example:

```
readonly GIT_BRANCH="my-git-branch"
readonly WORKLOAD_NAME="my-workload-name"
readonly GITHUB_REPO="github-repository-url"
readonly DEVELOPER_WORKSPACE_NAME="my-developer-namespace"

tanzu apps workload create "${WORKLOAD_NAME}" \
  --namespace "${DEVELOPER_WORKSPACE_NAME}" \
  --git-branch "${GIT_BRANCH}" \
  --git-repo "${GITHUB_REPO}" \
  --label apps.tanzu.vmware.com/has-tests=true \
  --label app.kubernetes.io/part-of="${WORKLOAD_NAME}" \
  --param-yaml testing_pipeline_matching_labels='{"apps.tanzu.vmware.com/pipeline":"je
nkins-pipeline"}' \
  --param-yaml testing_pipeline_params='{"secret-name":"jenkins-secret", "job-name":
"jenkins-job", "job-params":"[{"name":"GIT_URL", "value":"https://github.com/spring-pr
```

```
ojects/spring-petclinic"}, {"name":"GIT_BRANCH", "value":"main"}]"}'\
  --type web
```

Where:

- `GIT_URL` is the URL of your GitHub repository.
- `GIT_BRANCH` is the branch you want to target.

The value of the `job-params` parameter is a list of zero-or-more parameters that are sent to the Jenkins job. The parameter is entered into the `Workload` as a list of name-value pairs as shown in the example above.

Important

None of the fields in the `Workload` resource are implicitly passed to the Jenkins job. You have to set them in the `job-params` explicitly. An exception to this is the `SOURCE_URL` and `SOURCE_REVISION` parameters are sent to the Jenkins job implicitly by the Jenkins Adapter trigger application. For example, you can use the `SOURCE_REVISION` to verify which commit SHA to test. See Making a Jenkins Test Job earlier for details about how to use the Git URL and source revision in a Jenkins test job.

Watch the quoting of the `job-params` value closely. In the earlier `tanzu apps workload create` example, the `job-params` value is a string with a JSON structure in it. The value of the `--param-yaml testing_pipeline_params` parameter is a JSON string. Add backslash (`\`) escape characters before the double quote characters (`"`) in the `job-params` value.

Example output from the `tanzu apps workload create` command:

```
Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    app.kubernetes.io/part-of: my-workload-name
      7 + |    apps.tanzu.vmware.com/has-tests: "true"
      8 + |  name: my-workload-name
      9 + |  namespace: developer-namespace
     10 + |spec:
     11 + |  params:
     12 + |  - name: testing_pipeline_matching_labels
     13 + |    value:
     14 + |      apps.tanzu.vmware.com/pipeline: jenkins-pipeline
     15 + |  - name: testing_pipeline_params
     16 + |    value:
     17 + |      job-name: jenkins-job
     18 + |      job-params:
     19 + |      - name: param1
     20 + |        value: value1
     21 + |      secret-name: my-secret
     22 + |  source:
     23 + |    git:
     24 + |      ref:
     25 + |        branch: my-branch
     26 + |      url: https://my-source-code-repository
```

Building container images with Supply Chain Choreographer

This topic describes the methods you can use to build container images for Supply Chain Choreographer for Tanzu.

Methods for building container images

You can build a container image by using:

- A Maven artifact. See Building from source
- A Dockerfile based build. See Dockerfile-based builds
- Tanzu Build Service with buildpacks. See Tanzu Build Service Integration

Building from source with Supply Chain Choreographer

You can build from source by providing source code for the workload with any Supply Chain package.

You can provide source code for the workload from one of three places:

1. A Git repository.
2. A directory in your local computer's file system.
3. A Maven repository.

```
Supply Chain

-- fetch source                 * either from Git or local directory
  -- test
    -- build
      -- scan
        -- apply-conventions
          -- push config
```

This document provides details about each approach.

Note

To provide a prebuilt container image instead of building the application from the beginning by using the supply chain, see Using an existing image.

Git source

To provide source code from a Git repository to the supply chains, you must fill `workload.spec.source.git`. With the Tanzu CLI, you can do so by using the following flags:

- `--git-branch`: branch within the Git repository to checkout
- `--git-commit`: commit SHA within the Git repository to checkout
- `--git-repo`: Git URL to remote source code
- `--git-tag`: tag within the Git repository to checkout

For example, after installing `ootb-supply-chain-basic`, to create a `Workload` the source code for which comes from the `main` branch of the `github.com/vmware-tanzu/application-accelerator-samples` Git repository, and the subdirectory `tanzu-java-web-app` run:

```
tanzu apps workload create tanzu-java-web-app \
  --app tanzu-java-web-app \
  --type web \
```

```
  --git-repo https://github.com/vmware-tanzu/application-accelerator-samples \
  --sub-path tanzu-java-web-app \
  --git-branch main
```

Expect to see the following output:

```
Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    app.kubernetes.io/part-of: tanzu-java-web-app
      7 + |    apps.tanzu.vmware.com/workload-type: web
      8 + |  name: tanzu-java-web-app
      9 + |  namespace: default
     10 + |spec:
     11 + |  source:
     12 + |    git:
     13 + |      ref:
     14 + |        branch: main
     15 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
     16 + |    subPath: tanzu-java-web-app
```

Important

The Git repository URL must include the scheme: `http://`, `https://`, or `ssh://`.

Private `GitRepository`

To fetch source code from a repository that requires credentials, you must provide those by using a Kubernetes secret object that the `GitRepository` object created for that workload references. See How It Works to learn more about detecting changes to the repository.

```
Workload/tanzu-java-web-app
└─GitRepository/tanzu-java-web-app
                  └────────> secretRef: {name: GIT-SECRET-NAME}
                                                  |
                                        either a default from TAP installation or
                                             source_credentials_secret Workload paramete
r
```

Platform operators who install the Out of the Box Supply Chain packages by using Tanzu Application Platform profiles can customize the default name of the secret (`git-ssh`) by editing the corresponding `ootb_supply_chain*` property in the `tap-values.yaml` file:

```
ootb_supply_chain_basic:
  source:
    credentials_secret: GIT-SECRET-NAME
```

For platform operators who install the `ootb-supply-chain-*` package individually by using `tanzu package install`, they can edit the `ootb-supply-chain-*-values.yml` as follows:

```
source:
  credentials_secret: GIT-SECRET-NAME
```

You can also override the default secret name directly in the workload by using the `source_credentials_secret` parameter, regardless of how Tanzu Application Platform is installed. You can use the `--param` flag in Tanzu CLI. For example:

```
tanzu apps workload create tanzu-java-web-app \
  --app tanzu-java-web-app \
  --type web \
  --git-repo https://github.com/vmware-tanzu/application-accelerator-samples \
  --sub-path tanzu-java-web-app \
  --git-branch main \
  --param source_credentials_secret=SECRET-NAME
```

Expect to see the following output:

```
Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    app.kubernetes.io/part-of: tanzu-java-web-app
      7 + |    apps.tanzu.vmware.com/workload-type: web
      8 + |  name: tanzu-java-web-app
      9 + |  namespace: default
     10 + |spec:
     11 + |  params:
     12 + |  - name: source_credentials_secret  #! parameter that overrides the default
     13 + |    value: GIT-SECRET-NAME     #! secret name
     14 + |  source:
     15 + |    git:
     16 + |      ref:
     17 + |        branch: main
     18 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
     19 + |    subPath: tanzu-java-web-app
```

Note

A secret reference is only provided to `GitRepository` if `source_credentials_secret` is set to a non-empty string in some fashion, either by a package property or a workload parameter. To force a `GitRepository` to not reference a secret, set the value to an empty string (`""`).

After defining the name of the Kubernetes secret, you can define the secret.

HTTP(S) Basic-authentication and Token-based authentication

Use HTTP(S) transport:

1. Ensure that the repository in the `Workload` specification uses `http://` or `https://` schemes in any URLs that relate to the repositories. For example, `https://github.com/my-org/my-repo` instead of `github.com/my-org/my-repo` or `ssh://github.com:my-org/my-repo`.
2. In the same namespace as the workload, create a Kubernetes secret object of type `kubernetes.io/basic-auth` with the name matching the one expected by the supply chain. For example:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: GIT-SECRET-NAME
     annotations:
       tekton.dev/git-0: GIT-SERVER        # ! required
   type: kubernetes.io/basic-auth
   stringData:
   ```

```
  username: GIT-USERNAME
  password: GIT-PASSWORD
```

HTTPS with a Custom CA Certificate

For Git repositories hosted with a custom CA, setup a Kubernetes secret with a custom CA. For more information, see HTTPS with a Custom CA Certificate.

SSH authentication

Aside from using HTTP(S) as a transport, you can also use SSH:

1. Ensure that the repository URL in the workload specification uses `ssh://` as the scheme in the URL, for example, `ssh://git@github.com:my-org/my-repo.git`
2. Create a Kubernetes secret object of type `kubernetes.io/ssh-auth`:

```
apiVersion: v1
kind: Secret
metadata:
  name: GIT-SECRET-NAME
  annotations:
    tekton.dev/git-0: GIT-SERVER
type: kubernetes.io/ssh-auth
stringData:
  ssh-privatekey: SSH-PRIVATE-KEY     # private key with push-permissions
  identity: SSH-PRIVATE-KEY           # private key with pull permissions
  identity.pub: SSH-PUBLIC-KEY        # public of the `identity` private key
  known_hosts: GIT-SERVER-PUBLIC-KEYS # git server public keys
```

How it works

With the `workload.spec.source.git` filled, the supply chain takes care of managing a child `GitRepository` object that keeps track of commits made to the Git repository stated in `workload.spec.source.git`.

For each revision found, `gitrepository.status.artifact` gets updated providing information about an HTTP endpoint that the controller makes available for other components to fetch the source code from within the cluster.

The digest of the latest commit:

```
apiVersion: source.toolkit.fluxcd.io/v1
kind: GitRepository
metadata:
  name: tanzu-java-web-app
spec:
  gitImplementation: go-git
  ignore: '!.git'
  interval: 1m0s
  ref: {branch: main}
  timeout: 20s
  url: https://github.com/vmware-tanzu/application-accelerator-samples
status:
  artifact:
    checksum: 375c2daee5fc8657c5c5b49711a8e94d400994d7
    lastUpdateTime: "2022-04-07T15:02:30Z"
    path: gitrepository/default/tanzu-java-web-app/d85df1fc.tar.gz
    revision: main/d85df1fc28c6b86ca54bd613f55991645d3b257c
    url: http://source-controller.flux-system.svc.cluster.local./gitrepository/defaul
t/tanzu-java-web-app/d85df1fc.tar.gz
  conditions:
```

```
  - lastTransitionTime: "2022-04-07T15:02:30Z"
    message: 'Fetched revision: main/d85df1fc28c6b86ca54bd613f55991645d3b257c'
    reason: GitOperationSucceed
    status: "True"
    type: Ready
  observedGeneration: 1
```

Cartographer passes the artifact URL and revision to further components in the supply chain. Those components must consume the source code from an internal URL where a tarball with the source code is fetched, without having to process any Git-specific details in multiple places.

Workload parameters

You can pass the following parameters by using the workload object's `workload.spec.params` field to override the default behavior of the `GitRepository` object created for keeping track of the changes to a repository:

- `gitImplementation`: name of the Git implementation (`go-git`) to fetch the source code.
- `source_credentials_secret`: name of the secret in the same namespace as the workload where credentials to fetch the repository are found.

You can also customize the following parameters with defaults for the whole cluster. Do this by using properties for either `tap-values.yaml` when installing supply chains by using Tanzu Application Platform profiles, or `ootb-supply-chain-*-values.yml` when installing the OOTB packages individually):

- `source.credentials_secret`: the same as `source_credentials_secret` workload parameter

Local source

You can provide source code from a local directory such as, from a directory in the developer's file system. The Tanzu CLI provides two flags to specify the source code location in the file system and where the source code is pushed to as a container image:

- `--local-path`: path on the local file system to a directory of source code to build for the workload
- `--source-image`: destination image repository where source code is staged before being built

This way, whether the cluster the developer targets is local (a cluster in the developer's machine) or not, the source code is made available by using a container image registry.

For example, if a developer has source code under the current directory (`.`) and access to a repository in a container image registry, you can create a workload as follows:

```
tanzu apps workload create tanzu-java-web-app \
  --app tanzu-java-web-app \
  --type web \
  --local-path . \
  --source-image $REGISTRY/test
```

```
Publish source in "." to "REGISTRY-SERVER/REGISTRY-REPOSITORY"?
It may be visible to others who can pull images from that repository

  Yes

Publishing source in "." to "REGISTRY-SERVER/REGISTRY-REPOSITORY"...
Published source

Create workload:
```

```
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    app.kubernetes.io/part-of: tanzu-java-web-app
      7 + |    apps.tanzu.vmware.com/workload-type: web
      8 + |  name: tanzu-java-web-app
      9 + |  namespace: default
     10 + |spec:
     11 + |  source:
     12 + |    image: REGISTRY-SERVER/REGISTRY-REPOSITORY:latest@<digest>
```

Where:

- `REGISTRY-SERVER` is the container image registry.
- `REGISTRY-REPOSITORY` is the repository in the container image registry.

Authentication

Both the cluster and the developer's machine must be configured to properly provide credentials for accessing the container image registry where the local source code is published to.

Developer

The Tanzu CLI must push the source code to the container image registry indicated by `--source-image`. To do so, the CLI must find the credentials, so the developer must configure their machine accordingly.

To ensure credentials are available, use `docker` to make the necessary credentials available for the Tanzu CLI to perform the image push. Run:

```
docker login REGISTRY-SERVER -u REGISTRY-USERNAME -p REGISTRY-PASSWORD
```

Supply chain components

Aside from the developer's ability to push source code to the container image registry, the cluster must also have the proper credentials, so it can pull that container image, unpack it, run tests, and build the application.

To provide the cluster with the credentials, point the ServiceAccount used by the workload at the Kubernetes secret that contains the credentials.

If the registry that the developer targets is the same one for which credentials were provided while setting up the workload namespace, no further action is required. Otherwise, follow the same steps as recommended for the application image.

How it works

A workload specifies that source code must come from an image by setting `workload.spec.source.image` to point at the registry provided by using `--source-image`. Instead of having a GitRepository object created, an ImageRepository object is instantiated, with its specification filled in such a way to keep track of images pushed to the registry provided by the user.

Take the following workload as an example:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
```

```
  name: app
  labels:
    app.kubernetes.io/part-of: app
    apps.tanzu.vmware.com/workload-type: web
spec:
  source:
    image: 10.188.0.3:5000/test:latest
```

Instead of a `GitRepository` object, an `ImageRepository` is created:

```
  Workload/app
  │
- ├─GitRepository/app
+ ├─ImageRepository/app
  │
  ├─Image/app
  │ ├─Build/app-build-1
  │ │ └─Pod/app-build-1-build-pod
  │ ├─PersistentVolumeClaim/app-cache
  │ └─SourceResolver/app-source
  │
  ├─PodIntent/app
  │
  ├─ConfigMap/app
  │
  └─Runnable/app-config-writer
    └─TaskRun/app-config-writer-2zj7w
      └─Pod/app-config-writer-2zj7w-pod
```

`ImageRepository` provides the same semantics as `GitRepository`, except that it looks for source code in container image registries rather than Git repositories.

Maven Artifact

This approach aids integration with existing CI systems, such as Jenkins, and can pull artifacts from existing Maven repositories, including Jfrog Artifactory.

There are no dedicated fields in the `Workload` resource for specifying the Maven artifact configuration. You must fill in the `name`/`value` pairs in the `params` structure.

For example:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: my-workload
  labels:
    apps.tanzu.vmware.com/workload-type: web
spec:
  params:
  - name: maven
    value:
      groupId: com.example
      artifactId: springboot-initial
      version: RELEASE      # latest 'RELEASE' or a specific version (e.g.: '1.2.2')
      type: jar             # optional (defaults to 'jar')
      classifier: sources   # optional
```

There are two ways to create a workload that defines a specific version of a Maven artifact as source in the Tanzu CLI.

The first way is to define the source through CLI flags. For example:

```
tanzu apps workload apply my-workload \
      --maven-artifact springboot-initial \
      --maven-version 2.6.0 \
      --maven-group com.example \
      --type web --app spring-boot-initial -y
```

Another flag that can be used alongside the others in this type of command is `--maven-type`, which refers to the Maven packaging type and defaults to `jar` if not specified.

The second one is through complex params (in JSON or YAML format). To specify the Maven info with this method, run:

```
tanzu apps workload apply my-workload \
      --param-yaml maven='{"artifactId": "springboot-initial", "version": "2.6.0", "gr
oupId": "com.example"}'\
      --type web --app spring-boot-initial -y
```

To create a workload that defines the `RELEASE` version of a maven artifact as source, run:

```
tanzu apps workload apply my-workload \
      --param-yaml maven='{"artifactId": "springboot-initial", "version": "RELEASE",
"groupId": "com.example"}'\
      --type web --app spring-boot-initial -y
```

The Maven repository URL and required credentials are defined in the supply chain, not the workload. For more information, see Installing OOTB Basic.

Maven Repository Secret

The MavenArtifact only supports authentication using basic authentication.

Additionally, MavenArtifact supports security using the TLS protocol. The Application Operator can configure the MavenArtifact to use a custom, or self-signed certificate authority (CA).

The MavenArtifact expects that all of the earlier credentials are provided in one secret, formatted as shown later:

```
---
apiVersion: v1
kind: Secret
metadata:
  name: maven-credentials
type: Opaque
data:
  username: <BASE64>  # basic auth user name
  password: <BASE64>  # basic auth password
  caFile: <BASE64>    # PEM Encoded certificate data for custom CA
```

You cannot use the Tanzu CLI to create secrets such as this, but you can use the kubectl CLI instead.

For example:

```
kubectl create secret generic maven-credentials \
  --from-literal=username=literal-username \
  --from-file=password=/path/to/file/with/password.txt \
  --from-file=caFile=/path/to/ca-certificate.pem
```

Use Dockerfile-based builds with Supply Chain Choreographer

This topic explains how you can use Dockerfile-based builds with Supply Chain Choreographer.

For source-based supply chains, when you specify the `dockerfile` parameter in a workload, the builds switch from using Kpack to using kaniko. Source-based supply chains are supply chains that don't take a pre-built image. kaniko is an open-source tool for building container images from a Dockerfile without running Docker inside a container.

Use Dockerfile-based builds with Supply Chain Choreographer

Parameter name	Description	Example
`dockerfile`	The relative path to the Dockerfile file in the build context.	`./Dockerfile`
`docker_build_context`	The relative path to the directory where the build context is.	`.`
`docker_build_extra_args`	The list of flags to pass directly to kaniko, such as providing arguments to a build.	`- --build-arg=MY_KEY=MY_VALUE`

Example 1

To build a container image from the `github.com/my-foo/bar` repository where the Dockerfile resides in the root of that repository, you can switch from using Kpack to building from that Dockerfile by passing the `dockerfile` parameter:

```
$ tanzu apps workload create my-foo \
  --git-repo https://github.com/my-foo/bar \
  --git-branch dev \
  --label app.kubernetes.io/part-of=foo \
  --param dockerfile=./Dockerfile \
  --type web

  Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    apps.tanzu.vmware.com/workload-type: web
      7 + |  name: my-foo
      8 + |  namespace: dev
      9 + |spec:
     10 + |  params:
     11 + |  - name: dockerfile
     12 + |    value: ./Dockerfile
     13 + |  source:
     14 + |    git:
     15 + |      ref:
     16 + |        branch: dev
     17 + |      url: https://github.com/my-foo/bar
```

Example 2

If the context to be used for the build must be set to a different directory within the repository, use the `docker_build_context` parameter to change that:

```
$ tanzu apps workload create my-foo \
  --git-repo https://github.com/my-foo/bar \
  --git-branch dev \
  --param dockerfile=MyDockerfile \
  --param docker_build_context=./src
```

Important

This feature has no platform operator configurations to be passed through the `tap-values.yaml` file, but if `ootb-supply-chain-*.registry.ca_cert_data` or `shared.ca_cert_data` is configured in `tap-values`, the certificates are considered when pushing the container image.

OpenShift

kaniko can perform container image builds without a Docker daemon or privileged containers. It does require the use of:

- Capabilities usually dropped from the more restrictive SecurityContextConstraints (SCC) enabled by default in OpenShift.
- The root user.

To overcome the limitations imposed by the default unprivileged SCC, Tanzu Application Platform installs:

- `SecurityContextConstraints/ootb-templates-kaniko-restricted-v2-with-anyuid` with enough extra privileges for kaniko to operate.
- `ClusterRole/ootb-templates-kaniko-restricted-v2-with-anyuid` to permit the use of SCC to any actor binding to that cluster role.

Each developer namespace needs a role binding that binds the role to an actor: `ServiceAccount`. For example:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: workload-kaniko-scc
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: ootb-templates-kaniko-restricted-v2-with-anyuid
subjects:
  - kind: ServiceAccount
    name: default
```

With the SCC created and the ServiceAccount bound to the role that permits the use of the SCC, OpenShift accepts the pods created to run kaniko to build the container images.

For more information, see Set up developer namespaces to use your installed packages.

For more information about SCC, see the Openshift documentation.

Tanzu Kubernetes Grid and clusters with PSA enabled

Tanzu Kubernetes Grid v1.26 and later clusters with the Pod Security admission feature enabled and set to `enforce` cannot run kaniko without configuration changes. This is because the webhook

requires containers to run as a non-root user and kaniko needs to run as a root user. This kaniko limitation relates to how image builds are run.

To workaround this limitation, label the namespace that kaniko runs as `privileged`. For example:

```
pod-security.kubernetes.io/enforce: privileged
```

For more information about this kaniko limitation, see the kaniko documentation.

For more information about the Pod Security admission feature, see the Kubernetes documentation.

Tanzu Build Service integration for Supply Chain Choreographer

This topic describes how you can configure and use the Tanzu Build Service integration for Supply Chain Choreographer.

By default, the Out of the Box supply chains (`ootb-supply-chain-*`) in Tanzu Application Platform make use of Tanzu Build Service for building container images out of source code.

You can configure a **platform operator** by using `tap-values.yaml`:

1. The default container image registry where application images must be pushed:

   ```
   ootb_supply_chain_basic:
     registry:
       server: <>
       repository: <>
   ```

2. The name of the Kpack `ClusterBuilder` used by default:

   ```
   ootb_supply_chain_basic:
     cluster_builder: my-custom-cluster-builder
   ```

You can configure an **application operator** by using `Workload`:

- `spec.build.env` are the environment variables used during the build:

  ```
  kind: Workload
  apiVersion: carto.run/v1alpha1
  metadata:
  name: tanzu-java-web-app
  spec:
  # ...
  build:
    env:
      - name: PORT
        value: "8080"
      - name: CA_CERTIFICATE
        valueFrom:
          secretKeyRef:
            name: secret-in-the-same-namespace-as-workload
            key: crt.pem
  ```

- `spec.params.clusterBuilder` is the name of the ClusterBuilder to use for builds of that Workload:

  ```
  kind: Workload
  apiVersion: carto.run/v1alpha1
  metadata:
  name: tanzu-java-web-app
  ```

```
spec:
# ...
params:
  - name: clusterBuilder
    value: nodejs-cluster-builder
```

- `spec.params.buildServiceBindings` is the object carrying the definition of a list of service bindings to use at build time:

```
---
kind: Workload
apiVersion: carto.run/v1alpha1
metadata:
name: tanzu-java-web-app
spec:
# ...
params:
  - name: buildServiceBindings
    value:
      - name: settings-xml
        kind: Secret
        apiVersion: v1
---
apiVersion: v1
kind: Secret
metadata:
name: settings-xml
type: service.binding/maven
stringData:
type: maven
provider: sample
settings.xml: <settings>...</settings>
```

Note

See the Kpack ServiceBinding documentation in GitHub for more details about build-time service bindings.

these configuration only take effect when Kpack is used for building a container image. If you use Dockerfile-based builds by leveraging the `dockerfile` parameter, see dockerfile-based builds for more information.

Configure and deploy to multiple environments with custom parameters

This topic describes how to use carvel packages, Git repositories, and Flux CD to deploy workloads to multiple environments with Supply Chain Choreographer. By using a continuous delivery (CD) tool, you can apply Carvel packages to a runtime.

Overview

Flux CD is the VMware recommended CD tool. You can configure different parameters for each environment, such as replicas or host names. When you edit package parameters and commit them to a Git repository, Flux CD watches the Git repository and applies the package to your runtime environments.

Feature limits

To configure and deploy to multiple environments with custom parameters, ensure that your supply chains are compatible with the feature limits.

This feature is in beta and has the following limitation:

- Innerloop development is not supported.

Using Carvel packages

You can configure your supply chain to outputs Carvel packages and deliver configuration for each environment. For information about using Carvel, see Carvel Package Supply Chains (beta).

Using GitOps delivery with Flux CD

You can deliver packages created by the Carvel package supply chain, and add them to clusters, by using a GitOps repository. For information about this delivery method, see Use Gitops Delivery with Flux CD (beta).

Using GitOps delivery with Carvel App

Alternatively, you can deliver packages created by the Carvel package supply chain, and add them to clusters by using a GitOps repository. For information about this delivery method, see Use Gitops Delivery with Carvel App

Using GitOps delivery with Argo CD

You can deliver packages created by the Carvel package supply chain, and add them to clusters by using a GitOps repository. For information about this delivery method, see Use Gitops Delivery with Argo CD.

Configuring blue-green deployment

You can use blue-green deployment to transfer user traffic from one version of an app to a later version while both are running. For information about setting up blue-green deployment, see Use blue-green deployment with Contour and PackageInstall (beta).

Configuring canary deployment

You can use canary deployment to gradually shift traffic from one version of an application to a later version, and at the same time, perform analysis to verify if the later version is either promoted or rolled back. For information about setting up canary deployments, see Use canary deployment with Contour and Carvel packages for Supply Chain Choreographer.

Carvel Package Supply Chains (beta)

This topic explains what Carvel Package Supply Chains do, how they work, how operators can enable them, and how to create a `Workload` that uses them. You can use the Carvel Package Supply Chains with Supply Chain Choreographer to deliver applications to multiple production environments with configuration for each environment.

The Out of the Box Basic Supply Chain, Out of the Box Supply Chain with Testing, and Out of the Box Supply Chain with Testing and Scanning packages provide variations of the OOTB supply chains that output Carvel Packages.

Overview of the Carvel Package Supply Chains

The out of the box Supply Chains output a `Deliverable` object. These `Deliverable`s are deployed to a cluster by the Out of the Box Delivery Supply Chain. The Carvel Package Supply Chains instead output a Carvel `Package` object to a GitOps repository. These `Packages` have configurable parameters such as `hostname` and `replicas` that are configured per environment. GitOps tools such as Flux CD and Argo CD can deploy the `Package`s onto multiple environments.

Note

The Kubernetes resources created for your `Workload` are the same for Carvel Package supply chains, but with additional, optional resources such as `networking.k8s.io/v1 Ingress`.

What do the Carvel Package Supply Chains do?

There are two Carvel Package Supply Chains per package: - Out of the Box Basic Supply Chain contains `source-to-url-package` and `basic-image-to-url-package`. - Out of the Box Supply Chain with Testing contains `source-test-to-url-package` and `testing-image-to-url-package`. - Out of the Box Supply Chain with Testing and Scanning contains `source-test-scan-to-url-package` and `scanning-image-scan-to-url-package`

These supply chains are identical to their non Carvel Package counterparts, except for three resources:

- A new `carvel-package` resource is added. The supply chain stamps out a Tekton Task that bundles all application Kubernetes resources into a Carvel `Package`.
- `config-writer` is modified to write the Carvel `Package` to a GitOps repository.
- `deliverable` resource is removed.

When a `Workload` is created and all Supply Chain resources are stamped out, a Carvel `Package` is written to the GitOps repository at the path `<package_name>/packages/<package_id>.yaml`. `<package_name>` by default to `<workload_name>.<workload_namespace>.tap`, and is customized with the `name_suffix` parameter. `<package_id>` is a SemVer compatible version generated by the Bash command `$(date "+%Y%m%d%H%M%S.0.0")`.

For example:

```
app.default.tap/
  packages/
    20230321004057.0.0.yaml
```

Note

By default, the `<package_name>` directory is created in the root directory of the GitOps repository. You can optionally create this directory at a subpath by configuring the `gitops_subpath` parameter.

For example, the following Carvel `Package` definition is stored in `<package_id>.yaml`:

```
apiVersion: data.packaging.carvel.dev/v1alpha1
kind: Package
metadata:
  name: app.default.tap.20230321004057.0.0
spec:
```

```
refName: app.default.tap
version: 20230321004057.0.0
releaseNotes: |
  Release v20230321004057.0.0 of package app.default.tap
template:
  spec:
    fetch:
    - imgpkgBundle:
        image: # imgpkg bundle containing all Kubernetes configuration
    template:
    - ytt:
        paths:
        - .
    - kbld:
        paths:
        - .imgpkg/images.yml
        - '-'
    deploy:
    - kapp: {}
valuesSchema:
  openAPIv3:
    type: object
    additionalProperties: false
    properties:
      workload_name:
        title: Workload name
        type: string
        default: ""
      replicas:
        title: Replicas
        type: integer
        default: 1
      port:
        title: Port
        type: integer
        default: 8080
      hostname:
        title: Hostname
        type: string
        default: ""
      cluster_issuer:
        title: Cluster Issuer
        type: string
        default: tap-ingress-selfsigned
      http_route:
        type: object
        additionalProperties: false
        nullable: true
        properties:
          gateways:
            type: array
            items:
              type: object
              additionalProperties: false
              properties:
                protocol:
                  type: string
                  default: ""
                name:
                  type: string
                  default: ""
            default:
            - protocol: https
              name: default-gateway
```

By default, the Carvel `Package` generated by the Supply Chain has configurable parameters.

For the `server` workload type, the default parameters are:

- `workload_name`: Name of the workload. Required. Default is `""`.
- `replicas`: Number of pods that you want for the `apps/v1 Deployment`. Default is `1`.
- `hostname`: Host name for the `networking.k8s.io/v1 Ingress`. If you don't need ingress, leave host name as an empty string. If host name is not in the file, you see an error.
- `port`: Port for the `networking.k8s.io/v1 Ingress`. Default is `8080`.
- `cluster_issuer`: CertManager Issuer to use to generate certificate for Kubernetes `Ingress`. Default is `"tap-ingress-selfsigned"`.
- `http_route`: Configuration of `gateway.networking.k8s.io/v1beta1 HttpRoute`. This is intended for use with Tanzu Application Engine. VMware does not recommend this setting when running on Tanzu Application Platform.

For the `web` workload type, the default parameters are:

- `workload_name`: Name of the workload. Required. Default is `""`.

For the `worker` workload type, the default parameters are:

- `workload_name`: Name of the workload. Required. Default is `""`.
- `replicas`: Number of pods that you want for the `apps/v1 Deployment`. Default is `1`.

You can configure the Supply Chain to produce Carvel `Packages` with custom parameters. See the installation section.

When a new commit is pushed to the source code Git Repository, such as `source-to-url-package`, or a new pre-built image is created, like `basic-image-to-url-package`, the Supply Chain stamps out a new version of the Carvel `Package`. This definition is written to `<package_name>/packages/<package_id>.yaml` with a new `<package_id>`.

The Carvel `Package` stored in GitOps repositories are deployed to multiple run clusters using GitOps tools, such as Flux CD or Argo CD. See Deploy Carvel Packages using Flux CD Kustomization.

Installing the Carvel Package Supply Chains as an Operator

This section describes operator tasks for enabling and configuring the Carvel Package Supply Chains.

Prerequisites

The Carvel Package Supply Chains require access to a GitOps repository and credentials. See GitOps versus RegistryOps.

Installation

In `tap-values`, configure any Out of the Box Supply Chain and the Out of the Box Templates package with the following parameters:

Note

If you are installing Out of the Box Supply Chain with Testing, replace `ootb_supply_chain_basic` with `ootb_supply_chain_testing` in all of the following steps. If you are installing Out of the Box Supply Chain with Testing and Scanning, replace `ootb_supply_chain_basic` with `ootb_supply_chain_testing_scanning` in all of the following steps.

1. (Required) Enable the Carvel Package workflow.

```
ootb_supply_chain_basic:
  carvel_package:
    workflow_enabled: true
```

2. (Optional) Set a GitOps subpath. This verifies the path in your GitOps repository where Carvel Packages are written. Defaults to `""`. See Template reference.

```
ootb_supply_chain_basic:
  carvel_package:
    workflow_enabled: true
    gitops_subpath: path/to/my/dir
```

3. (Optional) Set a name suffix. Carvel Package names are chosen using the `<workload_name>.<workload_namespace>.<name_suffix>` template. Defaults to `tap`. See Template reference.

```
ootb_supply_chain_basic:
  carvel_package:
    workflow_enabled: true
    name_suffix: vmware.com
```

4. (Optional) Enable OpenAPIv3 Schema Definition Generation. Defaults to `true`. See Template reference.

```
ootb_supply_chain_basic:
  carvel_package:
    workflow_enabled: true
    openapiv3_enabled: true
```

   If the size of the resulting OpenAPIv3 specification exceeds roughly 3 KB, the Supply Chain does not function. See the known issue.

5. (Optional) Configure the Out of the Box Templates with custom Carvel Package parameters. See Template reference.

   - Packages created by the Carvel Package Supply Chains contain the YTT Schema and YTT Overlays.
   - Supply Chain decides which YTT Schema and YTT Overlays to use based on the `selector` field. Operators can specify as many labels as they want under `selector.matchLabels`. The `selector` with the highest amount of matching labels is used.
     - A `selector` is only considered a match if all labels under `selector.matchLabels` match. If there is more than one matching `selector`, the `selector` with more labels is used. If there is more than one matching `selector`, and they have the same amount of labels, the Supply Chain fails. If there is no matching `selector`, the Supply Chain fails.
   - The `ootb_templates.carvel_package.openapiv3_enabled = true` package has a generated OpenAPIv3 API specification.
   - Setting `ootb_templates.carvel_package.parameters` overrides the default Carvel Package parameters. If an operator wants to leave the parameters the same for `web` and `worker` types, but add a parameter for `server`, they must copy the default `ootb_templates.carvel_package.parameters` from the template reference, and then make the modification.

```
ootb_templates:
  carvel_package:
```

```
    parameters:
    - selector:
        matchLabels:
          apps.tanzu.vmware.com/workload-type: server
      schema: |
        #@data/values-schema
        ---
        #@schema/title "Replicas"
        #@schema/desc "Number of replicas."
        replicas: 1
      overlays: |
        #@ load("@ytt:overlay", "overlay")
        #@ load("@ytt:data", "data")
        #@overlay/match by=overlay.subset({"apiVersion":"apps/v1", "kind": "Dep
loyment"})
        ---
        spec:
          #@overlay/match missing_ok=True
          replicas: #@ data.values.replicas
    # copy the existing parameters for worker + web here
```

6. Configure your Out of the Box Supply Chain package with your GitOps parameters. See GitOps versus RegistryOps.
7. Install the Out of the Box Supply Chain package.

Verifying the Carvel Package Supply Chains are Installed

1. Run `kubectl get ClusterSupplyChains`.
2. Confirm you see both Carvel Package supply chains with status `Ready: True`:
    - Out of the Box Basic Supply Chain contains `source-to-url-package` and `basic-image-to-url-package`.
    - Out of the Box Supply Chain with Testing contains `source-test-to-url-package` and `testing-image-to-url-package`.
    - Out of the Box Supply Chain with Testing and Scanning contains `source-test-scan-to-url-package` and `scanning-image-scan-to-url-package`

Using the Carvel Package Supply Chains as a Developer

This section describes developer tasks for using the Carvel Package Supply Chains.

Prerequisites

Your operator must install the Carvel Package Supply Chains.

You must create your workload in a developer namespace. See Developer namespace.

Create a Workload

To use the Carvel Package Supply Chains, you must add the label `apps.tanzu.vmware.com/carvel-package-workflow=true` to your workload.

1. Use the `--label apps.tanzu.vmware.com/carvel-package-workflow=true` Tanzu CLI flag.

   For example:

```
tanzu apps workload create tanzu-java-web-app \
--namespace DEVELOPER_NAMESPACE \
--app tanzu-java-web-app \
--type server \
```

```
--label apps.tanzu.vmware.com/carvel-package-workflow=true \
--image springcommunity/spring-framework-petclinic
```

Expect to see the following output:

```
Create workload:
    1 + |---
    2 + |apiVersion: carto.run/v1alpha1
    3 + |kind: Workload
    4 + |metadata:
    5 + |  labels:
    6 + |    app.kubernetes.io/part-of: tanzu-java-web-app
    7 + |    apps.tanzu.vmware.com/carvel-package-workflow: "true"
    8 + |    apps.tanzu.vmware.com/workload-type: server
    9 + |  name: tanzu-java-web-app
   10 + |  namespace: DEVELOPER_NAMESPACE
   11 + |spec:
   12 + |  image: springcommunity/spring-framework-petclinic
```

You can override most parameters set by the operator.

Note

Developers cannot override `carvel_package_parameters` as it presents a security risk.

1. (Optional) Set a GitOps subpath. This verifies the path in your GitOps repository where Carvel Packages are written. Defaults to `""`. See Template reference.

   Set this parameter by modifying `workload.spec.params.carvel_package_gitops_subpath`, using the `--param carvel_package_gitops_subpath=path/to/my/dir` Tanzu CLI flag.

2. (Optional) Set a name suffix. Carvel Package names are chosen using the template `<workload_name>.<workload_namespace>.<name_suffix>`. Defaults to `tap`. See Template reference.

   Set this parameter by modifying `workload.spec.params.carvel_package_name_suffix`, using the `--param carvel_package_name_suffix=vmware.com` Tanzu CLI flag.

3. (Optional) Enable OpenAPIv3 specification generation. Defaults to `true`. See Template reference.

   Set this parameter by modifying `workload.spec.params.carvel_package_openapiv3_enabled`, using the `--param carvel_package_openapiv3_enabled=false` Tanzu CLI flag.

4. (Optional) You can override GitOps parameters. See GitOps versus RegistryOps.

Verify the Carvel Package was Created

Verify that you see a Carvel `Package` stored in your GitOps repository. For example, at the path `tanzu-java-web-app.default.tap/packages/20230321004057.0.0.yaml` you see a valid Carvel `Package` definition.

Git commit SHA in Package version

By default, the Git revision identifier for the software in the Carvel Package is added to the package version, in the format `+build.${git_commit}`, in the following cases:

1. The `source-to-url-package` builds the package and the image building resource is either `kpack-template` or the `kaniko-template`. These templates add the label Git commit SHA of

the source code used to the label `org.opencontainers.image.revision` on the OCI image container.

2. An OCI image container is used with the label `org.opencontainers.image.revision`. The package version might look similar to `20230518150903.0.0+build.6db88c7` where `6db88c7` is the Git commit SHA.

Next Steps

You can deploy the Carvel `Package` using tools such as Flux CD or Argo CD. See Deploy Carvel Packages using Flux CD Kustomization.`

Use Gitops delivery with a Carvel app (beta)

This topic explains how you can deliver Carvel `Packages`, created by the Carvel Package Supply Chains, from a GitOps repository to one or more run clusters using Carvel App. You can use Carvel Package Supply Chains with Supply Chain Choreographer.

Prerequisites

To use GitOps Delivery with Carvel App, you must complete the following prerequisites:

- You must create a `Workload` that uses the Carvel Package supply chains. For information about Carvel Packages, see Carvel Package Supply Chains. You must have at least one Carvel `Package` generated by this `Workload` stored in your GitOps repository.
- You must have at least one run cluster. run clusters serve as your deployment environments. They can either be Tanzu Application Platform clusters, or Kubernetes clusters, but they must have kapp-controller and Contour installed. See the Carvel documentation and the Contour documentation.
- If you plan to use a build cluster to control the deployment on all of the run clusters, you must create a Build cluster that has network access to your run clusters. If you intend to deploy directly on the run cluster without using a build cluster, a build cluster is only necessary for building the package.

Set up run cluster namespaces

Each run cluster must have a namespace and `ServiceAccount` with the correct permissions to deploy the Carvel `Packages`, `PackageInstalls` and Kubernetes `Secrets`. Create a namespace and `ServiceAccount` with the following permissions:

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: RUN-CLUSTER-NS
  name: app-package-and-pkgi-install-role
rules:
  - apiGroups: ["data.packaging.carvel.dev"]
    resources: ["packages"]
    verbs: ["get", "list", "create", "update", "delete", "patch"]
  - apiGroups: ["packaging.carvel.dev"]
    resources: ["packageinstalls"]
    verbs: ["get", "list", "create", "update", "delete", "patch"]
  - apiGroups: [""]
    resources: ["secrets"]
    verbs: ["get", "list", "create", "update", "delete", "patch"]
```

Where `RUN-CLUSTER-NS` is the run cluster namespace you want to use.

If your run cluster is a Tanzu Application Platform cluster, see Set up developer namespaces to use your installed packages.

If your run cluster is not a Tanzu Application Platform cluster, the `ServiceAccount` must also have the following permissions:

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: RUN-CLUSTER-NS
  name: app-cr-role
rules:
- apiGroups: ["apps"]
  resources: ["deployments"]
  verbs: ["get", "list", "create", "update", "delete"]
- apiGroups: [""]
  resources: ["configmaps", "services"]
  verbs: ["get", "list", "create", "update", "delete"]
- apiGroups: ["networking.k8s.io"]
  resources: ["ingresses"]
  verbs: ["get", "list", "create", "update", "delete"]
```

Where `RUN-CLUSTER-NS` is the name of your run cluster you want to create a namespace with.

Create Carvel PackageInstalls and secrets

For each Carvel `Package` and each run cluster, you must create a Carvel `PackageInstall` and a `Secret`. The Carvel `PackageInstall` and the `Secret` is stored in your GitOps repository and deployed to run clusters by the Carvel `App`.

The following example shows GitOps repository structure after completing the procedures in this section:

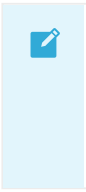
```
app.default.tap/
  packages/
    20230321004057.0.0.yaml  # Package
  staging/
    packageinstall.yaml      # PackageInstall
    params.yaml              # Secret
  prod/
    packageinstall.yaml      # PackageInstall
    params.yaml              # Secret
```

1. For each run cluster, create a `Secret` that has the values for each `Package` parameter. To see the configurable properties of the `Package`, inspect the `Package` CR's valuesSchema. See Carvel Package Supply Chains. Store the `Secret` in your GitOps repository at `PACKAGE-NAME/RUN-CLUSTER/params.yaml`.

   ```
   ---
   apiVersion: v1
   kind: Secret
   metadata:
    name: app-values
   stringData:
    values.yaml: |
      ---
      workload_name: app
      replicas: 2
      hostname: app.mycompany.com
   ```

Where:

- `PACKAGE-NAME` is the name of your Carvel package you want to use.
- `RUN-CLUSTER` is the name of the run cluster you want to use with the package.



Note

You must set a value for the `workload_name` parameter. You can skip setting other fields to use the default parameter values.

1. For each run cluster, create a `PackageInstall`. Reference the `Secret` you created earlier. Store the `PackageInstall` in your GitOps repository at `PACKAGE-NAME/RUN-CLUSTER/packageinstall.yaml`.

```
---
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
 name: app
spec:
 serviceAccountName: RUN-CLUSTER-NS-SA # ServiceAccount on run cluster with per
missions to deploy Package, see "Set up run Cluster Namespaces"
 packageRef:
   refName: app.default.tap # name of the Package
   versionSelection:
     constraints: 20230321004057.0.0 # version of the Package
 values:
 - secretRef:
     name: app-values # Secret created in previous step
```

Where:

- `PACKAGE-NAME` is the name of your Carvel package you want to use.
- `RUN-CLUSTER` is the name of the run cluster you want to use with the package.
- `RUN-CLUSTER-NS-SA` is the ServiceAccount on your run cluster with permissions to deploy the package.

To continuously deploy the latest version of your `Package`, set `versionSelection.constraints: >=0.0.0`. To revert to a previous version, update the `versionSelection.constraints:` field and annotate the PackageInstall:

```
packaging.carvel.dev/downgradable: ""
```

See the Carvel documentation.

1. Push the `PackageInstalls` and `Secrets` to your GitOps repository.

Create an app

1. You must give the build cluster access to the run clusters. On the build cluster create a `Secret` containing the run cluster's kubeconfig for each run cluster:

```
kubectl create secret generic RUN-CLUSTER-kubeconfig \
   -n BUILD-CLUSTER-NS \
   --from-file=value.yaml=PATH-TO-RUN-CLUSTER-KUBECONFIG
```

Where:

- `RUN-CLUSTER` is the name of the run cluster you want to use with your app.
- `BUILD-CLUSTER-NS` is the namespace of the build cluster you want to use.
- `PATH-TO-RUN-CLUSTER-KUBECONFIG` is the location of your run cluster kubeconfig.
- Each Carvel `App` custom resource (CR) must specify either a service account, by using `spec.serviceAccountName`, in the same namespace where the App CR is located on the Build cluster. Or specify a `Secret` with kubeconfig contents for a target destination run cluster, by using `spec.cluster.kubeconfigSecretRef.name`, to explicitly provide the privileges required for managing app resources. The example in this section uses a target run cluster.
- The Carvel App custom resource represents a collection of Kubernetes resources that kapp-controller can fetch and deploy to a cluster. The `App` points at the Git repository branch where kapp-controller resources, such as `PackageRepository` and `Packages`, are defined. By default, an `App` custom resource syncs the cluster with its fetch source every 30 seconds to prevent the cluster state from drifting from its source of truth. Create the following `App` on your Build cluster:

```
---
apiVersion: kappctrl.k14s.io/v1alpha1
kind: App
metadata:
 name: hello-app-app
 namespace: BUILD-CLUSTER-NS
spec:
 # specifies that app should be deployed to destination cluster;
 # by default, cluster is same as where this resource resides
 cluster:
   # specifies namespace in destination cluster
   namespace: ns2
   # specifies secret containing kubeconfig
   kubeconfigSecretRef:
     # specifies secret name within app's namespace
     name: cluster1
     # specifies key that contains kubeconfig
     key: value
 fetch:
 - git:
     url: # GitOps repo URL ex: https://github.com/mycompany/my-gitops
     ref: # GitOps repo branchex: origin/main
     subPath: PATH-FOR-PACKAGES # ex: hello-app.dev.tap/packages/
 - git:
     url: # GitOps repo URL ex: https://github.com/mycompany/my-gitops
     ref: # GitOps repo branch ex: origin/main
     subPath: PATH-FOR-PACKAGE-INSTALLS # ex: hello-app.dev.tap/runcluster1
 template:
 - ytt: {}

  deploy:
 - kapp:
     intoNs: DESIRED-NAMESPACE
     rawOptions: ["--dangerous-allow-empty-list-of-resources=true"]
```

  Where:

  - `DESIRED-NAMESPACE` is the namespace you want to use with your app.
  - `PATH-FOR-PACKAGE-INSTALLS` is the package install path.
  - `PATH-FOR-PACKAGES` is the package path.
  - `BUILD-CLUSTER-NS` is the build cluster namespace.

> **Note**
>
> The fetch section includes entries for all the locations in the GitOps repository to deploy, and append with other run clusters if needed.

Verify applications

To verify your installation:

1. Target a run cluster. Confirm that all Packages from the GitOps repository are deployed:

   ```
   kubectl get packages -A
   ```

2. Target a run cluster. Confirm that all PackageInstalls are reconciled:

   ```
   kubectl get packageinstalls -A
   ```

You can access your application on each run cluster.

Use Gitops delivery with Flux CD (beta)

This topic explains how you can deliver Carvel `Packages`, created by the Carvel Package Supply Chains, from a GitOps repository to one or more run clusters using Flux CD and Supply Chain Choreographer.

Prerequisites

To use Gitops Delivery with Flux CD, you must complete the following prerequisites:

- You must create a `Workload` that uses a Carvel Package Supply Chain. You must have at least one Carvel `Package` generated by this `Workload` stored in your GitOps repository. See Carvel Package Supply Chains (beta).
- You must have at least one run cluster. Run clusters are your deployment environments. They can either be Tanzu Application Platform clusters, or regular Kubernetes clusters, but they must have kapp-controller and Contour installed. See the Carvel documentation and the Contour documentation.
- To use a build cluster to control the deployment on all the run clusters, you must create a build cluster that has network access to your run clusters. You must also install Flux CD Kustomize Controller. See Install Kustomize Controller Prerequisite. If you intend to deploy directly on the run cluster without a build cluster, a build cluster is only necessary for building the package.

Install Kustomize Controller Prerequisite

As mentioned earlier, to use a build cluster to control the deployment on run clusters, you must install Flux CD kustomize-controller. You can do the installation using the flux cli. Use release v2.1.2.

> **Caution**
>
> Newer releases of Kustomize Controller are not compatible with Tanzu Application Platform v1.7.x and later.

To install Flux CD:

```
# https://fluxcd.io/flux/get-started/
brew install fluxcd/tap/flux@0.41  (for example, when on macOS)
flux install -n flux-system --components kustomize-controller --version v2.1.2
```

Set up run cluster namespaces

Each run cluster must have a namespace and `ServiceAccount` with the correct permissions to deploy the Carvel `Packages`, `PackageInstalls` and Kubernetes `Secrets`. Create a namespace and `ServiceAccount` with the following permissions:

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: <run-cluster-ns>
  name: app-package-and-pkgi-install-role
rules:
  - apiGroups: ["data.packaging.carvel.dev"]
    resources: ["packages"]
    verbs: ["get", "list", "create", "update", "delete", "patch"]
  - apiGroups: ["packaging.carvel.dev"]
    resources: ["packageinstalls"]
    verbs: ["get", "list", "create", "update", "delete", "patch"]
  - apiGroups: [""]
    resources: ["secrets"]
    verbs: ["get", "list", "create", "update", "delete", "patch"]
```

If your run cluster is a Tanzu Application Platform cluster, see Set up developer namespaces to use your installed packages.

If your run cluster is not a Tanzu Application Platform cluster, the `ServiceAccount` must also have the following permissions:

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: <run-cluster-ns>
  name: app-cr-role
rules:
- apiGroups: ["apps"]
  resources: ["deployments"]
  verbs: ["get", "list", "create", "update", "delete"]
- apiGroups: [""]
  resources: ["configmaps", "services"]
  verbs: ["get", "list", "create", "update", "delete"]
- apiGroups: ["networking.k8s.io"]
  resources: ["ingresses"]
  verbs: ["get", "list", "create", "update", "delete"]
```

Create Carvel PackageInstalls and secrets

For each Carvel `Package` and each run cluster, you must create a Carvel `PackageInstall` and a `Secret`. The Carvel `PackageInstall` and the `Secret` are stored in your GitOps repository and deployed to run clusters by Flux CD.

The following example shows GitOps repository structure after completing this section:

```
app.default.tap/
  packages/
    20230321004057.0.0.yaml  # Package
  staging/
    packageinstall.yaml      # PackageInstall
    params.yaml              # Secret
  prod/
    packageinstall.yaml      # PackageInstall
    params.yaml              # Secret
```

1. For each run cluster, create a `Secret` that has the values for each `Package` parameter. You can see the configurable properties of the `Package` by inspecting the `Package` CR's valuesSchema, or in the Carvel Package Supply Chains documentation. Store the `Secret` in your GitOps repository at `<package_name>/<run_cluster>/params.yaml`.

   ```
   ---
   apiVersion: v1
   kind: Secret
   metadata:
    name: app-values
   stringData:
    values.yaml: |
      ---
      workload_name: app
      replicas: 2
      hostname: app.mycompany.com
   ```

   Note

   You must set a value for the `workload_name` parameter. You can skip setting other fields to use the default parameter values.

2. For each run cluster, create a `PackageInstall`. Reference the `Secret` you created earlier. Store the `PackageInstall` in your GitOps repository at `<package_name>/<run_cluster>/packageinstall.yaml`.

   ```
   ---
   apiVersion: packaging.carvel.dev/v1alpha1
   kind: PackageInstall
   metadata:
    name: app
   spec:
    serviceAccountName: <run-cluster-ns-sa> # ServiceAccount on run cluster with p
   ermissions to deploy Package, see "Set up run Cluster Namespaces"
    packageRef:
      refName: app.default.tap # name of the Package
      versionSelection:
        constraints: 20230321004057.0.0 # version of the Package
    values:
    - secretRef:
        name: app-values # Secret created in previous step
   ```

To continuously deploy the latest version of your `Package`, set `versionSelection.constraints: >=0.0.0`. To revert to a previous version, update the `versionSelection.constraints:` field and annotate the PackageInstall:

```
packaging.carvel.dev/downgradable: ""
```

See the Carvel documentation.

1. Push the newly created `PackageInstalls` and `Secrets` to your GitOps repository.

Create Flux CD GitRepository and Flux CD Kustomizations on the build cluster

Configure Flux CD on the Build cluster to deploy your `Packages`, `PackageInstalls`, and `Secrets` to each of your run clusters.

1. Give your Build cluster access to your run clusters. On the Build cluster, for each run cluster, create a `Secret` containing the run cluster's kubeconfig. Create the Kubernetes `Secret` in the same namespace as the Kustomization resource:

```
kubectl create secret generic <run-cluster>-kubeconfig \
   -n <build-cluster-ns> \
   --from-file=value.yaml=<path-to-run-cluster-kubeconfig>
```

Note

The kubeconfig must be self-contained and not rely on binaries, or environment and credential files from the kustomize-controller pod. Kubeconfigs that specify a command to provide client credentials such as `gke-gcloud-auth-plugin` won't work without a custom per-provider installation of kustomize-controller.

2. Configure your Build cluster to clone the GitOps repository. On the Build cluster, create the following Flux CD `GitRepository`:

```
---
apiVersion: source.toolkit.fluxcd.io/v1
kind: GitRepository
metadata:
 name: <package-name>-gitops-repo
 namespace: <build-cluster-ns>
spec:
 url: # GitOps repo URL
 ignore: |
   !.git
 interval: 30s
 ref:
   branch: # GitOps repo branch
 timeout: 60s

 # only required if GitOps repo is private (recommended). The secret below shou
ld be present in the same namespace as the GitRepository.
 secretRef:
   name: <package-name>-gitops-auth

# only required if GitOps repo is private (recommended)
---
apiVersion: v1
kind: Secret
metadata:
 name: <package-name>-gitops-auth
 namespace: <build-cluster-ns>
type: Opaque
data:
 username: # base64 encoded GitHub (or other git remote) username
 password: # base64 encoded GitHub (or other git remote) personal access token
```

3. Configure your Build cluster to deploy your `Package` to the run clusters. For each run cluster, on the Build cluster, create the following Flux CD `Kustomization`:

```
---
apiVersion: kustomize.toolkit.fluxcd.io/v1beta2
kind: Kustomization
metadata:
 name: <package-name>-<run-cluster>-packages
 namespace: <build-cluster-ns>
spec:
 sourceRef:
   kind: GitRepository
   name: <package-name>-gitops-repo
   namespace: <build-cluster-ns>
 path: "./<package-name>/packages"
 interval: 5m
 timeout: 5m
 prune: true
 wait: true

 # where to deploy
 kubeConfig:
   secretRef:
     name: <run-cluster>-kubeconfig
 targetNamespace: <run-cluster-ns>
 serviceAccountName: <run-cluster-ns-sa>
```

Note

The Kustomization resource does not accept a `metadata.name` field longer than 63 characters.

4. Configure your Build cluster to deploy your `PackageInstalls` and `Secrets` to the run clusters. For each run cluster, on the Build cluster, create the following Flux CD `Kustomization`:

```
---
apiVersion: kustomize.toolkit.fluxcd.io/v1beta2
kind: Kustomization
metadata:
 # for the second run cluster, for example hello-app-prod2-packages
 name: <package-name>-<run-cluster>-packageinstalls
 namespace: <build-cluster-ns>
spec:
 sourceRef:
   kind: GitRepository
   name: <package-name>-gitops-repo
   namespace: <build-cluster-ns>
 path: "./<package-name>/<run-cluster>"
 interval: 5m
 timeout: 5m
 prune: true
 wait: true

 # where to deploy
 kubeConfig:
   secretRef:
     name: <run-cluster>-kubeconfig
 targetNamespace: <run-cluster-ns>
 serviceAccountName: <run-cluster-ns-sa>
```

Note

The Kustomization resource does not accept a `metadata.name` field longer than 63 characters.

Verify installation

To verify your installation:

1. On your Build cluster, confirm that your Flux CD GitRepository and Kustomizations are reconciling:

   ```
   kubectl get gitrepositories,kustomizations -A
   ```

2. Target a run cluster. Confirm that all Packages from the GitOps repository are deployed:

   ```
   kubectl get packages -A
   ```

3. Target a run cluster. Confirm that all PackageInstalls are reconciled:

   ```
   kubectl get packageinstalls -A
   ```

Now you can access your application on each run cluster.

Use Gitops delivery with Argo CD (beta)

This topic explains how you can deliver Carvel `Packages`, created by the Carvel Package Supply Chains, from a GitOps repository to one or more run clusters using Argo CD for Supply Chain Choreographer.

Prerequisites

To use Gitops Delivery with Argo CD, you must complete the following prerequisites:

- Create a `Workload` that uses a Carvel Package Supply Chain. You must have at least one Carvel `Package` generated by this `Workload` stored in your GitOps repository. See Carvel Package Supply Chains (beta).
- Have at least one run cluster. Run clusters serve as your deployment environments. They can either be Tanzu Application Platform clusters, or Kubernetes clusters, but they must have kapp-controller and Contour installed. See the Carvel documentation and the Contour documentation.
- Create a build cluster that has network access to your run clusters to use a build cluster to control the deployment on all the run clusters. You must also install Argo CD. If you intend to deploy directly on the run cluster without a build cluster, a build cluster is only necessary for building the package.

Set up run cluster namespaces

Each run cluster must have a namespace and `ServiceAccount` with the correct permissions to deploy the Carvel `Packages`.

If your run cluster is a Tanzu Application Platform cluster, see Set up developer namespaces to use installed packages.

If your run cluster is not a Tanzu Application Platform cluster, create a namespace and `ServiceAccount` with the following permissions:

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: <run-cluster-ns>
  name: app-cr-role
rules:
- apiGroups: ["apps"]
  resources: ["deployments"]
  verbs: ["get", "list", "create", "update", "delete"]
- apiGroups: [""]
  resources: ["configmaps", "services"]
  verbs: ["get", "list", "create", "update", "delete"]
- apiGroups: ["networking.k8s.io"]
  resources: ["ingresses"]
  verbs: ["get", "list", "create", "update", "delete"]
```

Create Carvel PackageInstalls and secrets

For each Carvel `Package` and run cluster, you must create a Carvel `PackageInstall` and a `Secret`. The Carvel `PackageInstall` and the `Secret` are stored in your GitOps repository and deployed to run clusters by Flux CD.

The following example shows a GitOps repository structure after completing this section:

```
app.default.tap/
  packages/
    20230321004057.0.0.yaml  # Package
  staging/
    packageinstall.yaml      # PackageInstall
    params.yaml              # Secret
  prod/
    packageinstall.yaml      # PackageInstall
    params.yaml              # Secret
```

For each run cluster:

1. Create a `Secret` that has the values for each `Package` parameter. You can view the configurable properties of the `Package` by inspecting the `Package` CR's valuesSchema, or in the Carvel Package Supply Chains documentation. Store the `Secret` in your GitOps repository at `<package_name>/<run_cluster>/params.yaml`.

   Note

   You can skip this step to use the default parameter values.

   ```
   ---
   apiVersion: v1
   kind: Secret
   metadata:
    name: app-values
   stringData:
    values.yaml: |
      ---
      workload_name: app
      replicas: 2
      hostname: app.mycompany.com
   ```

2. Create a `PackageInstall`. Reference the `Secret` you created earlier. Store the `PackageInstall` in your GitOps repository at `<package_name>/<run_cluster>/packageinstall.yaml`.

Note

If you skipped creation of the `Secret`, omit the `values` key.

```
---
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
 name: app
spec:
 serviceAccountName: <run-cluster-ns-sa> # ServiceAccount on run cluster with p
ermissions to deploy Package, see "Set up run Cluster Namespaces"
 packageRef:
   refName: app.default.tap # name of the Package
   versionSelection:
     constraints: 20230321004057.0.0 # version of the Package
 values:
 - secretRef:
     name: app-values # Secret created in previous step
```

Note

To continuously deploy the latest version of your `Package`, you can set `versionSelection.constraints: >=0.0.0`

3. Push the `PackageInstalls` and `Secrets` to your GitOps repository.

Create an Argo CD application on the Build cluster

Configure Argo CD on the Build cluster to deploy your `Packages`, `PackageInstalls`, and `Secrets` to each run cluster:

1. Register a cluster's credentials to Argo CD. This is only necessary when deploying to an external cluster.
    - First list all clusters contexts in your current kubeconfig:

```
kubectl config get-contexts -o name
```

    - Choose a context name from the list and supply it to the argocd cluster. This command installs a ServiceAccount, argocd-manager, into the kube-system namespace of that kubectl context, binding the service account to an admin-level ClusterRole. Argo CD uses this service account token to perform its management tasks, such as deployment and monitoring.

    For example, for `run-cluster1` context, run:

```
argocd cluster add run-cluster-1
```

Note

> You can modify the rules of the argocd-manager-role role so that it only has create, update, patch, delete privileges to a limited set of namespaces, groups, kinds. However get, list, and watch privileges are required at the cluster-scope.

2. Create an application from a Git repository.
    - Set the current namespace to argocd:

```
kubectl config set-context --current --namespace=argocd
```

    - Create a hello-world-app:

```
argocd app create hello-world-app --repo https://github.com/mycompany/gitops-re
po
```

3. Deploy the application.
    - After you create the application, you can view its status:

```
argocd app get hello-world-app
```

    The output is similar to the following:

```
    Name:               hello-world-app
    Server:             https://kubernetes.default.svc
    Namespace:          default
    URL:                https://10.97.164.88/applications/hello-world-app
    Repo:               https://github.com/mycompany/gitops-repo.git
    Target:
    Path:               hello-world-app
    Sync Policy:        <none>
    Sync Status:        OutOfSync from  (1ff8a67)
    Health Status:      Missing

    GROUP  KIND        NAMESPACE  NAME                 STATUS     HEALTH
    apps   Deployment  default    hello-world-app-dep  OutOfSync  Missing
           Service     default    hello-world-app-svc  OutOfSync  Missing
    The application status is initially in OutOfSync state since the applicatio
n has yet to be deployed, and no Kubernetes resources have been created. To syn
c (deploy) the application, run:
```

    This command retrieves the manifests from the repository and performs a kubectl apply. The hello-world-app app is running and you can now view its resource components, logs, events, and health status.

```
argocd app sync hello-world-app
```

Verify installation

To verify your installation:

1. On your Build cluster, confirm that your Flux CD GitRepository and Kustomizations are reconciling:

```
kubectl get gitrepositories,kustomizations -A
```

2. Target a run cluster. Confirm that all Packages from the GitOps repository are deployed:

```
kubectl get packages -A
```

3. Target a run cluster. Confirm that all PackageInstalls are reconciled:

```
kubectl get packageinstalls -A
```

Now you can access your application on each run cluster.

Use blue-green deployment with Contour and PackageInstall for Supply Chain Choreographer (beta)

Blue-green deployment is an application delivery model that lets you gradually transfer user traffic from one version of your app to a later version while both are running in production. This topic outlines how to use blue-green deployment with Packages and PackageInstalls.

Prerequisites

To use blue-green deployment, you must complete the following prerequisites:

- Complete the prerequisites in Configure and deploy to multiple environments with custom parameters.
- Configure Carvel for your supply chain. See Carvel Package Supply Chains (beta).
- Configure Flux CD for your supply chain. See Deploy Package and PackageInstall using Flux CD Kustomization.

Add HTTPProxy to the blue deployment

The following example deploys a sample application, `hello-app`, to production using a Carvel Package and PackageInstall.

1. Create a Contour HTTPProxy resource to route traffic to the `hello-app` service from the URL `www.hello-app.mycompany.com`.

```
apiVersion: projectcontour.io/v1
kind: HTTPProxy
metadata:
  name: www
  namespace: prod
spec:
  virtualhost:
    fqdn: www.hello-app.mycompany.com
  routes:
    - conditions:
      - prefix: /
        services:
        - name: hello-app
          port: 8080
```

Note

The services names used in HTTPProxy has to match the names of existing services. In this case, the name `hello-app` matches the service installed by the PackageInstall.

2. Apply the HTTPProxy to your cluster:

```
kubectl apply -f httpproxy.yaml
```

3. Verify that the HTTPProxy is present and the route serves traffic to your app.

   ```
   kubectl get HTTPProxy --namespace=prod
   ```

   This displays a list of all the HTTPproxies in the current namespace with their current names.

   ```
   kubectl get HTTPProxy --namespace=prod
   NAMESPACE        NAME                      FQDN
   TLS SECRET            STATUS    STATUS DESCRIPTION
   prod             www                       www.hello-app.mycompany.com
   hello-app-cert        valid     Valid HTTPProxy
   ```

Create the green deployment

After a new version of the package is added to the GitOps repository, create a new PackageInstall for v1.0.1 to create the green deployment.

1. Create a `green-secret.yaml` file with a secret that contains the following ytt overlay.

   ```
   ---
   apiVersion: v1
   kind: Secret
   metadata:
     name: green-overlay-secret
     namespace: prod
   stringData:
     custom-package-overlay.yaml: |
       #@ load("@ytt:overlay", "overlay")
       #@ kd = overlay.subset({"apiVersion":"apps/v1", "kind": "Deployment"})
       #@ ks = overlay.subset({"apiVersion":"v1", "kind": "Service"})
       #@ ki = overlay.subset({"apiVersion":"networking.k8s.io/v1", "kind": "Ingre
   ss"})
       #@ na = overlay.subset({"metadata":{"name":"hello-app"}})

       #@overlay/match by=overlay.and_op(kd, na)
       ---
       metadata:
         #@overlay/replace
         name: hello-app-green

       #@overlay/match by=overlay.and_op(ks, na)
       ---
       metadata:
         #@overlay/replace
         name: hello-app-green

       #@overlay/match by=overlay.and_op(ki, na)
       ---
       metadata:
         #@overlay/replace
         name: hello-app-green
   ```

   This secret changes the names of the service and deployment in the Carvel Package to allow you to install another version of the app in the same namespace.

2. Apply the secret to your cluster by running:

   ```
   kubectl apply -f green-secret.yaml
   ```

3. Create a parameter secret for the new PackageInstall:

```
---
apiVersion: v1
kind: Secret
metadata:
  name: green-dev-values
  namespace: prod
stringData:
  values.yaml: |
    ---
    replicas: 2
    hostname: hello-app-green.mycompany.com
```

4. Apply the parameter secret to your cluster by running:

```
kubectl apply -f green-dev-values.yaml
```

5. Create a PackageInstall to include the `ext.packaging.carvel.dev/ytt-paths-from-secret-name.x` annotation to reference your new overlay secret.

```
---
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: green.hello-app.dev.tap
  namespace: prod
  annotations:
    ext.packaging.carvel.dev/ytt-paths-from-secret-name.0: green-overlay-secret
spec:
  serviceAccountName: default
  packageRef:
    refName: hello-app.dev.tap
    versionSelection:
      constraints: "1.0.1"
  values:
  - secretRef:
      name: green-dev-values
```

Divide traffic between the blue and green deployments

Use the following procedure to divide traffic between your blue and green deployments.

1. Update the HTTPproxy created with the blue deployment to route traffic to both the blue and green deployments. The names of the services must match the names of the already created services.

```
---
apiVersion: projectcontour.io/v1
kind: HTTPProxy
metadata:
  name: www
  namespace: prod
spec:
  virtualhost:
    fqdn: www.hello-app.mycompany.com
  routes:
  - conditions:
    - prefix: /
      services:
      - name: hello-app-green
        port: 8080
```

```
        weight: 20
      - name: hello-app
        port: 8080
        weight: 80
```

2. Update the weights of traffic for each service by editing the HTTPProxy.
3. Access the service several times and confirm both versions are serving traffic in the same percentage.

   ```
   curl -k https://www.hello-app.mycompany.com
   ```

After the new green app is ready to handle the complete load and the `-green` version is not required, use the following steps to remove the old version and rename the new version:

1. Ensure that all the traffic is using the correct version of the app. For example:

   ```
   apiVersion: projectcontour.io/v1
   kind: HTTPProxy
   metadata:
     name: www
     namespace: prod
   spec:
     virtualhost:
       fqdn: www.hello-app.mycompany.com
     routes:
       - conditions:
         - prefix: /
           services:
           - name: hello-app-green
             port: 8080
             weight: 100 # all traffic routed to the green app
   ```

2. Identify the name of the deployment and service that are part of the PackageInstall you no longer need:

   ```
   kubectl get PackageInstall --namespace=prod
   ```

   This displays a list of all the deployments and services in the current Kubernetes namespace, with their current names. For example:

   ```
   NAME                      PACKAGE NAME       PACKAGE VERSION      DESCRIPTION
   green.hello-app.dev.tap   hello-app.dev.tap   1.0.1            Reconcile succee
   ded
   hello-app.dev.tap         hello-app.dev.tap   1.0.0            Reconcile succee
   ded
   ```

3. Delete the PackageInstall:

   ```
   kubectl delete PackageInstall hello-app.dev.tap --namespace=prod
   ```

4. Rename the service and deployments without the green prefix. For example, update the overlay secret:

   ```
   ---
   apiVersion: v1
   kind: Secret
   metadata:
     name: overlay-secret
     namespace: prod
   stringData:
   ```

```
  custom-package-overlay.yaml: |
    #@ load("@ytt:overlay", "overlay")
    #@ load("@ytt:data", "data")

    #@ kd = overlay.subset({"apiVersion":"apps/v1", "kind": "Deployment"})
    #@ ks = overlay.subset({"apiVersion":"v1", "kind": "Service"})
    #@ ki = overlay.subset({"apiVersion":"networking.k8s.io/v1", "kind": "Ingre
ss"})
    #@ na = overlay.subset({"metadata":{"name":"hello-app-green"}})

    #@overlay/match by=overlay.and_op(kd, na)
    ---
    metadata:
      #@overlay/replace
      name: hello-app
    #@overlay/match by=overlay.and_op(ks, na)
    ---
    metadata:
      #@overlay/replace
      name: hello-app
    #@overlay/match by=overlay.and_op(ki, na)
    ---
    metadata:
      #@overlay/replace
      name: hello-app
    ---
    apiVersion: projectcontour.io/v1
    kind: HTTPProxy
    metadata:
      name: www
      namespace: prod
    spec:
      virtualhost:
        fqdn: www.hello-app.mycompany.com
      routes:
        - conditions:
          - prefix: /
            services:
            - name: hello-app # note the name is changed back
              port: 8080
              weight: 100
```

5. Update your PackageInstall to include the `ext.packaging.carvel.dev/ytt-paths-from-secret-name.x` annotation to reference your new overlay secret. For example:

```
---
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: green.hello-app.dev.tap
  namespace: prod
  annotations:
    ext.packaging.carvel.dev/ytt-paths-from-secret-name.0: overlay-secret
spec:
  serviceAccountName: default
  packageRef:
    refName: hello-app.dev.tap
    versionSelection:
      constraints: "1.0.1"
  values:
  - secretRef:
      name: hello-app-values
```

6. After the deployment is complete, you can delete the secrets with the overlays.

Verify application

To verify the name of the deployment and service that are part of the PackageInstall:

1. Verify your application by running:

   ```
   kubectl get PackageInstall --namespace=prod
   ```

   This displays a list of all the deployments and services in the current Kubernetes namespace with their current names. For example:

   ```
   NAME                     PACKAGE NAME       PACKAGE VERSION      DESCRIPTION
   hello-app.dev.tap       hello-app.dev.tap     1.0.1            Reconcile succee
   ded
   ```

   The name is back to the original name and the version is `1.0.1`.

Use canary deployment with Contour and Carvel packages for Supply Chain Choreographer (beta)

This topic tells you how to automate canary releases using Contour ingress controller and Flagger with Packages and PackageInstalls.

Canary deployment is an incremental approach to releasing an application where traffic is divided between the existing deployed version and a new version. It introduces the new version to a smaller group of users before extending it to the entire user base.

Prerequisites

To use canary deployment, you must complete the following prerequisites:

- Complete the prerequisites in Configure and deploy to multiple environments with custom parameters.
- Configure your Supply Chain to output Carvel Packages. See Output Carvel Packages from your Supply Chain.
- Deploy Carvel Packages using the tool of your choice:
  - Deploy Carvel Packages using Carvel App CR
  - Deploy Carvel Packages using Flux CD Kustomization
  - Deploy Carvel Packages using Argo CD
- Install Flagger on your Kubernetes cluster. See Flagger Install on Kubernetes.

How to use Contour ingress controller and Flagger to create a canary release

Flagger is a tool that facilitates a controlled process of shifting traffic to a canary deployment while monitoring essential performance metrics, such as the success rate of HTTP requests, average request duration, and the health of the application's pods. By analyzing these key indicators, Flagger decides whether to promote the canary deployment to a wider audience or abort it if any issues arise.

For information about creating canary releases, see Contour canary Deployments in the Flagger documentation.

Using Contour ingress controller and Flagger to create a canary release involves the following steps:

1. Install Flagger on your Kubernetes cluster.
2. Add the label `app.kubernetes.io/name` to your `Workload`.

   The target deployment must have a single label selector in the format `app: <DEPLOYMENT-NAME>`. In addition to `app`, Flagger supports `name` and `app.kubernetes.io/name` selectors.

   Note

   You can use `tanzu apps workload apply <WORKLOAD-NAME> --label "app.kubernetes.io/name=<WORKLOAD-NAME>"` to edit an existing workload and add the required label.

3. Create a canary resource that defines a canary release with progressive traffic shifting.
   - To configure the canary resource, you must specify the Kubernetes Deployment that corresponds to your Workload. Ensure that you set `spec.targetRef.name` to match the name of your Tanzu Application Platform Workload, which is the same as its Kubernetes Deployment name.
   - Flagger generates some Kubernetes objects. The primary deployment represents the stable release of your application. It receives all incoming traffic while the target deployment is scaled down to zero. Flagger monitors changes in the target deployment, including secrets and configmaps. Before promoting the new version as the primary release, Flagger conducts a thorough canary analysis to ensure that it is stabile.
   - In the canary resource, you can define the canary analysis. Flagger uses this analysis definition to verify the duration of the canary phase, which runs periodically until it reaches the maximum traffic weight or the specified number of iterations.
   - During each iteration, Flagger executes webhooks, evaluates metrics, and checks for any exceeded failed checks threshold. If the threshold is surpassed, indicating potential issues, Flagger takes immediate action to halt the analysis and roll back the canary. If no issues are found, Flagger rolls out the new version as the primary release.

   An example of a canary resource created for a workload, called `tanzu-java-web-app`, is deployed in the namespace `dev-namespace`. Replace `myapps.tanzu.biz` with your own domain:

```
apiVersion: flagger.app/v1beta1
kind: Canary
metadata:
  name: tanzu-java-web-app
  namespace: dev-namespace
spec:
  # deployment reference
  targetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: tanzu-java-web-app
  service:
    # name of the Kubernetes service generated by Flagger. Defaults to spec.tar
getRef.name
    name: tanzu-java-web-app-flagger-service
    # service port
    port: 80
```

```
    # container port
    targetPort: 8080
    # Contour request timeout
    timeout: 15s
    # Contour retry policy
    retries:
      attempts: 3
      perTryTimeout: 5s
      # supported values for retryOn - https://projectcontour.io/docs/main/conf
ig/api/#projectcontour.io/v1.RetryOn
      retryOn: "5xx"
  # define the canary analysis timing and KPIs
  analysis:
    # schedule interval (default 60s)
    interval: 30s
    # max number of failed metric checks before rollback
    threshold: 5
    # max traffic percentage routed to canary
    # percentage (0-100)
    maxWeight: 50
    # canary increment step
    # percentage (0-100)
    stepWeight: 5
    # Contour Prometheus checks
    metrics:
      - name: request-success-rate
        # minimum req success rate (non 5xx responses)
        # percentage (0-100)
        thresholdRange:
          min: 99
        interval: 1m
      - name: request-duration
        # maximum req duration P99 in milliseconds
        thresholdRange:
          max: 500
        interval: 30s
    # testing
    webhooks:
      - name: acceptance-test
        type: pre-rollout
        url: http://flagger-loadtester.test/
        timeout: 30s
        metadata:
          type: bash
          cmd: "curl -s http://tanzu-java-web-app-flagger-service-canary.dev-na
mespace | grep Greetings"
      - name: load-test
        url: http://flagger-loadtester.test/
        type: rollout
        timeout: 5s
        metadata:
          cmd: "hey -z 1m -q 10 -c 2 -host tanzu-java-web-app.myapps.tanzu.biz
http://envoy.tanzu-system-ingress"
```

- Save the resource you created as `canary.yaml` and apply it to your cluster:

```
export WORKLOAD_NAMESPACE=dev-namespace
kubectl apply -n $WORKLOAD_NAMESPACE -f canary.yaml
```

- In the earlier example, you use the load testing service `flagger-loadtester` to generate traffic during the canary analysis.
- To install the load testing service:

```
kubectl create ns test
kubectl apply -k https://github.com/fluxcd/flagger//kustomize/tester?ref=main
```

4. Add an HTTPProxy to include the proxy generated by Flagger.

   The following example HTTPProxy resource references `tanzu-java-web-app-flagger-service` and is deployed in the namespace `dev-namespace`:

   ```
   apiVersion: projectcontour.io/v1
   kind: HTTPProxy
   metadata:
     name: tanzu-java-web-app-ingress
     namespace: dev-namespace
   spec:
     virtualhost:
       fqdn: tanzu-java-web-app.myapps.tanzu.biz
     includes:
     - name: tanzu-java-web-app-flagger-service
       namespace: dev-namespace
       conditions:
         - prefix: /
   ```

   Save the resource you created as `httpproxy.yaml` and then apply it to your cluster:

   ```
   export WORKLOAD_NAMESPACE=dev-namespace
   kubectl apply -n $WORKLOAD_NAMESPACE -f httpproxy.yaml
   ```

   Confirm the HTTPProxy created by Contour has `Valid` status:

   ```
   kubectl get httpproxies -n $WORKLOAD_NAMESPACE
   ```

5. Make changes to GitOps repository and observe the progressive delivery in action

   As changes are made to your GitOps repository, the GitOps tools in place in your environment, such as Flux CD and Argo CD, deploy the new `Package`s onto your clusters. Flagger detects any changes to the target deployment, including secrets and configmaps, and starts a new rollout. The new version is either promoted or rolled back.

   You can monitor the traffic shifting with:

   ```
   watch kubectl get canary -n $WORKLOAD_NAMESPACE
   kubectl describe canary
   CANARY-RESOURCE-NAME -n $WORKLOAD_NAMESPACE
   ```

   Where `CANARY-RESOURCE-NAME` is the name of the canary resource you want to use.

References and further reading

The following topics give you more information about canary:

- Deployment Strategies
- Contour canary Deployments
- Flagger - How it works

Use blue-green deployment with Flagger for Supply Chain Choreographer (beta)

This topic tells you how to use blue-green deployment with Packages and PackageInstalls using Flagger for Supply Chain Choreographer (beta).

Overview

Blue-green deployment is an application delivery model that lets you gradually transfer user traffic from one version of your app to a later version while both are running in production.

Prerequisites

To use blue-green deployment, you must complete the following prerequisites:

- Complete the prerequisites in Configure and deploy to multiple environments with custom parameters.
- Configure Carvel for your supply chain. See Carvel Package Supply Chains (beta).
- Configure a GitOps tool to deploy the package. See Deploy Package and PackageInstall using Flux CD Kustomization or ArgoCD.

Deployment name

To get the name of the deployment you want to use:

1. Identify the name of the deployment and service that are part of the PackageInstall:

   ```
   kubectl get deployment --namespace=prod
   ```

   This displays a list of all the deployments and services in the current Kubernetes namespace, with their current names. For example:

   ```
   NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
   hello-app-00001-deployment    1/1     1            1           103
   ```

Flagger Canary resource

Create a canary custom resource using the Flagger instructions using the deployment name from the earlier step.

Flagger creates three ClusterIP services: `hello-app.dev.tap-primary`, `hello-app.dev.tap-canary`, `hello-app.dev.tap`. It also creates a shadow deployment named `app-primary` that represents the blue version. When a new version is detected, Flagger scales up the green version and runs the conformance tests, the tests target the `hello-app.dev.tap-canary` ClusterIP service to reach the green version. If the conformance tests are passing, Flagger starts the load tests and validates them with custom Prometheus queries. If the load test analysis is successful, Flagger promotes the new version to `hello-app.dev.tap-primary` and scales down the green version. Flagger extends a canary analysis through custom metrics and webhooks for running load tests, acceptance tests, or any other custom validation.

Flagger includes a load generation tool to ensure that the canary version of an application has enough traffic to generate metrics for analysis. Without an HPA (Horizontal Pod Autoscaler), Flagger does not dynamically scale up or down the new version based on traffic weight. This implies that the number of pods for the new version remains constant throughout deployment. Consequently, performance issues like high latency, errors, or timeouts can arise if the new version receives more traffic than it can handle. This situation can also impact the metrics and webhooks that Flagger uses to validate the new version, potentially leading to a rollback or promotion delay.

Workloads of type `Web` use Knative, which has its own custom autoscaler and doesn't use HPA by default, however you can configure the Knative to use it.

Use an existing image with Supply Chain Choreographer

This topic describes how you can use an existing image with Supply Chain Choreographer.

For apps that build container images in a predefined way, the supply chains in the Out of the Box packages enable you to specify a prebuilt image. This uses the same stages as any other workload.

Requirements for prebuilt images

Supply chains aim at Knative as the runtime for the container image you provide. Your app must adhere to the following Knative standards:

- **Container port listens on port 8080**

  The Knative service is created with the container port set to `8080` in the pod template spec Therefore, your container image must have a socket listening on `8080`.

  ```
  ports:
    - containerPort: 8080
      name: user-port
      protocol: TCP
  ```

- **Non-privileged user ID**

  By default, the container initiated as part of the pod is run as user 1000.

  ```
  securityContext:
    runAsUser: 1000
  ```

- **Arguments other than the image's default ENTRYPOINT**

  In most cases the container image runs using the `ENTRYPOINT` it was configured with. In the case of Dockerfiles, the combination of `ENTRYPOINT` and `CMD`.

  If you need extra configuration for your image, use `--env` flags with the `tanzu apps workload create` command or modify `spec.env` in your `workload.yaml` file.

- **Credentials for pulling the container image at runtime**

  The image you provide is not relocated to an internal container image registry. Any components associated with the image must have the necessary credentials to pull it. For the service accounts used for the workload and deliverable, you must attach a secret that contains the credentials to pull the container image.

  If the image is hosted in a registry that has certificates signed by a private certificate authority, the components of the supply chains, delivery, and the Kubernetes nodes in the run cluster must trust the certificate.

Configure your workload to use a prebuilt image

To select a prebuilt image, set the `spec.image` field in your `workload.yaml` file with the name of the container image that contains the app to deploy by running:

```
tanzu apps workload create WORKLOAD-NAME \
  --app APP-NAME \
  --type TYPE \
  --image IMAGE
```

Where:

- `WORKLOAD-NAME` is the name you choose for your workload.

- `APP-NAME` is the name of your app.
- `TYPE` is the type of your app.
- `IMAGE` is the container image that contains the app you want to deploy.

For example, if you have an image named `IMAGE`, you can create a workload with the flag mentioned earlier:

```
tanzu apps workload create tanzu-java-web-app \
  --app tanzu-java-web-app \
  --type web \
  --image IMAGE
```

Expected output:

```
Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    app.kubernetes.io/part-of: hello-world
      7 + |    apps.tanzu.vmware.com/workload-type: web
      8 + |  name: tanzu-java-web-app
      9 + |  namespace: default
     10 + |spec:
     11 + |  image: IMAGE
```

When you run `tanzu apps workload create` command with the `--image` field, the source resolution and build phases of the supply chain are skipped.

Examples

The following examples show ways that you can build container images for a Java-based app and complete the supply chains to a running service.

Using a Dockerfile

Using a Dockerfile is the most common way of building container images. You can select a base image, on top of which certain operations must occur, such as compiling code, and mutate the contents of the file system to a final container image that has a build of your app and any required runtime dependencies.

Here you use the `maven` base image for compiling your app code, and then the minimal distroless `java17-debian11` image for providing a JRE that can run your app when it is built.

After building the image, you push it to a container image registry, and then reference it in the workload.

1. Create a Dockerfile that describes how to build your app and make it available as a container image:

   ```
   ARG BUILDER_IMAGE=maven
   ARG RUNTIME_IMAGE=gcr.io/distroless/java17-debian11


   FROM $BUILDER_IMAGE AS build

           ADD . .
           RUN unset MAVEN_CONFIG && ./mvnw clean package -B -DskipTests
   ```

```
FROM $RUNTIME_IMAGE AS runtime

        COPY --from=build /target/demo-0.0.1-SNAPSHOT.jar /demo.jar
        CMD [ "/demo.jar" ]
```

2. Push the container image to a container image registry by running:

```
docker build -t IMAGE .
docker push IMAGE
```

3. Create a workload by running:

```
tanzu apps workload create tanzu-java-web-app \
  --type web \
  --app tanzu-java-web-app \
  --image IMAGE
```

Expected output:

```
Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    app.kubernetes.io/part-of: hello-world
      7 + |    apps.tanzu.vmware.com/workload-type: web
      8 + |  name: tanzu-java-web-app
      9 + |  namespace: default
     10 + |spec:
     11 + |  image: IMAGE
```

4. Run the following workload:

```
tanzu apps workload get tanzu-java-web-app
```

Expected output:

```
# tanzu-java-web-app: Ready
---
lastTransitionTime: "2022-04-06T19:32:46Z"
message: ""
reason: Ready
status: "True"
type: Ready

Workload pods
NAME                                                  STATUS      RESTARTS   A
GE
tanzu-java-web-app-00001-deployment-7d7df5ccf5-k58rt   Running     0          3
2s
tanzu-java-web-app-config-writer-xjmvw-pod             Succeeded   0          8
9s

Workload Knative Services
NAME                READY   URL
tanzu-java-web-app   Ready   http://tanzu-java-web-app.default.example.com
```

Using Spring Boot's `build-image` Maven target

You can use Spring Boot's `build-image` target to build a container image that runs your app. The `build-image` target must use a Dockerfile.

For example, using the same sample repository as mentioned before (https://github.com/vmware-tanzu/application-accelerator-samples/tree/main/tanzu-java-web-app):

1. Build the image by running the following command from the root of the repository:

   ```
   IMAGE=ghcr.io/kontinue/hello-world:tanzu-java-web-app
   ./mvnw spring-boot:build-image -Dspring-boot.build-image.imageName=$IMAGE
   ```

   Expected output:

   ```
   [INFO] Scanning for projects...
   [INFO]
   [INFO] --------------------------< com.example:demo >--------------------------
   [INFO] Building demo 0.0.1-SNAPSHOT
   [INFO] --------------------------------[ jar ]---------------------------------
   [INFO]
   ...
   [INFO]
   [INFO] Successfully built image 'ghcr.io/kontinue/hello-world:tanzu-java-web-ap
   p'
   [INFO]
   [INFO] ------------------------------------------------------------------------
   [INFO] BUILD SUCCESS
   [INFO] ------------------------------------------------------------------------
   [INFO] Total time:  39.257 s
   [INFO] Finished at: 2022-04-06T19:40:16Z
   [INFO] ------------------------------------------------------------------------
   ```

2. Push the image you built to the container image registry by running:

   ```
   IMAGE=ghcr.io/kontinue/hello-world:tanzu-java-web-app
   docker push $IMAGE
   ```

   Expected output:

   ```
   The push refers to repository [ghcr.io/kontinue/hello-world]
   1dc94a70dbaa: Preparing
   ...
   9d6787a516e7: Pushed
   tanzu-java-web-app: digest: sha256:7140722ea396af69fb3d0ad12e9b4419bc3e67d9c5d8
   a2f6a1421decc4828ace size: 4497
   ```

After you push the container image, you see the same results as building the image using a Dockerfile.

For more information about building container images for a Spring Boot app, see Spring Boot with Docker

About Out of the Box Supply Chains

In Tanzu Application Platform, the `ootb-supply-chain-basic`, `ootb-supply-chain-testing`, and `ootb-supply-chain-testing-scanning` packages each receive a new supply chain that provides a prebuilt container image for your app.

```
ootb-supply-chain-basic

    (cluster)  basic-image-to-url   ClusterSupplyChain            (!) new
    ^          source-to-url        ClusterSupplyChain
```

```
ootb-supply-chain-testing

    (cluster)  testing-image-to-url  ClusterSupplyChain           (!) new
    ^          source-test-to-url    ClusterSupplyChain


ootb-supply-chain-testing-scanning

    (cluster)  scanning-image-scan-to-url    ClusterSupplyChain   (!) new
    ^          source-test-scan-to-url       ClusterSupplyChain
```

To leverage the supply chains that expect a prebuilt image, you must set the `spec.image` field in the workload to the name of the container image that contains the app to deploy.

The new supply chains use a Cartographer feature that lets VMware increase the specificity of supply chain selection by using the `matchFields` selector rule.

The selection takes place as follows:

- *ootb-supply-chain-basic*
    - From source: label `apps.tanzu.vmware.com/workload-type: web`
    - Prebuilt image: label `apps.tanzu.vmware.com/workload-type: web` **and** set `spec.image` in the `workload.yaml`
- *ootb-supply-chain-testing*
    - From source: labels `apps.tanzu.vmware.com/workload-type: web` and `apps.tanzu.vmware.com/has-tests: true`
    - Prebuilt image: label `apps.tanzu.vmware.com/workload-type: web` **and** set `spec.image` in the `workload.yaml`
- *ootb-supply-chain-testing-scanning*
    - From source: labels `apps.tanzu.vmware.com/workload-type: web` and `apps.tanzu.vmware.com/has-tests: true`
    - Prebuilt image: label `apps.tanzu.vmware.com/workload-type: web` **and** set `spec.image` in the `workload.yaml`

Workloads that already work with the supply chains before Tanzu Application Platform v1.1 continue to work with the same supply chain. Workloads that bring a prebuilt container image must set `spec.image` in the `workload.yaml`.

Understanding the supply chain for a prebuilt image

An `ImageRepository` object is created to keep track of new images pushed under that name. `ImageRepository` makes the image available to further resources in the supply chain, providing the final digest of the latest image.

Whenever a new image is pushed to the workload's image location, the `ImageRepository` detects the change. The image is then available to further resources by updating its `imagerepository.status.artifact.revision` with an absolute reference to that image.

For example, if you create a workload using an image named `hello-world`, tagged `tanzu-java-web-app` hosted under `ghcr.io` in the `kontinue` repository:

```
tanzu apps workload create tanzu-java-web-app \
  --app tanzu-java-web-app \
  --type web \
  --image ghcr.io/kontinue/hello-world:tanzu-java-web-app
```

After a couple seconds, you see the `ImageRepository` object created to keep track of images named `ghcr.io/kontinue/hello-world:tanzu-java-web-app`:

```
Workload/tanzu-java-web-app
├─ImageRepository/tanzu-java-web-app
├─PodIntent/tanzu-java-web-app
├─ConfigMap/tanzu-java-web-app
└─Runnable/tanzu-java-web-app-config-writer
  └─TaskRun/tanzu-java-web-app-config-writer-p2lzv
    └─Pod/tanzu-java-web-app-config-writer-p2lzv-pod
```

If you inspect the status in `status.resources` in the `workload.yaml`, you see the `image-provider` resource promoting the image it found to further resources:

```
apiVersion: carto.run/v1alpha1
kind: Workload
spec:
  image: ghcr.io/kontinue/hello-world:tanzu-java-web-app
status:
  resources:
    - name: image-provider
      outputs:
        # output being made available to further resources in the supply chain
        # (in this case, the latest image it found under that name).
        #
        - name: image
          lastTransitionTime: "2022-04-01T15:05:01Z"
          preview: ghcr.io/kontinue/hello-world:tanzu-java-web-app@sha256:9fb930a...

      # reference to the object managed by the supply chain for this
      # resource
      #
      stampedRef:
        apiVersion: source.apps.tanzu.vmware.com/v1alpha1
        kind: ImageRepository
        name: tanzu-java-web-app
        namespace: workload

      # reference to the template that defined how this object should look
      # like
      #
      templateRef:
        apiVersion: carto.run/v1alpha1
        kind: ClusterImageTemplate
        name: image-provider-template
```

The image found by the `ImageRepository` object is carried through the supply chain to the final configuration. This is pushed to either a Git repository or image registry so that it is deployed in a run cluster.

Note

The image name matches the image name supplied in the `spec.image` field in the `workload.yaml`, but also includes the digest of the latest image found under the tag. If a new image is pushed to the same tag, you see the `ImageRepository` resolving the name to a different digest corresponding to the new image pushed.

Use Git authentication with Supply Chain Choreographer

This topic describes how you can use Git authentication with Supply Chain Choreographer.

You can either fetch or push source code from or to a repository that requires credentials. You must provide credentials through a Kubernetes secret object referenced by the intended Kubernetes object created for performing the action.

The following sections provide details about how to appropriately set up Kubernetes secrets for carrying those credentials forward to the proper resources.

Important

For HTTP, HTTPS, and SSH, do not use the same server for multiple secrets to avoid a Tekton error.

Pulling Source Code

For the supply chain to pull source code it must reference a secret with Git credentials. This secret must exist in the same namespace as the workload.

```
apiVersion: v1
kind: Secret
metadata:
   name: NAME-OF-THE-SECRET
   namespace: SOME-NAMESPACE
spec: ...
```

You must provide the name of this secret to the supply chain, either as a tap-value or as a workload parameter.

tap-value example:

```
ootb_supply_chain_basic:
   source:
      credentials_secret: NAME-OF-THE-SECRET
```

Workload parameter value:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  namespace: SOME-NAMESPACE
spec:
  params:
    - name: source_credentials_secret
      value: NAME-OF-THE-SECRET
```

Pushing Build Configuration

For the supply chain to push build configuration to a Gitops repository, the supply chain must reference a service account and this service account must in turn reference a secret with Git credentials.

The secret can be different from the secret used for pulling source code, with different credentials to a different repository.

For example, a secret:

```
apiVersion: v1
kind: Secret
metadata:
```

```
   name: NAME-OF-A-SECRET
   namespace: SOME-NAMESPACE
   annotations:
     tekton.dev/git-0: GIT-SERVER        # ! required. example: https://github.com
spec: ...
```

referenced by a service account:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: SOME-SA-NAME
  namespace: SOME-NAMESPACE
secrets:
  - name: registry-credentials
  - name: tap-registry
  - name: NAME-OF-A-SECRET
imagePullSecrets:
  - name: registry-credentials
  - name: tap-registry
```

You must provide the name of this service account to the supply chain, either as a tap-value or as a workload parameter.

tap-value example:

```
ootb_supply_chain_basic:
   service_account: SOME-SA-NAME
```

Workload parameter example:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  namespace: SOME-NAMESPACE
spec:
  params:
    - name: serviceAccount
      value: SOME-SA-NAME
```

Note

If you've used Namespace Provisioner to set up your Developer Namespace where your workload is created, use the `namespace_provisioner.default_parameters.supply_chain_service_account.secrets` property in your `tap-values.yaml`. For example:

```
```yaml
namespace_provisioner:
  default_parameters:
    supply_chain_service_account:
      secrets:
      - GIT-SECRET-NAME
```
```

Namespace Provisioner manages the service account and manual edits to it do not persist.

Pulling Build Configuration

The delivery must pull the build configuration that was pushed by the supply chain. It must reference a secret with Git credentials (similar to how the supply chain pulls source code). This secret must exist in the same namespace as the deliverable. The credentials in this secret must be valid for the repository to which the supply chain pushed configuration.

```
apiVersion: v1
kind: Secret
metadata:
   name: NAME-OF-A-SECRET
   namespace: A-NAMESPACE
spec: ...
```

You must provide the name of this secret, either as a tap-value or as a deliverable parameter.

tap-value example:

```
ootb_delivery_basic:
   source:
      credentials_secret: NAME-OF-A-SECRET
```

Deliverable parameter example:

```
apiVersion: carto.run/v1alpha1
kind: Deliverable
metadata:
  namespace: A-NAMESPACE
spec:
  params:
    - name: source_credentials_secret
      value: NAME-OF-A-SECRET
```

HTTP

For any action upon an HTTP or HTTPS based repository, create a Kubernetes secret object of type `kubernetes.io/basic-auth` as follows:

```
apiVersion: v1
kind: Secret
metadata:
  name: SECRET-NAME
  annotations:
    tekton.dev/git-0: GIT-SERVER        # ! required
type: kubernetes.io/basic-auth          # ! required
stringData:
  username: GIT-USERNAME
  password: GIT-PASSWORD
```

For example, assuming you have a repository called `kontinue/hello-world` on GitHub that requires authentication, and you have an access token with the privileges of reading the contents of the repository, you can create the secret as follows:

```
apiVersion: v1
kind: Secret
metadata:
  name: git-secret
  annotations:
    tekton.dev/git-0: https://github.com
type: kubernetes.io/basic-auth
stringData:
```

```
  username: GITHUB-USERNAME
  password: GITHUB-ACCESS-TOKEN
```

Note

In this example, you use an access token because GitHub deprecates basic authentication with plain user name and password. For more information, see Creating a personal access token on GitHub.

HTTPS with a Custom CA Certificate

In addition to the shared.ca_cert_data field, you must add the certificate to the secret used to access the Git repository. The only platform tested with custom CA certificates is GitLab.

You set up the secret similarly to the section above, but the `caFile` field specifies a certificate authority.

```
apiVersion: v1
kind: Secret
metadata:
 name: SECRET-NAME
 annotations:
   tekton.dev/git-0: GIT-SERVER        # ! required
type: kubernetes.io/basic-auth          # ! required
stringData:
 username: GIT-USERNAME
 password: GIT-PASSWORD
 caFile: |
  -----BEGIN CERTIFICATE-----
  ...
  -----END CERTIFICATE-----
```

SSH

Aside from using HTTP or HTTPS as a transport, the supply chains also allow you to use SSH.

Important

To use the pull request feature, you must use HTTP or HTTPS authentication with an access token.

1. To provide the credentials for any Git operations with SSH, create the Kubernetes secret as follows:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: GIT-SECRET-NAME
     annotations:
       tekton.dev/git-0: GIT-SERVER
   type: kubernetes.io/ssh-auth          # ! required
   stringData:
     ssh-privatekey: SSH-PRIVATE-KEY     # private key with push-permissions
     identity: SSH-PRIVATE-KEY           # private key with pull permissions
     identity.pub: SSH-PUBLIC-KEY        # public of the `identity` private key
     known_hosts: GIT-SERVER-PUBLIC-KEYS # Git server public keys
   ```

2. Generate a new SSH keypair: `identity` and `identity.pub`.

```
ssh-keygen -t ecdsa -b 521 -C "" -f "identity" -N ""
```

3. Go to your Git provider and add the `identity.pub` as a deployment key for the repository of interest or add to an account that has access to it. For example, for GitHub, visit `https://github.com/<repository>/settings/keys/new`.

Note

Keys of type SHA-1/RSA are recently deprecated by GitHub.

4. Gather public keys from the provider, for example, GitHub:

```
ssh-keyscan github.com > ./known_hosts
```

5. Create the Kubernetes secret by using the contents of the files in the first step:

```
apiVersion: v1
kind: Secret
metadata:
  name: GIT-SECRET-NAME
  annotations: {tekton.dev/git-0: GIT-SERVER}
type: kubernetes.io/ssh-auth
stringData:
  ssh-privatekey: SSH-PRIVATE-KEY
  identity: SSH-PRIVATE-KEY
  identity.pub: SSH-PUBLIC-KEY
  known_hosts: GIT-SERVER-PUBLIC-KEYS
```

For example, edit the credentials:

```
apiVersion: v1
kind: Secret
metadata:
  name: git-ssh
  annotations: {tekton.dev/git-0: github.com}
type: kubernetes.io/ssh-auth
stringData:
  ssh-privatekey: |
    -----BEGIN OPENSSH PRIVATE KEY-----
    AAAA
    ....
    ....
    -----END OPENSSH PRIVATE KEY-----
  known_hosts: |
    <known hosts entrys for git provider>
  identity: |
    -----BEGIN OPENSSH PRIVATE KEY-----
    AAAA
    ....
    ....
    -----END OPENSSH PRIVATE KEY-----
  identity.pub: ssh-ed25519 AAAABBBCCCCDDDDeeeeFFFF user@example.com
```

More information about Git

For information about Git, see Git Reference.

Using Azure DevOps as a Git provider with your supply chains

This topic describes how you can Azure DevOps as a Git provider with your Supply Chain Choreographer supply chains.

Overview

There are two uses for Git in a supply chain:

- As a source of code to build and deploy applications
- As a repository of configuration created by the build cluster which is deployed on a run or production cluster

Azure DevOps differs from other Git providers in the following ways:

- Azure DevOps requires Git clients to support multi-ack.
- Azure DevOps repository paths differ from other Git providers.

For information about how Azure DevOps is different from other Git providers, see Gitops write path templates.

The operator requires special configuration to integrate Azure DevOps repositories into a supply chain.

Azure authentication

You can use Azure authentication with Supply Chain Choreographer.

For information about configuring secrets to authenticate with your Azure DevOps Git repository, see Use Git authentication with Supply Chain Choreographer.

Azure http and https authentication requires:

```
username: "_token"
password: AZURE-USER-TOKEN
```

Where `AZURE-USER-TOKEN` is your Azure personal access token. See Azure Devops Personal Access Tokens in the Microsoft documentation.

Using Azure DevOps as a repository for committed code

Developers can use Azure DevOps to commit source code to a repository that the supply chain pulls.

Azure DevOps example

The following example uses the Azure DevOps source repository:

`https://dev.azure.com/my-company/app/_git/app`

You can configure the supply chain by using `tap-values`:

```
ootb_supply_chain_testing_scanning:
  git_implementation: go-git
```

or by using workload parameter:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  ...
spec:
  params:
    - name: gitImplementation
      value: go-git
```

Using Azure DevOps as a GitOps repository

The supply chain commits Kubernetes configuration to a Git repository. This configuration is then applied to another cluster. This is the GitOps promotion pattern.

You must construct a path and configure your Git implementation to read and write to an Azure DevOps repository.

GitOps write path example

The following example uses the Azure DevOps Git repository:

`https://dev.azure.com/vmware-tanzu/tap/_git/tap`

Set the `gitops_server_kind` workload parameters to `azure`.

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  ...
spec:
  params:
    - name: gitops_server_kind
      value: azure
    ...
```

Set other GitOps values in either tap-values or in the workload parameters.

- By using tap-values:

  ```
  ootb_supply_chain_testing_scanning:
    gitops:
      server_address: https://dev.azure.com
      repository_owner: vmware-tanzu/tap
      repository_name: tap
  ```

- By using the workload parameters:

  ```
  apiVersion: carto.run/v1alpha1
  kind: Workload
  metadata:
    ...
  spec:
    params:
      - name: gitops_server_address
        value: https://dev.azure.com
      - name: gitops_repository_owner
        value: vmware-tanzu/tap
      - name: gitops_repository_name
        value: tap
      ...
  ```

Gitops write path templates

Azure DevOps and Git use different URL structures.

For example, the Git clone URL of an Azure DevOps repository is structured as:

`https://dev.azure.com/<org_name>/<project_name>/_git/<repository_name>`

GitHub uses the following address structure:

`https://github.com/<org_name>/<repository_name>`

In Azure DevOps, a project can have multiple repositories, but the project name and repository name are often the same.

The config-writer and config-writer-and-pull-requester templates accept three parameters to build the path of the repository. For Azure DevOps, configure them as follows:

- gitops_server_address: `https://dev.azure.com`
- gitops_repository_owner: `<org_name>/<project_name>`
- gitops_repository_name: `<repository_name>`

Configure the template parameters as follows:

- `gitops.server_address` tap-value during the Out of the Box Supply Chains package installation or `gitops_server_address` configured as a workload parameter.
- `gitops.repository_owner` tap-value during the Out of the Box Supply Chains package installation or `gitops_repository_owner` configured as a workload parameter.
- `gitops.repository_name` tap-value during the Out of the Box Supply Chains package installation or `gitops_repository_name` configured as a workload parameter.

To properly contruct the write path, the template parameter `gitops_server_kind` must be configured as `azure`.

Configure`gitops_server_kind` as a workload parameter.

Note

When you use pull requests with GitOps, you can set the type of server with the tap-value `gitops.pull_request.server_kind`. See GitOps versus RegistryOps.

For information about configuring the GitOps write operations, see GitOps versus RegistryOps.

Using GitLab as a Git provider with your supply chains

This topic describes how to use GitLab as a Git provider with your Supply Chain Choreographer supply chains.

Overview

There are two uses for Git in a supply chain:

- As a source of code to build and deploy applications
- As a repository of configuration created by the build cluster which is deployed on a run or production cluster

Using GitLab as a repository for committed code

This section tells you how developers can use GitLab to commit source code to a repository that the supply chain pulls.

GitLab example

The following example uses the GitLab source repository:

`https://gitlab.example.com/my-org/repository.git`

You can configure the supply chain by using `tap-values.yaml`:

```
ootb_supply_chain_testing_scanning:
  git_implementation: go-git
```

or by using a workload parameter:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  ...
spec:
  params:
    - name: gitImplementation
      value: go-git
```

Using GitLab as a GitOps repository

The supply chain commits Kubernetes configuration to a Git repository. This configuration is then applied to another cluster. This is the GitOps promotion process.

You must construct a path and configure your Git implementation to read and write to an GitLab repository.

GitOps write path example

The following example uses the GitLab Git repository:

`https://gitlab.example.com/my-org/repository.git`

1. Set the `gitops_server_kind` workload parameters to `gitlab`.

   ```
   apiVersion: carto.run/v1alpha1
   kind: Workload
   metadata:
     ...
   spec:
     params:
       - name: gitops_server_kind
         value: gitlab
       ...
   ```

2. Set other GitOps values in either `tap-values.yaml` or in the workload parameters.

   By using `tap-values.yaml`:

   ```
   ootb_supply_chain_testing_scanning:
     gitops:
       server_address: https://gitlab.example.com
       repository_owner: my-org
       repository_name: repository
   ```

   By using the workload parameters:

   ```
   apiVersion: carto.run/v1alpha1
   kind: Workload
   metadata:
   ```

```
    ...
  spec:
    params:
      - name: gitops_server_address
        value: https://gitlab.example.com
      - name: gitops_repository_owner
        value: my-org
      - name: gitops_repository_name
        value: repository
      ...
```

GitLab read example

The following example uses the GitLab repository:

`https://gitlab.example.com/my-org/repository.git`

You can configure the delivery `tap-values.yaml`:

```
ootb_delivery_basic:
  git_implementation: go-git
```

or the deliverable parameter:

```
apiVersion: carto.run/v1alpha1
kind: Deliverable
metadata:
  ...
spec:
  params:
    - name: gitImplementation
      value: go-git
```

GitLab over HTTPS with a custom CA certificate

When using HTTPS with a custom certificate authority, you must configure the Git secret both in `tap-values.yaml` and the Git secret used by the GitRepository.

1. Set the shared.ca_cert_data in `tap-values.yaml`. You must set the Git secret in the `caFile` field.

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: SECRET-NAME
     annotations:
       tekton.dev/git-0: GIT-SERVER        # ! required
   type: kubernetes.io/basic-auth          # ! required
   stringData:
     username: GIT-USERNAME
     password: GIT-PASSWORD
     caFile: |
     -----BEGIN CERTIFICATE-----
     ...
     -----END CERTIFICATE-----
   ```

2. Associate the secret with the `ServiceAccount`.

   ```
   apiVersion: v1
   kind: ServiceAccount
   metadata:
   name: default
   secrets:
   ```

```
- name: registry-credentials
- name: tap-registry
- name: GIT-SECRET-NAME
imagePullSecrets:
- name: registry-credentials
- name: tap-registry
```

For information about authentication, see Git Authentication.

Author your supply chains

The Out of the Box Supply Chain, Delivery Basic, and Templates Supply Chain Choreographer packages give you Kubernetes objects that cover a reference path to production. Because VMware recognizes that you have your own needs, these objects are customizable, including individual templates for each resource, whole supply chains, or delivery objects.

Depending on how you installed Tanzu Application Platform, there are different ways to customize the Out of the Box Supply Chains. The following sections describe the ways supply chains and templates are authored within the context of profile-based Tanzu Application Platform installations.

Providing your own supply chain

To create a new supply chain and make it available for workloads, ensure that the supply chain does not conflict with those installed on the cluster, as those objects are cluster-scoped.

If this is your first time creating a supply chain, follow the tutorials from the Cartographer documentation.

Any supply chain installed in a Tanzu Application Platform cluster might encounter two possible cases of collisions:

- **object name**: Supply chains are cluster scoped, such as any Cartographer resource prefixed with `Cluster`. So the name of the custom supply chain must be different from the ones provided by the Out of the Box packages.

  Either create a supply chain whose name is different, or remove the installation of the corresponding `ootb-supply-chain-*` from the Tanzu Application Platform.

- **workload selection**: A workload is reconciled against a particular supply chain based on a set of selection rules as defined by the supply chains. If the rules for the supply chain to match a workload are ambiguous, the workload does not make any progress.

  Either create a supply chain whose selection rules are different from the ones the Out of the Box Supply Chain packages use, or remove the installation of the corresponding `ootb-supply-chain-*` from Tanzu Application Platform.

  See Selectors.

For Tanzu Application Platform 1.2, the following selection rules are in place for the supply chains of the corresponding packages:

- *ootb-supply-chain-basic*
  - ClusterSupplyChain/**basic-image-to-url**
    - label `apps.tanzu.vmware.com/workload-type: web`
    - `workload.spec.image` text box set
  - ClusterSupplyChain/**source-to-url**
    - label `apps.tanzu.vmware.com/workload-type: web`
  - ClusterSupplyChain/**basic-image-to-url-package (experimental)**

        - label `apps.tanzu.vmware.com/workload-type: server`
        - label `apps.tanzu.vmware.com/carvel-package-workflow: true`
    - ClusterSupplyChain/**source-to-url-package (experimental)**
        - label `apps.tanzu.vmware.com/workload-type: server`
        - label `apps.tanzu.vmware.com/carvel-package-workflow: true`
- *ootb-supply-chain-testing*
    - ClusterSupplyChain/**testing-image-to-url**
        - label `apps.tanzu.vmware.com/workload-type: web`
        - `workload.spec.image` text box set
    - ClusterSupplyChain/**source-test-to-url**
        - label `apps.tanzu.vmware.com/workload-type: web`
        - label `apps.tanzu.vmware.com/has-test: true`
- *ootb-supply-chain-testing-scanning*
    - ClusterSupplyChain/**scanning-image-scan-to-url**
        - label `apps.tanzu.vmware.com/workload-type: web`
        - `workload.spec.image` text box set
    - ClusterSupplyChain/**source-test-scan-to-url**
        - label `apps.tanzu.vmware.com/workload-type: web`
        - label `apps.tanzu.vmware.com/has-test: true`

For details about how to edit an existing supply chain, see Modifying an Out of the Box Supply Chain section.

You can exclude a supply chain package from the installation to prevent the conflicts mentioned earlier, by using the `excluded_packages` property in `tap-values.yaml`. For example:

```
# add to exclued_packages `ootb-*` packages you DON'T want to install
# excluded_packages:
  - ootb-supply-chain-basic.apps.tanzu.vmware.com
  - ootb-supply-chain-testing.apps.tanzu.vmware.com
  - ootb-supply-chain-testing-scanning.apps.tanzu.vmware.com
# comment out remove the `supply_chain` property
#
# supply_chain: ""
```

Providing your own templates

Similar to supply chains, Cartographer templates (`Cluster*Template` resources) are cluster-scoped, so you must ensure that the new templates submitted to the cluster do not conflict with those installed by the `ootb-templates` package.

The following set of objects are provided by `ootb-templates`:

- ClusterConfigTemplate/**config-template**
- ClusterConfigTemplate/**convention-template**
- ClusterDeploymentTemplate/**app-deploy**
- ClusterImageTemplate/**image-provider-template**
- ClusterImageTemplate/**image-scanner-template**
- ClusterImageTemplate/**kpack-template**

- Task/**kaniko-build**
- ClusterImageTemplate/**kaniko-template**
- ClusterRole/**ootb-templates-app-viewer**
- ClusterRole/**ootb-templates-deliverable**
- ClusterRole/**ootb-templates-workload**
- ClusterRunTemplate/**tekton-source-pipelinerun**
- ClusterSourceTemplate/**delivery-source-template**
- ClusterSourceTemplate/**source-scanner-template**
- ClusterSourceTemplate/**source-template**
- ClusterSourceTemplate/**testing-pipeline**
- Task/**git-writer**
- Task/**image-writer**
- ClusterTemplate/**config-writer-template**
- ClusterTemplate/**deliverable-template**
- Task/**carvel-package (experimental)**
- ClusterConfigTemplate/**carvel-package (experimental)**
- ClusterTemplate/**package-config-writer-and-pull-requester-template (experimental)**
- ClusterTemplate/**package-config-writer-template (experimental)**

Before submitting your own, either ensure that the name and resource has no conflicts with those installed by `ootb-templates`, or exclude from the installation the template you want to override by using the `excluded_templates` property of `ootb-templates`.

For example, perhaps you want to override the `ClusterConfigTemplate` named `config-template` to provide your own with the same name, so that you don't need to edit the supply chain. In `tap-values.yaml`, you can exclude template provided by Tanzu Application Platform:

```
ootb_templates:
  excluded_templates:
    - 'config-writer'
```

For details about how to edit an existing template, see Modifying an Out of the Box Supply template section.

Modifying an Out of the Box Supply Chain

To change the shape of a supply chain or the template that it points to, do the following:

1. Copy one of the reference supply chains.
2. Remove the old supply chain. See preventing Tanzu Application Platform supply chains from being installed.
3. Edit the supply chain object.
4. Submit the modified supply chain to the cluster.

Example

In this example, you have a new `ClusterImageTemplate` object named `foo` that you want use for building container images instead of the out of the box object that makes use of Kpack. The supply

chain that you want to apply the modification to is `source-to-url` provided by the `ootb-supply-chain-basic` package.

1. Find the image that contains the supply chain definition:

```
kubectl get app ootb-supply-chain-basic \
  -n tap-install \
  -o jsonpath={.spec.fetch[0].imgpkgBundle.image}
```

```
registry.tanzu.vmware.com/tanzu-application-platform/tap-packages@sha256:f2ad40
1bb3e850940...
```

2. Pull the contents of the bundle into a directory named `ootb-supply-chain-basic`:

```
imgpkg pull \
  -b registry.tanzu.vmware.com/tanzu-application-platform/tap-packages@sha256:f
2ad401bb3e850940... \
  -o ootb-supply-chain-basic
```

```
Pulling bundle 'registry.tanzu.vmware.com/tanzu-...
  Extracting layer 'sha256:542f2bb8eb946fe9d2c8a...

Locating image lock file images...
The bundle repo (registry.tanzu.vmware.com/tanzu...

Succeeded
```

3. Inspect the files obtained:

```
tree ./ootb-supply-chain-basic/
```

```
./ootb-supply-chain-basic/
├── config
│   ├── supply-chain-image.yaml
│   └── supply-chain.yaml
└── values.yaml
```

4. Edit the supply chain that you want to exchange the template with another:

```
--- a/supply-chain.yaml
+++ b/supply-chain.yaml
@@ -52,7 +52,7 @@ spec:
   - name: image-builder
     templateRef:
       kind: ClusterImageTemplate
-      name: kpack-template
+      name: foo
     params:
       - name: serviceAccount
         value: #@ data.values.service_account
```

5. Submit the supply chain to Kubernetes:

   The supply chain definition found in the bundle expects the values you provided by using `tap-values.yaml` to be interpolated by using YTT before they are submitted to Kubernetes. So before applying the modified supply chain to the cluster, use YTT to interpolate those values. After that, run:

```
ytt \
  --ignore-unknown-comments \
  --file ./ootb-supply-chain-basic/config \
```

```
  --data-value registry.server=REGISTRY-SERVER \
  --data-value registry.repository=REGISTRY-REPOSITORY |
  kubectl apply -f-
```

Important

The modified supply chain does not outlive the destruction of the cluster. VMware recommends that you save it, for example, in a Git repository to install on every cluster where you expect the supply chain to exist.

Modifying an Out of the Box Supply template

The Out of the Box Templates package (`ootb-templates`) includes all of the templates and shared Tekton tasks used by the supply chains shipped by using `ootb-supply-chain-*` packages. Any template that you want to edit, for example, to change details about the resources that are created based on them, is part of this package.

The workflow for updating a template is as follows:

1. Copy one of the reference templates from `ootb-templates`.
2. Exclude that template from the set of objects provided by `ootb-templates`. For more information, see `excluded_templates` in Providing your Own Templates.
3. Edit the template.
4. Submit the modified template to the cluster.

Note

You don’t need to change anything related to supply chains, because you’re preserving the name of the object referenced by the supply chain.

Example

In this example, you want to update the `ClusterImageTemplate` object called `kpack-template`, which provides a template for creating `kpack/Image`s to hardcode an environment variable.

1. Exclude the `kpack-template` from the set of templates that `ootb-templates` installs by upating `tap-values.yaml`:

   ```
   ootb_templates:
   excluded_templates: ['kpack-template']
   ```

2. Find the image that contains the templates:

   ```
   kubectl get app ootb-templates \
     -n tap-install \
     -o jsonpath={.spec.fetch[0].imgpkgBundle.image}
   ```

   ```
   registry.tanzu.vmware.com/tanzu-application-platform/tap-packages@sha256:a5e177
   f38d7287f2ca7ee2afd67ff178645d8f1b1e47af4f192a5ddd6404825e
   ```

3. Pull the contents of the bundle into a directory named `ootb-templates`:

   ```
   imgpkg pull \
     -b registry.tanzu.vmware.com/tanzu-application-platform/tap-packages@sha256:a
   ```

```
5e177f38d7.. \
  -o ootb-templates
```

```
Pulling bundle 'registry.tanzu.vmware.com/tanzu-...
  Extracting layer 'sha256:a5e177f38d7...

Locating image lock file images...
The bundle repo (registry.tanzu.vmware.com/tanzu...

Succeeded
```

4. Confirm that you downloaded all the templates:

```
tree ./ootb-templates
```

```
./ootb-templates
├── config
│   ├── cluster-roles.yaml
│   ├── config-template.yaml
│   ├── kpack-template.yaml         # ! the one we want to modify
...
│   └── testing-pipeline.yaml
└── values.yaml
```

5. Change the property you want to change:

```
--- a/config/kpack-template.yaml
+++ b/config/kpack-template.yaml
@@ -65,6 +65,8 @@ spec:
         subPath: #@ data.values.workload.spec.source.subPath
       build:
         env:
+        - name: FOO
+          value: BAR
         - name: BP_OCI_SOURCE
           value: #@ data.values.source.revision
         #@ if/end param("live-update"):
```

6. Submit the template.

The name of the template is preserved but the contents are changed. So after the template is submitted, the supply chains are all embedded to the build of the application container images that have `FOO` environment variable.

Live modification of supply chains and templates

Preceding sections covered how to update supply chains or templates installed in a cluster. This section shows how you can experiment by making small changes in a live setup with `kubectl edit`.

When you install Tanzu Application Platform by using profiles, a `PackageInstall` object is created. This in turn creates a set of children `PackageInstall` objects for installing the individual components that make up the platform.

```
PackageInstall/tap
└─App/tap
  ├─ PackageInstall/cert-manager
  ├─ PackageInstall/cartographer
  ├─ ...
  └─ PackageInstall/tekton-pipelines
```

Because the installation is based on Kubernetes primitives, `PackageInstall` tries to achieve the state where all packages are installed.

This is great but presents challenges for modifying the contents of some of the objects that the installation submits to the cluster. Namely, such modifications cause the original definition persisting instead of the changes.

For this reason, before you perform any customization to the Out of the Box packages, you must pause the top-level `PackageInstall/tap` object. Run:

```
kubectl edit -n tap-install packageinstall tap
```

```
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: tap
  namespace: tap-install
spec:
  paused: true                    # ! set this field to `paused: true`.
  packageRef:
    refName: tap.tanzu.vmware.com
    versionSelection:
# ...
```

With the installation of Tanzu Application Platform paused, all of the installed components are still there, but changes to those children `PackageInstall` objects are not overwritten.

Now you can pause the `PackageInstall` objects that relate to the templates or supply chains you want to edit.

For example:

- To edit templates: `kubectl edit -n tap-install packageinstall ootb-templates`
- To edit the basic supply chains: `kubectl edit -n tap-install packageinstall ootb-supply-chain-basic`

setting `packageinstall.spec.paused: true`.

With the installations paused, further live changes to templates or supply chains are persisted until you revert the `PackageInstall`s to not being paused. To persist the changes, follow the steps outlined in the earlier sections.

Adding custom behavior to Supply Chains

Most behaviors in supply chains are supplied by Kubernetes controllers. For example, Cloud Native Buildpacks are created by the kpack controller when a kpack Image object is created. Sometimes there is need for behavior and no controller for it exists. In these instances, you might want to write a script that uses a CLI tool, or interact with an external API. To do this, you can bring the behavior to the supply chain by using Tekton.

You can look at the kaniko image-building as an example. You create a Tekton Task `kaniko-build` with instructions for how to build a Docker image using kaniko given a set of parameters. The Task has a set of steps. Each step refers to a container image and a set of instructions to run on the image. For example, it can be a Linux image against which a set of bash instructions are run. The Task is installed on the cluster in the `tap-tasks` namespace.

You create the ClusterImageTemplate `kaniko-template` to create Tekton TaskRuns. TaskRuns are immutable, so you add the `lifecycle: tekton` field to the template's specifications. This ensures two things:

- When inputs to the template change, rather than updating the TaskRun, a new TaskRun is created.
- Only the values from the most recently created TaskRun that is successful are propagated forward in the supply chain.

To learn more about the `lifecycle: tekton` field, see the Cartographer tutorial Lifecycle: Templating Objects That Cannot Update. To learn more about Tekton, see the Tekton documentation.

Tekton Tasks on a cluster with Pod Security Admission

Kubernetes administrators can enable the Pod Security Admission controller to restrict the behavior of pods in a clear consistent fashion. For more information, see the Kubernetes documentation. If this is the case on a cluster, the Tekton Tasks must be altered to adhere to the security context.

Include a PSA-compliant `stepTemplate`

The task specification must include a `stepTemplate` field with the following properties defined:

```
stepTemplate:
  securityContext:
    allowPrivilegeEscalation: false
    runAsUser: 1000
    capabilities:
      drop:
        - ALL
    seccompProfile:
      type: "RuntimeDefault"
    runAsNonRoot: true
```

For more information about `stepTemplate`, see the Tekton documentation.

Write to available directories

To be PSA-compliant, `stepTemplate` enforces that it is run as a non-root user. Task authors must ensure that there is no attempt to write to protected directories.

For example, a user might want to run a Go test and for their Task to use the Golang image.

In that image, the default value for the `HOME` environment variable is `/root`. This means that when running a command such as `go test`, binaries are created in a subdirectory of the root directory. However, the user does not have permission to create binaries in a subdirectory of the root directory.

To address this example, the task author can include a step `env` to override the default value:

```
    steps:
    - image: golang
      name: test
      env:
      - name: HOME
        value: /go
```

Knowing which directories are safe depends on the image you use. In this example, the `/go` directory exists and is not protected on the `golang` image, but the directory does not exist on an alpine image.

Reference guides for Supply Chain Choreographer for Tanzu

This topic describes the reference guides you can use for Supply Chain Choreographer for Tanzu.

Reference guides

The following reference guides apply to Supply Chain Choreographer for Tanzu:

- Tanzu Build Service Integration
- Use GitOps or RegistryOps with Supply Chain Choreographer
- Delivery reference for Supply Chain Choreographer
- Delivery reference for Supply Chain Choreographer
- ClusterRunTemplate reference for Supply Chain Choreographer
- Template reference for Supply Chain Choreographer
- Supply chains for Supply Chain Choreographer
- Workload Reference for Supply Chain Choreographer
- Events reference for Supply Chain Choreographer

Events reference for Supply Chain Choreographer

This topic describes each event you can view with Supply Chain Choreographer.

Events are emitted when Choreographer edits resources or notices a change in their output or healthy state. Don't treat events like logs, however they can offer valuable insight into what's happening in a supply chain over time. For example, very high occurrences of events in a short period of time might be a sign of slow application-level processing due to many page faults and a lack of storage resources.

Events are published on Workload, Deliverable, and Runnable resources. You can view them manually using:

```
kubectl describe workload.carto.run <workload-name> -n <workload-ns>
kubectl describe runnable.carto.run <runnable-name> -n <runnable-ns>
kubectl describe deliverable.carto.run <deliverable-name> -n <deliverable-ns>
```

Events

The following sections define the different events.

StampedObjectApplied

This event is emitted every time Choreographer creates or updates a resource. The created or updated resource is referenced in the event message.

Example messages:

```
Created object [gitrepositories.source.toolkit.fluxcd.io/my-project]
Updated object [apps.kappctrl.k14s.io/my-project-app]
```

StampedObjectRemoved

This event is emitted every time Choreographer deletes a resource. This currently only occurs when Runnable resources expire. The deleted object is referenced in the event message.

Example message:

```
Deleted object [task.tekton.dev/my-project-a737bdf]
```

ResourceOutputChanged

This event is emitted every time Choreographer recognizes a new output from a resource.

Example message:

```
[source-provider] found a new output in [imagerepositories.source.apps.tanzu.vmware.com/app]
```

ResourceHealthyStatusChanged

This event is emitted every time Choreographer recognizes that the healthy status of a resource has changed.

Example message:

```
[image-provider] found healthy status in [images.kpack.io/app] changed to [True]
[source-provider] found healthy status in [[gitrepositories.source.toolkit.fluxcd.io/my-project]] changed to [False]
```

Workload Reference for Supply Chain Choreographer

This topic describes the fields you can use for Supply Chain Choreographer workloads.

Standard Fields

Cartographer workloads have standard fields leveraged by supply chains. See Cartographer's Reference Documentation in the Cartographer documentation.

Labels

Workload labels affect which supply chain is selected. For information about which template is defined for a particular reference, see Selectors in the Cartographer documentation. Individual templates can also use workload labels.

OOTB Supply Chains use the following workload labels:

- `apps.tanzu.vmware.com/has-tests` by Source-Test-to-URL and Source-Test-Scan-to-URL.
- `apps.tanzu.vmware.com/workload-type` by all supply chains.
- `apis.apps.tanzu.vmware.com/register-api` by the Api-Descriptors Template.
- `apps.tanzu.vmware.com/carvel-package-workflow` by source-to-url-package (experimental) and basic-image-to-url-package (experimental).

Parameters

The OOTB templates are configured with parameters from the supply chain or workload. For information about Cartographer parameters, including precedence rules, see Parameters in the Cartographer documentation.

What parameters are relevant depends on the supply chain that selects the workload, for two reasons:

1. The OOTB supply chains refer to overlapping sets of templates. A workload selected by the Source-to-URL supply chain can provide a `scanning_image_template` parameter, but the supply chain does not refer to a template that leverages that parameter.
2. You can write Supply Chains to provide a parameter value to a template and prevent the workload from overriding the value. See Further Information in the Cartographer documentation.

The following list of parameters are respected by some OOTB supply chains. Each provides the templates that respect the parameter. The reference for the template details which supply chains include the template.

- gitImplementation: source-template
- source_credentials_secret: source-template
- gitops_credentials_secret: deliverable-template
- gitops_ssh_secret: source-template, deliverable-template, external-deliverable-template
- serviceAccount: source-template, image-provider-template, kpack-template, kaniko-template, convention-template, config-writer-template, config-writer-and-pull-requester-template, deliverable-template, external-deliverable-template
- maven: source-template
- testing_pipeline_matching_labels: testing-pipeline
- testing_pipeline_params: testing-pipeline
- scanning_source_template: source-scanner-template
- scanning_source_policy: source-scanner-template
- clusterBuilder: kpack-template
- buildServiceBindings: kpack-template
- live-update: kpack-template, convention-template
- dockerfile: kaniko-template
- docker_build_context: kaniko-template
- docker_build_extra_args: kaniko-template
- scanning_image_template: image-scanner-template
- scanning_image_policy: image-scanner-template
- annotations: convention-template, service-bindings, api-descriptors
- debug: convention-template
- ports: server-template
- api-descriptors: api-descriptors
- gitops_branch: config-writer-template, config-writer-and-pull-requester-template, deliverable-template, external-deliverable-template
- gitops_user_name: config-writer-template, config-writer-and-pull-requester-template
- gitops_user_email: config-writer-template, config-writer-and-pull-requester-template
- gitops_commit_message: config-writer-template, config-writer-and-pull-requester-template

- gitops_repository: config-writer-template, deliverable-template, external-deliverable-template
- gitops_repository_prefix: config-writer-template, deliverable-template, external-deliverable-template
- gitops_server_address: config-writer-template, config-writer-and-pull-requester-template, deliverable-template, external-deliverable-template
- gitops_repository_owner: config-writer-template, config-writer-and-pull-requester-template, deliverable-template, external-deliverable-template
- gitops_repository_name: config-writer-template, config-writer-and-pull-requester-template, deliverable-template, external-deliverable-template
- gitops_commit_branch: config-writer-and-pull-requester-template
- gitops_pull_request_title: config-writer-and-pull-requester-template
- gitops_pull_request_body: config-writer-and-pull-requester-template
- gitops_server_kind: config-writer-and-pull-requester-template
- carvel_package_gitops_subpath (experimental): carvel-package, package-config-writer-template, package-config-writer-and-pull-requester-template
- carvel_package_name_suffix (experimental): carvel-package, package-config-writer-template, package-config-writer-and-pull-requester-template
- carvel_package_openapiv3_enabled (experimental): carvel-package

Service Account

To create the templated objects, Cartographer needs a reference to a service account with permissions to manage resources. This service account might be provided in the workload's `.spec.serviceAccountName` field or in the supply chain's `spec.serviceAccountRef` field. See Service Account and Workload and Supply Chain Custom Resources in the Cartographer documentation. When using the Tanzu CLI to create a workload, specify this service account's name with the `--service-account` flag.

After the templated objects are created, they often need a service account with permissions to do work. In the OOTB Templates and Supply Chains, the parameter `serviceAccount` must reference the service account for these objects. When using the Tanzu CLI to create a workload, specify this service account's name with `--param serviceAccount=...`.

Supply chains for Supply Chain Choreographer

This topic describes the parameters for supply chains that you can use with Supply Chain Choreographer.

Tanzu Application Platform includes a number of supply chains packages, each of which installs two ClusterSupplyChains. You can only install one supply chain package at a time.

The supply chains provide some parameters to the referenced templates. The parameters provided by the workload might override the parameters in this topic.

Source-to-URL

Purpose

- Fetches application source code

- Builds it into an image
- Writes the Kubernetes configuration necessary to deploy the application
- Commits that configuration to either a Git repository or a container image registry

Resources

This section describes the templates and their parameters.

source-provider

Refers to source-template.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `source_credentials_secret` from tap-value `source.credentials_secret`. Overridable by workload.

image-provider

Refers to kaniko-template when the workload provides a parameter `dockerfile`. Refers to kpack-template otherwise.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `registry` from tap-value `registry`. NOT overridable by workload.
- `clusterBuilder` from tap-value `cluster_builder`. Overridable by workload.
- `dockerfile` value `./Dockerfile`. Overridable by workload.
- `docker_build_context` value `./`. Overridable by workload.
- `docker_build_extra_args` value `[]`. Overridable by workload.

Common resources

- Config-Provider
- App-Config
- Service-Bindings
- Api-Descriptors
- Config-Writer
- Deliverable

Parameters provided to all resources

- `maven_repository_url` from tap-value `maven.repository.url`. NOT overridable by workload.
- `maven_repository_secret_name` from tap-value `maven.repository.secret_name`. NOT overridable by workload.
- See Params provided by all Supply Chains to all Resources

Package

Out of the Box Supply Chain Basic

More information

See Install Out of the Box Supply Chain Basic for information about setting tap-values at installation time.

Source-Test-to-URL

- Fetches application source code
- Runs user defined tests against the code
- Builds the code into an image
- Writes the Kubernetes configuration necessary to deploy the application
- Commits that configuration to either a Git repository or a container image registry

Resources

source-provider

Refers to source-template.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `source_credentials_secret` from tap-value `source.credentials_secret`. Overridable by workload.

source-tester

Refers to testing-pipeline.

No parameters are provided by the supply-chain.

image-provider

Refers to kaniko-template when the workload provides a parameter `dockerfile`. Refers to kpack-template otherwise.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `registry` from tap-value `registry`. NOT overridable by workload.
- `clusterBuilder` from tap-value `cluster_builder`. Overridable by workload.
- `dockerfile` value `./Dockerfile`. Overridable by workload.
- `docker_build_context` value `./`. Overridable by workload.
- `docker_build_extra_args` value `[]`. Overridable by workload.

Common resources

- Config-Provider
- App-Config
- Service-Bindings
- Api-Descriptors
- Config-Writer

- Deliverable

Parameters provided to all resources

- `maven_repository_url` from tap-value `maven.repository.url`. NOT overridable by workload.
- `maven_repository_secret_name` from tap-value `maven.repository.secret_name`. NOT overridable by workload.
- See Params provided by all Supply Chains to all Resources.

Package

Out of the Box Supply Chain Testing

More information

See Install Out of the Box Supply Chain with Testing for information about setting tap-values at installation time.

Source-Test-Scan-to-URL

- Fetches application source code
- Runs user defined tests against the code
- Scans the code for vulnerabilities
- Builds the code into an image
- Scans the image for vulnerabilities
- Writes the Kubernetes configuration necessary to deploy the application
- Commits that configuration to either a Git repository or an image registry

Resources

source-provider

Refers to source-template.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `source_credentials_secret` from tap-value `source.credentials_secret`. Overridable by workload.

source-tester

Refers to testing-pipeline.

No parameters are provided by the supply-chain.

source-scanner

Refers to source-scanner-template.

Parameters provided:

- `scanning_source_policy` from tap-value `scanning.source.policy`. Overridable by workload.

- `scanning_source_template` from tap-value `scanning.source.template`. Overridable by workload.

image-provider

Refers to kaniko-template when the workload provides a parameter `dockerfile`. Refers to kpack-template otherwise.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `registry` from tap-value `registry`. NOT overridable by workload.
- `clusterBuilder` from tap-value `cluster_builder`. Overridable by workload.
- `dockerfile` value `./Dockerfile`. Overridable by workload.
- `docker_build_context` value `./`. Overridable by workload.
- `docker_build_extra_args` value `[]`. Overridable by workload.

image-scanner

Refers to image-scanner-template.

Parameters provided:

- `scanning_image_policy` from tap-value `scanning.image.policy`. Overridable by workload.
- `scanning_image_template` from tap-value `scanning.image.template`. Overridable by workload.

Common resources

- Config-Provider
- App-Config
- Service-Bindings
- Api-Descriptors
- Config-Writer
- Deliverable

Parameters provided to all resources

- `maven_repository_url` from tap-value `maven.repository.url`. NOT overridable by workload.
- `maven_repository_secret_name` from tap-value `maven.repository.secret_name`. NOT overridable by workload.
- See Params provided by all Supply Chains to all Resources

Package

Out of the Box Supply Chain Testing Scanning

More information

See Install Out of the Box Supply Chain with Testing and Scanning for information about setting tap-values at installation time.

Basic-Image-to-URL

- Fetches a prebuilt image.
- Writes the Kubernetes configuration necessary to deploy the application.
- Commits that configuration to either a Git repository or an image registry.

Resources

image-provider

Refers to image-provider-template.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.

Common resources

- Config-Provider
- App-Config
- Service-Bindings
- Api-Descriptors
- Config-Writer
- Deliverable

Parameters provided to all resources

- See Params provided by all Supply Chains to all Resources

Package

Out of the Box Supply Chain Basic

More information

See Install Out of the Box Supply Chain Basic for information about setting tap-values at installation time.

Testing-Image-to-URL

- Fetches a prebuilt image.
- Writes the Kubernetes configuration necessary to deploy the application.
- Commits that configuration to either a Git repository or an image registry.

Resources

image-provider

Refers to image-provider-template.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.

Common resources

- Config-Provider
- App-Config
- Service-Bindings
- Api-Descriptors
- Config-Writer
- Deliverable

Parameters provided to all resources

- See Params provided by all Supply Chains to all Resources

Package

Out of the Box Supply Chain Testing

More information

See Install Out of the Box Supply Chain with Testing for information about setting tap-values at installation time.

Scanning-image-scan-to-URL

- Fetches a prebuilt image.
- Scans the image for vulnerabilities.
- Writes the Kubernetes configuration necessary to deploy the application.
- Commits the configuration to either a Git repository or an image registry.

Resources

image-provider

Refers to image-provider-template.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.

image-scanner

Refers to image-scanner-template.

Parameters provided:

- `scanning_image_policy` from tap-value `scanning.image.policy`. Overridable by workload.
- `scanning_image_template` from tap-value `scanning.image.template`. Overridable by workload.

Common resources

- Config-Provider
- App-Config

- Service-Bindings
- Api-Descriptors
- Config-Writer
- Deliverable

Parameters provided to all resources

- See Params provided by all Supply Chains to all Resources

Package

Out of the Box Supply Chain Testing Scanning

More information

See Install Out of the Box Supply Chain with Testing and Scanning for information about setting tap-values at installation time.

Source-to-URL-Package (experimental)

Purpose

- Fetches the application source code.
- Builds the source code into an image.
- Bundles the Kubernetes configuration necessary to deploy the application into a Carvel Package.
- Commits the Package to a Git Repository.

Resources

This section describes the templates and their parameters.

source-provider

Refers to source-template.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `source_credentials_secret` from tap-value `source.credentials_secret`. Overridable by workload.

image-provider

Refers to kaniko-template when the workload provides a parameter `dockerfile`. Refers to kpack-template otherwise.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `registry` from tap-value `registry`. NOT overridable by workload.
- `clusterBuilder` from tap-value `cluster_builder`. Overridable by workload.
- `dockerfile` value `./Dockerfile`. Overridable by workload.

- `docker_build_context` value `./`. Overridable by workload.
- `docker_build_extra_args` value `[]`. Overridable by workload.

carvel-package

Refers to carvel-package.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `registry` from tap-value `registry`. NOT overridable by workload.

package-config-writer

Refers to the package-config-writer-and-pull-requester-template when the tap-value `gitops.commit_strategy` is `pull_request`. Otherwise, this resource refers to the package-config-writer-template.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `registry` from tap-value `registry`. NOT overridable by workload.

Common resources

- Config-Provider
- App-Config
- Service-Bindings
- Api-Descriptors

Parameters provided to all resources

- `maven_repository_url` from tap-value `maven.repository.url`. NOT overridable by workload.
- `maven_repository_secret_name` from tap-value `maven.repository.secret_name`. NOT overridable by workload.
- `carvel_package_gitops_subpath` from tap-value `carvel_package.gitops_subpath`. Overridable by workload.
- `carvel_package_name_suffix` from tap-value `carvel_package.name_suffix`. Overridable by workload.
- See Params provided by all Supply Chains to all Resources

Package

Out of the Box Supply Chain Basic

More information

See Install Out of the Box Supply Chain Basic for information about setting tap-values at installation time.

Basic-Image-to-URL-Package (experimental)

- Fetches a prebuilt image.

- Bundles the Kubernetes configuration necessary to deploy the application into a Carvel Package.
- Commits the Package to a Git Repository.

Resources

image-provider

Refers to image-provider-template.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.

carvel-package

Refers to carvel-package.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `registry` from tap-value `registry`. NOT overridable by workload.

package-config-writer

Refers to the package-config-writer-and-pull-requester-template when the tap-value `gitops.commit_strategy` is `pull_request`. Otherwise, this resource refers to the package-config-writer-template

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `registry` from tap-value `registry`. NOT overridable by workload.

Common resources

- Config-Provider
- App-Config
- Service-Bindings
- Api-Descriptors
- Config-Writer
- Deliverable

Parameters provided to all resources

- `carvel_package_gitops_subpath` from tap-value `carvel_package.gitops_subpath`. Overridable by workload.
- `carvel_package_name_suffix` from tap-value `carvel_package.name_suffix`. Overridable by workload.
- See Params provided by all Supply Chains to all Resources

Package

Out of the Box Supply Chain Basic

More information

See Install Out of the Box Supply Chain Basic for information about setting tap-values at installation time.

Resources common to all OOTB supply chains

config-provider

Refers to convention-template.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.

app-config

The tap-values field `supported_workloads` defines which templates are referred to by this resource. Default configuration is:

```
supported_workloads:
- type: web
  cluster_config_template_name: config-template
- type: server
  cluster_config_template_name: server-template
- type: worker
  cluster_config_template_name: worker-template
```

The workload's `apps.tanzu.vmware.com/workload-type` label determines which template is used at this step. For example, when the workload has a label `apps.tanzu.vmware.com/workload-type:web`, the supply chain references `config-template`.

No parameters are provided by the supply-chain.

service-bindings

Refers to the service-binding template.

No parameters are provided by the supply-chain.

api-descriptors

Refers to the api-descriptors template.

No parameters are provided by the supply-chain.

config-writer

Refers to the config-writer-and-pull-requester-template when the tap-value `gitops.commit_strategy` is `pull_request`. Otherwise, this resource refers to the config-writer-template

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `registry` from tap-value `registry`. NOT overridable by workload.

deliverable

Refers to the external-deliverable-template when the tap-value `external_delivery` evaluates to `true`. Otherwise the resource refers to the deliverable-template.

Parameters provided:

- `registry` from tap-value `registry`. NOT overridable by workload.

Parameters provided by all supply chains to all resources

All of the following parameters are overridable by the workload.

- `gitops_branch` from tap-value `gitops.branch`
- `gitops_user_name` from tap-value `gitops.username`
- `gitops_user_email` from tap-value `gitops.email`
- `gitops_commit_message` from tap-value `gitops.commit_message`
- `gitops_ssh_secret` from tap-value `gitops.ssh_secret`
- `gitops_credentials_secret` from tap-value `gitops.credentials_secret`
- `gitops_repository_prefix` from tap-value `gitops.repository_prefix` when present.
- `gitops_server_address` from tap-value `gitops.server_address` when present.
- `gitops_repository_owner` from tap-value `gitops.repository_owner` when present.
- `gitops_repository_name` from tap-value `gitops.repository_name` when present.
- `gitops_server_kind` from tap-value `gitops.pull_request.server_kind` when present.
- `gitops_commit_branch` from tap-value `gitops.pull_request.commit_branch` when present.
- `gitops_pull_request_title` from tap-value `gitops.pull_request.pull_request_title` when present.
- `gitops_pull_request_body` from tap-value `gitops.pull_request.pull_request_body` when present.

Template reference for Supply Chain Choreographer

This topic describes the objects from templates that you can use with Supply Chain Choreographer.

All the objects referenced in this topic are Cartographer Templates packaged in Out of the Box Templates.

This topic describes:

- The purpose of the templates
- The one or more objects that the templates create
- The supply chains that include the templates
- The parameters that the templates use

source-template

Purpose

Creates an object to fetch source code and make that code available to other objects in the supply chain. See Building from Source.

Used by

- Source-to-URL in the `source-provider` step.
- Source-Test-to-URL in the `source-provider` step.

- Source-Test-Scan-to-URL in the `source-provider` step.
- Source-to-URL-Package (experimental) in the `source-provider` step.

Creates

The source-template creates one of three objects, either:

- GitRepository. Created if the workload has `.spec.source.git` defined.
- MavenArtifact. Created if the template is provided a value for the parameter `maven`.
- ImageRepository. Created if the workload has `.spec.source.image` defined.

GitRepository

`GitRepository` makes source code from a particular commit available as a tarball in the cluster. Other resources in the supply chain can then access that code.

Parameters

Template reference for Supply Chain Choreographer

Parameter name	Meaning	Example
`gitImplementation`	The library used to fetch source code.	`- name: gitImplementation value: go-git`
`source_credentials_secret`	Name of the secret used to provide credentials for the Git repository. The secret with this name must exist in the same namespace as the `Workload`. The credentials must be sufficient to read the repository. See Git authentication.	`- name: source_credentials_secret value: git-credentials`
`gitops_ssh_secret`	Deprecated Only used if provided and source_credentials_secret is not provided. Name of the secret used to provide credentials for the Git repository. The secret with this name must exist in the same namespace as the `Workload`. The credentials must be sufficient to read the repository. See Git authentication.	`- name: gitops_ssh_secret value: git-credentials`

More information

For an example using the Tanzu CLI to create a Workload using GitHub as the provider of source code, see Create a workload from GitHub repository.

For information about GitRepository objects, see GitRepository.

ImageRepository

`ImageRepository` makes the contents of a container image available as a tarball on the cluster.

Parameters

Parameter name	Meaning	Example
`serviceAccount`	Name of the service account, providing credentials to `ImageRepository` for fetching container images. The service account must exist in the same namespace as the Workload.	`- name: serviceAccount value: default`

Note

When using the Tanzu CLI to configure this `serviceAccount` parameter, use `--param serviceAccount=...`. The similarly named `--service-account` flag sets a different value: the `spec.serviceAccountName` key in the Workload object.

More information

For information about the ImageRepository resource, see the ImageRepository reference documentation.

For information about how to use the Tanzu CLI to create a workload leveraging ImageRepository, see Create a workload from local source code.

MavenArtifact

`MavenArtifact` makes a pre-built Java artifact available to as a tarball on the cluster.

While the `source-template` leverages the workload's `.spec.source` field when creating a `GitRepository` or `ImageRepository` object, the creation of the `MavenArtifact` relies only on parameters in the Workload.

Parameters

Parameter name	Meaning	Example

`maven`	Points to the Maven artifact to fetch and the polling interval.	`- name: maven`
		`value:`
		`artifactId: springboot-initial`
		`groupId: com.example`
		`version: RELEASE`
		`classifier: sources # optional`
		`type: jar # optional`
		`artifactRetryTimeout: 1m0s # optional`

`maven_repository_url`	Specifies the Maven repository from which to fetch	`- name: maven_repository_url` `value: https://repo1.maven.org/maven2/`
`maven_repository_secret_name`	Specifies the secret containing credentials necessary to fetch from the Maven repository. The secret named must exist in the same workspace as the workload.	`- name: maven_repository_secret_name` `value: auth-secret`

More information

For information about the custom resource, see MavenArtifact reference docs.

For information about how to use the custom resource with the Tanzu Apps CLI plug-in, see Create a workload from a Maven repository artifact.

testing-pipeline

Purpose

Tests the source code provided in the supply chain. Testing depends on a user provided Tekton Pipeline. Parameters for this template allow for selection of the proper Pipeline and for specification of additional values to pass to the Pipeline.

Used by

- Source-Test-to-URL in the source-tester step.
- Source-Test-Scan-to-URL in the source-tester step.

These are used as the `source-tester` resource.

Creates

`testing-pipeline`creates a Runnable object. This Runnable provides inputs to the ClusterRunTemplate named tekton-source-pipelinerun.

Parameters

Parameter name	Meaning	Example

`testing_pipeline_matching_labels`	Set of labels to use when searching for Tekton Pipeline objects in the same namespace as the Workload. By default, a Pipeline labeled as `apps.tanzu.vmware.com/pipeline: test` is selected.	`- name: testing_pipeline_matching_labels`
		`value:`
		`apps.tanzu.vmware.com/pipeline: test`
		`my.company/language: golang`

`testing_pipeline_params`	Set of parameters to pass to the Tekton Pipeline. To this set of parameters, the template always adds the source URL and revision as `source-url` and `source-revision`.	`- name: testing_pipeline_params`
		`value:`
		`- name: verbose`
		`value: true`
		`- name: foo`
		`value: bar`

More information

For information about the ClusterRunTemplate that pairs with the Runnable, read tekton-source-pipelinerun

For information about the Tekton Pipeline that the user must create, read the OOTB Supply Chain Testing documentation of the Pipeline

source-scanner-template

Purpose

Scans the source code for vulnerabilities.

Used by

- Source-Test-Scan-to-URL in the source-scanner step.

This is used as the `source-scanner` resource.

Creates

SourceScan

Parameters

Parameter name	Meaning	Example
`scanning_source_template`	Name of the ScanTemplate object to use for running the scans. The ScanTemplate must be in the same namespace as the Workload.	`- name: scanning_source_template value: private-source-scan-template`
`scanning_source_policy`	Name of the ScanPolicy object to use when evaluating the scan results of a source scan. The ScanPolicy must be in the same namespace as the Workload.	`- name: scanning_source_policy value: allowlist-policy`

More information

For information about how to set up the Workload namespace with the ScanPolicy and ScanTemplate required for this resource, see Out of the Box Supply Chain with Testing and Scanning.

For information about the SourceScan custom resource, see SourceScan reference.

For information about how the artifacts found during scanning are catalogued, see Supply Chain Security Tools for Tanzu – Store.

image-provider-template

Purpose

Fetches a container image of a prebuilt application, specified in the workload's `.spec.image` field. This makes the content-addressable name, (e.g. the image name containing the digest) available to other resources in the supply chain.

Used by

- Basic-Image-to-URL in the image-provider step.
- Testing-Image-to-URL in the image-provider step.
- Scanning-Image-Scan-to-URL in the image-provider step.
- Basic-Image-to-URL-Package (experimental) in the image-provider step.

These are used as the `image-provider` resource.

Creates

ImageRepository.source.apps.tanzu.vmware.com

Parameters

Parameter name	Meaning	Example
`serviceAccount`	Name of the service account providing credentials for the target image registry. The service account must exist in the same namespace as the Workload.	`- name: serviceAccount value: default`

Note

When using the Tanzu CLI to configure this `serviceAccount` parameter, use `--param serviceAccount=...`. The similarly named `--service-account` flag sets a different value: the `spec.serviceAccountName` key in the Workload object.

More information

For information about the ImageRepository resource, see ImageRepository reference docs.

For information about prebuilt images, see Using a prebuilt image.

kpack-template

Purpose

Builds an container image from source code using cloud native buildpacks.

Used by

- Source-to-URL in the image-provider step.
- Source-Test-to-URL in the image-provider step.
- Source-Test-Scan-to-URL in the image-provider step.
- Source-to-URL-Package (experimental) in the image-provider step.

These are used as the `image-provider` resource when the workload parameter `dockerfile` is not defined.

Creates

Image.kpack.io

Parameters

Parameter name	Meaning	Example
`serviceAccount`	Name of the service account providing credentials for the configured image registry. `Image` uses these credentials to push built container images to the registry. The service account must exist in the same namespace as the Workload.	`- name: serviceAccount value: default`

Parameter name	Meaning	Example
`clusterBuilder`	Name of the Kpack Cluster Builder to use.	`- name: clusterBuilder value: nodejs-cluster-builder`
`buildServiceBindings`	Definition of a list of service bindings to make use at build time. For example, providing credentials for fetching dependencies from repositories that require credentials.	`- name: buildServiceBindings value: - name: settings-xml kind: Secret apiVersion: v1`
`live-update`	Enable the use of Tilt's live-update function.	`- name: live-update value: "true"`

Note

When using the Tanzu CLI to configure this `serviceAccount` parameter, use `--param serviceAccount=...`. The similarly named `--service-account` flag sets a different value: the `spec.serviceAccountName` key in the Workload object.

More information

For information about the integration with Tanzu Build Service, see Tanzu Build Service Integration.

For information about `live-update`, see Developer Conventions and Overview of Tanzu Developer Tools for IntelliJ.

For information about using Kpack builders with `clusterBuilder`, see Builders.

For information about `buildServiceBindings`, see Service Bindings.

kaniko-template

Purpose

Build an image for source code that includes a Dockerfile.

Used by

- Source-to-URL in the image-provider step.
- Source-Test-to-URL in the image-provider step.

- Source-Test-Scan-to-URL in the image-provider step.
- Source-to-URL-Package (experimental) in the image-provider step.

These are used as the `image-provider` resource when the workload parameter `dockerfile` is defined.

Creates

A taskrun.tekton.dev provides configuration to the Tekton Task `kaniko-build` which builds an image with kaniko.

This template uses the lifecycle: tekton flag to create new immutable objects rather than updating the previous object.

Parameters

Parameter name	Meaning	Example
`dockerfile`	relative path to the Dockerfile file in the build context	`./Dockerfile`
`docker_build_context`	relative path to the directory where the build context is	`.`
`docker_build_extra_args`	List of flags to pass directly to kaniko,such as providing arguments to a build.	`- --build-arg=FOO=BAR`
`serviceAccount`	Name of the service account to use for providing Docker credentials. The service account must exist in the same namespace as the Workload. The service account must have a secret associated with the credentials. See Configuring authentication for Docker in the Tekton documentation.	`- name: serviceAccount value: default`
`registry`	Specification of the registry server and repository in which the built image is placed.	`- name: registry value: server: index.docker.io repository: web-team`

More information

For information about how to use Dockerfile-based builds and limits associated with the function, see Dockerfile-based builds.

For information about `lifecycle:tekton`, read Cartographer Lifecycle.

image-scanner-template

Purpose

Scans the container image for vulnerabilities, persists the results in a store, and prevents the image from moving forward if CVEs are found which are not compliant with its referenced ScanPolicy.

Used by

- Source-Test-Scan-to-URL in the image-scanner step.
- Scanning-Image-Scan-to-URL in the image-scanner step.

Creates

ImageScan.scanning.apps.tanzu.vmware.com

Parameters

Parameter name	Meaning	Example
`scanning_image_template`	Name of the ScanTemplate object for running the scans against a container image. The ScanTemplate must be in the same namespace as the Workload.	`- name: scanning_image_template value: private-image-scan-template`
`scanning_image_policy`	Name of the ScanPolicy object for evaluating the scan results of an image scan. The ScanPolicy must be in the same namespace as the Workload.	`- name: scanning_image_policy value: allowlist-policy`

More information

For information about the ImageScan custom resource, see ImageScan reference.

For information about how the artifacts found during scanning are catalogued, see Supply Chain Security Tools for Tanzu – Store.

convention-template

Purpose

Create the PodTemplateSpec for the Kubernetes configuration (e.g. the knative service or kubernetes deployment) which are applied to the cluster.

Used by

- Source-to-URL in the config-provider step.
- Basic-Image-to-URL in the config-provider step.
- Source-Test-to-URL in the config-provider step.
- Testing-Image-to-URL in the config-provider step.
- Source-Test-Scan-to-URL in the config-provider step.
- Scanning-Image-Scan-to-URL in the config-provider step.

- Source-to-URL-Package (experimental) in the config-provider step.
- Basic-Image-to-URL-Package (experimental) in the config-provider step.

Creates

Creates a PodIntent object. The PodIntent leverages conventions installed on the cluster. The PodIntent object is responsible for generating a PodTemplateSpec. The PodTemplateSpec is used in app configs, such as knative services and deployments, to represent the shape of the pods to run the application in containers.

Parameters

Parameter name	Meaning	Example
`serviceAccount`	Name of the serviceAccount providing necessary credentials to `PodIntent`. The serviceAccount must be in the same namespace as the Workload. The serviceAccount is set as the `serviceAccountName` in the podtemplatespec. The credentials associated with the serviceAccount must allow fetching the container image used to inspect the metadata passed to convention servers.	`- name: serviceAccount` `value: default`

`annotations`	Extra set of annotations to pass down to the PodTemplateSpec.	`- name: annotations`
		`value:`
		`name: my-application`
		`version: v1.2.3`
		`team: store`

`debug`	Put the workload in debug mode.	`- name: debug` `value: "true"`
`live-update`	Enable live-updating of the code (for innerloop development).	`- name: live-update` `value: "true"`

Note

When using the Tanzu CLI to configure this `serviceAccount` parameter, use `--param serviceAccount=...`. The similarly named `--service-account` flag sets a different value: the `spec.serviceAccountName` key in the Workload object.

More information

For information about `PodTemplateSpec`, see PodTemplateSpec in the Kubernetes documentation.

For information about conventions, see Cartographer Conventions.

For information about the two convention servers enabled by default in Tanzu Application Platform installations, see Developer Conventions and Spring Boot conventions.

config-template

Purpose

For workloads with the label `apps.tanzu.vmware.com/workload-type: web`, define a knative service.

Used by

- Source-to-URL in the app-config step.
- Basic-Image-to-URL in the app-config step.
- Source-Test-to-URL in the app-config step.
- Testing-Image-to-URL in the app-config step.
- Source-Test-Scan-to-URL in the app-config step.
- Scanning-Image-Scan-to-URL in the app-config step.

Creates

A ConfigMap, in which the data field has a key `delivery.yaml` whose value is the definition of a knative service.

Parameters

None

More information

See workload types for more details about the three different types of workloads.

worker-template

Purpose

For workloads with the label `apps.tanzu.vmware.com/workload-type: worker`, define a Kubernetes Deployment.

Used by

- Source-to-URL in the app-config step.
- Basic-Image-to-URL in the app-config step.

- Source-Test-to-URL in the app-config step.
- Testing-Image-to-URL in the app-config step.
- Source-Test-Scan-to-URL in the app-config step.
- Scanning-Image-Scan-to-URL in the app-config step.

Creates

A ConfigMap, in which the data field has a key `delivery.yaml` whose value is the definition of a Kubernetes Deployment.

Parameters

None

More information

For information about the three different types of workloads, see workload types.

server-template

Purpose

For workloads with the label `apps.tanzu.vmware.com/workload-type: server`, define a Kubernetes Deployment and a Kubernetes Service.

Used by

- Source-to-URL in the app-config step.
- Basic-Image-to-URL in the app-config step.
- Source-Test-to-URL in the app-config step.
- Testing-Image-to-URL in the app-config step.
- Source-Test-Scan-to-URL in the app-config step.
- Scanning-Image-Scan-to-URL in the app-config step.
- Source-to-URL-Package (experimental) in the app-config step.
- Basic-Image-to-URL-Package (experimental) in the app-config step.

Creates

A ConfigMap, in which the data field has a key `delivery.yaml` whose value is the definitions of a Kubernetes Deployment and a Kubernetes Service to expose the pods.

Parameters

Parameter name	Meaning	Example

`ports`	Set of network ports to expose from the application to the Kubernetes cluster.	`- name: ports`
		`value:`
		`- containerPort: 2025`
		`name: smtp`
		`port: 25`

More information

For information about the three different types of workloads, see workload types.

For information about the ports parameter, see server-specific Workload parameters.

service-bindings

Purpose

Adds ServiceBindings to the set of Kubernetes configuration files.

Used by

- Source-to-URL in the service-bindings step.
- Basic-Image-to-URL in the service-bindings step.
- Source-Test-to-URL in the service-bindings step.
- Testing-Image-to-URL in the service-bindings step.
- Source-Test-Scan-to-URL in the service-bindings step.
- Scanning-Image-Scan-to-URL in the service-bindings step.
- Source-to-URL-Package (experimental) in the service-bindings step.
- Basic-Image-to-URL-Package (experimental) in the service-bindings step.

Creates

A ConfigMap. This template consumes input of multiple deployment YAML files and enriches the input with ResourceClaims and ServiceBindings if the workload contains serviceClaims.

Parameters

Parameter name	Meaning	Example

`annotations`	Extra set of annotations to pass down to the ServiceBinding and ResourceClaim objects.	`- name: annotations`
		`value:`
		`name: my-application`
		`version: v1.2.3`
		`team: store`

More information

For an example of using `--service-ref`, see the Tanzu CLI Command Reference documentation.

For an overview of the function, see Consume services on Tanzu Application Platform.

api-descriptors

Purpose

The `api-descriptor` resource takes care of adding an APIDescriptor to the set of Kubernetes objects to deploy such that API auto registration takes place.

Used by

- Source-to-URL in the api-descriptors step.
- Basic-Image-to-URL in the api-descriptors step.
- Source-Test-to-URL in the api-descriptors step.
- Testing-Image-to-URL in the api-descriptors step.
- Source-Test-Scan-to-URL in the api-descriptors step.
- Scanning-Image-Scan-to-URL in the api-descriptors step.
- Source-to-URL-Package (experimental) in the api-descriptors step.
- Basic-Image-to-URL-Package (experimental) in the api-descriptors step.

Creates

A ConfigMap. This template consumes input of multiple YAML files and enriches the input with an APIDescriptor if the workload has a label `apis.apps.tanzu.vmware.com/register-api` == to `true`.

Parameters

Parameter name	Meaning	Example

`annotations`	Extra set of annotations to pass down to the APIDescriptor object.	`- name: annotations`
		`value:`
		`name: my-application`
		`version: v1.2.3`
		`team: store`

`api_descriptor`	Information used to fill the state that you want of the APIDescriptor object (its spec).	`- name: api_descriptor`
		`value:`
		`type: openapi`
		`location:`
		`baseURL: http://petclinic-hard-coded.my-apps.tapdemo.vmware.com/`
		`path: "/v3/api`
		`owner: team-petclinic`
		`system: pet-clinics`
		`description: "example"`

More information

For information about API auto registration, see Use API Auto Registration.

config-writer-template

Purpose

Persist in an external system, such as a registry or git repository, the Kubernetes configuration passed to the template.

Used by

- Source-to-URL in the config-writer step.
- Basic-Image-to-URL in the config-writer step.
- Source-Test-to-URL in the config-writer step.
- Testing-Image-to-URL in the config-writer step.
- Source-Test-Scan-to-URL in the config-writer step.
- Scanning-Image-Scan-to-URL in the config-writer step.

Creates

A runnable which creates a Tekton TaskRun that refers either to the Tekton Task `git-writer` or the Tekton Task `image-writer`.

Parameters

Parameter name	Meaning	Example
`serviceAccount`	Name of the service account which provides the credentials to the registry or repository. The service account must exist in the same namespace as the Workload.	`- name: serviceAccount` `value: default`
`gitops_branch`	Name of the branch to push the configuration to.	`- name: gitops_branch` `value: main`
`gitops_user_name`	User name to use in the commits.	`- name: gitops_user_name` `value: "Alice Lee"`
`gitops_user_email`	User email address to use in the commits.	`- name: gitops_user_email` `value: alice@example.com`

Parameter name	Meaning	Example
`gitops_commit_message`	Message to write as the body of the commits produced for pushing configuration to the Git repository.	`- name: gitops_commit_message` `value: "ci bump"`
`gitops_repository`	The full repository URL to which the configuration is committed. **DEPRECATED**	`- name: gitops_repository` `value: "https://github.com/vmware-tanzu/cartographer"`
`gitops_repository_prefix`	The prefix of the repository URL. **DEPRECATED**	`- name: gitops_repository` `value: "https://github.com/vmware-tanzu/"`
`gitops_server_address`	The server URL of the Git repository to which configuration is applied.	`- name: gitops_server_address` `value: "https://github.com/"`
`gitops_repository_owner`	The owner/organization to which the repository belongs.	`- name: gitops_repository_owner` `value: vmware-tanzu`
`gitops_repository_name`	The name of the repository.	`- name: gitops_repository_name` `value: cartographer`

`registry`	Specification of the registry server and repository in which the configuration is placed.	`- name: registry`
		`value:`
		`server: index.docker.io`
		`repository: web-team`
		`ca_cert_data:`
		`-----BEGIN CERTIFICATE-----`
		`MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...`
		`-----END CERTIFICATE-----`

More information

For information about operating this template, see Gitops vs RegistryOps and the config-writer-and-pull-requester-template.

config-writer-and-pull-requester-template

Purpose

Persist the passed in Kubernetes configuration to a branch in a repository and open a pull request to another branch. This process allows for manual review of configuration before deployment to a cluster.

Used by

- Source-to-URL in the config-writer step.
- Basic-Image-to-URL in the config-writer step.
- Source-Test-to-URL in the config-writer step.
- Testing-Image-to-URL in the config-writer step.
- Source-Test-Scan-to-URL in the config-writer step.
- Scanning-Image-Scan-to-URL in the config-writer step.

Creates

A Tekton TaskRun refers to the Tekton Task `commit-and-pr`.

Parameters

Parameter name	Meaning	Example
`serviceAccount`	Name of the service account which provides the credentials to the registry or repository. The service account must exist in the same namespace as the Workload.	`- name: serviceAccount` `value: default`
`gitops_commit_branch`	Name of the branch to which configuration is pushed.	`- name: gitops_commit_branch` `value: feature`
`gitops_branch`	Name of the branch to which a pull request is opened.	`- name: gitops_branch` `value: main`
`gitops_user_name`	User name to use in the commits.	`- name: gitops_user_name` `value: "Alice Lee"`

Parameter name	Meaning	Example
`gitops_user_email`	User email address to use in the commits.	`- name: gitops_user_email` `value: alice@example.com`
`gitops_commit_message`	Message to write as the body of the commits produced for pushing configuration to the Git repository.	`- name: gitops_commit_message` `value: "ci bump"`
`gitops_pull_request_title`	Title of the pull request to be opened.	`- name: gitops_pull_request_title` `value: "ready for review"`
`gitops_pull_request_body`	Body of the pull request to be opened.	`- name: gitops_pull_request_body` `value: "generated by supply chain"`
`gitops_server_address`	The server URL of the Git repository to which configuration is applied.	`- name: gitops_server_address` `value: "https://github.com/"`
`gitops_repository_owner`	The owner/organization to which the repository belongs.	`- name: gitops_repository_owner` `value: vmware-tanzu`
`gitops_repository_name`	The name of the repository.	`- name: gitops_repository_name` `value: cartographer`
`gitops_server_kind`	The kind of Git provider	`- name: gitops_server_kind` `value: gitlab`

Parameter name	Meaning	Example

`ca_cert_data`	The string contents of the ssl certificate of the git server	`- name: ca_cert_data`
		`value:`
		`-----BEGIN CERTIFICATE-----`
		`MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...`
		`-----END CERTIFICATE-----`

More information

For information about the operation of this template, see Gitops vs RegistryOps and the config-writer-template.

deliverable-template

Purpose

Create a deliverable which pairs with a Delivery to deploy Kubernetes configuration on the cluster.

Used by

- Source-to-URL in the deliverable step.
- Basic-Image-to-URL in the deliverable step.
- Source-Test-to-URL in the deliverable step.
- Testing-Image-to-URL in the deliverable step.
- Source-Test-Scan-to-URL in the deliverable step.
- Scanning-Image-Scan-to-URL in the deliverable step.

Creates

A Deliverable preconfigured with reference to a repository or registry from which to fetch Kubernetes configuration.

Parameters

Parameter name	Meaning	Example
`serviceAccount`	Name of the service account providing the necessary permissions for the Delivery to create children objects. Populates the Deliverable's serviceAccount parameter. The service account must be in the same namespace as the Deliverable.	`- name: serviceAccount` `value: default`

Parameter name	Meaning	Example
`gitops_credentials_secret`	Name of the secret where credentials exist for fetching the configuration from a Git repository. Populates the Deliverable's `source_credentials_secret` parameter (The Workload's GitOps repository is the Deliverable's source reposiotor). The secret must be in the same namespace as the Deliverable.	`- name: gitops_credentials_secret` `value: git-secret`
`gitops_ssh_secret`	Deprecated. Use gitops_credentials_secret instead. Name of the secret where credentials exist for fetching the configuration from a Git repository. Populates the Deliverable's gitops_ssh_secret parameter. The secret must be in the same namespace as the Deliverable.	`- name: gitops_ssh_secret` `value: git-secret`
`gitops_branch`	Name of the branch from which to fetch the configuration.	`- name: gitops_branch` `value: main`
`gitops_repository`	The full repository URL to which the configuration is fetched. **DEPRECATED**	`- name: gitops_repository` `value: "https://github.com/vmware-tanzu/cartographer"`
`gitops_repository_prefix`	The prefix of the repository URL. **DEPRECATED**	`- name: gitops_repository` `value: "https://github.com/vmware-tanzu/"`
`gitops_server_address`	The server URL of the Git repository from which configuration is fetched.	`- name: gitops_server_address` `value: "https://github.com/"`
`gitops_repository_owner`	The owner/organization to which the repository belongs.	`- name: gitops_repository_owner` `value: vmware-tanzu`

Parameter name	Meaning	Example
`gitops_repository_name`	The name of the repository.	`- name: gitops_repository_name` `value: cartographer`

`registry`	Specification of the registry server and repository from which the configuration is fetched.	`- name: registry`
		`value:`
		`server: index.docker.io`
		`repository: web-team`
		`ca_cert_data:`
		`-----BEGIN CERTIFICATE-----`
		`MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...`
		`-----END CERTIFICATE-----`

Note

When using the Tanzu CLI to configure this `serviceAccount` parameter, use `--param serviceAccount=...`. The similarly named `--service-account` flag sets a different value: the `spec.serviceAccountName` key in the Workload object.

More information

For information about the ClusterDelivery shipped with `ootb-delivery-basic`, see Out of the Box Delivery Basic.

external-deliverable-template

Purpose

Create a definition of a deliverable which a user can manually applied to an external kubernetes cluster. When a properly configured Delivery is installed on that external cluster, the Deliverable will pair with the Delivery to deploy Kubernetes configuration on the cluster. For example, the OOTB Delivery.

Used by

- Source-to-URL in the deliverable step.
- Basic-Image-to-URL in the deliverable step.
- Source-Test-to-URL in the deliverable step.

- Testing-Image-to-URL in the deliverable step.
- Source-Test-Scan-to-URL in the deliverable step.
- Scanning-Image-Scan-to-URL in the deliverable step.

Creates

A configmap in which the `.data` field has a key `deliverable` for which the value is the YAML definition of a Deliverable.

Parameters

Parameter name	Meaning	Example
`serviceAccount`	Name of the service account providing the necessary permissions for the Delivery to create children objects. Populates the Deliverable's serviceAccount parameter. The service account must be in the same namespace as the Deliverable.	`- name: serviceAccount` `value: default`
`gitops_ssh_secret`	Name of the secret where credentials exist for fetching the configuration from a Git repository. Populates the Deliverable's gitops_ssh_secret parameter. The secret must be in the same namespace as the Deliverable.	`- name: gitops_ssh_secret` `value: ssh-secret`
`gitops_branch`	Name of the branch from which to fetch the configuration.	`- name: gitops_branch` `value: main`
`gitops_repository`	The full repository URL to which the configuration is fetched. **DEPRECATED**	`- name: gitops_repository` `value: "https://github.com/vmware-tanzu/cartographer"`
`gitops_repository_prefix`	The prefix of the repository URL. **DEPRECATED**	`- name: gitops_repository` `value: "https://github.com/vmware-tanzu/"`
`gitops_server_address`	The server URL of the Git repository from which configuration is fetched.	`- name: gitops_server_address` `value: "https://github.com/"`

Parameter name	Meaning	Example
`gitops_repository_owner`	The owner/organization to which the repository belongs.	`- name: gitops_repository_owner` `value: vmware-tanzu`
`gitops_repository_name`	The name of the repository.	`- name: gitops_repository_name` `value: cartographer`

`registry`	Specification of the registry server and repository from which the configuration is fetched.	`- name: registry`
		`value:`
		`server: index.docker.io`
		`repository: web-team`
		`ca_cert_data:`
		`-----BEGIN CERTIFICATE-----`
		`MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...`
		`-----END CERTIFICATE-----`

More information

For information about the ClusterDelivery shipped with `ootb-delivery-basic`, see Out of the Box Delivery Basic.

For information about using the Deliverable object in a multicluster environment, see Getting started with multicluster Tanzu Application Platform.

delivery-source-template

Purpose

Continuously fetches Kubernetes configuration files from a Git repository or container image registry and makes them available on the cluster.

Used by

- Delivery-Basic

Creates

The source-template creates one of three objects, either: - GitRepository. Created if the deliverable has `.spec.source.git` defined. - ImageRepository. Created if the deliverable has

`.spec.source.image` defined.

GitRepository

`GitRepository` makes source code from a particular commit available as a tarball in the cluster. Other resources in the supply chain can then access that code.

Parameters

Parameter name	Meaning	Example
`gitImplementation`	The library used to fetch source code.	`- name: gitImplementation value: go-git`
`source_credentials_secret`	Name of the secret used to provide credentials for the Git repository. The secret with this name must exist in the same namespace as the `Deliverable`. The credentials must be sufficient to read the repository. See Git authentication.	`- name: source_credentials_secret value: git-credentials`

More information

For an example using the Tanzu CLI to create a Workload using GitHub as the provider of source code, see Create a workload from GitHub repository.

For information about GitRepository objects, see GitRepository.

ImageRepository

`ImageRepository` makes the contents of a container image available as a tarball on the cluster.

Parameters

Parameter name	Meaning	Example
`serviceAccount`	Name of the service account, providing credentials to `ImageRepository` for fetching container images. The service account must exist in the same namespace as the Deliverable.	`- name: serviceAccount value: default`

More information

For information about the ImageRepository resource, see ImageRepository reference docs.

app-deploy

Purpose

Applies Kubernetes configuration to the cluster.

Used by

- Delivery-Basic

Creates

A kapp App.

Parameters

Parameter name	Meaning	Example
`serviceAccount`	Name of the service account providing the necessary privileges for `App` to apply the Kubernetes objects to the cluster. The service account must be in the same namespace as the Deliverable.	`- name: serviceAccount` `value: default`
`gitops_sub_path` (deprecated)	Sub directory within the configuration bundle that is used for looking up the files to apply to the Kubernetes cluster. **DEPRECATED**	`- name: gitops_sub_path` `value: ./config`

Note

The `gitops_sub_path` parameter is deprecated. Use `deliverable.spec.source.subPath` instead.

More information

For details about RBAC and how `kapp-controller` makes use of the ServiceAccount provided through the Deliverable's `serviceAccount` parameter, see kapp-controller's Security Model.

carvel-package (experimental)

Purpose

Bundles Kubernetes configuration into a Carvel Package.

Used by

- Source-to-URL-Package (experimental) in the carvel-package step.
- Basic-Image-to-URL-Package (experimental) in the carvel-package step.

Creates

A taskrun.tekton.dev which provides configuration to the `carvel-package` Tekton Task which bundles Kubernetes configuration into a Carvel Package.

This template uses the lifecycle: tekton flag to create new immutable objects rather than updating the previous object.

Parameters

Parameter name	Meaning	Example
`serviceAccount`	Name of the service account to use for providing Docker credentials. The service account must exist in the same namespace as the Workload. The service account must have a secret associated with the credentials. See Configuring authentication for Docker in the Tekton documentation.	`- name: serviceAccount` `value: default`

`registry`	Specification of the registry server and repository in which the built image is placed.	`- name: registry`
		`value:`
		`server: index.docker.io`
		`repository: web-team`

`carvel_package_gitops_subpath`	Specifies the subpath to which Carvel Packages should be written.	`- name: carvel_package_gitops_subpath` `value: path/to/my/dir`
`carvel_package_name_suffix`	Specifies the suffix to append to the Carvel Package name. The format is WORKLOAD_NAME.WORKLOAD_NAMESPACE.carvel_package_name_suffix The full Carvel Package name must be a valid DNS subdomain name as defined in RFC 1123.	`- name: carvel_package_name_suffix` `value: vmware.com`

Parameter name	Meaning	Example

`carvel_package_parameters`	Specifies the custom Carvel Package parameters	`- name: carvel_package_parameters`
		`  value: |`
		`  - selector:`
		`  matchLabels:`
		`  apps.tanzu.vmware.com/workload-type: server`
		`  schema: |`
		`  #@data/values-schema`
		`  ---`
		`  #@schema/title "Workload name"`
		`  #@schema/example "tanzu-java-web-app"`
		`  #@schema/validation min_len=1`
		`  workload_name: ""`
		
		`  #@schema/title "Replicas"`
		`  replicas: 1`
		
		`  #@schema/title "Port"`
		`  port: 8080`
		
		`  #@schema/title "Hostname"`
		`  #@schema/example "app.tanzu.vmware.com"`
		`  hostname: ""`
		
		`  #@schema/title "Cluster Issuer"`
		`  cluster_issuer: "tap-ingress-selfsigned"`
		
		`  #@schema/nullable`
		`  http_route:`
		`  #@schema/default [{"protocol": "https", "name": "default-gateway"}]`
		`  gateways:`
		`  - protocol: ""`
		`  name: ""`
		`  overlays: |`
		`  #@ load("@`

Parameter name	Meaning	Example
		ytt:overlay", "overlay") #@ load("@ytt:data", "data") #@overlay/match by=overlay.subset({"apiVersion":"apps/v1", "kind": "Deployment"}) --- spec: #@overlay/match missing_ok=True replicas: #@ data.values.replicas #@ if data.values.http_route != None: --- apiVersion: gateway.networking.k8s.io/v1beta1 kind: HTTPRoute metadata: name: #@ data.values.workload_name + "-route" spec: parentRefs: #@ for/end gateway in data.values.http_route.gateways: - group: gateway.networking.k8s.io kind: Gateway name: #@ gateway.name sectionName: #@ gateway.protocol + "-" + data.values.workload_name rules: - backendRefs: - name: #@ data.values.workload_name port: #@ data.values.port #@ elif data.values.hostname != "": --- apiVersio

Parameter name	Meaning	Example
		n: networking.k8s.io/v1 kind: Ingress metadata: name: #@ data.values.workload_name annotations: cert-manager.io/cluster-issuer: #@ data.values.cluster_issuer ingress.kubernetes.io/force-ssl-redirect: "true" kubernetes.io/ingress.class: contour kapp.k14s.io/change-rule: "upsert after upserting Services" labels: app.kubernetes.io/component: "run" carto.run/workload-name: #@ data.values.workload_name spec: tls: - secretName: #@ data.values.workload_name hosts: - #@ data.values.hostname rules: - host: #@ data.values.hostname http: paths: - pathType: Prefix path: / backend: service: name: #@ data.values.workload_name port: number: #@ data.value

Parameter name	Meaning	Example
		s.port #@ end - selector: matchLabels: apps.tanzu.vmware.com/workload-type: web schema: | #@data/values-schema --- #@schema/validation min_len=1 workload_name: "" overlays: "" - selector: matchLabels: apps.tanzu.vmware.com/workload-type: worker schema: | #@data/values-schema --- #@schema/validation min_len=1 workload_name: "" replicas: 1 overlays: | #@ load("@ytt:overlay", "overlay") #@ load("@ytt:data", "data") #@overlay/match by=overlay.subset({"apiVersion":"apps/v1", "kind": "Deployment"}) --- spec: #@overlay/match missing_ok=True replicas: #@ data.values.replicas
`carvel_package_openapiv3_enabled`	Specifies whether the Carvel Package should include a generated OpenAPIv3 specification	- name: carvel_package_openapiv3_enabled value: true

More information

To read more about `lifecycle:tekton`, read Cartographer Lifecycle.

package-config-writer-template (experimental)

Purpose

Persist in an external git repository the Carvel Package Kubernetes configuration passed to the template.

Used by

- Source-to-URL-Package (experimental) in the config-writer step.
- Basic-Image-to-URL-Package (experimental) in the config-writer step.

Creates

A runnable which creates a Tekton TaskRun that refers either to the Tekton Task `git-writer`.

Parameters

Parameter name	Meaning	Example
`serviceAccount`	Name of the service account which provides the credentials to the registry or repository. The service account must exist in the same namespace as the Workload.	`- name: serviceAccount` `value: default`
`gitops_branch`	Name of the branch to push the configuration to.	`- name: gitops_branch` `value: main`
`gitops_user_name`	User name to use in the commits.	`- name: gitops_user_name` `value: "Alice Lee"`
`gitops_user_email`	User email address to use in the commits.	`- name: gitops_user_email` `value: alice@example.com`
`gitops_commit_message`	Message to write as the body of the commits produced for pushing configuration to the Git repository.	`- name: gitops_commit_message` `value: "ci bump"`

Parameter name	Meaning	Example
`gitops_repository`	The full repository URL to which the configuration is committed. **DEPRECATED**	`- name: gitops_repository` `value: "https://github.com/vmware-tanzu/cartographer"`
`gitops_repository_prefix`	The prefix of the repository URL. **DEPRECATED**	`- name: gitops_repository` `value: "https://github.com/vmware-tanzu/"`
`gitops_server_address`	The server URL of the Git repository to which configuration is applied.	`- name: gitops_server_address` `value: "https://github.com/"`
`gitops_repository_owner`	The owner/organization to which the repository belongs.	`- name: gitops_repository_owner` `value: vmware-tanzu`
`gitops_repository_name`	The name of the repository.	`- name: gitops_repository_name` `value: cartographer`

`registry`	Specification of the registry server and repository in which the configuration is placed.	`- name: registry`
		`value:`
		`server: index.docker.io`
		`repository: web-team`
		`ca_cert_data:`
		`-----BEGIN CERTIFICATE-----`
		`MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...`
		`-----END CERTIFICATE-----`

Parameter name	Meaning	Example
`carvel_package_gitops_subpath`	Specifies the subpath to which Carvel Packages should be written.	`- name: carvel_package_gitops_subpath` `value: path/to/my/dir`
`carvel_package_name_suffix`	Specifies the suffix to append to the Carvel Package name. The format is WORKLOAD_NAME.WORKLOAD_NAMESPACE.carvel_package_name_suffix The full Carvel Package name must be a valid DNS subdomain name as defined in RFC 1123.	`- name: carvel_package_name_suffix` `value: vmware.com`

More information

See Gitops vs RegistryOps for more information about the operation of this template and of the package-config-writer-and-pull-requester-template (experimental).

package-config-writer-and-pull-requester-template (experimental)

Purpose

Persist the passed in Carvel Package Kubernetes configuration to a branch in a repository and open a pull request to another branch. (This process allows for manual review of configuration before deployment to a cluster)

Used by

- Source-to-URL-Package (experimental) in the config-writer step.
- Basic-Image-to-URL-Package (experimental) in the config-writer step.

Creates

A Tekton TaskRun which refers to the Tekton Task `commit-and-pr`.

Parameters

Parameter name	Meaning	Example
`serviceAccount`	Name of the service account which provides the credentials to the registry or repository. The service account must exist in the same namespace as the Workload.	`- name: serviceAccount` `value: default`

`gitops_commit_branch`	Name of the branch to which configuration is pushed.	`- name: gitops_commit_branch` `value: feature`
`gitops_branch`	Name of the branch to which a pull request is opened.	`- name: gitops_branch` `value: main`
`gitops_user_name`	User name to use in the commits.	`- name: gitops_user_name` `value: "Alice Lee"`
`gitops_user_email`	User email address to use in the commits.	`- name: gitops_user_email` `value: alice@example.com`
`gitops_commit_message`	Message to write as the body of the commits produced for pushing configuration to the Git repository.	`- name: gitops_commit_message` `value: "ci bump"`
`gitops_pull_request_title`	Title of the pull request to be opened.	`- name: gitops_pull_request_title` `value: "ready for review"`
`gitops_pull_request_body`	Body of the pull request to be opened.	`- name: gitops_pull_request_body` `value: "generated by supply chain"`
`gitops_server_address`	The server URL of the Git repository to which configuration is applied.	`- name: gitops_server_address` `value: "https://github.com/"`
`gitops_repository_owner`	The owner/organization to which the repository belongs.	`- name: gitops_repository_owner` `value: vmware-tanzu`

`gitops_repository_name`	The name of the repository.	`- name: gitops_repository_name` `value: cartographer`
`gitops_server_kind`	The kind of Git provider	`- name: gitops_server_kind` `value: gitlab`
`carvel_package_gitops_subpath`	Specifies the subpath to which Carvel Packages should be written.	`- name: carvel_package_gitops_subpath` `value: path/to/my/dir`
`carvel_package_name_suffix`	Specifies the suffix to append to the Carvel Package name. The format is WORKLOAD_NAME.WORKLOAD_NAMESPACE.carvel_package_name_suffix The full Carvel Package name must be a valid DNS subdomain name as defined in RFC 1123.	`- name: carvel_package_name_suffix` `value: vmware.com`

`ca_cert_data`	The string contents of the ssl certificate of the git server	`- name: ca_cert_data`
		`value:`
		`-----BEGIN CERTIFICATE-----`
		`MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...`
		`-----END CERTIFICATE-----`

More information

See Gitops vs RegistryOps for more information about the operation of this template and of the package-config-writer-template (experimental).

ClusterRunTemplate reference for Supply Chain Choreographer

This topic lists the objects you can use with Supply Chain Choreographer. All the objects referenced in this topic are Cartographer ClusterRunTemplates packaged in Out of the Box Templates. This topic describes the one or more objects they create, the supply chains that include them, and the parameters they use.

tekton-source-pipelinerun

Purpose

Tests source code.

Used by

- testing-pipeline

Creates

This ClusterRunTemplate creates a Tekton PipelineRun referring to the user's Tekton Pipeline.

Inputs

ClusterRunTemplate reference for Supply Chain Choreographer

Input name	Meaning	Example

`tekton-params`	Set of parameters to pass to the Tekton Pipeline	`- name: source-url`
		`value: https://github.com/vmware-tanzu/cartographer.git`
		`- name: source-revision`
		`value: e4a53f49a92fc913d26f8cc23d59102a51a5e635`
		`- name: verbose`
		`value: true`
		`- name: foo`
		`value: bar`

More information

For information about the runnable created in the OOTB Testing and OOTB Testing and Scanning, see testing-pipeline.

For information about the Tekton Pipeline that the user must create, see Tekton/Pipeline.

Delivery reference for Supply Chain Choreographer

This topic describes the delivery parameters and templates you can use with Supply Chain Choreographer.

Tanzu Application Platform delivery package installs a single ClusterDelivery.

The delivery provides some parameters to the templates. The parameters provided by the deliverable might override some of the delivery parameters in this topic. For more information about how parameters work, including precedence rules, see the Cartographer documentation.

delivery-basic

Purpose

- Fetches Kubernetes configuration created by a supply chain.
- Deploys the configuration on the cluster.

Resources

The following resources describe the templates.

source-provider

Refers to delivery-source-template.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by deliverable.

Deployer

Refers to app-deploy template.

Parameter provided:

- `serviceAccount` from tap-value `service_account`. Overridable by deliverable.

Package

Refers to Out of the Box Delivery Basic.

More information

For information about setting `tap-values.yaml` at installation time, see Install Out of the Box Delivery Basic.

Use Git with Supply Chain Choreographer

This topic explains how you can use Git with Supply Chain Choreographer.

Overview

The out of the box supply chains and delivery use Git in three ways:

- To fetch the developers source code, using the template.
- To store complete Kubernetes configuration, the write side of GitOps, using template 1, template 2, template 3 (experimental), and template 4 (experimental).
- To fetch stored Kubernetes configuration, the read side of GitOps, from either the same or a different Kubernetes cluster, using the template.

Supported Git Repositories

Tanzu Application Platform supports three Git providers:

- GitHub
- GitLab
- Azure DevOps

Related Articles

Git Authentication walks through the objects, such as secrets and service accounts, you can create on a cluster to allow successful supply chain Git operations. This includes the configuring a custom CA certificate.

GitOps versus RegistryOps discusses the two methods of storing built Kubernetes configuration, either in a Git repository or a container image registry, and walks through the parameters that must be provided for each.

Configuration for Azure DevOps: discusses configuration necessary for working with this Git provider.

Use GitOps or RegistryOps with Supply Chain Choreographer

You can use GitOps or RegistryOps to manage your Kubernetes configuration with Supply Chain Choreographer.

Regardless of the supply chain that a workload goes through, in the end, some Kubernetes configuration is pushed to an external entity, either to a Git repository or to a container image registry.

For example:

```
Supply Chain

  -- fetch source
    -- test
      -- build
        -- scan
          -- apply-conventions
            -- push config        * either to Git or Registry
```

This topic dives into the specifics of that last phase of the supply chains by pushing configuration to a Git repository or a container image registry.

Note

For more information about providing source code either from a local directory or Git repository, see Building from Source.

GitOps

The GitOps approach differs from local iteration in that GitOps configures the supply chains to push the Kubernetes configuration to a remote Git repository. This allows users to compare configuration changes and promote those changes through environments by using GitOps principles.

Typically associated with an outerloop workflow, the GitOps approach is only activated if a collection of parameters are set:

- `gitops.server_address` during the Out of the Box Supply Chains package installation or `gitops_server_address` configured as a workload parameter.
- `gitops.repository_owner` during the Out of the Box Supply Chains package installation or `gitops_repository_owner` configured as a workload parameter.
- `gitops.repository_name` during the Out of the Box Supply Chains package installation or `gitops_repository_name` configured as a workload parameter.

With all three values set, Kubernetes configuration is written to the specified repository. If a value is set at installation and the corresponding workload parameter is also set, the value of the workload parameter is respected.

In the repository, files are located in the `./config/{workload-namespace}/{workload-name}` directory. This allows multiple workloads to commit configuration to the same repository.

Examples

`tap-values.yaml`

```
gitops:
  server_address:
  repository_owner:
  repository_name:
```

`workload`

```
  name: incrediApp
  namespace: awesomeTeam
  params:
    - name: gitops_server_address
      value: https://github.com/
    - name: gitops_repository_owner
      value: vmware-tanzu
    - name: gitops_repository_name
      value: cartographer
```

Resulting GitOps repository: `https://github.com/vmware-tanzu/cartographer`

Directory containing configuration: `./config/awesomeTeam/incrediApp`

`tap-values.yaml`:

```
gitops:
  server_address: https://github.com/
  repository_owner: vmware-tanzu
  repository_name: cartographer
```

`workload`:

```
  name: superApp
  namespace: awesomeTeam
```

Resulting GitOps repository: `https://github.com/vmware-tanzu/cartographer`

Directory containing configuration: `./config/awesomeTeam/superApp`

`tap-values.yaml`:

```
gitops:
  server_address: https://github.com/
  repository_owner: vmware-tanzu
```

`workload`:

```
  name: superApp
  namespace: awesomeTeam
  params:
     - name: gitops_repository_owner
       value: buildpacks-community
     - name: gitops_repository_name
       value: kpack
```

Resulting GitOps repository: `https://github.com/buildpacks-community/kpack`

Directory containing configuration: `./config/awesomeTeam/superApp`

`tap-values.yaml`:

```
gitops:
  server_address:
  repository_owner:
  repository_name:
```

`workload`:

```
  name: superApp
  namespace: awesomeTeam
  params:
     - name: gitops_repository_owner
       value: buildpacks-community
     - name: gitops_repository_name
       value: kpack
```

Resulting GitOps repository: Fails to resolve as some, but not all, of the three required values are provided.

Deprecated parameters

The following parameters are deprecated and no longer recommended for specifying GitOps repositories:

- `gitops.repository_prefix`: configured during the Out of the Box Supply Chains package installation.
- `gitops_repository`: configured as a workload parameter.

For example, assuming the installation of the supply chain packages through Tanzu Application Platform profiles and a `tap-values.yaml`:

```
ootb_supply_chain_basic:
  registry:
    server: REGISTRY-SERVER
    repository: REGISTRY-REPOSITORY

  gitops:
    repository_prefix: https://github.com/my-org/
```

Workloads in the cluster with the Kubernetes configuration produced throughout the supply chain are pushed to the repository whose name is formed by concatenating `gitops.repository_prefix` with the name of the workload. In this case, for example, `https://github.com/my-org/$(workload.metadata.name).git`.

```
Supply Chain
  param:
      - gitops_repository_prefix: GIT-REPO_PREFIX


workload-1:
  `git push` to GIT-REPO-PREFIX/workload-1.git

workload-2:
  `git push` to GIT-REPO-PREFIX/workload-2.git

...

workload-n:
  `git push` to GIT-REPO-PREFIX/workload-n.git
```

Alternatively, you can force a workload to publish the configuration in a Git repository by providing the `gitops_repository` parameter to the workload:

```
tanzu apps workload create tanzu-java-web-app \
  --app tanzu-java-web-app \
  --type web \
  --git-repo https://github.com/vmware-tanzu/application-accelerator-samples \
  --sub-path tanzu-java-web-app \
  --git-branch main \
  --param gitops_credentials_secret=GIT-SECRET-NAME \
  --param gitops_repository=https://github.com/my-org/config-repo
```

In this case, at the end of the supply chain, the configuration for this workload is published to the repository provided under the `gitops_repository` parameter.

If you use deprecated parameters, Kubernetes configuration is committed to the `./config` directory in the repository. This can lead to collisions if two workloads specify the same repository, or two workloads in different namespaces have the same name and the `gitops.repository_prefix` is set in `tap-values.yaml`.

If the deprecated values are set and any of the suggested GitOps values are set, the deprecated values are ignored.

Examples

`tap-values.yaml`:

```
gitops:
  repository_prefix: https://github.com/vmware-tanzu
```

`workload`:

```
  name: superApp
  namespace: awesomeTeam
```

Resulting GitOps repository: `https://github.com/vmware-tanzu/incrediApp`

Directory containing configuration: `./config`

`tap-values.yaml`:

```
gitops:
  server_address: https://github.com/
  repository_owner: vmware-tanzu
  repository_name: cartographer
```

`workload`:

```
  name: superApp
  namespace: awesomeTeam
  params:
    - name: gitops_repository
      value: https://github.com/buildpacks-community/kpack
```

Resulting GitOps repository: `https://github.com/vmware-tanzu/cartographer` The deprecated parameter `gitops_repository` is ignored.

Directory containing configuration: `./config/awesomeTeam/superApp`

`tap-values.yaml`:

```
gitops:
  repository_prefix: https://github.com/vmware-tanzu
```

`workload`:

```
  name: superApp
  namespace: awesomeTeam
  params:
    - name: gitops_repository_owner
      value: buildpacks-community
    - name: gitops_repository_name
      value: kpack
```

Resulting GitOps repository: Fails to resolve as some, but not all, of the three GitOps values are provided. The deprecated value `repository_prefix` is ignored because suggested values are present.

Pull requests

In the standard `git-ops` approach, configuration is pushed to a repository and is immediately applied to a cluster by any deliverable watching that repository. Operators might want to manually review configuration before applying it to the cluster. To do this, operators must specify a `pull_request` commit strategy. You can use this strategy with the following Git providers:

- GitHub
- GitLab
- Azure DevOps

Authentication

The pull request approach requires HTTP(S) authentication with a token.

The pull request function is not a part of the Git specification, but most Git server providers include it. You must authenticate with those providers using a token.

In the Kubernetes secret that holds the Git credentials, the password text box must contain a token. When generating a token, ensure that it has the proper scope:

- On GitHub, the token must have a Repo scope.
- On GitLab, the token must have an API scope.

To use the `pull_request` commit strategy, set the following parameters:

- `commit_strategy` == `pull_request` configured during the Out of the Box Supply Chains package installation.
- `gitops.pull_request.server_kind` configured during the Out of the Box Supply Chains package installation or `gitops_server_kind` configured as a workload parameter. Supported values are `github`, `gitlab`, and `azure`.
- `gitops.pull_request.commit_branch` configured during the Out of the Box Supply Chains package installation or `gitops_commit_branch` configured as a workload parameter.
- `gitops.pull_request.pull_request_title` configured during the Out of the Box Supply Chains package installation or `gitops_pull_request_title` configured as a workload parameter.
- `gitops.pull_request.pull_request_body` configured during the Out of the Box Supply Chains package installation or `gitops_pull_request_body` configured as a workload

parameter.

If a value is set at both installation and in a workload parameter, the workload parameter is respected.

The recommended value for `commit_branch` is an empty string. This generates a new branch for each commit based on a hash of the time when the commit is created. This prevents collisions between multiple workloads using a single Git repository.

For example, using the following Tanzu Application Platform values:

```
ootb_supply_chain_basic:
   gitops:
     server_address: https://github.com/
     repository_owner: vmware-tanzu
     repository_name: cartographer
     branch: main
     commit_strategy: pull_request
     pull_request:
       server_kind: github
       commit_branch: ""
       pull_request_title: ready for review
       pull_request_body: generated by supply chain
```

In a workload with the name `app` in the `dev` namespace, you find:

A commit to the `https://github.com/vmware-tanzu/cartographer` repository on a branch with a random name. For example, `MTY1MTYxMzE0NQo=`. There is a pull request open to merge this branch into the base branch `main`.

Authentication

Regardless of how the supply chains are configured, if the repository prefix or repository name is configured to push to Git, you must provide credentials for the remote provider by using a Kubernetes secret in the same namespace as the workload attached to the workload `ServiceAccount`.

Because the operation of pushing requires elevated permissions, credentials are required by both public and private repositories.

HTTP(S) Basic authentication or Token-based authentication

If the repository at which configuration is published uses `https://` or `http://` as the URL scheme, the Kubernetes secret must provide the credentials for that repository as follows:

```
apiVersion: v1
kind: Secret
metadata:
  name: GIT-SECRET-NAME #   - operators should write this name into the
                        #   `gitops.credentials_secret` property in `tap-values.yaml`
                        #   - developers can override by using the workload parameter
                        #     named `gitops_credentials_secret`.
  annotations:
    tekton.dev/git-0: GIT-SERVER        # ! required
type: kubernetes.io/basic-auth          # ! required
stringData:
  username: GIT-USERNAME
  password: GIT-PASSWORD
  # ! Optional, required if the git repository is signed by a certificate authority no
t in the system trust store
  caFile: |
    CADATA-BASE64
```

Both the Tekton annotation and the `basic-auth` secret type must be set. `GIT-SERVER` must be prefixed with the appropriate URL scheme and the Git server. For example, for `https://github.com/vmware-tanzu/cartographer`, `https://github.com` must be provided as the `GIT-SERVER`.

To use the pull request approach, the password text box must contain a token. See Pull Requests.

After the `Secret` is created, attach it to the `ServiceAccount` used by the workload. For example:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: default
secrets:
  - name: registry-credentials
  - name: tap-registry
  - name: GIT-SECRET-NAME
imagePullSecrets:
  - name: registry-credentials
  - name: tap-registry
```

For more information about the credentials and setting up the Kubernetes secret, see Git Authentication's HTTP section.

SSH

If the repository to which configuration is published uses `https://` or `http://` as the URL scheme, the Kubernetes secret must provide the credentials for that repository as follows:

```
apiVersion: v1
kind: Secret
metadata:
  name: GIT-SECRET-NAME #   - operators should write this name into the
                        #   `gitops.credentials_secret` property in `tap-values.yaml`
                        #   - developers can override by using the workload parameter
                        #     named `gitops_credentials_secret`.
  annotations:
    tekton.dev/git-0: GIT-SERVER
type: kubernetes.io/ssh-auth
stringData:
  ssh-privatekey: SSH-PRIVATE-KEY     # private key with push-permissions
  identity: SSH-PRIVATE-KEY           # private key with pull permissions
  identity.pub: SSH-PUBLIC-KEY        # public of the `identity` private key
  known_hosts: GIT-SERVER-PUBLIC-KEYS # git server public keys
```

After the `Secret` is created, attach it to the `ServiceAccount` used by the workload. For example:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: default
secrets:
  - name: registry-credentials
  - name: tap-registry
  - name: GIT-SECRET-NAME
imagePullSecrets:
  - name: registry-credentials
  - name: tap-registry
```

Note

If you've used Namespace Provisioner to set up your Developer Namespace where you workload is created, use the `namespace_provisioner.default_parameters.supply_chain_service_account.secrets` property in your `tap-values.yaml`. For example:

```yaml
namespace_provisioner:
  default_parameters:
    supply_chain_service_account:
      secrets:
      - GIT-SECRET-NAME
```

Namespace Provisioner manages the service account and manual edits to it do not persist.

For information about the credentials and setting up the Kubernetes secret, see Git Authentication's SSH section.

GitOps workload parameters

While installing `ootb-*`, operators can configure `gitops.repository_prefix` to indicate what prefix the supply chain must use when forming the name of the repository to push to the Kubernetes configurations produced by the supply chains.

To change the behavior to use GitOps, set the source of the source code to a Git repository. As the supply chain progresses, configuration is pushed to a repository named `$(gitops.repository_prefix) + $(workload.name)`.

For example, configure `gitops.repository_prefix` to `git@github.com/foo/` and create a workload as follows:

```
tanzu apps workload create tanzu-java-web-app \
  --git-branch main \
  --git-repo https://github.com/vmware-tanzu/application-accelerator-samples \
  --sub-path tanzu-java-web-app \
  --label app.kubernetes.io/part-of=tanzu-java-web-app \
  --type web
```

Expect to see the following output:

```
Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    apps.tanzu.vmware.com/workload-type: web
      7 + |    app.kubernetes.io/part-of: tanzu-java-web-app
      8 + |  name: tanzu-java-web-app
      9 + |  namespace: default
     10 + |spec:
     11 + |  source:
     12 + |    git:
     13 + |      ref:
     14 + |        branch: main
     15 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
     16 + |    subPath: tanzu-java-web-app
```

As a result, the Kubernetes configuration is pushed to `git@github.com/foo/tanzu-java-web-app.git`.

Regardless of the setup, developers can also manually override the repository where configuration is pushed to by tweaking the following parameters:

- `gitops_credentials_secret`: Name of the secret in the same namespace as the workload where SSH credentials exist for pushing the configuration produced by the supply chain to a Git repository. Example: `git-secret`
- `gitops_ssh_secret`: Deprecated: Name of the secret in the same namespace as the workload where SSH credentials exist for pushing the configuration produced by the supply chain to a Git repository. Example: `git-secret`
- `gitops_repository`: SSH URL of the Git repository to push the Kubernetes configuration produced by the supply chain to. Example: `ssh://git@foo.com/staging.git`
- `gitops_branch`: Name of the branch to push the configuration to. Example: `main`
- `gitops_commit_message`: Message to write as the body of the commits produced for pushing configuration to the Git repository. Example: `ci bump`
- `gitops_user_name`: User name to use in the commits. Example: `Alice Lee`
- `gitops_user_email`: User email address to use in the commits. Example: `alice@example.com`

Read more on Git

See Git Reference

RegistryOps

RegistryOps is typically used for inner loop flows where configuration is treated as an artifact from quick iterations by developers. In this scenario, at the end of the supply chain, configuration is pushed to a container image registry in the form of an imgpkg bundle. You can think of it as a container image whose sole purpose is to carry arbitrary files.

To enable this mode of operation, the supply chains must be configured **without** the following parameters being configured during the installation of the `ootb-` packages or overwritten by the workload by using the following parameters:

- `gitops_repository_prefix`
- `gitops_repository`

If none of the parameters are set, the configuration is pushed to the same container image registry as the application image. That is, to the registry configured under the `registry: {}` section of the `ootb-` values.

For example, assuming the installation of Tanzu Application Platform by using profiles, configure the `ootb-supply-chain*` package as follows:

```
ootb_supply_chain_basic:
  registry:
    server: REGISTRY-SERVER
    repository: REGISTRY_REPOSITORY
```

The Kubernetes configuration produced by the supply chain is pushed to an image named after `REGISTRY-SERVER/REGISTRY-REPOSITORY` including the workload name.

In this scenario, no extra credentials must be set up, because the secret containing the credentials for the container image registry were already configured during the setup of the workload namespace.

Overview of Supply Chain Security Tools for VMware Tanzu - Policy Controller

Supply Chain Security Tools - Policy Controller is a security tool that helps you ensure that the container images in their registry have not been tampered with. Policy Controller is a Kubernetes Admission Controller that allows you to apply policies to verify signatures on container images before being admitted to a cluster.

The Policy Controller:

- Verifies signatures on container images used by Kubernetes resources
- Enforces policies to allow or deny images being admitted a cluster
- Allows operators to define multiple policies in the cluster
- Allows operators to select which `namespaces` to enforce policies against
- Supports `cosign` signatures and keyless signing
- Supports storing public keys in a KMS

It enforces its policies against all resources that create `Pod`s as part of their life cycle:

- `Pod`
- `ReplicaSet`
- `Deployment`
- `Job`
- `StatefulSet`
- `DaemonSet`
- `CronJob`

Note

This component is the successor to `Supply Chain Security Tools - Sign`, which is deprecated. Support and maintenance for `Supply Chain Security Tools - Sign` continues. Monitor Release Notes for updates.

Supply Chain Security Tools - Policy Controller is based on Sigstore's Policy Controller and is compatible only with `cosign` signatures. See Cosign and Policy Controller in GitHub. For information about image signing and verification, see Sigstore open source community and the cosign project in GitHub.

The Policy Controller component is a policy enforcement tool only. It does not sign images. Operators can configure image signing for their containers in several ways, including:

- By using Tanzu Build Service
- By using kpack
- By integrating cosign into their build pipelines

Image signatures generated by `cosign` are stored in the same registry location as the image itself unless configured with the `COSIGN_REPOSITORY` environment variable. Policy Controller uses registry credentials provided in the admission request, Service Account, or `signaturePullSecrets` defined in the policy to connect to the registry to verify a signature.

Important

This component does not work with insecure registries.

To Install Supply Chain Security Tools - Policy Controller, see Install Supply Chain Security Tools - Policy Controller

Overview of Supply Chain Security Tools for VMware Tanzu - Policy Controller

Supply Chain Security Tools - Policy Controller is a security tool that helps you ensure that the container images in their registry have not been tampered with. Policy Controller is a Kubernetes Admission Controller that allows you to apply policies to verify signatures on container images before being admitted to a cluster.

The Policy Controller:

- Verifies signatures on container images used by Kubernetes resources
- Enforces policies to allow or deny images being admitted a cluster
- Allows operators to define multiple policies in the cluster
- Allows operators to select which `namespaces` to enforce policies against
- Supports `cosign` signatures and keyless signing
- Supports storing public keys in a KMS

It enforces its policies against all resources that create `Pod`s as part of their life cycle:

- `Pod`
- `ReplicaSet`
- `Deployment`
- `Job`
- `StatefulSet`
- `DaemonSet`
- `CronJob`

Note

This component is the successor to `Supply Chain Security Tools - Sign`, which is deprecated. Support and maintenance for `Supply Chain Security Tools - Sign` continues. Monitor Release Notes for updates.

Supply Chain Security Tools - Policy Controller is based on Sigstore's Policy Controller and is compatible only with `cosign` signatures. See Cosign and Policy Controller in GitHub. For information about image signing and verification, see Sigstore open source community and the cosign project in GitHub.

The Policy Controller component is a policy enforcement tool only. It does not sign images. Operators can configure image signing for their containers in several ways, including:

- By using Tanzu Build Service
- By using kpack
- By integrating cosign into their build pipelines

Image signatures generated by `cosign` are stored in the same registry location as the image itself unless configured with the `COSIGN_REPOSITORY` environment variable. Policy Controller uses registry credentials provided in the admission request, Service Account, or `signaturePullSecrets` defined in the policy to connect to the registry to verify a signature.

Important

This component does not work with insecure registries.

To Install Supply Chain Security Tools - Policy Controller, see Install Supply Chain Security Tools - Policy Controller

Install Supply Chain Security Tools - Policy Controller

You install Supply Chain Security Tools - Policy Controller as part of Tanzu Application Platform's Full, Iterate, and Run profiles. You can use the instructions in this topic to manually install SCST - Policy Controller.

Note

Follow the steps in this topic if you do not want to use a profile to install Supply Chain Security Tools - Policy Controller. For more information about profiles, see Components and installation profiles.

Prerequisites

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- A container image registry that supports TLS connections.

Important

This component does not work with not secure registries.

- For keyless authorities support, you must set `policy.tuf_enabled: true`. By default, the public official Sigstore The Update Framework (TUF) server is used. To target an alternative Sigstore stack, specify `policy.tuf_mirror` and `policy.tuf_root`.
- If you are installing in an air-gapped environment and require keyless authorities, you must deploy a Sigstore Stack on the cluster or be accessible from the air-gapped environment.
- During configuration, you provide a cosign public key to validate signed images. The Policy Controller only supports ECDSA public keys. An example cosign public key is provided that can validate an image from the public cosign registry. To provide your own key and images, follow the Cosign Quick Start Guide in GitHub.

Caution

This component rejects `pods` if they are not correctly configured. Test your configuration in a test environment before applying policies to your production cluster.

Install

To install Supply Chain Security Tools - Policy Controller:

1. List version information for the package by running:

```
tanzu package available list policy.apps.tanzu.vmware.com --namespace tap-insta
ll
```

   For example:

```
$ tanzu package available list policy.apps.tanzu.vmware.com --namespace tap-ins
tall
- Retrieving package versions for policy.apps.tanzu.vmware.com...
  NAME                           VERSION        RELEASED-AT
  policy.apps.tanzu.vmware.com  1.2.0          2023-10-01 20:00:00 -0400 EDT
```

2. (Optional) Make changes to the default installation settings by running:

```
tanzu package available get policy.apps.tanzu.vmware.com/VERSION --values-schem
a --namespace tap-install
```

   Where `VERSION` is the version number you discovered. For example, `1.2.0`.

   For example:

```
$ tanzu package available get policy.apps.tanzu.vmware.com/1.2.0 --values-schem
a --namespace tap-install
| Retrieving package details for policy.apps.tanzu.vmware.com/1.2.0...

KEY                         DEFAULT        TYPE     DESCRIPTION
custom_cas                  <nil>          array    List of custom CA contents t
hat should be included in the application container for registry communication.
                                                    An array of items containing
a ca_content field with the PEM-encoded contents of a certificate authority.
deployment_namespace        cosign-system  string   Deployment namespace specifi
es the namespace where this component should be deployed to.
                                                    If not specified, "cosign-sy
stem" is assumed.
fail_on_empty_authorities  true           boolean  Configure if a ClusterImageP
olicy will fail or allow empty authorities
limits_cpu                  200m           string   The CPU limit defines a hard
ceiling on how much CPU time
                                                    that the Policy Controller m
anager container can use.
                                                    https://kubernetes.io/docs/c
oncepts/configuration/manage-resources-containers/#meaning-of-cpu

no_match_policy             deny           string   The action when no policy ma
tches the admitting image digest. Valid values are "warn", "allow", or "deny".
quota.pod_number            6              string   The maximum number of Policy
Controller Pods allowed to be created with the priority class
                                                    system-cluster-critical. Thi
s value must be enclosed in quotes (""). If this value is not
                                                    specified then a default val
ue of 6 is used.
replicas                    1              integer  The number of replicas to be
created for the Policy Controller. This value must not be enclosed
                                                    in quotes. If this value is
not specified then a default value of 1 is used.
requests_memory             20Mi           string   The memory request defines t
he minium memory amount for the Policy Controller manager.
                                                    https://kubernetes.io/docs/c
oncepts/configuration/manage-resources-containers/#meaning-of-memory
```

```
tuf_root                                    string   The root.json file content o
f the TUF mirror

custom_ca_secrets          <nil>            array    List of custom CA secrets th
at should be included in the application container for registry communication.
                                                    An array of secret reference
s each containing a secret_name field with the secret name to be referenced
                                                    and a namespace field with t
he name of the namespace where the referred secret resides.
limits_memory              200Mi            string   The memory limit defines a h
ard ceiling on how much memory
                                                    that the Policy Controller m
anager container can use.
                                                    https://kubernetes.io/docs/c
oncepts/configuration/manage-resources-containers/#meaning-of-memory

requests_cpu               20m              string   The CPU request defines the
minimum CPU time for the Policy
                                                    Controller manager. During C
PU contention, CPU request is used as
                                                    a weighting where higher CPU
requests are allocated more CPU time.
                                                    https://kubernetes.io/docs/c
oncepts/configuration/manage-resources-containers/#meaning-of-cpu

tuf_mirror                                  string   TUF mirror address
```

3. Create a file named `scst-policy-values.yaml` and add the settings you want to customize:

    - `custom_ca_secrets`: If your container registries are secured by self-signed certificates, this setting controls which secrets are added to the application container as custom certificate authorities (CAs). `custom_ca_secrets` consists of an array of items. Each item contains two text boxes: the `secret_name` text box defines the name of the secret, and the `namespace` text box defines the name of the namespace where said secret is stored.

      For example:

      ```
      custom_ca_secrets:
      - secret_name: first-ca
        namespace: ca-namespace
      - secret_name: second-ca
        namespace: ca-namespace
      ```

      Note

      This setting is allowed even if `custom_cas` is defined.

    - `custom_cas`: This setting enables adding certificate content in PEM format. The certificate content is added to the application container as custom certificate authorities (CAs) to communicate with registries deployed with self-signed certificates. `custom_cas` consists of an array of items. Each item contains a single text box named `ca_content`. The value of this text box must be a PEM-formatted certificate authority. The certificate content must be defined as a YAML block, preceded by the literal indicator (`|`) to preserve line breaks and ensure that the certificates are interpreted correctly.

      For example:

```
custom_cas:
- ca_content: |
    ----- BEGIN CERTIFICATE -----
    first certificate content here...
    ----- END CERTIFICATE -----
- ca_content: |
    ----- BEGIN CERTIFICATE -----
    second certificate content here...
    ----- END CERTIFICATE -----
```

> **Note**
>
> This setting is allowed even if `custom_ca_secrets` is defined.

- `deployment_namespace`: This setting controls the namespace to which this component is deployed. When not specified, the namespace `cosign-system` is assumed. This component creates the specified namespace to deploy required resources. Select a namespace that is not used by any other components.
- `limits_cpu`: This setting controls the maximum CPU resource allocated to the Policy admission controller. The default value is “200m”. See Kubernetes documentation.
- `limits_memory`: This setting controls the maximum memory resource allocated to the Policy admission controller. The default value is “200Mi”. See Kubernetes documentation.
- `quota.pod_number`: This setting controls the maximum number of pods that are allowed in the deployment namespace with the `system-cluster-critical` priority class. This priority class is added to the pods to prevent preemption of this component’s pods in case of node pressure.

  The default value for this text box is `6`. If your use requires more than 6 pods, change this value to allow the number of replicas you intend to deploy.

  > **Note**
  >
  > VMware recommends to run this component with a critical priority level to prevent the cluster from rejecting all admission requests if the component’s `pod`s are evicted due to resource limits.

- `replicas`: This setting controls the default amount of replicas deployed by this component. The default value is `1`.

  For production environments: VMware recommends you increase the number of replicas to `3` to ensure that the availability of the component and better admission performance.
- `requests_cpu`: This setting controls the minimum CPU resource allocated to the Policy admission controller. During CPU contention, this value is used as a weighting where higher values indicate more CPU time is allocated. The default value is `20m`. See CPU resource units in the Kubernetes documentation.
- `requests_memory`: This setting controls the minimum memory resource allocated to the Policy admission controller. The default value is `20Mi`. See Memory resource units in the Kubernetes documentation.
- `tuf_enabled`: This setting defines whether the TUF initialization is done on startup. It is required for keyless verification support. The default value is `false`, which means

that keyless authorities of `ClusterImagePolicy` are not supported. Also, policy-controller does not have an external dependency on setup.

- `tuf_root`: The root.json file content of the TUF mirror.
- `tuf_mirror`: This setting defines the TUF mirror address which is used for doing the initialization.
- `no_match_policy`: The action when no policy matches the admitting image digest. Valid values are `"warn"`, `"allow"`, or `"deny"`. Default value is `"deny"`
- `fail_on_empty_authorities`: Failing or allowing empty authorities when adding a new `ClusterImagePolicy`. Default value is `true`.

4. Install the package:

```
tanzu package install policy-controller \
  --package policy.apps.tanzu.vmware.com \
  --version VERSION \
  --namespace tap-install \
  --values-file scst-policy-values.yaml
```

Where `VERSION` is the version number you discovered earlier. For example, `1.2.0`.

For example:

```
$ tanzu package install policy-controller \
    --package policy.apps.tanzu.vmware.com \
    --version 1.2.0 \
    --namespace tap-install \
    --values-file scst-policy-values.yaml

  Installing package 'policy.apps.tanzu.vmware.com'
  Getting package metadata for 'policy.apps.tanzu.vmware.com'
  Creating service account 'policy-controller-tap-install-sa'
  Creating cluster admin role 'policy-controller-tap-install-cluster-role'
  Creating cluster role binding 'policy-controller-tap-install-cluster-rolebind
ing'
  Creating package resource
  Waiting for 'PackageInstall' reconciliation for 'policy-controller'
  'PackageInstall' resource install status: Reconciling
  'PackageInstall' resource install status: ReconcileSucceeded
  'PackageInstall' resource successfully reconciled

  Added installed package 'policy-controller'
```

After you run the commands earlier the policy controller is running.

Policy Controller is now installed, but it does not enforce any policies by default. Policies must be explicitly configured on the cluster. To configure signature verification policies, see Configuring Supply Chain Security Tools - Policy.

Configure Supply Chain Security Tools - Policy

This topic describes how you can configure Supply Chain Security Tools - Policy. SCST - Policy requires extra configuration steps to verify your container images.

Admission of Images

An image is admitted after it is validated against a policy with matching image pattern, and where at least one valid signature is obtained from the authorities provided in a matched ClusterImagePolicy.

If more than one policy exists with a matching image pattern, *ALL* of the policies must have at least one passing authority for the image.

Including Namespaces

The Policy Controller only validates resources in namespaces that have chosen to opt-in. This is done by adding the label `policy.sigstore.dev/include: "true"` to the namespace resource.

```
kubectl label namespace my-secure-namespace policy.sigstore.dev/include=true
```


Caution

Without a Policy Controller ClusterImagePolicy applied, there are fallback behaviors where images are validated against the public Sigstore Rekor and Fulcio servers by using a keyless authority flow. Therefore, if the deploying image is signed publicly by a third-party using the keyless authority flow, the image is admitted as it can validate against the public Rekor and Fulcio. To avoid this behavior, develop, and apply a ClusterImagePolicy that applies to the images being deployed in the namespace.

Create a `ClusterImagePolicy` resource

The cluster image policy is a custom resource containing the following properties:

`images`

In a ClusterImagePolicy, `spec.images` specifies a list of glob matching patterns. These patterns are matched against the image digest in `PodSpec` for resources attempting deployment.

Policy Controller defines the following globs by default:

- If `*` is specified, the `glob` matching behavior is `index.docker.io/library/*`.
- If `*/*` is specified, the `glob` matching behavior is `index.docker.io/*/*`.

With these defaults, you require the `glob` pattern `**` to match against all images. If your image is hosted on Docker Hub, include `index.docker.io` as the host for the glob.

A sample ClusterImagePolicy which matches against all images using glob:

```
apiVersion: policy.sigstore.dev/v1beta1
kind: ClusterImagePolicy
metadata:
  name: image-policy
spec:
  images:
  - glob: "**"
```

`mode`

In a ClusterImagePolicy, `spec.mode` specifies the action of a policy:

- `enforce`: The default behavior. If the policy fails to validate the image, the policy fails.
- `warn`: If the policy fails to validate the image, validation error messages are converted to warnings and the policy passes.

A sample of a ClusterImagePolicy which has `warn` mode configured.

```
---
apiVersion: policy.sigstore.dev/v1beta1
kind: ClusterImagePolicy
metadata:
  name: POLICY-NAME
spec:
  mode: warn
```

Where `POLICY-NAME` is the name of the policy you want to configure your ClusterImagePolicy with.

When `enforce` mode is set, an image that fails validation is not admitted.

Sample output message:

```
error: failed to patch: admission webhook "policy.sigstore.dev" denied the request: va
lidation failed: failed policy: POLICY-NAME: spec.template.spec.containers[0].image
IMAGE-REFERENCE signature key validation failed for authority authority-0 for IMAGE-RE
FERENCE: GET IMAGE-SIGNATURE-REFERENCE: DENIED: denied; denied
failed policy: POLICY-NAME: spec.template.spec.containers[1].image
IMAGE-REFERENCE signature key validation failed for authority authority-0 for IMAGE-RE
FERENCE: GET IMAGE-SIGNATURE-REFERENCE: DENIED: denied; denied
```

When `warn` mode is set, an image that fails validation is admitted.

Sample output message:

```
Warning: failed policy: POLICY-NAME: spec.template.spec.containers[0].image
Warning: IMAGE-REFERENCE signature key validation failed for authority authority-0 for
IMAGE-REFERENCE: GET IMAGE-SIGNATURE-REFERENCE: DENIED: denied; denied
Warning: failed policy: POLICY-NAME: spec.template.spec.containers[1].image
Warning: IMAGE-REFERENCE signature key validation failed for authority authority-0 for
IMAGE-REFERENCE: GET IMAGE-SIGNATURE-REFERENCE: DENIED: denied; denied
```

If you don't want a `Warning` output message, you can configure a `static.action pass` authority to allow expected unsigned images. For example, you may want to allow unsigned images if your policy controller runs on a development environment, and you need to iterate quickly. For information about static action authorities, see Static Action.

match

You can use `match` to filter resources using group, version, kind, or labels in a selected namespace to enforce the defined policy. If the list of matching resources is empty, all core resources are used by default.

For example, you can filter all `v1 cronjobs` with the label `app: tap` in a namespace that is labeled for policy enforcement:

```
spec:
  match:
  - group: batch
    resource: cronjobs
    version: v1
    selector:
      matchLabels:
        app: tap
```

authorities

Authorities listed in the `authorities` block of the ClusterImagePolicy are `key` or `keyless` specifications.

`key`

Each `key` authority can contain a PEM-encoded ECDSA public key, a `secretRef`, or a `kms` path.

The policy resyncs with KMS referenced every 10 hours. Any updates to the secret in KMS is pulled in during the refresh. To force a resync, the policy must be deleted and recreated.

Important

Only ECDSA public keys are supported.

```
spec:
  authorities:
    - key:
        data: |
          -----BEGIN PUBLIC KEY-----
          ...
          -----END PUBLIC KEY-----
    - key:
        secretRef:
          name: secretName
    - key:
        kms: KMSPATH
```

Where `KMSPATH` is the name of the KMS path you want to configure in your key authority.

Note

The secret referenced in `key.secretRef.name` must be created in the `cosign-system` namespace or the namespace where the Policy Controller is installed. This secret must only contain one `data` entry with the public key.

`keyless`

Note

Keyless support is deactivated by default. See Install Supply Chain Security Tools - Policy Controller.

Each keyless authority can contain a Fulcio URL, a Rekor URL, a certificate, or an array of identities.

Identities are represented with a combination of `issuer` or `issuerRegExp` with `subject` or `subjectRegExp`.

- `issuer`: Defines the issuer for this identity.
- `issuerRegExp`: Specifies a regular expression to match the issuer for this identity.
- `subject`: Defines the subject for this identity.
- `subjectRegExp`: Specifies a regular expression to match the subject for this identity.

An example of keyless authority structure:

```
spec:
  authorities:
    - keyless:
        url: https://fulcio.example.com
```

```
        ca-cert:
          data: Certificate Data
        identities:
          - issuer: https://accounts.google.com
            subjectRegExp: .*@example.com
          - issuer: https://token.actions.githubusercontent.com
            subject: https://github.com/mycompany/*/.github/workflows/*@*
      ctlog:
        url: https://rekor.example.com
    - keyless:
        url: https://fulcio.example.com
        ca-cert:
          secretRef:
            name: secretName
        identities:
          - issuerRegExp: .*kubernetes.default.*
            subjectRegExp: .*kubernetes.io/namespaces/default/serviceaccounts/default
```

The authorities are evaluated using the `any of` operator to admit container images. For each pod, the Policy Controller iterates over the list of containers and init containers. For every policy that matches against the images, they must each have at least one valid signature obtained using the authorities specified. If an image does not match any policy, the Policy Controller does not admit the image.

`static.action`

ClusterImagePolicy authorities are configured to always `pass` or `fail` with `static.action`.

Sample `ClusterImagePolicy` with static action `fail`.

```
apiVersion: policy.sigstore.dev/v1beta1
kind: ClusterImagePolicy
metadata:
  name: POLICY-NAME
spec:
  authorities:
  - static:
      action: fail
```

Where `POLICY-NAME` is the name of the policy you want to configure your ClusterImagePolicy with.

A sample output of static action `fail`:

```
error: failed to patch: admission webhook "policy.sigstore.dev" denied the request: va
lidation failed: failed policy: POLICY-NAME: spec.template.spec.containers[0].image
IMAGE-REFERENCE disallowed by static policy
failed policy: POLICY-NAME: spec.template.spec.containers[1].image
IMAGE-REFERENCE disallowed by static policy
```

Images that are unsigned in a namespace with validation enabled are admitted with an authority with static action `pass`.

This applies when you are configuring a policy with `static.action pass` for `tap-packages` images. Another policy is then configured to validate signed images produced by Tanzu Build Service. This allows images from `tap-packages`, which are unsigned and required by the platform, to be admitted while still validating signed built images from Tanzu Build Service. See Configure your supply chain to sign and verify your image builds.

If `Warning` messages are desirable for admitted images where validation failed, you can configure a policy with `warn` mode and valid authorities. For information about ClusterImagePolicy modes, see Mode.

Provide credentials for the package

There are three ways the package reads credentials to authenticate to registries protected by authentication:

1. Reading `imagePullSecrets` directly from the resource being admitted. See Container image pull secrets in the Kubernetes documentation.
2. Reading `imagePullSecrets` from the service account the resource is running as. See Arranging for imagePullSecrets to be automatically attached in the Kubernetes documentation.
3. Reading a `secretRef` from the `ClusterImagePolicy` resource's `signaturePullSecrets` when specifying the cosign signature source.

Authentication can fail for the following scenarios:

- A not valid credential is specified in the `imagePullSecrets` of the resource or in the service account the resource runs as.
- A not valid credential is specified in the `ClusterImagePolicy` `signaturePullSecrets` text box.

Provide secrets for authentication in your policy

You can provide secrets for authentication as part of the policy configuration. The `oci` location is the image location or a remote location where signatures are configured to be stored during signing. The `signaturePullSecrets` is available in the `cosign-system` namespace or the namespace where the Policy Controller is installed.

By default, `imagePullSecrets` from the resource or service account is used while the default `oci` location is the image location.

See the following example:

```
spec:
  authorities:
    - key:
        data: |
          -----BEGIN PUBLIC KEY-----
          ...
          -----END PUBLIC KEY-----
      source:
        - oci: registry.example.com/project/signature-location
          signaturePullSecrets:
            - name: MY-SECRET
    - keyless:
        url: https://fulcio.example.com
      source:
        - oci: registry.example.com/project/signature-location
          signaturePullSecrets:
            - name: MY-SECRET
```

Where `MY-SECRET` is the name of the secret you want to use with your credentials.

VMware recommends using a set of credentials with the least amount of privilege that allows reading the signature stored in your registry.

Verify your configuration

A sample policy:

```
apiVersion: policy.sigstore.dev/v1beta1
kind: ClusterImagePolicy
metadata:
  name: image-policy
spec:
  images:
  - glob: "gcr.io/projectsigstore/cosign*"
  authorities:
  - name: official-cosign-key
    key:
      data: |
        -----BEGIN PUBLIC KEY-----
        MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEhyQCx0E9wQWSFI9ULGwy3BuRklnt
        IqozONbbdbqz11hlRJy9c7SG+hdcFl9jE9uE/dwtuwU2MqU9T/cN0YkWww==
        -----END PUBLIC KEY-----
```

When using the sample policy, run these commands to verify your configuration:

1. Verify that the Policy Controller admits the signed image that validates with the configured public key. Run:

   ```
   kubectl run cosign \
     --image=gcr.io/projectsigstore/cosign:v1.2.1 \
     --dry-run=server
   ```

   For example:

   ```
   $ kubectl run cosign \
     --image=gcr.io/projectsigstore/cosign:v1.2.1 \
     --dry-run=server
   pod/cosign created (server dry run)
   ```

   If you are using vSphere with Tanzu or OpenShift, you must add some overrides:

   ```
   $ kubectl run cosign \
     --image=gcr.io/projectsigstore/cosign:v1.2.1 \
     --overrides='{"spec": {"securityContext": {"seccompProfile": {"type": "Runtim
   eDefault"}}, "containers": [{"name": "cosign", "securityContext": {"allowPrivil
   egeEscalation": false, "runAsNonRoot": true, "capabilities": {"drop": ["AL
   L"]}}}]}}' \
     --override-type strategic \
     --dry-run=server
   pod/cosign created (server dry run)
   ```

2. Verify that the Policy Controller rejects the unmatched image. Run:

   ```
   kubectl run busybox --image=busybox --dry-run=server
   ```

   For example:

   ```
   $ kubectl run busybox --image=busybox --dry-run=server
     Error from server (BadRequest): admission webhook "policy.sigstore.dev" denie
   d the request: validation failed: no matching policies: spec.containers[0].imag
   e
     index.docker.io/library/busybox@sha256:3614ca5eacf0a3a1bcc361c939202a974b4902
   b9334ff36eb29ffe9011aaad83
   ```

   In the output, it did not specify which authorities were used as there was no policy found that matched the image. Therefore, the image fails to validate for a signature and fails to deploy.

3. Verify that the Policy Controller rejects a matched image signed with a different key than the one configured. Run:

```
kubectl run cosign-fail \
  --image=gcr.io/projectsigstore/cosign:v0.3.0 \
  --dry-run=server
```

For example:

```
$ kubectl run cosign-fail \
    --image=gcr.io/projectsigstore/cosign:v0.3.0 \
    --dry-run=server
  Error from server (BadRequest): admission webhook "policy.sigstore.dev" denie
d the request: validation failed: failed policy: image-policy: spec.containers
[0].image
  gcr.io/projectsigstore/cosign@sha256:135d8c5e27bdc917f04b415fc947d7d5b1137f99
bb8fa00bffc3eca1856e9c52 failed to validate public keys with authority official
-cosign-key for gcr.io/projectsigstore/cosign@sha256:135d8c5e27bdc917f04b415fc9
47d7d5b1137f99bb8fa00bffc3eca1856e9c52: no matching signatures:
```

In the output, it specifies which authorities were used for validation when a policy was found that matched the image. In this case, the authority used was `official-cosign-key`. If no name is specified, it is defaulted to `authority-#`.

Overview of Supply Chain Security Tools - Scan

This topic gives you an overview of use cases, features, and CVEs for Supply Chain Security Tools (SCST) - Scan.

Overview

With Supply Chain Security Tools (SCST) - Scan, you can build and deploy secure, trusted software that complies with your corporate security requirements. SCST - Scan provides scanning and gatekeeping capabilities that Application and DevSecOps teams can incorporate early in their path to production. This is the best practice for reducing security risk and ensuring more efficient remediation.

Language support

For information about the languages and frameworks that are supported by Tanzu Application Platform components, see the Language and framework support in Tanzu Application Platform table.

Use cases

The following use cases apply to SCST - Scan:

- Scan source code repositories and images for known Common Vulnerabilities and Exposures (CVEs) as part of your software supply chain at build time.
- Scan container images produced by your supply chain and any running image in your Tanzu Application Platform clusters for newly reported vulnerabilities after the initial image build scan.
- Use one of the available scan integrations or create your own to use your existing vulnerability scanning platforms.
- Analyze supply chain scan results against user-defined policies.

- Store vulnerability scan results in SCST - Store for long-term archival and reporting.

SCST - Scan versions

There are two versions of SCST - Scan:

SCST - Scan 1.0

SCST - Scan 1.0 has been in the testing and scanning supply chain since it was introduced with Tanzu Application Platform v1.0. SCST - Scan 1.0 can scan a workload for vulnerabilities, submit the scan results to SCST - Store for long-term storage and reporting, and compare the results against a policy defined by the user. This is all included in a tightly coupled scan job that is executed as part of the testing and scanning supply chain.

This tight coupling of the capabilities made it difficult to develop and maintain integrations for the vast ecosystem of vulnerability scanning platforms. To simplify this integration process, VMware introduced SCST - Scan 2.0.

Although other scan integrations are available, the default configuration for SCST - Scan 1.0 is the open-source Anchore Grype.

SCST - Scan 2.0

SCST - Scan 2.0 was introduced in the Tanzu Application Platform v1.5 release as an Alpha and is now GA in Tanzu Application Platform v1.8. This iteration of SCST - Scan focuses on simplifying the integration experience by decoupling SCST - Store submission and policy from the scanning task. This allows integration to be simplified and more focused on the task of scanning workloads for vulnerabilities.

SCST - Scan 2.0 can scan container images after the initial creation of the workload. This allows you to have visibility in the security posture of images as new vulnerabilities are reported. For more information, see Recurring Scanning. This capability can be used with Scan 1.0 or Scan 2.0.

SCST - Scan 2.0 includes the default configuration to use open-source Aqua Security Trivy as the image scanner, and a Grype template is also included for backward compatibility. Examples of other integrations, and how to build your own integration with this simplified interface, are provided in documentation.

Determine which version to use

In Tanzu Application Platform v1.8, both SCST - Scan v1.0 and SCST - Scan v2.0 are supported. In a future release, SCST - Scan v1.0 will be deprecated and replaced with SCST - Scan v2.0. To help facilitate this transition, VMware is slowly making SCST - Scan 2.0 the default component. The default version varies depending on the environment Tanzu Application Platform is installed on:

Installation Environment	Default Component	Detail
Online Installation	SCST - Scan v1.0	SCST - Scan v1.0 remains the default with the option to opt in to Scan 2.0.
Offline Installation	SCST - Scan v2.0	SCST - Scan v2.0 is the default due to a simplified air-gapped experience with Trivy.
Azure Installation	SCST - Scan v1.0	SCST - Scan v1.0 remains the default with the option to opt in to Scan v2.0.
AWS Installation	SCST - Scan v1.0	SCST - Scan v1.0 remains the default with the option to opt in to SCST - Scan v2.0.

If you require a policy to block a workload in a supply chain based on detected vulnerabilities, use SCST - Scan v1.0.

If you want to create a scan integration for a scan tool that does not exist, use SCST - Scan v2.0 as the process is greatly simplified. For more information, see Bring your own scanner with Supply Chain Security Tools - Scan 2.0.

If you are using the Tanzu Supply Chain component, use SCST - Scan v2.0 as only SCST - Scan v2.0 is supported with Tanzu Supply Chain. For more information, see Overview of Tanzu Supply Chain.

Vulnerability Scanner limitations

Although vulnerability scanning is an important practice in DevSecOps and the benefits of it are widely recognized and accepted, some limits impact its efficacy. The following examples illustrate the limits that are prevalent in most scanners today:

Missed CVEs

One limit of all vulnerability scanners is that no one tool can find all CVEs, which means there is a risk that a missed CVE could be exploited. Some reasons for missed CVEs include:

- The scanner does not detect the vulnerability because it is a recently discovered vulnerability and the CVE databases are not updated yet.
- The scanner verifies different CVE sources based on the detected package type and OS.
- The scanner might not fully support a particular programming language, packaging system, or manifest format.
- The scanner might not implement binary analysis or fingerprinting.
- The detected component does not always include a canonical name and vendor, requiring the scanner to infer and attempt fuzzy matching.
- When vendors register impacted software with NVD, the provided information might not exactly match the values in the release artifacts.

False positives

Vulnerability scanners cannot always access the information to accurately identify whether a CVE exists. This often leads to an influx of false positives where the tool mistakenly flags something as a vulnerability. Unless you are specialized in security or are very familiar with a vulnerable component, assessing, and determining false positives is a challenging and time-consuming activity. Some reasons for a false positive flag include:

- A component is misidentified due to similar names.
- A sub-component is identified as the parent component.
- A component is correctly identified but the impacted function is not on a reachable code path.
- A component's impacted function is on a reachable code path but is not a concern due to the specific environment or configuration.
- The version of a component might be incorrectly flagged as impacted.
- The detected component does not always include a canonical name and vendor, requiring the scanner to infer and attempt fuzzy matching.

Mitigation measures

Although vulnerability scanning is not a perfect solution, it is an essential part of the process for keeping your organization secure. To maximize the benefits while minimizing the impact of the limits of vulnerability scanning, take the following steps to scan more continuously and comprehensively to identify and remediate zero-day vulnerabilities quickly:

- Scan early in the development cycle to ensure that you can address issues more efficiently. Tanzu Application Platform includes security practices such as source and container image vulnerability scanning earlier in the path to production for application teams.
- Scan any base images in use. Tanzu Application Platform image scanning can recognize and scan the OS packages from a base image.
- Scan running software in test, stage, and production environments at a regular cadence.
- Generate accurate provenance at any level so that scanners have a complete picture of the dependencies to scan. This is where a software bill of materials (SBoM) is used. To help you automate this process, VMware Tanzu Build Service, leveraging Cloud Native Buildpacks, generates an SBoM for buildpack-based projects. Because this SBoM is generated during the image-building stage, it is more accurate and complete than one generated earlier or later in the release life cycle. This is because it can highlight dependencies introduced at the time of build that might introduce the potential for compromise.
- Scan by using multiple scanners to maximize CVE coverage.
- Keep your dependencies up-to-date.
- To reduce the overall surface area of attack use smaller dependencies.
- Reduce the amount of third-party dependencies when possible.
- Use distroless base images when possible.
- Maintain a central record of false positives to ease CVE triaging and remediation efforts.

Overview of Supply Chain Security Tools - Scan

This topic gives you an overview of use cases, features, and CVEs for Supply Chain Security Tools (SCST) - Scan.

Overview

With Supply Chain Security Tools (SCST) - Scan, you can build and deploy secure, trusted software that complies with your corporate security requirements. SCST - Scan provides scanning and gatekeeping capabilities that Application and DevSecOps teams can incorporate early in their path to production. This is the best practice for reducing security risk and ensuring more efficient remediation.

Language support

For information about the languages and frameworks that are supported by Tanzu Application Platform components, see the Language and framework support in Tanzu Application Platform table.

Use cases

The following use cases apply to SCST - Scan:

- Scan source code repositories and images for known Common Vulnerabilities and Exposures (CVEs) as part of your software supply chain at build time.

- Scan container images produced by your supply chain and any running image in your Tanzu Application Platform clusters for newly reported vulnerabilities after the initial image build scan.
- Use one of the available scan integrations or create your own to use your existing vulnerability scanning platforms.
- Analyze supply chain scan results against user-defined policies.
- Store vulnerability scan results in SCST - Store for long-term archival and reporting.

SCST - Scan versions

There are two versions of SCST - Scan:

SCST - Scan 1.0

SCST - Scan 1.0 has been in the testing and scanning supply chain since it was introduced with Tanzu Application Platform v1.0. SCST - Scan 1.0 can scan a workload for vulnerabilities, submit the scan results to SCST - Store for long-term storage and reporting, and compare the results against a policy defined by the user. This is all included in a tightly coupled scan job that is executed as part of the testing and scanning supply chain.

This tight coupling of the capabilities made it difficult to develop and maintain integrations for the vast ecosystem of vulnerability scanning platforms. To simplify this integration process, VMware introduced SCST - Scan 2.0.

Although other scan integrations are available, the default configuration for SCST - Scan 1.0 is the open-source Anchore Grype.

SCST - Scan 2.0

SCST - Scan 2.0 was introduced in the Tanzu Application Platform v1.5 release as an Alpha and is now GA in Tanzu Application Platform v1.8. This iteration of SCST - Scan focuses on simplifying the integration experience by decoupling SCST - Store submission and policy from the scanning task. This allows integration to be simplified and more focused on the task of scanning workloads for vulnerabilities.

SCST - Scan 2.0 can scan container images after the initial creation of the workload. This allows you to have visibility in the security posture of images as new vulnerabilities are reported. For more information, see Recurring Scanning. This capability can be used with Scan 1.0 or Scan 2.0.

SCST - Scan 2.0 includes the default configuration to use open-source Aqua Security Trivy as the image scanner, and a Grype template is also included for backward compatibility. Examples of other integrations, and how to build your own integration with this simplified interface, are provided in documentation.

Determine which version to use

In Tanzu Application Platform v1.8, both SCST - Scan v1.0 and SCST - Scan v2.0 are supported. In a future release, SCST - Scan v1.0 will be deprecated and replaced with SCST - Scan v2.0. To help facilitate this transition, VMware is slowly making SCST - Scan 2.0 the default component. The default version varies depending on the environment Tanzu Application Platform is installed on:

Installation Environment	Default Component	Detail
Online Installation	SCST - Scan v1.0	SCST - Scan v1.0 remains the default with the option to opt in to Scan 2.0.

Installation Environment	Default Component	Detail
Offline Installation	SCST - Scan v2.0	SCST - Scan v2.0 is the default due to a simplified air-gapped experience with Trivy.
Azure Installation	SCST - Scan v1.0	SCST - Scan v1.0 remains the default with the option to opt in to Scan v2.0.
AWS Installation	SCST - Scan v1.0	SCST - Scan v1.0 remains the default with the option to opt in to SCST - Scan v2.0.

If you require a policy to block a workload in a supply chain based on detected vulnerabilities, use SCST - Scan v1.0.

If you want to create a scan integration for a scan tool that does not exist, use SCST - Scan v2.0 as the process is greatly simplified. For more information, see Bring your own scanner with Supply Chain Security Tools - Scan 2.0.

If you are using the Tanzu Supply Chain component, use SCST - Scan v2.0 as only SCST - Scan v2.0 is supported with Tanzu Supply Chain. For more information, see Overview of Tanzu Supply Chain.

Vulnerability Scanner limitations

Although vulnerability scanning is an important practice in DevSecOps and the benefits of it are widely recognized and accepted, some limits impact its efficacy. The following examples illustrate the limits that are prevalent in most scanners today:

Missed CVEs

One limit of all vulnerability scanners is that no one tool can find all CVEs, which means there is a risk that a missed CVE could be exploited. Some reasons for missed CVEs include:

- The scanner does not detect the vulnerability because it is a recently discovered vulnerability and the CVE databases are not updated yet.
- The scanner verifies different CVE sources based on the detected package type and OS.
- The scanner might not fully support a particular programming language, packaging system, or manifest format.
- The scanner might not implement binary analysis or fingerprinting.
- The detected component does not always include a canonical name and vendor, requiring the scanner to infer and attempt fuzzy matching.
- When vendors register impacted software with NVD, the provided information might not exactly match the values in the release artifacts.

False positives

Vulnerability scanners cannot always access the information to accurately identify whether a CVE exists. This often leads to an influx of false positives where the tool mistakenly flags something as a vulnerability. Unless you are specialized in security or are very familiar with a vulnerable component, assessing, and determining false positives is a challenging and time-consuming activity. Some reasons for a false positive flag include:

- A component is misidentified due to similar names.
- A sub-component is identified as the parent component.

- A component is correctly identified but the impacted function is not on a reachable code path.
- A component's impacted function is on a reachable code path but is not a concern due to the specific environment or configuration.
- The version of a component might be incorrectly flagged as impacted.
- The detected component does not always include a canonical name and vendor, requiring the scanner to infer and attempt fuzzy matching.

Mitigation measures

Although vulnerability scanning is not a perfect solution, it is an essential part of the process for keeping your organization secure. To maximize the benefits while minimizing the impact of the limits of vulnerability scanning, take the following steps to scan more continuously and comprehensively to identify and remediate zero-day vulnerabilities quickly:

- Scan early in the development cycle to ensure that you can address issues more efficiently. Tanzu Application Platform includes security practices such as source and container image vulnerability scanning earlier in the path to production for application teams.
- Scan any base images in use. Tanzu Application Platform image scanning can recognize and scan the OS packages from a base image.
- Scan running software in test, stage, and production environments at a regular cadence.
- Generate accurate provenance at any level so that scanners have a complete picture of the dependencies to scan. This is where a software bill of materials (SBoM) is used. To help you automate this process, VMware Tanzu Build Service, leveraging Cloud Native Buildpacks, generates an SBoM for buildpack-based projects. Because this SBoM is generated during the image-building stage, it is more accurate and complete than one generated earlier or later in the release life cycle. This is because it can highlight dependencies introduced at the time of build that might introduce the potential for compromise.
- Scan by using multiple scanners to maximize CVE coverage.
- Keep your dependencies up-to-date.
- To reduce the overall surface area of attack use smaller dependencies.
- Reduce the amount of third-party dependencies when possible.
- Use distroless base images when possible.
- Maintain a central record of false positives to ease CVE triaging and remediation efforts.

Scan Types for Supply Chain Security Tools - Scan

This topic tells you what scan types you can use with Scan Types for Supply Chain Security Tools (SCST) - Scan. The out-of-box test and scan supply chain supports the source and container image scan types.

Source scan

The source scan step in the test and scan supply chain performs a Software Composition Analysis (SCA) scan to inspect the open source dependencies of an application for vulnerabilities. You perform this by inspecting the file that the language uses for dependency declaration. For example:

Language	Dependency File
Spring	`pom.xml`

Language	Dependency File
.Net	`deps.json`
Node.JS	`packages.json`
Python	`requirements.txt`

Rather than declare specific dependency versions, some languages such as Spring, Java, and .Net resolve dependency versions at build time. For these languages, performing a SCA scan on the declaration file stored in the source code does not produce meaningful results, often creating false positives or false negatives.

Due to this, in Tanzu Application Platform v1.6, the source scan step is moved to an opt-in step in the supply chain.

Adding Source Scan to the Test and Scan Supply Chain

To add source scanning to the out-of-the-box test and scan supply chain, you can apply an overlay in the install `tap-values.yaml` file. This overlay adds the required resources to the supply chain in the correct location to opt-in to source scanning.

For information about how overlays work with Tanzu Application Platform, see Customize your package installation.

To add source scanning to the default out-of-the-box test and scan supply chain:

1. Create a `secret.yml` file with a `Secret` that contains your ytt overlay. For example:

```
apiVersion: v1
kind: Secret
metadata:
 name: ootb-supply-chain-testing-scanning-add-source-scanner
 namespace: tap-install
 annotations:
   kapp.k14s.io/change-group: "tap-overlays"
type: Opaque
stringData:
 ootb-supply-chain-testing-scanning-add-source-scanner.yaml: |
   #@ load("@ytt:overlay", "overlay")
   #@overlay/match by=overlay.subset({"metadata":{"name":"source-test-scan-to-u
rl"}, "kind": "ClusterSupplyChain"})
   ---
   spec:
     resources:
       #@overlay/match by=overlay.index(2)
       #@overlay/insert before=True
           - name: source-scanner
             params:
             - default: scan-policy
               name: scanning_source_policy
             - default: blob-source-scan-template
               name: scanning_source_template
             sources:
             - name: source
               resource: source-tester
             templateRef:
               kind: ClusterSourceTemplate
               name: source-scanner-template
           #@overlay/match by="name"
           - name: image-provider
             sources:
             #@overlay/match by="name"
```

```
            - name: source
              resource: source-scanner
```

For information about ytt overlays, see the Carvel documentation.

2. Apply the `Secret` to your cluster:

```
kubectl apply -f secret.yml
```

3. Update your values file to include a `package_overlays` field:

```
package_overlays:
- name: ootb-supply-chain-testing-scanning
  secrets:
  - name: ootb-supply-chain-testing-scanning-add-source-scanner
```

4. Update Tanzu Application Platform:

```
tanzu package installed update tap -p tap.tanzu.vmware.com -v 1.9.1  --values-f
ile tap-values.yaml -n tap-install
```

For a multicluster installation, you must apply this to the build profile, because that is where the scan components run. For information about Tanzu Application Platform profiles, see Installing Tanzu Application Platform package and profiles.

Container image scan

A container image scan inspects the contents of a built container image for vulnerabilities. This scan is performed on the container image after it is uploaded to the container image registry and at periodic intervals after the initial upload. Many container registries, such as Harbor and Docker Hub include this capability.

Tanzu Application Platform enables scanning container images for vulnerabilities as part of your supply chain, allowing you to prevent deployment of a container image if vulnerabilities are discovered that exceed your security policy.

Overview of Supply Chain Security Tools - Scan 1.0

This topic gives you an overview of use cases, features, and CVEs for Supply Chain Security Tools (SCST) - Scan 1.0

Overview

With Supply Chain Security Tools - Scan, you can build and deploy secure, trusted software that complies with your corporate security requirements. Supply Chain Security Tools (SCST) - Scan provides scanning and gatekeeping capabilities that Application and DevSecOps teams can incorporate early in their path to production as it is a known industry best practice for reducing security risk and ensuring more efficient remediation.

Language support

For information about the languages and frameworks that are supported by Tanzu Application Platform components, see the Language and framework support in Tanzu Application Platform table.

Use cases

The following use cases apply to SCST - Scan:

- Use your scanner as a plug-in to scan source code repositories and images for known Common Vulnerabilities and Exposures (CVEs) before deploying to a cluster.
- Identify CVEs by continuously scanning each new code commit or each new image built.
- Analyze scan results against user-defined policies by using Open Policy Agent.
- Produce vulnerability scan results and post them to the SCST - Store from where they are queried.

SCST - Scan features

The following SCST - Scan features enable the Use cases:

- Kubernetes controllers to run scan TaskRuns.
- Custom Resource Definitions (CRDs) for Image and Source Scan.
- CRD for a scanner plug-in. Example is available by using Anchore's Syft and Grype.
- CRD for policy enforcement.
- Enhanced scanning coverage by analyzing the Cloud Native Buildpack SBoMs that Tanzu Build Service images provide.

A Note on Vulnerability Scanners

Although vulnerability scanning is an important practice in DevSecOps and the benefits of it are widely recognized and accepted, remember that there are limits present that impact its efficacy. The following examples illustrate the limits that are prevalent in most scanners today:

Missed CVEs

One limit of all vulnerability scanners is that there is no one tool that can find 100% of all CVEs, which means there is always a risk that a missed CVE can be exploited. Some reasons for missed CVEs include:

- The scanner does not detect the vulnerability because it is recently discovered and the CVE databases that the scanner checks against are not updated yet.
- Scanners verify different CVE sources based on the detected package type and OS.
- The scanner might not fully support a particular programming language, packaging system or manifest format.
- The scanner might not implement binary analysis or fingerprinting.
- The detected component does not always include a canonical name and vendor, requiring the scanner to infer and attempt fuzzy matching.
- When vendors register impacted software with NVD, the provided information might not exactly match the values in the release artifacts.

False positives

Vulnerability scanners cannot always access the information to accurately identify whether a CVE exists. This often leads to an influx of false positives where the tool mistakenly flags something as a vulnerability when it isn't. Unless a user is specialized in security or is deeply familiar with what is deemed to be a vulnerable component by the scanner, assessing and determining false positives becomes a challenging and time-consuming activity. Some reasons for a false positive flag include:

- A component might be misidentified due to similar names.
- A sub-component might be identified as the parent component.
- A component is correctly identified but the impacted function is not on a reachable code path.
- A component's impacted function is on a reachable code path but is not a concern due to the specific environment or configuration.
- The version of a component might be incorrectly flagged as impacted.
- The detected component does not always include a canonical name and vendor, requiring the scanner to infer and attempt fuzzy matching.

So what can you do to protect yourselves and your software?

Although vulnerability scanning is not a perfect solution, it is an essential part of the process for keeping your organization secure. You can take the following measures to maximize the benefits while minimizing the impact of the limits:

- Scan more continuously and comprehensively to identify and remediate zero-day vulnerabilities quicker. You can achieve comprehensive scanning by:
    - scanning earlier in the development cycle to ensure that you can address issues more efficiently and do not delay a release. Tanzu Application Platform includes security practices such as source and container image vulnerability scanning earlier in the path to production for application teams.
    - scanning any base images in use. Tanzu Application Platform image scanning can recognize and scan the OS packages from a base image.
    - scanning running software in test, stage, and production environments at a regular cadence.
    - generating accurate provenance at any level so that scanners have a complete picture of the dependencies to scan. This is where a software bill of materials (SBoM) comes into play. To help you automate this process, VMware Tanzu Build Service, leveraging Cloud Native Buildpacks, generates an SBoM for buildpack-based projects. Because this SBoM is generated during the image building stage, it is more accurate and complete than one generated earlier or later in the release life cycle. This is because it can highlight dependencies introduced at the time of build that might introduce potential for compromise.
- Scan by using multiple scanners to maximize CVE coverage.
- Practice keeping your dependencies up-to-date.
- Reduce overall surface area of attack by:
    - using smaller dependencies.
    - reducing the amount of third-party dependencies when possible.
    - using distroless base images when possible.
- Maintain a central record of false positives to ease CVE triaging and remediation efforts.

Install Supply Chain Security Tools - Scan

This topic describes how you can install Supply Chain Security Tools - Scan from the Tanzu Application Platform package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install SCST - Scan. For information about profiles, see Components and installation profiles.

Prerequisites

Before installing SCST - Scan:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- Install Supply Chain Security Tools - Store for scan results to persist. The integration with SCST - Store are handled in:
    - **Single Cluster:** The SCST - Store is present in the same cluster where SCST - Scan and the `ScanTemplates` are present.
    - **Multi-Cluster:** The SCST - Store is present in a different cluster (e.g.: view cluster) where the SCST - Scan and `ScanTemplates` are present.
    - **Integration Deactivated:** The SCST - Scan deployment is not required to communicate with SCST - Store.

  For information about SCST - Store, see Using the Supply Chain Security Tools - Store.

Note

If you are installing SCST - Scan in a cluster with restricted Kubernetes Pod Security Standards, you must update configurations for the Tekton Pipelines package. See Troubleshooting.

Configure properties

When you install the SCST - Scan (Scan controller), you can configure the following optional properties:

Key	Default	Type	Description	ScanTemplate Version
resources.limits.cpu	250m	integer/string	Limits describes the maximum amount of CPU resources allowed.	*n/a*
resources.limits.memory	256Mi	integer/string	Limits describes the maximum amount of memory resources allowed.	*n/a*
resources.requests.cpu	100m	integer/string	Requests describes the minimum amount of CPU resources required.	*n/a*
resources.requests.memory	128Mi	integer/string	Requests describes the minimum amount of memory resources required.	*n/a*
namespace	scan-link-system	string	Deployment namespace for the Scan Controller	*n/a*
retryScanJobsSecondsAfterError	60	integer	Seconds to wait before retrying errored scans	v1.3.1 and later
caCertData	""	string	The custom certificates trusted by the scans' connections	v1.4.0 and later

Key	Default	Type	Description	ScanTemplate Version
controller.pullSecret	“controller-secret-ref”	string	Reference to the secret used for pulling the controller image from private registry. Set to empty if deploying from a public registry.	v1.5.0 and later
docker.import	true	Boolean	Import `controller.pullSecret` from another namespace (requires secretgen-controller). Set to false if the secret is present.	v1.5.0 and later
deployedThroughTmc	false	Boolean	Flag to configure multicluter property collectors to configure Insight Metadata Store credentials	v1.7.0 and later
insertUserAndGroupID	true	Boolean	Flag to add default pod security context runAsUser and runAsGroup to the scan job	v1.7.0 and later
metadataStore.exports.namespace	metadata-store-secrets	string	Namespace for metadata store secrets and exports	v1.7.0 and later
metadataStore.exports.toNamespace	“*”	string	Destination namespace for exported secrets, or "*" to allow any namespace to import	v1.7.0 and later
metadataStore.exports.toNamespaces	- ""	array of strings	List of destination namespaces for exported secrets	v1.7.0 and later
metadataStore.exports.ca.secretName	store-ca-cert	string	Name to use for created CA Cert secret of the Insight Metadata Store	v1.7.0 and later
metadataStore.exports.ca.pem	""	string	PEM-encoded CA certificate of the Insight Metadata Store	v1.7.0 and later
metadataStore.exports.auth.secretName	store-auth-token	string	Name to use for created auth secret for the Insight Metadata Store	v1.7.0 and later
metadataStore.exports.auth.token	""	string	Service account auth token with read-write access to the Insight Metadata Store	v1.7.0 and later

When you install the SCST - Scan (Grype scanner), you can configure the following optional properties:

Key	Default	Type	Description	ScanTemplate Version
resources.requests.cpu	250m	integer/string	Requests describes the minimum amount of CPU resources required.	
resources.requests.memory	128Mi	integer/string	Requests describes the minimum amount of memory resources required.	
scanner.serviceAccount	grype-scanner	string	Name of scan pod’s service ServiceAccount	
scanner.serviceAccountAnnotations	nil	object	Annotations added to ServiceAccount	
targetImagePullSecret	*n/a*	string	Reference to the secret used for pulling images from private registry	

Key	Default	Type	Description	ScanTemplate Version
targetSourceSshSecret	*n/a*	string	Reference to the secret containing SSH credentials for cloning private repositories	
namespace	default	string	Deployment namespace for the Scan Templates	*n/a*
metadataStore.url	https://metadata-store-app.metadata-store.svc.cluster.local:8443	string	URL of the Insight Metadata Store	v1.2.0 and earlier
metadataStore.authSecret.name	*n/a*	string	Name of deployed secret with key auth_token	v1.2.0 and earlier
metadataStore.authSecret.importFromNamespace	*n/a*	string	Namespace from which to import the Insight Metadata Store auth_token	v1.2.0 and earlier
metadataStore.caSecret.importFromNamespace	metadata-store	string	Namespace from which to import the Insight Metadata Store CA Cert	v1.2.0 and earlier
metadataStore.caSecret.name	app-tls-cert	string	Name of deployed secret with key ca.crt holding the CA Cert of the Insight Metadata Store	v1.2.0 and earlier
metadataStore.clusterRole	metadata-store-read-write	string	Name of the deployed ClusterRole for read/write access to the Insight Metadata Store deployed in the same cluster	v1.2.0

Install

There are two options for installing Supply Chain Security Tools – Scan

Option 1: Install to multiple namespaces with the Namespace Provisioner

The Namespace Provisioner enables operators to securely automate the provisioning of multiple developer namespaces in a shared cluster. To install Supply Chain Security Tools – Scan by using the Namespace Provisioner, see Namespace Provisioner.

The Namespace Provisioner can also create scan policies across multiple developer namespaces. See Customize installation in the Namespace Provisioner documentation for configuration steps.

Option 2: Install manually to each individual namespace

The installation for Supply Chain Security Tools – Scan involves installing two packages:

- Scan controller
- Grype scanner

The Scan controller enables you to use a scanner, in this case, the Grype scanner. Ensure that both the Grype scanner and the Scan controller are installed.

To install SCST - Scan (Scan controller):

1. List version information for the package by running:

```
tanzu package available list scanning.apps.tanzu.vmware.com --namespace tap-ins
```

```
tall
```

For example:

```
$ tanzu package available list scanning.apps.tanzu.vmware.com --namespace tap-i
nstall
/ Retrieving package versions for scanning.apps.tanzu.vmware.com...
  NAME                            VERSION       RELEASED-AT
  scanning.apps.tanzu.vmware.com   1.1.0
```

2. (Optional) Make changes to the default installation settings:

   If you are using Grype Scanner `v1.5.1 and later` or other supported scanners included with Tanzu Application Platform `v1.5.1 and later`, and do not want to use the default SCST - Store integration, deactivate the integration by appending the following field to the `values.yaml` file:

   ```
   ---
   metadataStore:
     url: "" # Deactivate Supply Chain Security Tools - Store integration
   ```

   If you are using Grype Scanner `v1.5.0` or other supported scanners included with Tanzu Application Platform `v1.5.0`, and do not want to use the default SCST - Store integration, deactivate the integration by appending the following field to the `values.yaml` file:

   ```
   ---
   metadataStore: {} # Deactivate Supply Chain Security Tools - Store integration
   ```

   If you are using Grype Scanner `v1.2.0 and earlier`, or the Snyk Scanner, the following scanning configuration deactivates the embedded SCST - Store integration with a `scan-values.yaml` file.

   ```
   ---
   metadataStore:
     url: "" # Deactivate Supply Chain Security Tools - Store integration
   ```

   If your Grype Scanner version is earlier than v1.2.0, the scanning configuration must configure the store parameters. See v1.1 Install Supply Chain Security Tools - Scan.

   Run to retrieve other configurable settings and append the key-value pair to the previous `scan-values.yaml` file:

   ```
   tanzu package available get scanning.apps.tanzu.vmware.com/VERSION --values-sch
   ema -n tap-install
   ```

   Where `VERSION` is your package version number. For example, `1.1.0`.

3. Install the package by running:

   ```
   tanzu package install scan-controller \
     --package scanning.apps.tanzu.vmware.com \
     --version VERSION \
     --namespace tap-install \
     --values-file scan-values.yaml
   ```

   Where `VERSION` is your package version number. For example, `1.1.0`.

To install SCST - Scan (Grype scanner):

Note

To install Grype in multiple namespaces, use a namespace provisioner. See Namespace Provisioner.

1. List version information for the package by running:

```
tanzu package available list grype.scanning.apps.tanzu.vmware.com --namespace t
ap-install
```

For example:

```
$ tanzu package available list grype.scanning.apps.tanzu.vmware.com --namespace
tap-install
/ Retrieving package versions for grype.scanning.apps.tanzu.vmware.com...
  NAME                                  VERSION       RELEASED-AT
  grype.scanning.apps.tanzu.vmware.com  1.1.0
```

2. (Optional) Make changes to the default installation settings:

To define the configuration for the SCST - Store integration in the `grype-values.yaml` file for the Grype Scanner:

```
---
namespace: "DEV-NAMESPACE" # The developer namespace where the ScanTemplates ar
e going to be deployed
metadataStore:
  url: "METADATA-STORE-URL" # The base URL where the Store deployment can be re
ached
  caSecret:
    name: "CA-SECRET-NAME" # The name of the secret containing the ca.crt
    importFromNamespace: "SECRET-NAMESPACE" # The namespace where Store is depl
oyed (if single cluster) or where the connection secrets were created (if multi
-cluster)
  authSecret:
    name: "TOKEN-SECRET-NAME" # The name of the secret containing the auth toke
n to connect to Store
    importFromNamespace: "SECRET-NAMESPACE" # The namespace where the connectio
n secrets were created (if multi-cluster)
```

Note In a single cluster, the connection between the scanning pod and the metadata store happens inside the cluster and does not pass through ingress. This is automatically configured. You do not need to provide an ingress connection to the store. For information about troubleshooting issues with scanner to metadata store connection configuration, see Troubleshooting Scanner to MetadataStore Configuration.

Important

You must either define both the `METADATA-STORE-URL` and `CA-SECRET-NAME`, or not define them as they depend on each other.

Where:

- `DEV-NAMESPACE` is the namespace where you want to deploy the `ScanTemplates`. This is the namespace where the scanning feature runs.
- `METADATA-STORE-URL` is the base URL where the Supply Chain Security Tools (SCST) - Store deployment is reached, for example, `https://metadata-store-app.metadata-store.svc.cluster.local:8443`.

- `CA-SECRET-NAME` is the name of the secret containing the ca.crt to connect to the SCST - Store deployment.
- `SECRET-NAMESPACE` is the namespace where SCST - Store is deployed, if you are using a single cluster. If you are using multicluster, it is where the connection secrets were created.
- `TOKEN-SECRET-NAME` is the name of the secret containing the authentication token to connect to the SCST - Store deployment when installed in a different cluster, if you are using multicluster. If built images are pushed to the same registry as the Tanzu Application Platform images, this can reuse the `tap-registry` secret created in Add the Tanzu Application Platform package repository as described earlier.

Run to retrieve other configurable settings and append the key-value pair to the previous `grype-values.yaml` file:

```
tanzu package available get grype.scanning.apps.tanzu.vmware.com/VERSION --valu
es-schema -n tap-install
```

Where `VERSION` is your package version number. For example, `1.1.0`.

For example:

```
$ tanzu package available get grype.scanning.apps.tanzu.vmware.com/1.1.0 --valu
es-schema -n tap-install
| Retrieving package details for grype.scanning.apps.tanzu.vmware.com/1.1.0...
  KEY                         DEFAULT  TYPE    DESCRIPTION
  namespace                   default  string  Deployment namespace for the Scan
Templates
  resources.limits.cpu        1000m    <nil>   Limits describes the maximum amou
nt of cpu resources allowed.
  resources.requests.cpu      250m     <nil>   Requests describes the minimum am
ount of cpu resources required.
  resources.requests.memory   128Mi    <nil>   Requests describes the minimum am
ount of memory resources required.
  targetImagePullSecret       <EMPTY>  string  Reference to the secret used for
pulling images from private registry.
  targetSourceSshSecret       <EMPTY>  string  Reference to the secret containin
g SSH credentials for cloning private repositories.
```

Important

If `targetSourceSshSecret` is not set, the private source scan template is not installed.

3. Install the package by running:

```
tanzu package install grype-scanner \
  --package grype.scanning.apps.tanzu.vmware.com \
  --version VERSION \
  --namespace tap-install \
  --values-file grype-values.yaml
```

Where `VERSION` is your package version number. For example, `1.1.0`.

For example:

```
$ tanzu package install grype-scanner \
  --package grype.scanning.apps.tanzu.vmware.com \
  --version 1.1.0 \
  --namespace tap-install \
```

```
  --values-file grype-values.yaml
/ Installing package 'grype.scanning.apps.tanzu.vmware.com'
| Getting namespace 'tap-install'
| Getting package metadata for 'grype.scanning.apps.tanzu.vmware.com'
| Creating service account 'grype-scanner-tap-install-sa'
| Creating cluster admin role 'grype-scanner-tap-install-cluster-role'
| Creating cluster role binding 'grype-scanner-tap-install-cluster-rolebinding'
/ Creating package resource
- Package install status: Reconciling

 Added installed package 'grype-scanner' in namespace 'tap-install'
```

Upgrade Supply Chain Security Tools - Scan

This topic describes how you can upgrade Supply Chain Security Tools - Scan from the Tanzu Application Platform package repository.

You can perform a fresh install of SCST - Scan by following the instructions in Install Supply Chain Security Tools - Scan.

Prerequisites

Before you upgrade SCST - Scan, upgrade the Tanzu Application Platform by following the instructions in Upgrading Tanzu Application Platform.

General Upgrades for SCST - Scan

When you're upgrading to any version of SCST - Scan these are some factors to accomplish this task:

1. Inspect the Release Notes for the version you're upgrading to. There you can find any breaking changes for the installation.
2. Get the values schema for the package version you're upgrading to by running:

   ```
   tanzu package available get scanning.apps.tanzu.vmware.com/$VERSION --values-sc
   hema -n tap-install
   ```

   Where `$VERSION` is the new version. This gives you insights on the values you can configure in your `tap-values.yaml` for the new version.

Upgrade a scanner in all namespaces

This section describes how to upgrade a supported scanner in all namespaces. The procedure is different depending on the installation method:

1. Installation by using Namespace Provisioner
2. Manual installation

Installation by using Namespace Provisioner

All scanners installed by the Namespace Provisioner in all managed namespaces are upgraded automatically. For example, if you upgrade your installation of Tanzu Application Platform and the version of Grype is updated, all Grype scanners installed by the Namespace Provisioner for all managed namespaces are automatically upgraded.

Manual installation

1. If a scanner, such as Grype Scanner, was installed as part of Tanzu Application Platform by using the full profile, run to upgrade:

   ```
   tanzu package installed update tap -p tap.tanzu.vmware.com -v VERSION --values-file tap-values.yaml -n tap-install
   ```

   Where `VERSION` is your Tanzu Application Platform version.

2. If a scanner, such as Grype Scanner, was installed by using component installation you must manually run:

   ```
   tanzu package installed update grype -p grype.scanning.apps.tanzu.vmware.com -v GRYPE-VERSION --values-file grype-values.yaml -n NAMESPACE
   ```

   Where:

   - `GRYPE-VERSION` is the version of Grype that you are upgrading to.
   - `NAMESPACE` is the namespace in which Grype is installed in.

Upgrade to Version v1.2.0

To upgrade from a previous version of SCST - Scan to the version `v1.2.0`:

1. Change the `SecretExports` from SCST - Store.

   SCST - Scan needs information to connect to the SCST - Store deployment, you must change where these secrets are exported to enable the connection with the version `v1.2.0` of SCST - Scan.

   - **For a single cluster deployment:**

     1. Edit the `tap-values.yaml` file you used to deploy SCST - Store to export the CA certificate to your developer namespace.

        ```
        metadata_store:
            ns_for_export_app_cert: "DEV-NAMESPACE"
        ```

        Note

        The `ns_for_export_app_cert` supports one namespace at a time. If you have multiple namespaces you can replace this value with a `*`, but this exports the CA certificate to all namespaces. Consider whether this increased visibility presents a risk.

     2. Update Tanzu Application Platform to apply the changes:

        ```
        tanzu package installed update tap -f tap-values.yaml -n tap-install
        ```

   - **For a multicluster deployment:**

     You must reapply the SecretExport by changing the `toNamespace: scan-link-system` to `toNamespace: DEV-NAMESPACE`:

     ```
     ---
     apiVersion: secretgen.carvel.dev/v1alpha1
     kind: SecretExport
     metadata:
     ```

```
  name: store-ca-cert
  namespace: metadata-store-secrets
spec:
  toNamespace: "DEV-NAMESPACE"
---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretExport
metadata:
  name: store-auth-token
  namespace: metadata-store-secrets
spec:
  toNamespace: "DEV-NAMESPACE"
```

2. Update your `tap-values.yaml` file.

   The installation of the SCST - Scan and the Grype scanner have some changes. The connection to the SCST - Store component have moved to the Grype scanner package. To deactivate the connection from the SCST - Scan, which is still present for backwards compatibility, but is deprecated and is removed in `v1.3.0`.

```
# Deactivate scan controller embedded Supply Chain Security Tools - Store integ
ration
scanning:
  metadataStore:
    url: ""

# Install Grype Scanner v1.2.0
grype:
  namespace: "DEV-NAMESPACE" # The developer namespace where the ScanTemplates
are going to be deployed
  metadataStore:
    url: "METADATA-STORE-URL" # The base URL where the Store deployment can be
reached
    caSecret:
      name: "CA-SECRET-NAME" # The name of the secret containing the ca.crt
      importFromNamespace: "SECRET-NAMESPACE" # The namespace where Store is de
ployed (if single cluster) or where the connection secrets were created (if mul
ti-cluster)
    authSecret:
      name: "TOKEN-SECRET-NAME" # The name of the secret containing the auth to
ken to connect to Store
      importFromNamespace: "SECRET-NAMESPACE" # The namespace where the connect
ion secrets were created (if multi-cluster)
```

   For more insights on how to install Grype, see Install Supply Chain Security Tools - Scan (Grype Scanner).

   Note

   If a mix of Grype templates, such as earlier than v1.2.0 and v1.2.0 and later, are used, both `scanning` and `grype` must configure the parameters. The secret must also export to both scan-link-system and the developer namespace. Do this by exporting to `*` or by defining multiple secrets and exports. If Grype is installed to multiple namespaces there must be corresponding exports. See Install Supply Chain Security Tools - Scan (Grype Scanner).

3. Update Tanzu Application Platform to apply the changes:

```
tanzu package installed update tap -f tap-values.yaml -n tap-install
```

4. Update the `ScanPolicy` to include the latest structure changes for `v1.2.0`.

   To update to the latest valid Rego File in the `ScanPolicy`, Enforce compliance policy using Open Policy Agent. `v1.2.0` introduced some breaking changes in the Rego File structure used for the `ScanPolicies`. For more information, see the Release Notes.

5. Verify the upgrade.

   You can run any `ImageScan` or `SourceScan` in your `DEV-NAMESPACE` where the Grype Scanner was installed, and it finishes. Here is a sample you can try to run to detect if everything upgraded.

   1. Create the `verify-upgrade.yaml` file in your system with the following content:

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: scan-policy
  labels:
    'app.kubernetes.io/part-of': 'enable-in-gui'
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible",
"UnknownSeverity"
    notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]
    ignoreCves := []

    contains(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
      severities := { e | e := match.ratings.rating.severity } | { e | e
:= match.ratings.rating[_].severity }
      some i
      fails := contains(notAllowedSeverities, severities[i])
      not fails
    }

    isSafe(match) {
      ignore := contains(ignoreCves, match.id)
      ignore
    }

    deny[msg] {
      comps := { e | e := input.bom.components.component } | { e | e := i
nput.bom.components.component[_] }
      some i
      comp := comps[i]
      vulns := { e | e := comp.vulnerabilities.vulnerability } | { e | e
:= comp.vulnerabilities.vulnerability[_] }
      some j
      vuln := vulns[j]
      ratings := { e | e := vuln.ratings.rating.severity } | { e | e := v
uln.ratings.rating[_].severity }
      not isSafe(vuln)
      msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
    }
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ImageScan
metadata:
  name: sample-public-image-scan
```

```
spec:
  registry:
    image: "nginx:1.16"
  scanTemplate: public-image-scan-template
  scanPolicy: scan-policy
```

2. Deploy the resources:

```
kubectl apply -f verify-upgrade.yaml -n DEV-NAMESPACE
```

3. View the scan results:

```
kubectl describe imagescan sample-public-image-scan -n DEV-NAMESPACE
```

If it is successful, the `ImageScan` goes to the `Failed` phase and shows the results of the scan in the `Status`.

Install another scanner for Supply Chain Security Tools - Scan

This topic describes how you can install scanners to work with Supply Chain Security Tools - Scan from the Tanzu Application Platform package repository.

Follow the instructions in this topic to install a scanner other than the out of the box Grype Scanner with SCST - Scan.

Prerequisites

Before installing a new scanner, install Supply Chain Security Tools - Scan. It must be present on the same cluster. The prerequisites for Scan are also required.

Install

To install a new scanner, follow these steps:

1. Complete scanner specific prerequisites for the scanner you're trying to install. For example, creating an API token to connect to the scanner.
    - Snyk Scanner (Beta) is available for image scanning.
    - Carbon Black Scanner (Beta) is available for image scanning.
2. List the available packages to discover what scanners you can use by running:

```
tanzu package available list --namespace tap-install
```

For example:

```
$ tanzu package available list --namespace tap-install
/ Retrieving available packages...
  NAME                                                DISPLAY-NAME
SHORT-DESCRIPTION
  grype.scanning.apps.tanzu.vmware.com                Grype Scanner for Supply
Chain Security Tools - Scan                      Default scan templates using A
nchore Grype
  snyk.scanning.apps.tanzu.vmware.com                 Snyk for Supply Chain Se
curity Tools - Scan                               Default scan templates using
Snyk
  carbonblack.scanning.apps.tanzu.vmware.com          Carbon Black Scanner for
```

```
Supply Chain Security Tools - Scan               Default scan templates using C
arbon Black
```

3. List version information for the scanner package by running:

   ```
   tanzu package available list SCANNER-NAME --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package available list snyk.scanning.apps.tanzu.vmware.com --namespace
   tap-install
   / Retrieving package versions for snyk.scanning.apps.tanzu.vmware.com...
     NAME                                VERSION           RELEASED-AT
     snyk.scanning.apps.tanzu.vmware.com   1.0.0-beta.2
   ```

4. (Optional) Confirm that the secret created in Step 1 for scanner specific prerequisites is created.

5. Create a `values.yaml` to apply custom configurations to the scanner:

   > **Note**
   >
   > This step might be required for some scanners but optional for others.

   To list the values you can configure for any scanner, run:

   ```
   tanzu package available get SCANNER-NAME/VERSION --values-schema -n tap-install
   ```

   Where:

   - `SCANNER-NAME` is the name of the scanner package you retrieved earlier.
   - `VERSION` is your package version number. For example, `snyk.scanning.apps.tanzu.vmware.com/1.0.0-beta.2`.

   For example:

   ```
   $ tanzu package available get snyk.scanning.apps.tanzu.vmware.com/1.0.0-beta.2
   --values-schema -n tap-install

   KEY                                          DEFAULT
   TYPE    DESCRIPTION
   metadataStore.authSecret.name
   string  Name of deployed Secret with key auth_token
   metadataStore.authSecret.importFromNamespace
   string  Namespace from which to import the Insight Metadata Store auth_token
   metadataStore.caSecret.importFromNamespace    metadata-store
   string  Namespace from which to import the Insight Metadata Store CA Cert
   metadataStore.caSecret.name                   app-tls-cert
   string  Name of deployed Secret with key ca.crt holding the CA Cert of the Insi
   ght Metadata Store
   metadataStore.clusterRole                     metadata-store-read-write
   string  Name of the deployed ClusterRole for read/write access to the Insight M
   etadata Store deployed in the same cluster
   metadataStore.url                             https://metadata-store-app.metada
   ta-store.svc.cluster.local:8443  string  Url of the Insight Metadata Store
   namespace                                     default
   string  Deployment namespace for the Scan Templates
   resources.requests.cpu                        250m
   <nil>   Requests describes the minimum amount of cpu resources required.
   resources.requests.memory                     128Mi
   <nil>   Requests describes the minimum amount of memory resources required.
   ```

```
resources.limits.cpu                            1000m
<nil>   Limits describes the maximum amount of cpu resources allowed.
snyk.tokenSecret.name
string  Reference to the secret containing a Snyk API Token as snyk_token.
targetImagePullSecret
string  Reference to the secret used for pulling images from private registry.
```

6. Define the `--values-file` flag to customize the default configuration:

   The `values.yaml` file you created earlier is referenced with the `--values-file` flag when running your Tanzu install command:

   ```
   tanzu package install REFERENCE-NAME \
     --package SCANNER-NAME \
     --version VERSION \
     --namespace tap-install \
     --values-file PATH-TO-VALUES-YAML
   ```

   Where:

   - `REFERENCE-NAME` is the name referenced by the installed package. For example, `grype-scanner`, `snyk-scanner`.
   - `SCANNER-NAME` is the name of the scanner package you retrieved earlier. For example, `snyk.scanning.apps.tanzu.vmware.com`.
   - `VERSION` is your package version number. For example, `1.0.0-beta.2`.
   - `PATH-TO-VALUES-YAML` is the path that points to the `values.yaml` file created earlier.

   For example:

   ```
   $ tanzu package install snyk-scanner \
     --package snyk.scanning.apps.tanzu.vmware.com \
     --version 1.1.0 \
     --namespace tap-install \
     --values-file values.yaml
   / Installing package 'snyk.scanning.apps.tanzu.vmware.com'
   | Getting namespace 'tap-install'
   | Getting package metadata for 'snyk.scanning.apps.tanzu.vmware.com'
   | Creating service account 'snyk-scanner-tap-install-sa'
   | Creating cluster admin role 'snyk-scanner-tap-install-cluster-role'
   | Creating cluster role binding 'snyk-scanner-tap-install-cluster-rolebinding'
   / Creating package resource
   - Package install status: Reconciling

    Added installed package 'snyk-scanner' in namespace 'tap-install'
   ```

Verify Installation

To verify the installation create an `ImageScan` or `SourceScan` referencing one of the newly added `ScanTemplates` for the scanner.

1. (Optional) Create a `ScanPolicy` formatted for the output specific to the scanner you are installing, to reference in the `ImageScan` or `SourceScan`.

   ```
   kubectl apply -n $DEV_NAMESPACE -f SCAN-POLICY-YAML
   ```

   Note

   As vulnerability scanners output different formats, the `ScanPolicies` can vary. For information about policies and samples, see Enforce compliance

policy using Open Policy Agent.

2. Retrieve available `ScanTemplates` from the namespace where the scanner is installed:

```
kubectl get scantemplates -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

For example:

```
$ kubectl get scantemplates
NAME                                AGE
blob-source-scan-template           10d
private-image-scan-template         10d
public-image-scan-template          10d
public-source-scan-template         10d
snyk-private-image-scan-template    10d
snyk-public-image-scan-template     10d
```

3. Create the following ImageScan YAML:

Note

Some scanners do not support both `ImageScan` and `SourceScan`.

```
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ImageScan
metadata:
  name: sample-scanner-public-image-scan
spec:
  registry:
    image: "nginx:1.16"
  scanTemplate: SCAN-TEMPLATE
  scanPolicy: SCAN-POLICY # Optional
```

Where:

- `SCAN-TEMPLATE` is the name of the installed `ScanTemplate` in the `DEV-NAMESPACE` you retrieved earlier.
- `SCAN-POLICY` it's an optional reference to an existing `ScanPolicy` in the same `DEV-NAMESPACE`.

For example:

```
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ImageScan
metadata:
  name: sample-snyk-public-image-scan
spec:
  registry:
    image: "nginx:1.16"
  scanTemplate: snyk-public-image-scan-template
  scanPolicy: snyk-scan-policy
```

4. Create the following SourceScan YAML:

Note

Some scanners do not support both `ImageScan` and `SourceScan`.

```
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: SourceScan
metadata:
  name: sample-scanner-public-source-scan
spec:
  git:
    url: "https://github.com/houndci/hound.git"
    revision: "5805c650"
  scanTemplate: SCAN-TEMPLATE
  scanPolicy: SCAN-POLICY # Optional
```

Where:

- `SCAN-TEMPLATE` is the name of the installed `ScanTemplate` in the `DEV-NAMESPACE` you retrieved earlier.
- `SCAN-POLICY` is an optional reference to an existing `ScanPolicy` in the same `DEV-NAMESPACE`.

For example:

```
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: SourceScan
metadata:
  name: sample-grype-public-source-scan
spec:
  git:
    url: "https://github.com/houndci/hound.git"
    revision: "5805c650"
  scanTemplate: public-source-scan-template
  scanPolicy: scan-policy
```

5. Apply the ImageScan and SourceScan YAMLs:

   To run the scans, apply them to the cluster by running these commands:

   `ImageScan`:

   ```
   kubectl apply -f PATH-TO-IMAGE-SCAN-YAML -n DEV-NAMESPACE
   ```

   Where `PATH-TO-IMAGE-SCAN-YAML` is the path to the YAML file created earlier.

   `SourceScan`:

   ```
   kubectl apply -f PATH-TO-SOURCE-SCAN-YAML -n DEV-NAMESPACE
   ```

   Where `PATH-TO-SOURCE-SCAN-YAML` is the path to the YAML file created earlier.

   For example:

   ```
   $ kubectl apply -f imagescan.yaml -n my-apps
   imagescan.scanning.apps.tanzu.vmware.com/sample-snyk-public-image-scan created

   $ kubectl apply -f sourcescan.yaml -n my-apps
   sourcescan.scanning.apps.tanzu.vmware.com/sample-grype-public-source-scan creat
   ed
   ```

6. To verify the integration, get the scan to see if it completed by running:

   For `ImageScan`:

```
kubectl get imagescan IMAGE-SCAN-NAME -n DEV-NAMESPACE
```

Where `IMAGE-SCAN-NAME` is the name of the `ImageScan` as defined in the YAML file created earlier.

For `SourceScan`:

```
kubectl get sourcescan SOURCE-SCAN-NAME -n DEV-NAMESPACE
```

Where `SOURCE-SCAN-NAME` is the name of the `SourceScan` as defined in the YAML file created earlier.

For example:

```
$ kubectl get imagescan sample-snyk-public-image-scan -n my-apps
NAME                              PHASE       SCANNEDIMAGE   AGE   CRITICAL   HIG
H   MEDIUM   LOW   UNKNOWN   CVETOTAL
sample-snyk-public-image-scan   Completed   nginx:1.16     26h   0          114
58       314   0         486

$ kubectl get sourcescan sample-grype-public-source-scan -n my-apps
NAME                                                                          PHASE
SCANNEDREVISION   SCANNEDREPOSITORY                        AGE     CRITICAL   HIG
H   MEDIUM   LOW   UNKNOWN   CVETOTAL
sourcescan.scanning.apps.tanzu.vmware.com/grypesourcescan-sample-public   Compl
eted   5805c650          https://github.com/houndci/hound.git   8m34s   21
121    112      9     0         263
```

Note

If you define a `ScanPolicy` for the scans and the evaluation finds a violation, the `Phase` is `Failed` instead of `Completed`. In both cases the scan finished.

7. Clean up:

```
kubectl delete -f PATH-TO-SCAN-YAML -n DEV-NAMESPACE
```

Where `PATH-TO-SCAN-YAML` is the path to the YAML file created earlier.

Install scanner to multiple namespaces

To install a Scanner to multiple namespaces, VMware recommends using a namespace provisioner. See Namespace Provisioner

Configure Tanzu Application Platform Supply Chain to use new scanner

In order to scan your images with the new scanner installed in the Out of the Box Supply Chain with Testing and Scanning, you must update your Tanzu Application Platform installation.

Add the `ootb_supply_chain_testing_scanning.scanning` section to your `tap-values.yaml` and perform a Tanzu Application Platform update.

You can define which `ScanTemplates` is used for both `SourceScan` and `ImageScan`. The default values are the Grype Scanner `ScanTemplates`, but they are overwritten by any other `ScanTemplate` present in your `DEV-NAMESPACE`. The same applies to the `ScanPolicies` applied to each kind of scan.

```
ootb_supply_chain_testing_scanning:
  scanning:
    image:
      template: IMAGE-SCAN-TEMPLATE
      policy: IMAGE-SCAN-POLICY
    source:
      template: SOURCE-SCAN-TEMPLATE
      policy: SOURCE-SCAN-POLICY
```

Note

For the Supply Chain to work properly, the `SOURCE-SCAN-TEMPLATE` must support blob files and the `IMAGE-SCAN-TEMPLATE` must support private images.

For example:

```
ootb_supply_chain_testing_scanning:
  scanning:
    image:
      template: snyk-private-image-scan-template
      policy: snyk-scan-policy
    source:
      template: blob-source-scan-template
      policy: scan-policy
```

Uninstall Scanner

To replace the scanner in the Supply Chain, follow the steps mentioned in Configure TAP Supply Chain to Use New Scanner. After the scanner is no longer required by the Supply Chain, you can remove the package by running:

```
tanzu package installed delete REFERENCE-NAME \
    --namespace tap-install
```

Where `REFERENCE-NAME` is the name you identified the package with, when installing in the Install section. For example, `grype-scanner`, `snyk-scanner`.

For example:

```
$ tanzu package installed delete snyk-scanner \
    --namespace tap-install
```

Other Available Scanner Integrations

In addition to providing the above supported integrations, VMware encourages the broader community to support VMware in our goal of integrating with customers' preferred CVE scanners.

Additional integrations:

- Prisma Scanner (Alpha) is available for source and image scanning.
- Trivy Scanner (Alpha) is available for source and image scanning.

Prerequisites for Snyk Scanner for Supply Chain Security Tools - Scan (Beta)

This topic describes the prerequisites you must complete to install Supply Chain Security Tools - Scan (Snyk Scanner) from the Tanzu Application Platform package repository.

Important

Snyk's image scanning capability is in beta. Snyk might only return a partial list of CVEs when scanning Buildpack images.

Prepare the Snyk Scanner configuration

1. Obtain a Snyk API Token from the Snyk documentation.
2. Create a Snyk secret YAML file and insert the base64 encoded Snyk API token into the `snyk_token`:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: snyk-token-secret
     namespace: my-apps
   data:
     snyk_token: BASE64-SNYK-API-TOKEN
   ```

   Where `BASE64-SNYK-API-TOKEN` is the Snyk API Token obtained earlier.
3. Apply the Snyk secret YAML file by running:

   ```
   kubectl apply -f YAML-FILE
   ```

   Where `YAML-FILE` is the name of the Snyk secret YAML file you created.
4. Define the `--values-file` flag to customize the default configuration. You must define the following fields in the `values.yaml` file for the Snyk Scanner configuration. You can add fields as needed to activate or deactivate behaviors. You can append the values to this file as shown later in this topic. Create a `values.yaml` file by using the following configuration:

   ```
   ---
   namespace: DEV-NAMESPACE
   targetImagePullSecret: TARGET-REGISTRY-CREDENTIALS-SECRET
   snyk:
     tokenSecret:
       name: SNYK-TOKEN-SECRET
   ```

   Where:

   - `DEV-NAMESPACE` is your developer namespace.

     Note

     To use a namespace other than the default namespace, ensure that the namespace exists before you install. If the namespace does not exist, the scanner installation fails.

   - `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the secret that contains the credentials to pull an image from a private registry for scanning.
   - `SNYK-TOKEN-SECRET` is the name of the secret you created that contains the `snyk_token` to connect to the Snyk API. This field is required.

The Snyk Scanner integration can work with or without the SCST - Store integration. The `values.yaml` file is slightly different for each configuration.

SCST - Store integration

Using SCST - Store Integration: To persist the results found by the Snyk Scanner, you can enable the SCST - Store integration by appending the fields to the `values.yaml` file.

The Grype and Snyk Scanner Integrations both enable the Metadata Store. To prevent conflicts, the configuration values are slightly different based on whether the Grype Scanner Integration is installed or not. If Tanzu Application Platform is installed using the Full Profile, the Grype Scanner Integration is installed, unless it is explicitly excluded.

- If the Grype Scanner Integration is installed in the same `dev-namespace` Snyk Scanner is installed:

```
#! ...
metadataStore:
  #! The url where the Store deployment is accessible.
  #! Default value is: "https://metadata-store-app.metadata-store.svc.cluster.l
ocal:8443"
  url: "STORE-URL"
  caSecret:
    #! The name of the secret that contains the ca.crt to connect to the Store
Deployment.
    #! Default value is: "app-tls-cert"
    name: "CA-SECRET-NAME"
    importFromNamespace: "" #! since both Snyk and Grype both enable store, one
must leave importFromNamespace blank
  #! authSecret is for multicluster configurations.
  authSecret:
    #! The name of the secret that contains the auth token to authenticate to t
he Store Deployment.
    name: "AUTH-SECRET-NAME"
    importFromNamespace: "" #! since both Snyk and Grype both enable store, one
must leave importFromNamespace blank
```

- If the Grype Scanner Integration is not installed in the same `dev-namespace` Snyk Scanner is installed:

```
#! ...
metadataStore:
  #! The url where the Store deployment is accessible.
  #! Default value is: "https://metadata-store-app.metadata-store.svc.cluster.l
ocal:8443"
  url: "STORE-URL"
  caSecret:
    #! The name of the secret that contains the ca.crt to connect to the Store
Deployment.
    #! Default value is: "app-tls-cert"
    name: "CA-SECRET-NAME"
    #! The namespace where the secrets for the Store Deployment live.
    #! Default value is: "metadata-store"
    importFromNamespace: "STORE-SECRETS-NAMESPACE"
  #! authSecret is for multicluster configurations.
  authSecret:
    #! The name of the secret that contains the auth token to authenticate to t
he Store Deployment.
    name: "AUTH-SECRET-NAME"
    #! The namespace where the secrets for the Store Deployment live.
    importFromNamespace: "STORE-SECRETS-NAMESPACE"
```

Without SCST - Store Integration: The SCST - Store integration is enabled by default. If you don't want to use this integration, deactivate the integration by appending the following field to the `values.yaml` file:

```
# ...
metadataStore:
   url: "" # Configuration is moved, so set this string to empty.
```

Sample ScanPolicy for Snyk in SPDX JSON format

1. Create a ScanPolicy YAML with a Rego file for scanner output in the SPDX JSON format. Here is a sample scan policy resource:

```
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: snyk-scan-policy
  labels:
    'app.kubernetes.io/part-of': 'enable-in-gui'
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "Unkn
ownSeverity"
    notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]
    ignoreCves := []

    contains(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
      fails := contains(notAllowedSeverities, match.relationships[_].ratedBy.ra
ting[_].severity)
      not fails
    }

    isSafe(match) {
      ignore := contains(ignoreCves, match.id)
      ignore
    }

    deny[msg] {
      vuln := input.vulnerabilities[_]
      ratings := vuln.relationships[_].ratedBy.rating[_].severity
      comp := vuln.relationships[_].affect.to[_]
      not isSafe(vuln)
      msg = sprintf("CVE %s %s %s", [comp, vuln.id, ratings])
    }
```

2. Apply the YAML file by running:

```
kubectl apply -n $DEV_NAMESPACE -f SCAN-POLICY-YAML
```

Note

The Snyk Scanner integration is only available for an image scan, not a source scan.

After all prerequisites are completed, follow the steps in Install another scanner for Supply Chain Security Tools - Scan to install the Snyk Scanner.

Prerequisites for Carbon Black Scanner for Supply Chain Security Tools - Scan (Beta)

This topic describes prerequisites you must complete to install Supply Chain Security Tools - Scan (Carbon Black Scanner) from the Tanzu Application Platform package repository. The Carbon Black Scanner integration is only available for an image scan, not a source scan.

Important

Carbon Black's image scanning capability is in beta. Carbon Black might only return a partial list of CVEs when scanning Buildpack images.

Prepare the Carbon Black Scanner configuration

To prepare the Carbon Black Scanner configuration before you install any scanners:

1. Obtain a Carbon Black API Token from Carbon Black Cloud.
2. Create a Carbon Black secret YAML file and insert the Carbon Black API configuration key. Obtain all values from your CBC console.

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: CARBONBLACK-CONFIG-SECRET
     namespace: my-apps
   stringData:
     cbc_api_id: CBC-API-ID
     cbc_api_key: CBC-API-KEY
     cbc_org_key: CBC-ORG-KEY
     cbc_saas_url: CBC-SAAS-URL
   ```

   Where:

   - `CBC-API-ID` is the API ID obtained from CBC.
   - `CBC-API-KEY` is the API Key obtained from CBC.
   - `CBC-ORG-KEY` is the Org Key of your CBC organization.
   - `CBC-SAAS-URL` is the CBC Backend URL.
3. Apply the Carbon Black secret YAML file by running:

   ```
   kubectl apply -f YAML-FILE
   ```

   Where `YAML-FILE` is the name of the Carbon Black secret YAML file you created.
4. Define the `--values-file` flag to customize the default configuration. Create a `values.yaml` file by using the following configuration:

   You must define the following fields in the `values.yaml` file for the Carbon Black Scanner configuration. You can add fields as needed to enable or deactivate behaviors. You can append the values to this file as shown later in this topic.

   ```
   ---
   namespace: DEV-NAMESPACE
   ```

```
targetImagePullSecret: TARGET-REGISTRY-CREDENTIALS-SECRET
carbonBlack:
  configSecret:
    name: CARBONBLACK-CONFIG-SECRET
```

- `DEV-NAMESPACE` is your developer namespace.

  Important

  To use a namespace other than the default namespace, ensure that the namespace exists before you install. If the namespace does not exist, the scanner installation fails.

- `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the secret that contains the credentials to pull an image from a private registry for scanning.
- `CARBONBLACK-CONFIG-SECRET` is the name of the secret you created that contains the Carbon Black configuration to connect to CBC. This field is required.

The Carbon Black Scanner integration can work with or without the SCST - Store integration. The `values.yaml` file is slightly different for each configuration.

SCST - Store integration

To Integrate:

1. Do one of the following procedures:
   - Use the Supply Chain Security Tools - Store
   - Without using the Supply Chain Security Tools - Store
2. Apply the YAML.

Using SCST - Store Integration

To persist the results found by the Carbon Black Scanner, you can enable the SCST - Store integration by appending the fields to the `values.yaml` file.

The Grype and Carbon Black Scanner Integrations both enable the Metadata Store. To prevent conflicts, the configuration values are slightly different based on whether the Grype Scanner Integration is installed or not. If Tanzu Application Platform was installed using the Full Profile, the Grype Scanner Integration was installed, unless it was explicitly excluded.

- If the Grype Scanner Integration is installed in the same `dev-namespace` Carbon Black Scanner is installed:

```
#! ...
metadataStore:
  #! The url where the Store deployment is accessible.
  #! Default value is: "https://metadata-store-app.metadata-store.svc.cluster.l
ocal:8443"
  url: "STORE-URL"
  caSecret:
    #! The name of the secret that contains the ca.crt to connect to the Store
Deployment.
    #! Default value is: "app-tls-cert"
    name: "CA-SECRET-NAME"
    importFromNamespace: "" #! since both Carbon Black and Grype both enable st
ore, one must leave importFromNamespace blank
  #! authSecret is for multicluster configurations.
  authSecret:
```

```
    #! The name of the secret that contains the auth token to authenticate to t
he Store Deployment.
    name: "AUTH-SECRET-NAME"
    importFromNamespace: "" #! since both Carbon Black and Grype both enable st
ore, one must leave importFromNamespace blank
```

- If the Grype Scanner Integration is not installed in the same `dev-namespace` Carbon Black Scanner is installed:

```
#! ...
metadataStore:
  #! The url where the Store deployment is accessible.
  #! Default value is: "https://metadata-store-app.metadata-store.svc.cluster.l
ocal:8443"
  url: "STORE-URL"
  caSecret:
    #! The name of the secret that contains the ca.crt to connect to the Store
Deployment.
    #! Default value is: "app-tls-cert"
    name: "CA-SECRET-NAME"
    #! The namespace where the secrets for the Store Deployment live.
    #! Default value is: "metadata-store"
    importFromNamespace: "STORE-SECRETS-NAMESPACE"
  #! authSecret is for multicluster configurations.
  authSecret:
    #! The name of the secret that contains the auth token to authenticate to t
he Store Deployment.
    name: "AUTH-SECRET-NAME"
    #! The namespace where the secrets for the Store Deployment live.
    importFromNamespace: "STORE-SECRETS-NAMESPACE"
```

Without SCST - Store Integration

The SCST - Store integration is enabled by default. If you don't want to use this integration, explicitly deactivate the integration by appending the following field to the `values.yaml` file:

```
# ...
metadataStore:
  url: "" # Disable Supply Chain Security Tools - Store integration
```

Sample ScanPolicy in CycloneDX format

1. Create a ScanPolicy YAML with a Rego file for scanner output in the CycloneDX format. For example:

```
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: scan-policy
  labels:
    'app.kubernetes.io/part-of': 'enable-in-gui'
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "Unkn
ownSeverity"
    notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]
    ignoreCves := []

    contains(array, elem) = true {
      array[_] = elem
```

```
    } else = false { true }

    isSafe(match) {
      severities := { e | e := match.ratings.rating.severity } | { e | e := mat
ch.ratings.rating[_].severity }
      some i
      fails := contains(notAllowedSeverities, severities[i])
      not fails
    }

    isSafe(match) {
      ignore := contains(ignoreCves, match.id)
      ignore
    }

    deny[msg] {
      comps := { e | e := input.bom.components.component } | { e | e := input.b
om.components.component[_] }
      some i
      comp := comps[i]
      vulns := { e | e := comp.vulnerabilities.vulnerability } | { e | e := com
p.vulnerabilities.vulnerability[_] }
      some j
      vuln := vulns[j]
      ratings := { e | e := vuln.ratings.rating.severity } | { e | e := vuln.ra
tings.rating[_].severity }
      not isSafe(vuln)
      msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
    }
```

2. Apply the YAML:

```
kubectl apply -n $DEV_NAMESPACE -f SCAN-POLICY-YAML
```

After you complete all prerequisites, install the Carbon Black Scanner. See Install another scanner for Supply Chain Security Tools - Scan.

Prerequisites for Prisma Scanner for Supply Chain Security Tools - Scan (Alpha)

This topic describes prerequisites you must complete to install SCST - Scan (Prisma) from the VMware package repository.

Important

This integration is in Alpha, which means that it is still in active development by the Tanzu Practices Global Tech Team and might be subject to change at any point. Users might encounter unexpected behavior.

Verify the latest alpha package version

Run this command to output a list of available tags.

```
imgpkg tag list -i projects.registry.vmware.com/tanzu_practice/tap-scanners-package/pr
isma-repo-scanning-bundle | sort -V
```

Use the latest version returned in place of the sample version in this topic. For example, `0.1.5-alpha.13` in the following output.

```
imgpkg tag list -i projects.registry.vmware.com/tanzu_practice/tap-scanners-package/pr
isma-repo-scanning-bundle | sort -V
0.1.4-alpha.11
0.1.4-alpha.12
0.1.4-alpha.15
0.1.5-alpha.11
0.1.5-alpha.12
0.1.5-alpha.13
```

Relocate images to a registry

VMware recommends relocating the images from VMware Tanzu Network registry to your own container image registry before installing. The Prisma Scanner is in the Alpha development phase, and not packaged as part of Tanzu Application Platform. It is hosted on the VMware Project Repository instead of VMware Tanzu Network. If you relocated the Tanzu Application Platform images, you can also relocate the Prisma Scanner package. If you don't relocate the images, the Prisma Scanner installation depends on VMware Tanzu Network for continued operation, and VMware Tanzu Network offers no uptime guarantees. The option to skip relocation is documented for evaluation and proof-of-concept only.

For information about supported registries, see each registry's documentation.

To relocate images from the VMware Project Registry to your registry:

1. Set up environment variables for installation:

   ```
   export IMGPKG_REGISTRY_HOSTNAME_0=registry.tanzu.vmware.com
   export IMGPKG_REGISTRY_USERNAME_0=MY-TANZUNET-USERNAME
   export IMGPKG_REGISTRY_PASSWORD_0=MY-TANZUNET-PASSWORD
   export INSTALL_REGISTRY_USERNAME=MY-REGISTRY-USER
   export INSTALL_REGISTRY_PASSWORD=MY-REGISTRY-PASSWORD
   export INSTALL_REGISTRY_HOSTNAME=MY-REGISTRY
   export VERSION=VERSION-NUMBER
   export INSTALL_REPO=TARGET-REPOSITORY
   ```

   Where:

   - `MY-REGISTRY-USER` is the user with write access to MY-REGISTRY.
   - `MY-REGISTRY-PASSWORD` is the password for `MY-REGISTRY-USER`.
   - `MY-REGISTRY` is your own registry.
   - `VERSION` is your Prisma Scanner version. For example, `0.1.4-alpha.12`.
   - `TARGET-REPOSITORY` is your target repository, a directory or repository on `MY-REGISTRY` that serves as the location for the installation files for Prisma Scanner.

2. Install the Carvel tool imgpkg CLI. See Deploying Cluster Essentials.
3. Relocate images with the imgpkg CLI by running:

   ```
   imgpkg copy -b projects.registry.vmware.com/tanzu_practice/tap-scanners-packag
   e/prisma-repo-scanning-bundle:${VERSION} --to-repo ${INSTALL_REGISTRY_HOSTNAM
   E}/${INSTALL_REPO}/prisma-repo-scanning-bundle
   ```

Add the Prisma Scanner package repository

Tanzu CLI packages are available on repositories. Adding the Prisma Scanning package repository makes the Prisma Scanning bundle and its packages available for installation.

Note

VMware recommends, but does not require, relocating images to a registry for installation. This section assumes that you relocated images to a registry. See the earlier section to fill in the variables.

VMware recommends installing the Prisma Scanner objects in the existing `tap-install` namespace to keep the Prisma Scanner grouped logically with the other Tanzu Application Platform components.

1. Add the Prisma Scanner package repository to the cluster by running:

```
tanzu package repository add prisma-scanner-repository \
  --url ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/prisma-repo-scanning-bundl
e:$VERSION \
  --namespace tap-install
```

2. Get the status of the Prisma Scanner package repository, and ensure that the status updates to `Reconcile succeeded` by running:

```
tanzu package repository get prisma-scanner-repository --namespace tap-install
```

For example:

```
$ tanzu package repository get prisma-scanning-repository --namespace tap-insta
ll
- Retrieving repository prisma-scanner-repository...
NAME:          prisma-scanner-repository
VERSION:       71091125
REPOSITORY:    projects.registry.vmware.com/tanzu_practice/tap-scanners-packag
e/prisma-repo-scanning-bundle
TAG:           0.1.4-alpha.12
STATUS:        Reconcile succeeded
REASON:
```

3. List the available packages by running:

```
tanzu package available list --namespace tap-install
```

For example:

```
$ tanzu package available list --namespace tap-install
/ Retrieving available packages...
  NAME                                                DISPLAY-NAME
SHORT-DESCRIPTION
  prisma.scanning.apps.tanzu.vmware.com               Prisma for Supply Chain
Security Tools - Scan                              Default scan templates using
Prisma
```

Prepare the Prisma Scanner configuration

Before installing the Prisma scanner, you must create the configuration and a Kubernetes secret that contains credentials to access Prisma Cloud.

Obtain Console URL and Access Keys and Token

The Prisma Scanner supports two methods of authentication:

1) Basic Authentication with API Key and Secret 2) Token Based Authentication

The steps to configure both are outlined to allow you to decide which option you use.

> **Note**
>
> The token method issued by Prisma Cloud has a expiration of 1 hour, so it requires frequent refreshing.

To obtain your Prisma Compute Console URL and Access Keys and Token. See Access keys in the Palo Alto Networks documentation.

> **Note**
>
> Generated tokens expire after an hour.

Access key and secret authentication

To create a Prisma secret, use the following instructions.

1. Create a Prisma secret YAML file and insert the base64 encoded Prisma API token into the `prisma_token`:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: PRISMA-ACCESS-KEY-SECRET
     namespace: APP-NAME
   data:
     username: BASE64-PRISMA-ACCESS-KEY-ID
     password: BASE64-PRISMA-ACCESS-KEY-PASSWORD
   ```

   Where:

   - `PRISMA-ACCESS-KEY-SECRET` is the name of your Prisma token secret.
   - `APP-NAME` is the namespace you want to use.
   - `BASE64-PRISMA-ACCESS-KEY-ID` is your base64 encoded Prisma Access Key ID.
   - `BASE64-PRISMA-ACCESS-KEY-PASSWORD` is your base64 encoded Prisma Access Key Password.

2. Apply the Prisma secret YAML file by running:

   ```
   kubectl apply -f YAML-FILE
   ```

   Where `YAML-FILE` is the name of the Prisma secret YAML file you created.

3. Define the `--values-file` flag to customize the default configuration. You must define the following fields in the `values.yaml` file for the Prisma Scanner configuration. You can add fields to activate or deactivate behaviors. You can append the values to this file as shown later in this topic. Create a `values.yaml` file by using the following configuration:

   ```
   ---
   namespace: DEV-NAMESPACE
   targetImagePullSecret: TARGET-REGISTRY-CREDENTIALS-SECRET
   prisma:
     url: PRISMA-URL
   ```

```
  basicAuth:
    name: PRISMA-ACCESS-KEY-SECRET
```

Where:

- `DEV-NAMESPACE` is your developer namespace.

> **Note**
>
> To use a namespace other than the default namespace, ensure that the namespace exists before you install. If the namespace does not exist, the scanner installation fails.

- `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the secret that contains the credentials to pull an image from a private registry for scanning.
- `PRISMA-URL` is the FQDN of your Twistlock server.
- `PRISMA-CONFIG-SECRET` is the name of the secret you created that contains the Prisma configuration to connect to Prisma. This field is required.

The Prisma integration can work with or without the SCST - Store integration. The `values.yaml` file is slightly different for each configuration.

Access Token Authentication

1. Create a Prisma secret YAML file and insert the base64 encoded Prisma API token into the `prisma_token`:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: PRISMA-TOKEN-SECRET
     namespace: APP-NAME
   data:
     prisma_token: BASE64-PRISMA-API-TOKEN
   ```

   Where:

   - `PRISMA-TOKEN-SECRET` is the name of your Prisma token secret.
   - `APP-NAME` is the namespace you want to use.
   - `BASE64-PRISMA-API-TOKEN` is the name of your base64 encoded Prisma API token.

2. Apply the Prisma secret YAML file by running:

   ```
   kubectl apply -f YAML-FILE
   ```

   Where `YAML-FILE` is the name of the Prisma secret YAML file you created.

3. Define the `--values-file` flag to customize the default configuration. You must define the following fields in the `values.yaml` file for the Prisma Scanner configuration. You can add fields as needed to activate or deactivate behaviors. You can append the values to this file as shown later in this topic. Create a `values.yaml` file by using the following configuration:

   ```
   ---
   namespace: DEV-NAMESPACE
   targetImagePullSecret: TARGET-REGISTRY-CREDENTIALS-SECRET
   prisma:
     url: PRISMA-URL
   ```

```
  tokenSecret:
    name: PRISMA-CONFIG-SECRET
```

Where:

- `DEV-NAMESPACE` is your developer namespace.

> **Note**
>
> To use a namespace other than the default namespace, ensure that the namespace exists before you install. If the namespace does not exist, the scanner installation fails.

- `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the secret that contains the credentials to pull an image from a private registry for scanning.
- `PRISMA-URL` is the FQDN of your Twistlock server.
- `PRISMA-CONFIG-SECRET` is the name of the secret you created that contains the Prisma configuration to connect to Prisma. This field is required.

SCST - Store integration

The Prisma Scanner integration can work with or without the SCST - Store integration. The `values.yaml` file is slightly different for each configuration.

When using SCST - Store integration, to persist the results found by the Prisma Scanner, you can enable the SCST - Store integration by appending the fields to the `values.yaml` file.

The Grype, Snyk, and Prisma Scanner Integrations enable the Metadata Store. To prevent conflicts, the configuration values are slightly different based on whether the Grype Scanner Integration is installed or not. If Tanzu Application Platform is installed using the Full Profile, the Grype Scanner Integration is installed unless it is explicitly excluded.

Multiple Scanners installed

In order to find your CA secret name and authentication token secret name as needed for your values.yaml when installing Prisma Scanner you must look at the configuration of a prior installed scanner in the same namespace as it already exists.

For information about how the scanner was likely initially created, see Set up multicluster Supply Chain Security Tools (SCST) - Store

An example `values.yaml` when there are other scanners already installed in the same `dev-namespace` where the Prisma Scanner is installed:

```
#! ...
metadataStore:
  #! The url where the Store deployment is accessible.
  #! Default value is: "https://metadata-store-app.metadata-store.svc.cluster.local:8443"
  url: "STORE-URL"
  caSecret:
    #! The name of the secret that contains the ca.crt to connect to the Store Deployment.
    #! Default value is: "app-tls-cert"
    name: "CA-SECRET-NAME"
    importFromNamespace: "" #! since both Prisma and Grype/Snyk both enable store, one must leave importFromNamespace blank
  #! authSecret is for multicluster configurations.
  authSecret:
```

```
    #! The name of the secret that contains the auth token to authenticate to the Stor
e Deployment.
    name: "AUTH-SECRET-NAME"
    importFromNamespace: "" #! since both Prisma and Grype/Snyk both enable store, one
must leave importFromNamespace blank
```

Where:

- `STORE-URL` is the URL where the Store deployment is accessible.
- `CA-SECRET-NAME` is the name of the secret that contains the ca.crt to connect to the Store Deployment. Default is `app-tls-cert`.
- `AUTH-SECRET-NAME` is the name of the secret that contains the authentication token to authenticate to the Store Deployment.

Prisma Only Scanner Installed

For information about creating and exporting secrets for the Metadata Store CA and authentication token referenced in the data values when installing Prisma Scanner, see Set up multicluster Supply Chain Security Tools (SCST) - Store.

An example `values.yaml` when no other scanner integrations installed in the same `dev-namespace` where the Prisma Scanner is installed:

```
#! ...
metadataStore:
  #! The url where the Store deployment is accessible.
  #! Default value is: "https://metadata-store-app.metadata-store.svc.cluster.local:84
43"
  url: "STORE-URL"
  caSecret:
    #! The name of the secret that contains the ca.crt to connect to the Store Deploym
ent.
    #! Default value is: "app-tls-cert"
    name: "CA-SECRET-NAME"
    #! The namespace where the secrets for the Store Deployment live.
    #! Default value is: "metadata-store"
    importFromNamespace: "STORE-SECRETS-NAMESPACE"
  #! authSecret is for multicluster configurations.
  authSecret:
    #! The name of the secret that contains the auth token to authenticate to the Stor
e Deployment.
    name: "AUTH-SECRET-NAME"
    #! The namespace where the secrets for the Store Deployment live.
    importFromNamespace: "STORE-SECRETS-NAMESPACE"
```

Where:

- `STORE-URL` is the URL where the Store deployment is accessible.
- `CA-SECRET-NAME` is the name of the secret that contains the `ca.crt` to connect to the Store Deployment. Default is `app-tls-cert`.
- `STORE-SECRETS-NAMESPACE` is the namespace where the secrets for the Store Deployment live. Default is `metadata-store`.
- `AUTH-SECRET-NAME` is the name of the secret that contains the authentication token to authenticate to the Store Deployment.

No Store Integration

If you do not want to enable the SCST - Store integration, explicitly deactivate the integration by appending the following field to the `values.yaml` file that is enabled by default:

```
# ...
metadataStore:
  url: "" # Deactivate Supply Chain Security Tools - Store integration
```

Prepare the ScanPolicy

To prepare the ScanPolicy, use the instructions in the following sections.

Sample ScanPolicy using Prisma Policies

The following sample ScanPolicy allows you to control whether the SupplyChain passes or fails based on the compliance and vulnerability rules configured in the Prisma Compute Console.

The policy reads the `complianceScanPassed` and `vulnerabilityScanPassed` fields returned from Prisma scanner output to control the results of the scan.

```
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: prisma-scan-policy
  labels:
    'app.kubernetes.io/part-of': 'enable-in-gui'
spec:
  regoFile: |
    package main

    import future.keywords.in

    deny[msg] {
      vulnerabilityAndComplianceScanResults := {e | e := input.bom.metadata.propertie
s.property[_]}
      some result in vulnerabilityAndComplianceScanResults
      failedScans:= "false" in result
      failedScans
      vulnerabilityMessages := { message |
        components := {e | e := input.bom.components.component} | {e | e := input.bom.
components.component[_]}
        some component in components
        vulnerabilities := {e | e := component.vulnerabilities.vulnerability} | {e | e
:= component.vulnerabilities.vulnerability[_]}
        some vulnerability in vulnerabilities
        ratings := {e | e := vulnerability.ratings.rating.severity} | {e | e := vulner
ability.ratings.rating[_].severity}
        formattedRatings := concat(", ", ratings)
        message := sprintf("Vulnerability - Component: %s CVE: %s Severity: %s", [comp
onent.name, vulnerability.id, formattedRatings])
      }
      complianceMessages := { message |
        compliances := {e | e := input.bom.metadata.component.compliances.compliance}
| {e | e := input.bom.metadata.component.compliances.compliance[_]}
        some compliance in compliances
        message := sprintf("Compliance - %s \\nId: %s Severity: %s Category: %s", [com
pliance.title, compliance.id, compliance.severity, compliance.category])
      }
      combinedMessages := complianceMessages | vulnerabilityMessages
      some message in combinedMessages
      msg := message
    }
```

Sample ScanPolicy using Local Policies

The following sample ScanPolicy allows you to control whether the SupplyChain passes or fails based on the Prisma Scanner CycloneDX vulnerability results returned from the Prisma Scanner.

```
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: prisma-scan-policy
  labels:
    'app.kubernetes.io/part-of': 'enable-in-gui'
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "UnknownSeve
rity"
    notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]
    ignoreCves := []

    contains(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
      severities := { e | e := match.ratings.rating.severity } | { e | e := match.rati
ngs.rating[_].severity }
      some i
      fails := contains(notAllowedSeverities, severities[i])
      not fails
    }

    isSafe(match) {
      ignore := contains(ignoreCves, match.id)
      ignore
    }

    deny[msg] {
      comps := { e | e := input.bom.components.component } | { e | e := input.bom.comp
onents.component[_] }
      some i
      comp := comps[i]
      vulns := { e | e := comp.vulnerabilities.vulnerability } | { e | e := comp.vulne
rabilities.vulnerability[_] }
      some j
      vuln := vulns[j]
      ratings := { e | e := vuln.ratings.rating.severity } | { e | e := vuln.ratings.r
ating[_].severity }
      not isSafe(vuln)
      msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
    }
```

Apply the YAML:

```
kubectl apply -n $DEV-NAMESPACE -f SCAN-POLICY-YAML
```

Where:

- `DEV-NAMESPACE` is the name of the developer namespace you want to use.
- `SCAN-POLICY-YAML` is the name of your SCST - Scan YAML.

Install Prisma Scanner

After all prerequisites are completed, install the Prisma Scanner. See Install another scanner for Supply Chain Security Tools - Scan.

Self-Signed Registry Certificate

When attempting to pull an image from a registry with a self-signed certificate during image scans additional configuration is necessary.

Tanzu Application Platform Values Shared CA

If your `tap-values.yaml` used during install has the following shared section filled out, Prisma Scanner uses this and enable it to connect to your registry without additional configuration.

```
shared:
   ca_cert_data: | # To be passed if using custom certificates.
      -----BEGIN CERTIFICATE-----
      MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...
      -----END CERTIFICATE-----
```

Secret within Developer Namespace

1. Create a secret that holds the registry's CA certificate data.

   An example of the secret:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
    name: prisma-registry-cert
    namespace: dev
   type: Opaque
   data:
    ca_cert_data: BASE64_CERT
   ```

2. Update your Prisma Scanner install values.yaml.

   Add `caCertSecret` to the root of your `prisma-values.yaml` when installing Prisma Scanner

   Example:

   ```
   namespace: dev
   targetImagePullSecret: tap-registry
   caCertSecret: prisma-registry-cert
   ```

Connect to Prisma through a Proxy

To connect to Prisma through a proxy, you must add `environmentVariables` configuration to your `prisma-values.yaml`.

Note All valid container `env` configurations are supported.

For example:

```
namespace: dev
targetImagePullSecret: tap-registry
environmentVariables:
- name: HTTP_PROXY
  value: "test.proxy.com"
- name: HTTPS_PROXY
  value: "test.proxy.com"
- name: NO_PROXY
  value: "127.0.0.1,.svc,.svc.cluster.local,demo.app"
```

Known Limits

- OpenShift is not supported

Install Trivy for Supply Chain Security Tools - Scan (alpha)

This topic describes how you can install SCST - Scan (Trivy) from the VMware package repository.

Important

This integration is in Alpha, which means that it is still in active development by the Tanzu Practice Global Tech Team and might be subject to change at any point. Users might encounter unexpected behavior.

Verify the latest alpha package version

Run the following command to output a list of available tags.

```
imgpkg tag list -i projects.registry.vmware.com/tanzu_practice/tap-scanners-package/tr
ivy-repo-scanning-bundle | sort -V
```

For example:

```
imgpkg tag list -i projects.registry.vmware.com/tanzu_practice/tap-scanners-package/tr
ivy-repo-scanning-bundle | sort -V

0.1.0-alpha.19
0.1.4-alpha.1
0.1.4-alpha.3
0.1.4-alpha.5
0.1.4-alpha.6
0.1.4-alpha.7
0.1.4-alpha.9
0.1.4-alpha.10
0.1.4-alpha.11
0.1.5-alpha.1
0.1.5-alpha.2
0.1.5-alpha.3
0.1.6-alpha.1
0.1.6-alpha.2
0.1.6-alpha.4
0.1.6-alpha.6
0.1.6-alpha.7
0.1.7-alpha.1
```

Use the latest alpha version available for your version of the Tanzu Application Platform.

For example, given the output above, if you are using TAP 1.4, use `0.1.4-alpha.11`. If you are using TAP 1.7, use `0.1.7-alpha.1`.

Relocate images to a registry

VMware recommends relocating the images from VMware Tanzu Network registry to your own container image registry before installing.

Trivy is in the Alpha development phase, is not packaged as part of the Tanzu Application Platform package, and is hosted on the VMware Project Repository instead of VMware Tanzu Network. If

you relocated the Tanzu Application Platform images, you might also want to relocate the Trivy package.

If you don't relocate the images, Trivy installation depends on VMware Tanzu Network for continued operation, and VMware Tanzu Network offers no uptime guarantees. The option to skip relocation is documented for evaluation and proof-of-concept only.

For information about supported registries, see the registry's documentation.

To relocate images from the VMware Project Registry to your registry:

1. Install Docker if it is not already installed.
2. Set up environment variables for installation by running:

   ```
   export INSTALL_REGISTRY_USERNAME=MY-REGISTRY-USER
   export INSTALL_REGISTRY_PASSWORD=MY-REGISTRY-PASSWORD
   export INSTALL_REGISTRY_HOSTNAME=MY-REGISTRY
   export VERSION=VERSION-NUMBER
   export INSTALL_REPO=TARGET-REPOSITORY
   ```

   Where:

   - `MY-REGISTRY-USER` is the user with write access to MY-REGISTRY.
   - `MY-REGISTRY-PASSWORD` is the password for `MY-REGISTRY-USER`.
   - `MY-REGISTRY` is your own registry.
   - `VERSION` is your Trivy version. For example, `0.1.4-alpha.6`.
   - `TARGET-REPOSITORY` is your target repository, a directory or repository on `MY-REGISTRY` that serves as the location for the installation files for Trivy.
3. Install the Carvel tool imgpkg CLI. See Deploying Cluster Essentials.
4. Relocate the images with the imgpkg CLI by running:

   ```
   imgpkg copy -b projects.registry.vmware.com/tanzu_practice/tap-scanners-packag
   e/trivy-repo-scanning-bundle:${VERSION} --to-repo ${INSTALL_REGISTRY_HOSTNAME}/
   ${INSTALL_REPO}/trivy-repo-scanning-bundle
   ```

Note

The VMware project repository does not require authentication, so you don't need to perform a Docker login.

Add Trivy package repository

Tanzu CLI packages are available on repositories. Adding the Trivy scanning package repository makes the Trivy scanning bundle and its packages available for installation.

Note

VMware recommends, but does not require, relocating images to a registry for installation. The following section requires that you relocated images to a registry. See the earlier section to fill in the variables.

VMware recommends installing Trivy objects in the existing `tap-install` namespace to keep Trivy grouped logically with the other Tanzu Application Platform components.

1. Add Trivy package repository to the cluster by running:

```
tanzu package repository add trivy-scanner-repository \
--url ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/trivy-repo-scanning-bundl
e:$VERSION \
--namespace tap-install
```

2. Get the status of Trivy package repository, and ensure that the status updates to `Reconcile succeeded` by running:

```
tanzu package repository get trivy-scanner-repository --namespace tap-install
```

For example:

```
tanzu package repository get trivy-scanner-repository --namespace tap-install

NAME:          trivy-scanner-repository
VERSION:       7750726
REPOSITORY:    projects.registry.vmware.com/tanzu_practice/tap-scanners-packag
e/trivy-repo-scanning-bundle
TAG:           0.1.4-alpha.6
STATUS:        Reconcile succeeded
REASON:
```

3. List the available packages by running:

```
tanzu package available list --namespace tap-install
```

For example:

```
$ tanzu package available list --namespace tap-install
/ Retrieving available packages...
NAME                                                DISPLAY-NAME
SHORT-DESCRIPTION
trivy.scanning.apps.tanzu.vmware.com                  trivy
Default scan templates using Trivy
```

Prepare Trivy configuration

Before installing the Trivy, you must create the configuration necessary to install Trivy.

1. Define the `--values-file` flag to customize the default configuration. You must define the following fields in the `values.yaml` file for the Trivy Scanner configuration. You can add fields as needed to activate or deactivate behaviors. You can append the values to the `values.yaml` file. Create a `values.yaml` file by using the following configuration:

```
---
namespace: DEV-NAMESPACE
targetImagePullSecret: TARGET-REGISTRY-CREDENTIALS-SECRET
targetSourceSshSecret: TARGET-SOURCE-SSH-SECRET
```

Where:

- `DEV-NAMESPACE` is your developer namespace.

Note

> To use a namespace other than the default namespace, ensure that the namespace exists before you install. If the namespace does not exist, the scanner installation fails.

- `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the secret that contains the credentials to pull an image from a private registry for scanning.
- `TARGET-SOURCE-SSH-SECRET` is the name of the secret containing SSH credentials for cloning private repositories

2. To see all available values, run the following command using the version that you want:

```
VERSION="0.1.4-alpha.6"
tanzu package available get trivy.scanning.apps.tanzu.vmware.com/$VERSION --val
ues-schema -n tap-install
```

Example output:

```
KEY                                          DEFAULT
TYPE    DESCRIPTION
environmentVariables                         <nil>
<nil>   Environment Variables you want added to the scan container to impact tr
ivy behavior
resources.limits.cpu                         1000m
string  Limits describes the maximum amount of cpu resources allowed.
resources.requests.cpu                       250m
string  Requests describes the minimum amount of cpu resources required.
resources.requests.memory                    128Mi
string  Requests describes the minimum amount of memory resources
scanner.docker.server
string  <nil>
scanner.docker.username
string  <nil>
scanner.docker.password
string  <nil>
scanner.pullSecret
string  <nil>
scanner.serviceAccount                       trivy-scanner
string  Name of scan pod's service ServiceAccount
scanner.serviceAccountAnnotations            <nil>
<nil>   Annotations added to ServiceAccount
trivy.cli.image.additionalArguments
string  additional arguments to be appended to the image scan command
trivy.cli.plugins.aqua.repositoryUrl
string  location of the aqua plugin tar in an OCI registry to be used in place
of the embedded version
trivy.cli.repositoryUrl
string  location of the CLI tar in an OCI registry to be used in place of the e
mbedded version
trivy.cli.source.additionalArguments
string  additional arguments to be appended to the fs scan command
trivy.db.repositoryUrl
string  location of the vulnerability database in an OCI registry to be used as
the download location prior to running a scan
caCertSecret
string  Reference to the secret containing the registry ca cert set as ca_cert_
data
metadataStore.authSecret.importFromNamespace
string  Namespace from which to import the Insight Metadata Store auth_token
metadataStore.authSecret.name
string  Name of deployed Secret with key auth_token
metadataStore.caSecret.importFromNamespace    metadata-store
string  Namespace from which to import the Insight Metadata Store CA Cert
```

```
metadataStore.caSecret.name                    app-tls-cert
string  Name of deployed Secret with key ca.crt holding the CA Cert of the Insi
ght Metadata Store
metadataStore.clusterRole                      metadata-store-read-write
string  Name of the deployed ClusterRole for read/write access to the Insight M
etadata Store deployed in the same cluster
metadataStore.url                              https://metadata-store-app.metada
ta-store.svc.cluster.local:8443  string  Url of the Insight Metadata Store
namespace                                      default
string  Deployment namespace for the Scan Templates
targetImagePullSecret
string  Reference to the secret used for pulling images from private registry
targetSourceSshSecret
string  Reference to the secret containing SSH credentials for cloning private
repositories
```

SCST - Store integration

Trivy integration can work with or without the SCST - Store integration. The `values.yaml` file is slightly different for each configuration.

To persist the results found by the Trivy, enable the SCST - Store integration by appending the SCST- scan fields to Trivy`values.yaml` file.

The Grype, Snyk, Prisma, Carbon Black, and Trivy integrations enable the Metadata Store. To prevent conflicts, the configuration values are slightly different based on whether another scanner integration is installed or not. If Tanzu Application Platform is installed using the Full Profile, the Grype Scanner Integration is installed unless it is explicitly excluded.

Multiple scanners installed

When installing Trivy, find your CA secret name and authentication token secret name for your `values.yaml` ny looking at the configuration of a prior installed scanner in the same namespace as it already exists.

For information about how the scanner was initially created, see Set up multicluster Supply Chain Security Tools (SCST) - Store.

The following example `values.yaml` has other scanners already installed in the same `dev-namespace` where Trivy is installed:

```
#! ...
metadataStore:
  #! The url where the Store deployment is accessible.
  #! Default value is: "https://metadata-store-app.metadata-store.svc.cluster.local:84
43"
  url: "STORE-URL"
  caSecret:
    #! The name of the secret that contains the ca.crt to connect to the Store Deploym
ent.
    #! Default value is: "app-tls-cert"
    name: "CA-SECRET-NAME"
    importFromNamespace: "" #! since both Trivy and Grype/Snyk both enable store, one
must leave importFromNamespace blank
  #! authSecret is for multicluster configurations.
  authSecret:
    #! The name of the secret that contains the auth token to authenticate to the Stor
e Deployment.
    name: "AUTH-SECRET-NAME"
    importFromNamespace: "" #! since both Trivy and Grype/Snyk both enable store, one
must leave importFromNamespace blank
```

Where:

- `STORE-URL` is the URL where the Store deployment is accessible.
- `CA-SECRET-NAME` is the name of the secret that contains the ca.crt to connect to the Store Deployment. Default is `app-tls-cert`.
- `AUTH-SECRET-NAME` is the name of the secret that contains the authentication token to authenticate to the Store Deployment.

Trivy is the only scanner installed

For a walk through of creating and exporting secrets for the Metadata Store CA and authentication token which referenced in the data values, see Set up multicluster Supply Chain Security Tools (SCST) - Store.

The following example `values.yaml` has no other scanner integrations installed in the same `dev-namespace` where Trivy is installed:

```
#! ...
metadataStore:
  #! The url where the Store deployment is accessible.
  #! Default value is: "https://metadata-store-app.metadata-store.svc.cluster.local:84
43"
  url: "STORE-URL"
  caSecret:
    #! The name of the secret that contains the ca.crt to connect to the Store Deploym
ent.
    #! Default value is: "app-tls-cert"
    name: "CA-SECRET-NAME"
    #! The namespace where the secrets for the Store Deployment live.
    #! Default value is: "metadata-store"
    importFromNamespace: "STORE-SECRETS-NAMESPACE"
  #! authSecret is for multicluster configurations.
  authSecret:
    #! The name of the secret that contains the auth token to authenticate to the Stor
e Deployment.
    name: "AUTH-SECRET-NAME"
    #! The namespace where the secrets for the Store Deployment live.
    importFromNamespace: "STORE-SECRETS-NAMESPACE"
```

Where:

- `STORE-URL` is the URL where the Store deployment is accessible.
- `CA-SECRET-NAME` is the name of the secret that contains the `ca.crt` to connect to the Store Deployment. Default is `app-tls-cert`.
- `STORE-SECRETS-NAMESPACE` is the namespace where the secrets for the Store Deployment live. Default is `metadata-store`.
- `AUTH-SECRET-NAME` is the name of the secret that contains the authentication token to authenticate to the Store Deployment.

No store integration

If you do not want to enable the SCST - Store integration, deactivate the integration by appending the following field to the `values.yaml` file that is enabled by default:

```
# ...
metadataStore:
  url: "" # Deactivate Supply Chain Security Tools - Store integration
```

Prepare the ScanPolicy

The following sample ScanPolicy allows you to control whether the SupplyChain passes or fails based on the CycloneDX vulnerability results returned from the Trivy.

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
   name: trivy-scan-policy
   labels:
      app.kubernetes.io/part-of: enable-in-gui
spec:
   regoFile: |
      package main

      import future.keywords.in
      import future.keywords.every

      # Accepted Values: "critical", "high", "medium", "low", unknown"
      notAllowedSeverities := ["critical", "high", "unknown"]
      notAllowedSet := {x | x := notAllowedSeverities[_]}
      ignoreCves := []

      isSafe(match) {
        severities := { e | e := match.ratings.rating.severity } | { e | e := match.ra
tings.rating[_].severity }
        every severity in severities {
            not severity in notAllowedSet
        }
      }

      isIgnored(match) {
        match.id in ignoreCves
      }

      deny[msg] {
        notAllowedVulnerabilities := { vulnerability |
          vulnerabilities := {e | e := input.bom.vulnerabilities.vulnerability} | {e |
e := input.bom.vulnerabilities.vulnerability[_]}
          some vulnerability in vulnerabilities
          not isIgnored(vulnerability)
          not isSafe(vulnerability)
        }
        formattedVulnerabilityMessages := { message |
          some vulnerability in notAllowedVulnerabilities
          ratings := {e | e := vulnerability.ratings.rating.severity} | {e | e := vuln
erability.ratings.rating[_].severity}
          formattedRatings := concat(", ", ratings)
          affectedComponents := {e | e := vulnerability.affects.target.ref} | {e | e :
= vulnerability.affects.target[_].ref}
          formattedComponents := concat("\\n", affectedComponents)
          message = sprintf("CVE: %s \\nRatings: %s\\nAffected Components: \\n%s", [vu
lnerability.id, formattedRatings, formattedComponents])
        }
        some formattedVulnerabilityMessage in formattedVulnerabilityMessages
        msg := formattedVulnerabilityMessage
      }
```

To prepare the ScanPolicy:

1. Apply the following to the YAML:

   ```
   kubectl apply -n $DEV-NAMESPACE -f SCAN-POLICY-YAML
   ```

Where:

- `DEV-NAMESPACE` is the name of the developer namespace you want to use.
- `SCAN-POLICY-YAML` is the name of your SCST - Scan YAML.

Install Trivy

After the following prerequisites are completed, install the Trivy:

- Prerequisites listed in Install another scanner for Supply Chain Security Tools - Scan.
- Install the ORAS CLI. See the ORAS documentation.

Air-gap configuration

This section explains how to configure Trivy in an air-gapped environment.

For information about additional flags and configuration, see Air-Gapped Environment in the Trivy documentation.

Relocate a Trivy database to your registry

Note

Using a relocated database means you are taking responsibility for keeping it up to date to ensure that security scans are relevant. Stale databases weaken your security posture.

If you have a host with access, you can use the ORAS CLI to perform a copy.

```
oras copy -r ghcr.io/aquasecurity/trivy-db:2 registry.company.com/project_name/trivy-d
b:2 # the tag of 2 is required

Copying 4a39b38cf2fd db.tar.gz
Copied  4a39b38cf2fd db.tar.gz
Copied ghcr.io/aquasecurity/trivy-db:2 => registry.company.com/project_name/trivy-db:2
Digest: sha256:ed57874a80499e858caac27fc92e4952346eb75a2774809ee989bcd2ce48897a
```

If not, you can use the ORAS CLI to download the database and manifest and then push to your registry.

1. Download the trivy-db.

   ```
   oras pull ghcr.io/aquasecurity/trivy-db:2
   ```

   Example output:

   ```
   oras pull ghcr.io/aquasecurity/trivy-db:2
   Downloading 1612cc15d377 db.tar.gz
   Downloaded  1612cc15d377 db.tar.gz
   Pulled ghcr.io/aquasecurity/trivy-db:2
   Digest: sha256:af903c7ddbe7516f18b06254b6297cf53c0ece918def07322925c71d2f694860
   ```

2. Download the manifest for trivy-db.

   ```
   oras manifest fetch ghcr.io/aquasecurity/trivy-db:2 > trivy-db-manifest.json
   ```

3. Add the media type to the manifest.

```
jq '.mediaType="application/vnd.oci.image.manifest.v1+json"' trivy-db-manifest.
json > updated-trivy-db-manifest.json
```

4. Push the prior downloaded trivy-db to your registry.

   ```
   oras push registry.company.com/project_name/trivy-db:2 ./db.tar.gz
   ```

   Example output:

   ```
   oras push registry.company.com/project_name/trivy-db:2 \
   ./db.tar.gz

   Uploading 1612cc15d377 db.tar.gz
   Uploaded  1612cc15d377 db.tar.gz
   Pushed registry.company.com/project_name/trivy-db:2
   Digest: sha256:41a7eeab8837e90d8a5afd56cfce73936e15d3db04c5294f992ecff9492971dc
   ```

5. Push the updated trivy-db manifest to your registry

   ```
   oras manifest push registry.company.com/project_name/trivy-db:2 updated-trivy-d
   b-manifest.json
   ```

   Example output:

   ```
   oras manifest push registry.company.com/project_name/trivy-db:2 updated-trivy-d
   b-manifest.json
   Pushed registry.company.com/project_name/trivy-db:2
   Digest: sha256:b51a2fccf38e723aac1a7217ba36ca52398b2b20e3d74c9d5089dfdcd9bb2f11
   ```

6. Cleanup files

   ```
   rm trivy-db-manifest.json updated-trivy-db-manifest.json db.tar.gz
   ```

7. Update data values with the database repository URL. Edit your `values.yaml` to add the following:

   ```
   trivy:
   db:
     repositoryUrl: "registry.company.com/project_name/trivy-db"
   ```

   The URL leaves off the tag of `2`.

Use another Trivy version

This section describes how to use a different Trivy CLI version than what is bundled with the package.

To use another Trivy version:

1. Install the ORAS CLI. See the ORAS documentation.
2. Download the version of the CLI you are interested in from their GitHub releases page. For example:
   https://github.com/aquasecurity/trivy/releases/download/v0.36.0/trivy_0.36.0_Linux-64bit.tar.gz

   ```
   wget -c https://github.com/aquasecurity/trivy/releases/download/v0.36.0/trivy_
   0.36.0_Linux-64bit.tar.gz -O trivy.tar.gz
   Length: 48363295 (46M) [application/octet-stream]
   Saving to: 'trivy.tar.gz'
   ```

```
trivy.tar.gz 100%[==>]  46.12M  50.7MB/s    in 0.9s

2023-01-25 10:47:55 (50.7 MB/s) - 'trivy.tar.gz' saved [48363295/48363295]
```

3. Relocate the CLI to your registry.

   Run the following to relocate the CLI to your registry:

```
oras push registry.company.com/project_name/trivy-cli:0.36.0 \
--artifact-type trivy/cli \
./trivy.tar.gz:application/gzip

Uploading 121f4d8282aa trivy.tar.gz
Uploaded  121f4d8282aa trivy.tar.gz
Pushed registry.company.com/project_name/trivy-cli:0.36.0
Digest: sha256:5bdb18378e8f66a72f4bef4964edeccfcc2f21883e7a6caca6dbf7a3d7233696
```

4. Edit your `values.yaml` to add the location of your CLI.

```
trivy:
cli:
  repositoryUrl: "registry.company.com/project_name/trivy-cli:0.36.0"
```

Use another Trivy Aqua plug-in version

Trivy Aqua plug-in enables Aqua SaaS integration with your Trivy scans.

To use another Trivy Aqua plug-in version:

1. Install the ORAS CLI. See the ORAS documentation.
2. Download the version of Trivy Aqua plug-in you want from the GitHub releases page. See trivy-plugin-aqua in GitHub.

   For example, v0.115.14 in GitHub:

```
TRIVY-AQUA-PLUGIN-VERSION="v0.115.6"
wget -c "https://github.com/aquasecurity/trivy-plugin-aqua/releases/download/
${TRIVY-AQUA-PLUGIN-VERSION}/linux_amd64_${TRIVY-AQUA-PLUGIN-VERSION}.tar.gz" -
O trivy-aqua-plugin.tar.gz

--2023-01-30 10:44:05--  https://github.com/aquasecurity/trivy-plugin-aqua/rele
ases/download/v0.115.6/linux_amd64_v0.115.6.tar.gz
HTTP request sent, awaiting response... 200 OK
Length: 50915539 (49M) [application/octet-stream]
Saving to: 'trivy-aqua-plugin.tar.gz'

trivy-aqua-plugin.tar.gz 100%[==>]  48.56M  35.3MB/s    in 1.4s

2023-01-30 10:44:07 (35.3 MB/s) - 'trivy-aqua-plugin.tar.gz' saved [50915539/50
915539]
```

3. The YAML file is a necessary component to tell Trivy it has the plug-in already installed. Download the plugin.yml file associated with Trivy Aqua plug-in version you downloaded.

```
TRIVY-AQUA-PLUGIN-VERSION="v0.115.6"
wget -c "https://raw.githubusercontent.com/aquasecurity/trivy-plugin-aqua/${TRI
VY-AQUA-PLUGIN-VERSION}/plugin.yaml" -O plugin.yaml

--2023-01-30 10:46:32--  https://raw.githubusercontent.com/aquasecurity/trivy-p
lugin-aqua/v0.115.6/plugin.yaml
HTTP request sent, awaiting response... 200 OK
```

```
Length: 909 [text/plain]
Saving to: ‘plugin.yaml’
plugin.yaml 100%[==>]     909  --.-KB/s    in 0s

2023-01-30 10:46:32 (54.2 MB/s) - ‘plugin.yaml’ saved [909/909]
```

4. Relocate the plug-in and YAML to your registry:

```
TRIVY-AQUA-PLUGIN-VERSION="v0.115.6"
REPOSITORY-URL="registry.company.com/project_name/trivy-aqua-plugin:$TRIVY-AQUA
-PLUGIN-VERSION"

oras push ${REPOSITORY-URL} \
--artifact-type trivy/aqua-plugin \
./trivy-aqua-plugin.tar.gz:application/gzip \
./plugin.yaml:text/yaml

Uploading 6fb65adbfde2 plugin.yaml
Uploading 7340855e31ff trivy-aqua-plugin.tar.gz
Uploaded  6fb65adbfde2 plugin.yaml
Uploaded  7340855e31ff trivy-aqua-plugin.tar.gz
Pushed registry.company.com/project_name/trivy-aqua-plugin:v0.115.6
Digest: sha256:791274e44b97fad98edf570205fddc1b0bc21c56d3d54565ad9475fd4da969ae
```

   Where:

   - `TRIVY-AQUA-PLUGIN-VERSION` is the version of Trivy Aqua plug-in you are using.
   - `REPOSITORY-URL` is the repository where you want to relocate the plug-in.

5. Edit your `values.yaml` to add the location of your CLI.

```
trivy:
plugins:
  aqua:
    repositoryUrl: "registry.company.com/project_name/trivy-aqua-plugin:v0.115.
6"
```

Integrate with the Aqua SaaS platform

To integrate with the Aqua SaaS platform:

1. In order to connect to the SaaS Platform you must have an API key. To create an API key:
   1. Log into Aqua SaaS.
   2. Enter CSPM.
   3. Click **Settings** -> **API Keys**.
   4. Click **Generate Key**.
   5. Save the information for the next steps.
2. To integrate with the Aqua SaaS Platform you must have an API key. You pass this to the scanner through environment variables, referenced in a secret. Create an auth secret:

   Example secret:

```
apiVersion: v1
kind: Secret
metadata:
name: aqua-creds
namespace: APP-NAMESPACE
stringData:
```

```
aqua-key: API-KEY
aqua-secret: API-KEY-SECRET
```

Where:

- `APP-NAMESPACE` is the developer namespace your app uses.
- `API-KEY` is the Aqua Platform API key.
- `API-KEY-SECRET` is the Aqua Platform API key's Secret.

3. Set environment variables to tell Trivy to connect and report to Aqua SaaS.

You can find plug-in options in the README.md in GitHub.

Here is an example of referencing your API key and secret from a Kubernetes Secret created earlier:

```
namespace: dev
targetImagePullSecret: registry-credentials
environmentVariables:
  - name: TRIVY-RUN-AS-PLUGIN
    value: aqua
  - name: AQUA-KEY
    valueFrom:
      secretKeyRef:
        name: aqua-creds
        key: aqua-key
  - name: AQUA-SECRET
    valueFrom:
      secretKeyRef:
        name: aqua-creds
        key: aqua-secret
```

Where:

- `TRIVY-RUN-AS-PLUGIN` is the Trivy plug-in you want to enable without using the subcommand.
- `AQUA-KEY` is the Aqua Platform API key.
- `AQUA-SECRET` is the Aqua Platform API key's Secret.

Self-signed registry certificate

You need additional configuration when attempting to pull an image from a registry with a self-signed certificate during image scans.

1. If your `tap-values.yaml` used during install has the following shared section filled out, Trivy uses this to connect to your registry without additional configuration. Use the following YAML with a Tanzu Application Platform values shared CA:

   ```
   shared:
   ca_cert_data: | # To be passed if using custom certificates.
       -----BEGIN CERTIFICATE-----
       MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...
       -----END CERTIFICATE-----
   ```

2. Create a secret that holds the registry's CA certificate data.

   An example secret:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
   ```

```
name: trivy-registry-cert
namespace: dev
type: Opaque
data:
ca_cert_data: BASE64_CERT
```

3. Update your Trivy install `trivy-values.yaml`.

   Add `caCertSecret` to the root of your `trivy-values.yaml`.

   For example:

```
namespace: dev
targetImagePullSecret: tap-registry
caCertSecret: trivy-registry-cert
```

Spec reference

This topic describes the specifications and custom resources you can use with Supply Chain Security Tools - Scan.

With the Scan Controller and Grype Scanner installed the following Custom Resource Definitions (CRDs) are now available:

```
$ kubectl get crds | grep scanning.apps.tanzu.vmware.com
imagescans.scanning.apps.tanzu.vmware.com                2021-09-09T15:22:07Z
scanpolicies.scanning.apps.tanzu.vmware.com              2021-09-09T15:22:07Z
scantemplates.scanning.apps.tanzu.vmware.com             2021-09-09T15:22:07Z
sourcescans.scanning.apps.tanzu.vmware.com               2021-09-09T15:22:07Z
```

For more information about installing SCST - Scan, see Installing Individual Packages.

About source and image scans

Both SourceScan (`sourcescans.scanning.apps.tanzu.vmware.com`) and ImageScan (`imagescans.scanning.apps.tanzu.vmware.com`) define what will be scanned, and ScanTemplate (`scantemplates.scanning.apps.tanzu.vmware.com`) will define how to run a scan. We have provided five custom resources (CRs) pre-installed for use. You can either use them as-is or as samples to create your own.

To view the pre-installed Scan Template CRs, run:

```
kubectl get scantemplates
```

You will see the following scan templates:

CR Name	Use Case
`public-source-scan-template`	Clones and scans source code from a public repository.
`private-source-scan-template`	Connects with SSH credentials to clone and scan source code from a private repository.
`public-image-scan-template`	Pulls and scans images from a public registry.
`private-image-scan-template`	Connects with the registry credentials to pull and scan images from a private registry.
`blob-source-scan-template`	To be used in a Supply Chain. Gets a `.tar.gz` available file with `wget`, uncompresses it, and scans the source code inside it.

By default, three scan templates are deployed (`public-source-scan-template`, `public-image-scan-template`, and `blob-source-scan-template`).

If `targetImagePullSecret` is set in `tap-values.yaml`, `private-image-scan-template` is also deployed. If `targetSourceSshSecret` is set in `tap-values.yaml`, `private-source-scan-template` is also deployed.

The private scan templates reference secrets created using the Docker server and credentials you provided, which means they are ready to use immediately.

For more information about the `SourceScan` and `ImageScan` CRDs and how to customize your own, refer to Configuring Code Repositories and Image Artifacts to be Scanned.

About policy enforcement around vulnerabilities found

The Scan Controller supports policy enforcement by using an Open Policy Agent (OPA) engine. ScanPolicy (`scanpolicies.scanning.apps.tanzu.vmware.com`) allows scan results to be validated for company policy compliance and can prevent source code from being built or images from being deployed.

For more information, see Configuring Policy Enforcement using Open Policy Agent (OPA).

Scan samples for Supply Chain Security Tools - Scan

This section provides samples on multiple use cases for SCST - Scan that you can copy to your cluster for testing purposes.

- Running a sample public image scan with compliance check
- Running a sample public source scan with compliance check
- Running a sample private image scan
- Running a sample private source scan
- Running a sample public source scan of a blob/tar file

Scan samples for Supply Chain Security Tools - Scan

This section provides samples on multiple use cases for SCST - Scan that you can copy to your cluster for testing purposes.

- Running a sample public image scan with compliance check
- Running a sample public source scan with compliance check
- Running a sample private image scan
- Running a sample private source scan
- Running a sample public source scan of a blob/tar file

Sample public image scan with compliance check for Supply Chain Security Tools - Scan

This topic includes an example public image scan with compliance check for SCST - Scan.

Public image scan

The following example performs an image scan on an image in a public registry. This image revision has 223 known vulnerabilities (CVEs), spanning a number of severities. ImageScan uses the

ScanPolicy to run a compliance check against the CVEs.

The policy in this example is set to only consider `Critical` severity CVEs as a violation, which returns 21 Critical Severity Vulnerabilities.

Note

This example ScanPolicy is deliberately constructed to showcase the features available and must not be considered an acceptable base policy.

In this example, the scan does the following:

- Finds all 223 of the CVEs
- Ignores any CVEs with severities that are not critical
- Indicates in the `Status.Conditions` that 21 CVEs have violated policy compliance

Define the ScanPolicy and ImageScan

Create `sample-public-image-scan-with-compliance-check.yaml`:

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: sample-scan-policy
  labels:
    'app.kubernetes.io/part-of': 'enable-in-gui'
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "UnknownSeve
rity"
    notAllowedSeverities := ["Critical"]
    ignoreCves := []

    contains(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
      severities := { e | e := match.ratings.rating.severity } | { e | e := match.rati
ngs.rating[_].severity }
      some i
      fails := contains(notAllowedSeverities, severities[i])
      not fails
    }

    isSafe(match) {
      ignore := contains(ignoreCves, match.id)
      ignore
    }

    deny[msg] {
      comps := { e | e := input.bom.components.component } | { e | e := input.bom.comp
onents.component[_] }
      some i
      comp := comps[i]
      vulns := { e | e := comp.vulnerabilities.vulnerability } | { e | e := comp.vulne
rabilities.vulnerability[_] }
      some j
      vuln := vulns[j]
```

```
      ratings := { e | e := vuln.ratings.rating.severity } | { e | e := vuln.ratings.r
ating[_].severity }
      not isSafe(vuln)
      msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
    }

---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ImageScan
metadata:
  name: sample-public-image-scan-with-compliance-check
spec:
  registry:
    image: "nginx:1.16"
  scanTemplate: public-image-scan-template
  scanPolicy: sample-scan-policy
```

(Optional) Set up a watch

Before deploying the resources to a user specified namespace, set up a watch in another terminal to view the progression:

```
watch kubectl get sourcescans,imagescans,pods,taskruns,scantemplates,scanpolicies -n D
EV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

For more information about setting up a watch, see Observing and Troubleshooting.

Deploy the resources

```
kubectl apply -f sample-public-image-scan-with-compliance-check.yaml -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

View the scan results

```
kubectl describe imagescan sample-public-image-scan-with-compliance-check -n DEV-NAMES
PACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

Note

The `Status.Conditions` includes a `Reason: EvaluationFailed` and `Message: Policy violated because of 21 CVEs`.

For more information about scan status conditions, see Viewing and Understanding Scan Status Conditions.

Edit the ScanPolicy

To edit the Scan Policy, see Step 5: Sample Public Source Code Scan with Compliance Check.

Clean up

To clean up, run:

```
kubectl delete -f sample-public-image-scan-with-compliance-check.yaml -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

Sample public source code scan with compliance check for Supply Chain Security Tools - Scan

This topic includes an example public source code scan with compliance check for SCST - Scan.

Public source scan

This example performs a source scan on a public repository. The source revision has 192 known Common Vulnerabilities and Exposures (CVEs), spanning several severities. SourceScan uses the ScanPolicy to run a compliance check against the CVEs.

The example policy is set to only consider `Critical` severity CVEs as violations, which returns 7 Critical Severity Vulnerabilities.

Note

This example ScanPolicy is deliberately constructed to showcase the features available and must not be considered an acceptable base policy.

For this example, the scan (at the time of writing):

- Finds all 192 of the CVEs.
- Ignores any CVEs that have severities that are not critical.
- Indicates in the `Status.Conditions` that 7 CVEs have violated policy compliance.

Run an example public source scan

To perform an example source scan on a public repository:

1. Create `sample-public-source-scan-with-compliance-check.yaml` to define the ScanPolicy and SourceScan:

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: sample-scan-policy
  labels:
    'app.kubernetes.io/part-of': 'enable-in-gui'
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "Unkn
ownSeverity"
    notAllowedSeverities := ["Critical"]
    ignoreCves := []

    contains(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
      severities := { e | e := match.ratings.rating.severity } | { e | e := mat
```

```
ch.ratings.rating[_].severity }
      some i
      fails := contains(notAllowedSeverities, severities[i])
      not fails
    }

    isSafe(match) {
      ignore := contains(ignoreCves, match.id)
      ignore
    }

    deny[msg] {
      comps := { e | e := input.bom.components.component } | { e | e := input.b
om.components.component[_] }
      some i
      comp := comps[i]
      vulns := { e | e := comp.vulnerabilities.vulnerability } | { e | e := com
p.vulnerabilities.vulnerability[_] }
      some j
      vuln := vulns[j]
      ratings := { e | e := vuln.ratings.rating.severity } | { e | e := vuln.ra
tings.rating[_].severity }
      not isSafe(vuln)
      msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
    }

---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: SourceScan
metadata:
  name: sample-public-source-scan-with-compliance-check
spec:
  git:
    url: "https://github.com/houndci/hound.git"
    revision: "5805c650"
  scanTemplate: public-source-scan-template
  scanPolicy: sample-scan-policy
```

2. (Optional) Before deploying the resources to a user specified namespace, set up a watch in another terminal to view the progression:

   ```
   watch kubectl get sourcescans,imagescans,pods,taskruns,scantemplates,scanpolici
   es -n DEV-NAMESPACE
   ```

   Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

   For more information, see Observing and Troubleshooting.

3. Deploy the resources by running:

   ```
   kubectl apply -f sample-public-source-scan-with-compliance-check.yaml -n DEV-NA
   MESPACE
   ```

   Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

4. When the scan completes, view the results by running:

   ```
   kubectl describe sourcescan sample-public-source-scan-with-compliance-check -n
   DEV-NAMESPACE
   ```

   The `Status.Conditions` includes a `Reason: EvaluationFailed` and `Message: Policy violated because of 7 CVEs`. For more information, see Viewing and Understanding Scan Status Conditions.

5. If the failing CVEs are acceptable or the build must be deployed regardless of these CVEs, the app is patched to remove the vulnerabilities. Update the `ignoreCVEs` array in the ScanPolicy to include the CVEs to ignore:

```
...
spec:
  regoFile: |
    package policies

    default isCompliant = false

    # Accepted Values: "UnknownSeverity", "Critical", "High", "Medium", "Low",
"Negligible"
    violatingSeverities := ["Critical"]
    # Adding the failing CVEs to the ignore array
    ignoreCVEs := ["CVE-2018-14643", "GHSA-f2jv-r9rf-7988", "GHSA-w457-6q6x-cgp
9", "CVE-2021-23369", "CVE-2021-23383", "CVE-2020-15256", "CVE-2021-29940"]
...
```

6. The changes applied to the new ScanPolicy trigger the scan to run again. Reapply the resources by running:

```
kubectl apply -f sample-public-source-scan-with-compliance-check.yaml -n DEV-NA
MESPACE
```

7. Re-describe the SourceScan CR by running:

```
kubectl describe sourcescan sample-public-source-scan-with-compliance-check -n
DEV-NAMESPACE
```

8. Ensure that `Status.Conditions` now includes a `Reason: EvaluationPassed` and `No CVEs were found that violated the policy`. You can update the `violatingSeverities` array in the ScanPolicy if you want. For reference, the Grype scan returns the following Severity spread of vulnerabilities:
    - Critical: 7
    - High: 88
    - Medium: 92
    - Low: 5
    - Negligible: 0
    - UnknownSeverity: 0
9. Clean up by running:

```
kubectl delete -f sample-public-source-scan-with-compliance-check.yaml -n DEV-N
AMESPACE
```

Sample private image scan for Supply Chain Security Tools - Scan

This example describes how you can perform a scan against an image located in a private registry for SCST - Scan.

Define the resources

Set up target image pull secret

1. Confirm that target image secret is configured. This is completed during Tanzu Application Platform installation. If the target image secret exists, see Create the private image scan.
2. If the target image secret was not configured, create a secret containing the credentials used to pull the target image you want to scan. For information about secret creation, see the Kubernetes documentation.

   ```
   kubectl create secret docker-registry TARGET-REGISTRY-CREDENTIALS-SECRET \
   --docker-server=YOUR-REGISTRY-SERVER \
   --docker-username=YOUR-NAME \
   --docker-password=YOUR-PASSWORD \
   --docker-email=YOUR-EMAIL \
   -n DEV-NAMESPACE
   ```

   Where:

   - `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the secret that is created.
   - `DEV-NAMESPACE` is the developer namespace where the scanner is installed.
   - `YOUR-REGISTRY-SERVER` is the registry server you want to use.
   - `YOUR-NAME` is the name associated with the secret.
   - `YOUR-PASSWORD` is the password associated with the secret.
   - `YOUR-EMAIL` is the email associated with the secret.
3. Update the `tap-values.yaml` file to include the name of secret created earlier.

   ```
   grype:
   targetImagePullSecret: "TARGET-REGISTRY-CREDENTIALS-SECRET"
   ```
4. Upgrade Tanzu Application Platform with the modified `tap-values.yaml` file.

   ```
   tanzu package installed update tap -p tap.tanzu.vmware.com -v ${TAP-VERSION}  -
   -values-file tap-values.yaml -n tap-install
   ```

   Where `TAP-VERSION` is the Tanzu Application Platform version.

Create the private image scan

Create `sample-private-image-scan.yaml`:

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ImageScan
metadata:
  name: sample-private-image-scan
spec:
  registry:
    image: IMAGE-URL
  scanTemplate: private-image-scan-template
```

Where `IMAGE-URL` is the URL of an image in a private registry.

(Optional) Set up a watch

Before deploying the resources to a user specified namespace, set up a watch in another terminal to view the progression:

```
watch kubectl get sourcescans,imagescans,pods,taskruns,scantemplates,scanpolicies -n D
EV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

For more information, see Observing and Troubleshooting.

Deploy the resources

```
kubectl apply -f sample-private-image-scan.yaml -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

View the scan results

When the scan completes, run:

```
kubectl describe imagescan sample-private-image-scan -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

Note

The `Status.Conditions` includes a `Reason: JobFinished` and `Message: The scan job finished`. See Viewing and Understanding Scan Status Conditions.

Clean up

```
kubectl delete -f sample-private-image-scan.yaml -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

View vulnerability reports

After completing the scans, query the Supply Chain Security Tools - Store to view your vulnerability results.

Sample private source scan for Supply Chain Security Tools - Scan

This example shows how you can perform a private source scan for SCST - Scan.

Define the resources

1. Create a Kubernetes secret with an SSH key for cloning a Git repository. See the Kubernetes documentation.

   ```
   cat <<EOF | kubectl create -f -
   apiVersion: v1
   kind: Secret
   metadata:
   name: SECRET-SSH-AUTH
   namespace: DEV-NAMESPACE
   annotations:
   ```

```
  tekton.dev/git-0: https://github.com
  tekton.dev/git-1: https://gitlab.com
type: kubernetes.io/ssh-auth
stringData:
ssh-privatekey: |
  -----BEGIN OPENSSH PRIVATE KEY-----
  ....
  ....
  -----END OPENSSH PRIVATE KEY-----
EOF
```

Where:

- `SECRET-SSH-AUTH` is the name of the secret that is being created.
- `DEV-NAMESPACE` is the developer namespace where the scanner is installed.
- `.stringData.ssh-privatekey` contains the private key with pull-permissions.

2. Update the `tap-values.yaml` file to include the name of secret created above.

```
grype:
targetSourceSshSecret: "SECRET-SSH-AUTH"
```

3. Upgrade Tanzu Application Platform with the modified `tap-values.yaml` file.

```
tanzu package installed update tap -p tap.tanzu.vmware.com -v ${TAP-VERSION}  -
-values-file tap-values.yaml -n tap-install
```

Where `TAP-VERSION` is the Tanzu Application Platform version.

4. Create `sample-private-source-scan.yaml`:

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: SourceScan
metadata:
name: sample-private-source-scan
spec:
git:
  url: URL
  revision: REVISION
  knownHosts: |
    KNOWN-HOSTS
scanTemplate: private-source-scan-template
```

Where:

- `URL` is the Git clone repository using SSH.
- `REVISION` is the commit hash.
- `KNOWN-HOSTS` are the SSH client stored host keys generated by ssh-keyscan.
  - For example, `ssh-keyscan github.com` produces:

```
github.com ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAq2A7hRGmdnm9tUDbO9I
DSwBK6TbQa+PXYPCPy6rbTrTtw7PHkccKrpp0yVhp5HdEIcKr6pLlVDBfOLX9QUsyC
OV0wzfjIJNlGEYsdlLJizHhbn2mUjvSAHQqZETYP81eFzLQNnPHt4EVVUh7VfDESU8
4KezmD5QlWpXLmvU31/yMf+Se8xhHTvKSCZIFImWwoG6mbUoWf9nzpIoaSjB+weqqU
UmpaaasXVal72J+UX2B+2RPW3RcT0eOzQgqlJL3RKrTJvdsjE3JEAvGq3lGHSZXy28
G3skua2SmVi/w4yCE6gbODqnTWlg7+wC604ydGXA8VJiS5ap43JXiUFFAaQ==
github.com ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAA
IbmlzdHAyNTYAAABBBEmKSENjQEezOmxkZMy7opKgwFB9nkt5YRrYMjNuG5N87uRgg
6CLrbo5wAdT/y6v0mKV0U2w0WZ2YB/++Tpockg=
```

```
github.com ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIOMqqnkVzrm0SdG6UOo
qKLsabgH5C9okWi0dh2l9GKJl
```

For example:

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: SourceScan
metadata:
name: sample-private-source-scan
spec:
git:
  url: git@github.com:acme/website.git
  revision: 25as5e7df56c6401111be514a2f3666179ba04d0
  knownHosts: |
    10.254.171.53 ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItb
POVVQF/CzuAeQNv4fZVf2pLxpGHle15zkpxOosckequUDxoq
scanTemplate: private-source-scan-template
```

(Optional) Set up a watch

Before deploying the resources to a user specified namespace, set up a watch in another terminal to view the progression:

```
watch kubectl get sourcescans,imagescans,pods,taskruns,scantemplates,scanpolicies -n D
EV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

See Observing and Troubleshooting.

Deploy the resources

```
kubectl apply -f sample-private-source-scan.yaml -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

View the scan status

After the scan has completed, run:

```
kubectl describe sourcescan sample-private-source-scan -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

Notice the `Status.Conditions` includes a `Reason: JobFinished` and `Message: The scan job finished`. See Viewing and Understanding Scan Status Conditions.

Clean up

```
kubectl delete -f sample-private-source-scan.yaml -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

View vulnerability reports

After completing the scans, query the Supply Chain Security Tools - Store to view your vulnerability results.

Sample public source scan of a blob for Supply Chain Security Tools - Scan

You can do a public source scan of a blob for SCST - Scan. This example performs a scan against source code in a `.tar.gz` file. This is helpful in a Supply Chain, where there is a `GitRepository` step that handles cloning a repository and outputting the source code as a compressed archive.

Define the resources

Create `public-blob-source-example.yaml`:

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: SourceScan
metadata:
  name: public-blob-source-example
spec:
  blob:
    url: "https://gitlab.com/nina-data/ckan/-/archive/master/ckan-master.tar.gz"
  scanTemplate: blob-source-scan-template
```

(Optional) Set up a watch

Before deploying the resources to a user specified namespace, set up a watch in another terminal to view the progression:

```
watch kubectl get sourcescans,imagescans,pods,taskruns,scantemplates,scanpolicies -n D
EV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

For more information, see Observing and Troubleshooting.

Deploy the resources

```
kubectl apply -f public-blob-source-example.yaml -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

View the scan results

When the scan completes, perform:

```
kubectl describe sourcescan public-blob-source-example -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

Notice the `Status.Conditions` includes a `Reason: JobFinished` and `Message: The scan job finished`.

For more information, see Viewing and Understanding Scan Status Conditions.

Clean up

```
kubectl delete -f public-blob-source-example.yaml -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

View vulnerability reports

After completing the scans, query the Supply Chain Security Tools - Store to view your vulnerability results.

Use vulnerability scanning in offline and air-gapped environments for Supply Chain Security Tools - Scan

This topic tells you how to use Grype in air-gapped (offline) environments for Supply Chain Security Tools (SCST) - Scan.

Note

For air-gapped (offline) environments, VMware recommends using Aqua Security's Trivy scanner with Scan 2.0. This greatly simplifies the installation and maintenance experiences over Anchore Grype and Scan 1.0. See here for more information on Scan 1.0 versus Scan 2.0.

Using Trivy with SCST - Scan 2.0

The Aqua Trivy vulnerability scanner uses two databases to perform vulnerability scans:

1. Vulnerability Database: This database contains the vulnerability information used to scan artifacts. This database is built every six hours on GitHub and is distributed using GitHub Container registry (GHCR).
2. Java Index Database: This database enables Trivy to identify the `groupId`, `artifactId`, and `version` of JAR files. It is built once a day on GitHub and distributed using GitHub Container registry (GHCR).

In an online installation of Tanzu Application Platform, these databases are automatically downloaded from GHCR (Github Container Registry) with each scan execution. To enable scanning in an offline environment, these two databases must be relocated to your private container image registry, and Trivy must be configured to use your private location instead of downloading from GHCR.

For more information about the Trivy vulnerability databases, see the Trivy documentation.

Relocate Trivy Images

The databases are stored in an OCI-compliant registry as OCI artifacts. To copy these to your private registry, Trivy recommends using the ORAS CLI.

To make the Trivy databases available in your private registry:

1. Create a repository within your private registry. This example uses aquasecurity to remain consistent with the GHCR repository.
2. Copy the vulnerability database from GHCR to your private registry:

   ```
   oras cp ghcr.io/aquasecurity/trivy-db:2 CONTAINER-REGISTRY/aquasecurity/trivy-db:2
   ```

> **Note**
>
> The Trivy scanner is hardcoded to use the `2` tag for the vulnerability database.

Where `CONTAINER-REGISTRY` is the URL for your registry. For example, `harbor.example.com`

3. Copy the Java index database from GHCR to your private registry:

```
oras cp ghcr.io/aquasecurity/trivy-java-db:1 harbor.CONTAINER-REGISTRY/aquasecu
rity/trivy-java-db:1
```

Where `CONTAINER-REGISTRY` is the URL for your registry. For example, `harbor.example.com`

> **Note**
>
> The Trivy scanner is hardcoded to use the `1` tag for the Java index database.

The databases are now available locally in your air-gapped environment. The next step is to configure the supply chain to use the air-gapped location for Trivy. For more information about using Trivy in an air-gapped environment, see the Trivy documentation.

Configure Trivy in the Supply Chain

Now that you have relocated the images, you must configure the scanning step of the supply chain to use Trivy with the SCST - Scan 2.0, and provide the location of the database artifacts in the private registry.

1. To enable SCST - Scan 2.0 with Trivy with private registries, update your `tap-values.yaml` file to specify the Trivy `ClusterImageTemplate`. For example:

```
ootb_supply_chain_testing_scanning:
  image_scanner_template_name: image-vulnerability-scan-trivy
  scanning:
    steps:
      env_vars:
      - name: TRIVY_DB_REPOSITORY
        value: CONTAINER-REGISTRY/aquasecurity/trivy-db
      - name: TRIVY_JAVA_DB_REPOSITORY
        value: CONTAINER-REGISTRY/aquasecurity/trivy-java-db
      - name: TRIVY_OFFLINE_SCAN
        value: "true"
```

Where `CONTAINER-REGISTRY` is the URL for your registry. For example, `harbor.example.com`

2. Update your Tanzu Application Platform installation by running:

```
tanzu package installed update tap -p tap.tanzu.vmware.com -v TAP-VERSION  --va
lues-file tap-values.yaml -n tap-install
```

Where `TAP-VERSION` is the version of Tanzu Application Platform installed.

Trivy is now configured to use resources in your air-gapped environment when executed with the supply chain. For more information about using Trivy in an air-gapped environment, see the Trivy documentation.

Using Grype with Scan 1.0

VMware recommends using Aqua Trivy with SCST - Scan 2.0 for offline or air-gapped environments due to the simplified setup and maintenance. However, if you require policy enablement based on vulnerability scanning, use the following steps to enable Grype scanning with SCST - Scan 1.0.

The Grype CLI attempts to perform two over-the-Internet calls:

- One to verify for later versions of the CLI.
- One to update the vulnerability database before scanning.

For the Grype CLI to function in an offline or air-gapped environment, the vulnerability database must be hosted in the environment. You must configure the Grype CLI with the internal URL.

The Grype CLI accepts environment variables to satisfy these needs.

Host the Grype vulnerability database

To host the Grype vulnerability database in an air-gapped environment:

1. Retrieve the Grype listing file from its public endpoint: https://toolbox-data.anchore.io/grype/databases/listing.json.
2. Create your own `listing.json` file.

   Note Different Grype versions require specific database schema versions. To avoid compatibility issues between different versions, include a database schema for each version. For example:

```
    {
      "available": {
        "1": [
          {
            "built": "2023-06-16T01:33:30Z",
            "version": 1,
            "url": "https://toolbox-data.anchore.io/grype/databases/vulnerabili
ty-db_v1_2023-06-16T01:33:30Z_1621f4169ffd15bea9e5.tar.gz",
            "checksum": "sha256:3f2c1b432945cca9a69b2e604f6fb231fec450fdd27f494
6fc5608692b63a9d1"
          }
        ],
        "2": [
          {
            "built": "2023-06-16T01:33:30Z",
            "version": 2,
            "url": "https://toolbox-data.anchore.io/grype/databases/vulnerabili
ty-db_v2_2023-06-16T01:33:30Z_d6eee5e78d9b78285e1a.tar.gz",
            "checksum": "sha256:7b7e3a2a7712c72b8c5cc777733c4d8d140d8cfee65e4f0
4540abbdfe3ef1f65"
          }
        ],
        "3": [
          {
            "built": "2023-06-16T01:33:30Z",
            "version": 3,
            "url": "https://toolbox-data.anchore.io/grype/databases/vulnerabili
ty-db_v3_2023-06-16T01:33:30Z_f96ae38a7b05987c3ece.tar.gz",
            "checksum": "sha256:8ea9fae3fda3bf3bf35bd5e5eb656fc127b59cd3c42db4c
36795556aab8a9cf0"
          }
        ],
        "4": [
          {
            "built": "2023-06-16T01:33:30Z",
            "version": 4,
```

```
            "url": "https://toolbox-data.anchore.io/grype/databases/vulnerabili
ty-db_v4_2023-06-16T01:33:30Z_13bba2fa8ff62b7f8b26.tar.gz",
            "checksum": "sha256:3b53d20241b88e5aa45feb817b325c53d6efbe9fa1fc5a6
7eeddaecafa7687e0"
          }
        ],
        "5": [
          {
            "built": "2023-06-16T01:33:30Z",
            "version": 5,
            "url": "https://toolbox-data.anchore.io/grype/databases/vulnerabili
ty-db_v5_2023-06-16T01:33:30Z_e07da3853f6db6eb1104.tar.gz",
            "checksum": "sha256:93d4d9d2f9e39f86570f832cf85b7149a949ca6f1613581
b10c12393509d884f"
          }
        ]
      }
    }
```

Where `url` points to a tarball containing Grype's `vulnerability.db` and `metadata.json` files.

3. Download and host the tarballs in your internal file server.

> **Note**
>
> Some storage solutions for internal file servers change the name of TAR files automatically because of their limits. Notice these modified names and reflect the changes in the `url`. Ensure that the timestamp in the name is correctly formatted because Grype parses the name of TAR artifact to get the timestamp.

4. Update the download `url` to point at your internal endpoint.

   For information about setting up an offline vulnerability database, see the Anchore Grype README in GitHub.

To enable Grype in offline air-gapped environments

1. Add the following to your `tap-values.yaml` file:

```
grype:
  db:
    dbUpdateUrl: INTERNAL-VULN-DB-URL
```

   Where `INTERNAL-VULN-DB-URL` is the URL that points to the internal file server.

2. Update Tanzu Application Platform:

```
tanzu package installed update tap -f tap-values.yaml -n tap-install
```

Configure Grype environmental variables

1. Create a secret that contains the ytt overlay to add the Grype environment variable to the ScanTemplates.

```
apiVersion: v1
kind: Secret
metadata:
  name: grype-airgap-environmental-variables
  namespace: tap-install
```

```
stringData:
  patch.yaml: |
    #@ load("@ytt:overlay", "overlay")

    #@overlay/match by=overlay.subset({"kind":"ScanTemplate"}),expects="1+"
    ---
    spec:
      template:
        initContainers:
          #@overlay/match by=overlay.subset({"name": "scan-plugin"}), expects
="1+"
          - name: scan-plugin
            #@overlay/match missing_ok=True
            env:
              #@overlay/append
              - name: GRYPE_CHECK_FOR_APP_UPDATE
                value: "false"
```

Where `spec.template.initContainers[]` specifies setting one or more environment variables in the `scan-plugin initContainer`.

Note

If you are using the Namespace Provisioner to provision a new developer namespace and want to apply a package overlay for Grype, you must import the overlay `Secret`. See Import overlay secrets.

Troubleshooting

ERROR failed to fetch latest cli version

Symptom

Error message:

```
ERROR failed to fetch latest version: Get "https://toolbox-data.anchore.io/grype/releases/latest/VERSION": dial tcp: lookup toolbox-data.anchore.io on [::1]:53: read udp [::1]:65010->[::1]:53: read: connection refused
```

Cause

The Grype CLI checks for later versions of the CLI by contacting the anchore endpoint over-the-Internet.

Solution

This message is a warning and the Grype scan still runs.

To deactivate this check, set the environment variable `GRYPE_CHECK_FOR_APP_UPDATE` to `false` by using a package overlay:

1. Create a secret that contains the ytt overlay to add the Grype environment variable to the ScanTemplates.

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: grype-airgap-deactivate-cli-check-overlay
   ```

```
  namespace: tap-install #! namespace where tap is installed
stringData:
  patch.yaml: |
    #@ load("@ytt:overlay", "overlay")

    #@overlay/match by=overlay.subset({"kind":"ScanTemplate"}),expects="1+"
    ---
    spec:
      template:
        initContainers:
          #@overlay/match by=overlay.subset({"name": "scan-plugin"}), expects
="1+"
          - name: scan-plugin
            #@overlay/match missing_ok=True
            env:
              #@overlay/append
              - name: GRYPE_CHECK_FOR_APP_UPDATE
                value: "false"
```

2. Configure `tap-values.yaml` to use `package_overlays`. Add the following to your `tap-values.yaml` file:

```
package_overlays:
  - name: "grype"
    secrets:
        - name: "grype-airgap-deactivate-cli-check-overlay"
```

3. Update Tanzu Application Platform:

```
tanzu package installed update tap -f tap-values.yaml -n tap-install
```

Database is too old

Symptom

Error message:

```
1 error occurred:
  * db could not be loaded: the vulnerability database was built N days/weeks ago (max
allowed age is 5 days)
```

Cause

Grype needs up-to-date vulnerability information to provide accurate matches. By default, it fails to run if the local database was not built in the last 5 days.

Solution

There are two options to resolve this:

1. Stale databases weaken your security posture. VMware recommends updating the database daily.
2. If you cannot update the database daily, configure the data staleness check using the environment variable `GRYPE_DB_MAX_ALLOWED_BUILT_AGE` and apply a package overlay:
    1. Create a secret that contains the ytt overlay to add the Grype environment variable to the ScanTemplates.

```
apiVersion: v1
kind: Secret
metadata:
  name: grype-airgap-override-stale-db-overlay
  namespace: tap-install #! namespace where tap is installed
stringData:
  patch.yaml: |
    #@ load("@ytt:overlay", "overlay")

    #@overlay/match by=overlay.subset({"kind":"ScanTemplate"}),expects="1
+"
    ---
    spec:
      template:
        initContainers:
          #@overlay/match by=overlay.subset({"name": "scan-plugin"}), exp
ects="1+"
          - name: scan-plugin
            #@overlay/match missing_ok=True
            env:
              #@overlay/append
              - name: GRYPE_DB_MAX_ALLOWED_BUILT_AGE #! see note on best
practices
                value: "120h"
```

Note

The default maximum allowed built age of Grype's vulnerability database is 5 days. This means that scanning with a 6 day old database causes the scan to fail. You can use the `GRYPE_DB_MAX_ALLOWED_BUILT_AGE` parameter to override the default in accordance with your security posture.

2. Configure `tap-values.yaml` to use `package_overlays`. Add the following to your `tap-values.yaml` file:

```
package_overlays:
  - name: "grype"
    secrets:
        - name: "grype-airgap-override-stale-db-overlay"
```

3. Update Tanzu Application Platform:

```
tanzu package installed update tap -f tap-values.yaml -n tap-install
```

Vulnerability database is invalid

Symptom

Error message:

```
scan-pod[scan-plugin]  1 error occurred:
scan-pod[scan-plugin]  * failed to load vulnerability db: vulnerability database is in
valid (run db update to correct): database metadata not found: /.cache/grype/db/5
```

Solution

Examine the `listing.json` file you created. This matches the format of the listing file. The listing file is located at Anchore Grype's public endpoint. See the Grype README.md in GitHub.

An example `listing.json`:

```
{
  "available": {
    "5": [
      {
        "built": "2023-03-28T01:29:38Z",
        "version": 5,
        "url": "https://toolbox-data.anchore.io/grype/databases/vulnerability-db_v5_20
23-03-28T01:29:38Z_e49d318c32a6113eed07.tar.gz",
        "checksum": "sha256:408ce2932f04dee929a5df524e92494f2d635c6b19e30ff9f0a50425b1
fc29a1"
      },
      .....
    ]
  }
}
```

Where:

- `5` refers to the Grype vulnerability database schema.
- `built` is the build timestamp in the format `yyyy-MM-ddTHH:mm:ssZ`.
- `url` is the download URL for the tarball containing the database. This points at your internal endpoint. The tarball contains the following files:
    - `vulnerability.db` is an SQLite file that is Grype's vulnerability database. Each time the data shape of the vulnerability database changes, a new schema is created. Different Grype versions require specific database schema versions. For example, Grype `v0.54.0` requires database schema version v5.
    - `metadata.json` file
- `checksum` is the SHA used to verify the database's integrity.

Verify these possible reasons why the vulnerability database is not valid:

1. The database schema is invalid. Confirm that the required database schema for the installed Grype version is used. Confirm that the top level version key matches the nested `version`. For example, the top level version `1` in the following snippet does not match the nested `version: 5`.

    ```
    {
      "available": {
        "1": [{
              "built": "2023-02-08T08_17_20Z",
              "version": 5,
              "url": "https://INTERNAL-ENDPOINT/PATH-TO-TARBALL/vulnerability-db_v
    5_2023-02-08T08_17_20Z_6ef73016d160043c630f.tar.gz",
              "checksum": "sha256:aab8d369933c845878ef1b53bb5c26ee49b91ddc5cd87c9e
    b57ffb203a88a72f"
        }]
      }
    }
    ```

    Where `PATH-TO-TARBALL` is the path to the tarball containing the vulnerability database.

    As stale databases weaken your security posture, VMware recommends using the newest entry of the relevant schema version in the `listing.json` file. See Anchore's grype-db in GitHub.

2. The `built` parameters in the `listing.json` file are incorrectly formatted. The proper format is `yyyy-MM-ddTHH:mm:ssZ`.
3. The `url` that you modified to point at an internal endpoint is not reachable from the cluster. For information about verifying connectivity, see Debug Grype database in a cluster.
4. Verify if there are syntax errors in the listing.json:

   ```
   grype db check
   ```

5. Validate the configured listing.json:

   ```
   grype db list -o raw
   ```

Debug Grype database in a cluster

1. Describe the failed source scan or image scan to verify the name of the ScanTemplate being used.
   - For `sourcescan`, run:

     ```
     kubectl describe sourcescan SCAN-NAME -n DEV-NAMESPACE
     ```

   - For `imagescan`, run:

     ```
     kubectl describe imagescan SCAN-NAME -n DEV-NAMESPACE
     ```

   Where `SCAN-NAME` is the name of the source or image scan that failed.
2. Pause reconciliation of the `grype.scanning.apps.tanzu.vmware.com` package:

   ```
   kctrl package installed pause -i <PACKAGE-INSTALL-NAME> -n tap-install
   ```

   Where `PACKAGE-INSTALL-NAME` is the name of the `grype.scanning.apps.tanzu.vmware.com` package, for example, grype
3. Edit the ScanTemplate's `scan-plugin` container to include a "sleep" entrypoint which allows you to troubleshoot inside the container:

   ```
   - name: scan-plugin
     volumeMounts:
       ...
     image: #@ data.values.scanner.image
     imagePullPolicy: IfNotPresent
     env:
       ...
     command: ["/bin/bash"]
     args:
     - "sleep 1800" # insert 30 min sleep here
   ```

4. Re-run the scan.
5. Get the name of the `scan-plugin` pod.

   ```
   kubectl get pods -n DEV-NAMESPACE
   ```

6. Get a shell to the container.

   ```
   kubectl exec --stdin --tty SCAN-PLUGIN-POD -c step-scan-plugin -- /bin/bash
   ```

Where `SCAN-PLUGIN-POD` is the name of the `scan-plugin` pod. For more information, see the Kubernetes documentation.

7. Inside the container, run Grype CLI commands to report database status and verify connectivity from the cluster to the mirror. See the Grype documentation in GitHub.
    - Report current status of Grype's database, such as location, build date, and checksum:

      ```
      grype db status
      ```

8. Ensure that the built parameters in the `listing.json` have timestamps in this proper format `yyyy-MM-ddTHH:mm:ssZ`.
9. After you complete troubleshooting, to trigger reconciliation, run:

   ```
   kctrl package installed kick -i <PACKAGE-INSTALL-NAME> -n tap-install
   ```

   Where `PACKAGE-INSTALL-NAME` is the name of the `grype.scanning.apps.tanzu.vmware.com` package, such as Grype.

Grype package overlays are not applied to scantemplates created by Namespace Provisioner

If you used the Namespace Provisioner to provision a new developer namespace and want to apply a package overlay for Grype, see Import overlay secrets.

Triage and Remediate CVEs for Supply Chain Security Tools - Scan

This topic explains how you can triage and remediate CVEs related to SCST - Scan.

Confirm that Supply Chain stopped due to failed policy enforcement

To confirm that Supply Chain failure is related to policy enforcement:

1. Verify that the status of the workload is `MissingValueAtPath` due to waiting on a `.status.compliantArtifact` from either the SourceScan or ImageScan:

   ```
   kubectl describe workload WORKLOAD-NAME -n DEVELOPER-NAMESPACE
   ```

2. Describe the SourceScan or ImageScan to determine what CVE(s) violated the ScanPolicy:

   ```
   kubectl describe sourcescan NAME -n DEVELOPER-NAMESPACE
   kubectl describe imagescan NAME -n DEVELOPER-NAMESPACE
   ```

Triage

The goal of triage is to analyze and prioritize the reported vulnerability data to discover the appropriate course of action to take at the remediation step. To remediate efficiently and appropriately, you need context on the vulnerabilities that are blocking your supply chain, the packages that are affected, and the impact they can have.

During triage, review which packages are impacted by the CVEs that violated your scan policy. Use the Supply Chain Choreographer in the Tanzu Developer Portal to visualize your supply chain,

including scans, scan policy, and CVEs. You can also use the Tanzu CLI Insight plug-in to query packages, vulnerabilities, and create vulnerability analysis. See Tanzu CLI Insight plug-in.

During this stage, VMware recommends reviewing information pertaining to the CVEs from sources such as the National Vulnerability Database or the release page of a package.

Remediation

After triage is complete, the next step is to remediate the blocking vulnerabilities quickly. Some common methods for CVE remediation are as follows:

- Updating the affected component to remove the CVE
- Amending the scan policy with an exception if you decide to accept the CVE and unblock your supply chain

Updating the affected component

Vulnerabilities that occur in older versions of a package might be resolved in later versions. Apply a patch by upgrading to a later version. You can further adopt security best practices by using your project's package manager tools, such as `go mod graph` for projects in Go, to identify transitive or indirect dependencies that can affect CVEs.

Amending the scan policy

If you decide to proceed without remediating the CVE, for example, when a CVE is evaluated to be a false positive or when a fix is not available, you can amend the ScanPolicy to ignore one or more CVEs. For information about common scanner limitations, see Note on Vulnerability Scanners. For information about templates, see Writing Policy Templates.

Under RBAC, users with the `app-operator-scanning` role that is part of the `app-operator` aggregate role, have permission to edit the ScanPolicy. See Detailed role permissions breakdown.

Observe Supply Chain Security Tools - Scan

This topic outlines observability and troubleshooting methods and issues you can use with SCST - Scan components.

Observability

The scans run inside a Tekton TaskRun where the TaskRun creates a pod. Both the TaskRun and pod are cleaned up after completion.

Before applying a new scan, you can set a watch on the TaskRuns, Pods, SourceScans, and Imagescans to observe their progression:

```
watch kubectl get sourcescans,imagescans,pods,taskruns,scantemplates,scanpolicies -n D
EV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

Troubleshoot Supply Chain Security Tools - Scan

This topic describes troubleshooting methods you can use with Supply Chain Security Tools (SCST) - Scan.

Debugging commands

Run these commands to get more logs and details about the errors around scanning. The TaskRuns and pods persist for a predefined amount of seconds before getting deleted. (`deleteScanJobsSecondsAfterFinished` is the tap pkg variable that defines this)

Debugging Tekton TaskRun

To retrieve TaskRun events:

```
kubectl describe taskrun TASKRUN-NAME -n DEV-NAMESPACE
```

Where:

- `TASKRUN-NAME` is the name of the TaskRun.
- `DEV-NAMESPACE` is the name of the developer namespace you want to use.

Debugging Scan pods

Run the following to get error logs from a pod when scan pods are in a failing state:

```
kubectl logs scan-pod-name -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the name of the developer namespace you want to use.

See here for more details about debugging Kubernetes pods.

The following is an example of a successful scan run output:

```
scan:
  cveCount:
    critical: 20
    high: 120
    medium: 114
    low: 9
    unknown: 0
  scanner:
    name: Grype
    vendor: Anchore
    version: v0.37.0
  reports:
  - /workspace/scan.xml
eval:
  violations:
  - CVE node-fetch GHSA-w7rc-rwvf-8q5r Low
store:
  locations:
  - https://metadata-store-app.metadata-store.svc.cluster.local:8443/api/sources?repo=
hound&sha=5805c6502976c10f5529e7f7aeb0af0c370c0354&org=houndci
```

A scan run that has an error means that one of the init containers: `scan-plugin`, `metadata-store-plugin`, `compliance-plugin`, `summary`, or any other additional containers had a failure.

To inspect for a specific init container in a pod:

```
kubectl logs scan-pod-name -n DEV-NAMESPACE -c init-container-name
```

Where `DEV-NAMESPACE` is the name of the developer namespace you want to use.

See Debug Init Containers in the Kubernetes documentation for debug init container tips.

Debugging SourceScan and ImageScan

To retrieve status conditions of an SourceScan and ImageScan, run:

```
kubectl describe sourcescan SOURCE-SCAN -n DEV-NAMESPACE
```

Where:

- `DEV-NAMESPACE` is the name of the developer namespace you want to use.
- `SOURCE-SCAN` is the name of the SourceScan you want to use.

```
kubectl describe imagescan IMAGE-SCAN -n DEV-NAMESPACE
```

Where:

- `DEV-NAMESPACE` is the name of the developer namespace you want to use.
- `IMAGE-SCAN` is the name of the ImageScan you want to use.

Under `Status.Conditions`, for a condition look at the “Reason”, “Type”, “Message” values that use the keyword “Error” to investigate issues.

Debugging Scanning within a SupplyChain

See here for Tanzu workload commands for tailing build and runtime logs and getting workload status and details.

Viewing the Scan-Controller manager logs

To retrieve scan-controller manager logs:

```
kubectl -n scan-link-system logs -f deployment/scan-link-controller-manager -c manager
```

Restarting Deployment

If you encounter an issue with the scan-link controller not starting, run the following to restart the deployment to see if it’s reproducible or flaking upon starting:

```
kubectl rollout restart deployment scan-link-controller-manager -n scan-link-system
```

Troubleshooting scanner to MetadataStore configuration

Insight CLI failed to post scan results to metadata store due to failed certificate verification

If you encounter this issue:

```
✖  Error: Post "https://metadata-store.tap.tanzu.example.com/api/sourceReport?": tls:
failed to verify certificate: x509: certificate signed by unknown authority
```

To ensure that the `caSecret` from the scanner `DEV-NAMESPACE` matches the `caSecret` from the `METADATASTORE-NAMESPACE` namespace:

1. In a single cluster, the connection between the scanning pod and the metadata store happens inside the cluster and does not pass through ingress. This is automatically configured. You do not need to provide an ingress connection to the store. If you provided an ingress connection to the store, delete it.
2. Get the `caSecret.name` depending if your setup is single or multicluster.
    1. If you are using a single cluster setup, the default value for `grype.metadataStore.caSecret.name` is `app-tls-cert`. See Install Supply Chain

Security Tools - Scan.

2. If you are using a multicluster setup, retrieve `grype.metadataStore.caSecret.name` from the Grype config:

```
grype:
metadataStore:
  caSecret:
    name: store-ca-cert
    importFromNamespace: metadata-store-secrets
```

Note `caSecret.name` is set to `store-ca-cert`. See Set up multicluster Supply Chain Security Tools (SCST) - Store.

3. Verify that the `CA-SECRET` secret exists in the `DEV-NAMESPACE`.

```
kubectl get secret CA-SECRET -n DEV-NAMESPACE
```

4. If the secret `CA-SECRET` doesn't exist in your `DEV-NAMESPACE`, verify that the `CA-SECRET` exists in the `METADATASTORE-NAMESPACE` namespace:

```
kubectl get secret CA-SECRET -n METADATASTORE-NAMESPACE
```

Where `METADATASTORE-NAMESPACE` is the namespace that contains the secret `CA-SECRET`. If you are using a single cluster, it is configured using the `metadata-store` namespace. If multicluster, it is configured using the `metadata-store-secrets`.

- If `CA-SECRET` doesn't exist in the metadata store namespace, configure the certificate. See Custom certificate configuration.

5. Check if the secretexport and secretimport exist and are reconciling successfully:

```
kubectl get secretexports.secretgen.carvel.dev -n `METADATASTORE-NAMESPACE`
kubectl get secretimports.secretgen.carvel.dev -n `DEV-NAMESPACE`
```

- SCST - Store creates the single cluster secretexport by default. See Deployment details and configuration.
- For information about creating the multicluster secretexport, see Set up multicluster Supply Chain Security Tools (SCST) - Store.

6. Verify that the `ca.crt` field in both secrets from `METADATASTORE-NAMESPACE` and `DEV-NAMESPACE` match, or that the `ca.crt` field of the secret in the `METADATASTORE-NAMESPACE` includes the `ca.crt` field of the `DEV-NAMESPACE` secret.

Confirm this by base64 decoding both secrets and verifying that there is a match:

```
kubectl get secret CA-SECRET -n DEV-NAMESPACE -o json | jq -r '.data."ca.crt"'
| base64 -d
kubectl get secret CA-SECRET -n METADATASTORE-NAMESPACE -o json | jq -r '.dat
a."ca.crt"' | base64 -d
```

The certificates in the `METADATASTORE-NAMESPACE` and `DEV-NAMESPACE` must have a match for the scanner to connect to the metadata-store.

Troubleshooting issues

Source scan missing in supply chain

The source scan step is opt-in in Tanzu Application Platform 1.6 to better support languages that resolve dependencies at build time. For information and how to opt-in to source scanning in the

out-of-the-box test and scan supply chain, see Scan Types for Supply Chain Security Tools - Scan.

Troubleshooting Grype in air gap Environments

For information about issues with Grype in air gap environments, see Use vulnerability scanning in offline and air-gapped environments.

Missing target SSH secret

Scanning source code from a private source repository requires an SSH secret present in the namespace and referenced as `grype.targetSourceSshSecret` in `tap-values.yaml`. See Installing the Tanzu Application Platform Package and Profiles.

If a private source scan is triggered and the secret cannot be found, the scan pod includes a `FailedMount` warning in Events with the message `MountVolume.SetUp failed for volume "ssh-secret" : secret "secret-ssh-auth" not found`, where `secret-ssh-auth` is the value specified in `grype.targetSourceSshSecret`.

Missing target image pull secret

Scanning an image from a private registry requires an image pull secret to exist in the Scan CRs namespace and be referenced as `grype.targetImagePullSecret` in `tap-values.yaml`. See Installing the Tanzu Application Platform Package and Profiles.

If a private image scan is triggered and the secret is not configured, the scan TaskRun's pod's `step-scan-plugin` container fails with the following error:

```
Error: GET https://dev.registry.tanzu.vmware.com/v2/vse-dev/spring-petclinic/manifest
s/sha256:128e38c1d3f10401a595c253743bee343967c81e8f22b94e30b2ab8292b3973f: UNAUTHORIZE
D: unauthorized to access repository: vse-dev/spring-petclinic, action: pull: unauthor
ized to access repository: vse-dev/spring-petclinic, action: pull
```

Deactivate Supply Chain Security Tools (SCST) - Store

SCST - Store is required to install SCST - Scan. If you install without the SCST - Store, you must edit the configurations to deactivate the Store:

```
---
metadataStore:
  url: ""
```

Install the package with the edited configurations by running:

```
tanzu package install scan-controller \
  --package scanning.apps.tanzu.vmware.com \
  --version VERSION \
  --namespace tap-install \
  --values-file tap-values.yaml
```

Resolving Incompatible Syft Schema Version

You might encounter the following error:

```
The provided SBOM has a Syft Schema Version which doesn't match the version that is su
pported by Grype...
```

This means that the Syft Schema Version from the provided SBOM doesn't match the version supported by the installed `grype-scanner`. There are two different methods to resolve this incompatibility issue:

- Install a version of Tanzu Build Service that provides an SBOM with a compatible Syft Schema Version. This is the method VMware recommends.
- Deactivate the `failOnSchemaErrors` in `grype-values.yaml`. See Install Supply Chain Security Tools - Scan. Although this change bypasses the check on Syft Schema Version, it does not resolve the incompatibility issue and produces a partial scanning result.

  ```
  syft:
    failOnSchemaErrors: false
  ```

Resolving incompatible scan policy

If your scan policy appears to not be enforced, it might be because the Rego file defined in the scan policy is incompatible with the scanner that is being used. For example, the Grype Scanner outputs in the CycloneDX XML format while the Snyk Scanner outputs SPDX JSON.

See Sample ScanPolicy for Snyk in SPDX JSON format for an example of a ScanPolicy formatted for SPDX JSON.

Could not find CA in secret

If you encounter the following issue, it might be due to not exporting `app-tls-cert` to the correct namespace:

```
{"level":"error","ts":"2022-06-08T15:20:48.43237873Z","logger":"setup","msg":"Could no
t find CA in Secret","err":"unable to set up connection to Supply Chain Security Tools
- Store"}
```

Configure `ns_for_export_app_cert` in your `tap-values.yaml` file.

```
metadata_store:
  ns_for_export_app_cert: "DEV-NAMESPACE"
```

Where `DEV-NAMESPACE` is the name of the developer namespace you want to use.

If there are multiple developer namespaces, use `ns_for_export_app_cert: "*"`.

Blob Source Scan is reporting wrong source URL

A Source Scan for a blob artifact can cause reporting in the `status.artifact` and `status.compliantArtifact` the wrong URL for the resource, passing the remote SSH URL instead of the cluster local fluxcd one. One symptom of this issue is the `image-builder` failing with a `ssh:// is an unsupported protocol` error message.

You can confirm you're having this problem by running `kubectl describe` in the affected resource and comparing the `spec.blob.url` value against the `status.artifact.blob.url`. The problem occurs if they are different URLs. For example:

```
kubectl describe sourcescan SOURCE-SCAN-NAME -n DEV-NAMESPACE
```

Where:

- `SOURCE-SCAN-NAME` is the name of the source scan you want to configure.
- `DEV-NAMESPACE` is the name of the developer namespace you want to use. And compare the output:

```
...
spec:
  blob:
```

```
    ...
    url: http://source-controller.flux-system.svc.cluster.local./gitrepository/sample/
repo/8d4cea98b0fa9e0112d58414099d0229f190f7f1.tar.gz
    ...
status:
  artifact:
    blob:
      ...
      url: ssh://git@github.com:sample/repo.git
  compliantArtifact:
    blob:
      ...
      url: ssh://git@github.com:sample/repo.git
```

Workaround: This problem happens in SCST - Scan `v1.2.0` when you use a Grype Scanner ScanTemplates earlier than `v1.2.0`, because this is a deprecated path. To fix this problem, upgrade your Grype Scanner deployment to `v1.2.0` or later. See Upgrading Supply Chain Security Tools - Scan for step-by-step instructions.

Resolving failing scans that block a Supply Chain

If the Supply Chain is not progressing due to CVEs found in either the SourceScan or ImageScan, see the CVE triage workflow in Triaging and Remediating CVEs.

Policy not defined in the Tanzu Developer Portal

If you encounter `No policy has been defined`, it might be because the Tanzu Application Platform GUI is unable to view the Scan Policy resource.

Confirm that the Scan Policy associated with a SourceScan or ImageScan exists. For example, the `scanPolicy` in the scan matches the name of the Scan Policy.

```
kubectl describe sourcescan NAME -n DEV-NAMESPACE
kubectl describe imagescan NAME -n DEV-NAMESPACE
kubectl get scanpolicy NAME -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the name of the developer namespace you want to use.

Add the `app.kubernetes.io/part-of` label to the Scan Policy. See Enable Tanzu Application Platform GUI to view ScanPolicy Resource.

Lookup error when connecting to SCST - Store

If your scan pod is failing, you might see the following connection error in the logs:

```
dial tcp: lookup metadata-store-app.metadata-store.svc.cluster.local on 10.100.0.10:5
3: no such host
```

A connection error while attempting to connect to the local cluster URL causes this error. If this is a multicluster deployment, set the `grype.metadataStore.url` property in your Build profile `values.yaml`. You must set the ingress domain of SCST - Store which is deployed in the View cluster. For information about this configuration, see How to configure Grype in the Build profile values file.

Sourcescan error with SCST - Store endpoint without a prefix

If your Source Scan resource is failing, the status might show this error:

```
Error: endpoint require 'http://' or 'https://' prefix
```

This is because the `grype.metadataStore.url` value in the Tanzu Application Platform profile `values.yaml` was not configured with the correct prefix. Verify that the URL starts with either `http://` or `https://`.

Deprecated pre-v1.2 templates

If the scan phase is in `Error` and the status condition message is:

```
Summary logs could not be retrieved: . error opening stream pod logs reader: container
summary is not valid for pod scan-grypeimagescan-sample-public-zmj2g-hqv5g
```

This error might be a consequence of using Grype Scanner ScanTemplates shipped with SCST - Scan v1.1 or earlier. These ScanTemplates are deprecated and are not supported in Tanzu Application Platform v1.4.0 and later.

There are two options to resolve this issue:

- Option 1: Upgrade to the latest Grype Scanner version. This automatically replaces the old ScanTemplates with the upgraded ScanTemplates.
- Option 2: Create a ScanTemplate. Follow the steps in Create a scan template.

Incorrectly configured self-signed certificate

The following error in the pod logs indicate that the self-signed certificate might be incorrectly configured:

```
x509: certificate signed by unknown authority
```

To resolve this issue, ensure that `shared.ca_cert_data` contains the required certificate. For an example of setting up the shared self-signed certificate, see Build profile.

For information about `shared.ca_cert_data`, see View possible configuration settings for your package.

Unable to pull scan controller and scanner images from a specified registry

The `docker` field and related sub-fields by SCST - Scan Controller, Grype Scanner, or Snyk Scanner are deprecated in Tanzu Application Platform v1.4.0 and removed in Tanzu Application Platform v1.9.1. Previously these fields might be used to populate the `registry-credentials` secret. You might encounter the following error during installation:

```
UNAUTHORIZED: unauthorized to access repository
```

The recommended migration path for users setting up their namespaces manually is to add registry credentials to both the developer namespace and the `scan-link-system` namespace, using these instructions.

Important

This step does not apply to users who used `--export-to-all-namespaces` when setting up the Tanzu Application Platform package repository.

Grype database not available

Before running a scan, the Grype scanner downloads a copy of its database. If the database fails to download, the following log entry might appear.

```
Vulnerability DB [no update available] New version of grype is available: 0.50.2 [000
0] WARN unable to check for vulnerability database update 1 error occurred: * failed t
o load vulnerability db: vulnerability database is corrupt (run db update to correct):
database metadata not found: ~/Library/Caches/grype/db/3
```

To resolve this issue, ensure that Grype has access to its vulnerability database:

- If you set up a mirror of the vulnerability database in Use vulnerability scanning in offline and air-gapped environments, verify that it is populated and reachable.
- If you did not set up a mirror, Grype manages its database behind the scenes. Verify that the cluster has access to https://anchore.com/.

This issue is unrelated to Supply Chain Security Tools for Tanzu – Store.

Scanner Pod restarts once in SCST - Scan `v1.5.0` or later

For SCST - Scan `v1.5.0` or later, you see scanner pods restart:

```
Pods
   NAME                                READY   STATUS      RESTARTS   AGE
   my-scan-45smk-pod                   0/9     Completed   1          14m
```

One restart in scanner pods is expected with successful scans. To support Tanzu Service Mesh (TSM) integration, jobs were replaced with TaskRuns. This restart is an artifact of how Tekton cleans up sidecar containers by patching the container specifications.

Reconciliation of SCST - Scan fails when upgrading to v1.9

When upgrading the SCST - Scan from a previous version to v1.9, you might see a reconciliation failure similar to:

```
NAME                                PACKAGE-NAME
PACKAGE-VERSION                 STATUS
scanning                            scanning.apps.tanzu.vmware.com
1.9.1-build.36081392+c072d305  Reconcile failed
```

If getting the package by running the command `tanzu package installed get scanning -n tap-install` you might see an error similar to:

```
STATUS:                  Reconcile failed
CONDITIONS:              - type: ReconcileFailed
  status: "True"
  reason: ""
  message: Error (see .status.usefulErrorMessage for details)
USEFUL-ERROR-MESSAGE:    ytt: Error: Overlaying data values (in following order: addit
ional data values):
One or more data values were invalid
====================================Given data value is not declared in schema
values.yaml:
    |
  2 |   url: ""
    |    = found: url
    = expected: a map item with the key named "exports" (from .ytt/data.yaml:52)
```

This is because the field `scanning.metadataStore.url` is removed. If this field is present in `tap-values.yaml` provided for the upgrade, the reconciliation fails. To resolve this problem, remove the field from the values file and run the upgrade command again.

Scanning in a cluster with restricted Kubernetes Pod Security Standards

As part of compliance with the restricted profile Kubernetes Pod Security Standards, you must set the `securityContext` of containers and initContainers. This applies to the `prepare` initContainers created by Tekton. When a pod does not meet pod Security Standards, it is not created and vulnerability scanning cannot proceed. For more information, see the Kubernetes documentation.

You might see an error message similar to the following when describing the TaskRun:

```
"scan-source-scan-with-passing-policy-zx46t-pod" is forbidden: violates PodSecurity "r
estricted:latest": allowPrivilegeEscalation != false (container "prepare" must set sec
urityContext.allowPrivilegeEscalation=false), unrestricted capabilities (container "pr
epare" must set securityContext.capabilities.drop=["ALL"]), seccompProfile (pod or con
tainer "prepare" must set securityContext.seccompProfile.type to "RuntimeDefault" or
"Localhost"). Maybe invalid TaskSpec. ScanPodError PodNotFound: no pod found
```

1. Update your Tekton Pipelines package configuration in your `tap-values.yaml` file with the following changes.

   ```
   tekton_pipelines:
       feature_flags:
           set_security_context: "true"
   ```

   Setting the `securityContext` resolves the `prepare` initContainer violation.

2. Update your Tanzu Application Platform installation by running:

   ```
   tanzu package installed update tap -p tap.tanzu.vmware.com -v TAP-VERSION  --va
   lues-file tap-values.yaml -n tap-install
   ```

   Where `TAP-VERSION` is the version of Tanzu Application Platform installed.

3. Re-run the scan.

Troubleshoot Rego files with a scan policy for Supply Chain Security Tools - Scan

This topic describes how you can use an example output to troubleshoot your Rego file for SCST - Scan. You use a Rego file in a scan policy custom resource. See Enforce compliance policy using Open Policy Agent.

For information about how to write Rego, see Open Policy Agent documentation.

Using the Rego playground

Use the Rego Playground, to evaluate your Rego file against an input. In this example, use the example output of an image or source scan custom resource.

Sample input in CycloneDX's XML re-encoded as JSON format

The following is an example scan custom resource output in CycloneDX's XML structure re-encoded as JSON. This example output contains CVEs at low, medium, high, and critical severities.

To troubleshoot using this example output:

1. Paste your Rego file and the example output into the Rego Playground.
2. Evaluate your Rego file against the example output and verify that your Rego file detects the intended CVEs. See this Rego example.

```
{
    "bom": {
        "-serialNumber": "urn:uuid:123",
        "-v": "http://cyclonedx.org/schema/ext/vulnerability/1.0",
        "-version": "1",
        "-xmlns": "http://cyclonedx.org/schema/bom/1.2",
        "components": {
            "component": [
                {
                    "-type": "library",
                    "licenses": {
                        "license": {
                            "name": "GPL-2"
                        }
                    },
                    "name": "adduser",
                    "version": "3.118",
                    "vulnerabilities": {
                        "vulnerability": [
                            {
                                "-ref": "urn:uuid:3d7c61c6-9cfa-494c-858a-9668a318ff2
3",
                                "advisories": {
                                    "advisory": "https://security-tracker.debian.org/t
racker/CVE-2011-3374"
                                },
                                "id": "CVE-2011-3374-a",
                                "ratings": {
                                    "rating": {
                                        "severity": "Low"
                                    }
                                },
                                "source": {
                                    "-name": "debian:10",
                                    "url": "http://cve.mitre.org/cgi-bin/cvename.cgi?n
ame=CVE-2011-3374"
                                }
                            },
                            {
                                "-ref": "urn:uuid:ebabaa92-2bf9-4d33-8181-595b0fdf55b
d",
                                "advisories": {
                                    "advisory": "https://security-tracker.debian.org/t
racker/CVE-2020-27350"
                                },
                                "id": "CVE-2020-27350-a",
                                "ratings": {
                                    "rating": {
                                        "severity": "Medium"
                                    }
                                },
                                "source": {
                                    "-name": "debian:10",
                                    "url": "http://cve.mitre.org/cgi-bin/cvename.cgi?n
ame=CVE-2020-27350"
                                }
                            },
                            {
                                "-ref": "urn:uuid:07c58c81-1e01-459d-9e9d-0e10456a9bf
0",
                                "advisories": {
                                    "advisory": "https://security-tracker.debian.org/t
racker/CVE-2020-3810"
                                },
                                "id": "CVE-2020-3810-a",
                                "ratings": {
```

```
                                    "rating": {
                                        "severity": "Medium"
                                    }
                                },
                                "source": {
                                    "-name": "debian:10",
                                    "url": "http://cve.mitre.org/cgi-bin/cvename.cgi?n
ame=CVE-2020-3810"
                                }
                            }
                        ]
                    }
                },
                {
                    "-type": "library",
                    "licenses": {
                        "license": [
                            {
                                "name": "GPL-2"
                            },
                            {
                                "name": "GPLv2+"
                            }
                        ]
                    },
                    "name": "apt",
                    "version": "1.8.2",
                    "vulnerabilities": {
                        "vulnerability": [
                            {
                                "-ref": "urn:uuid:3d7c61c6-9cfa-494c-858a-9668a318ff2
3",
                                "advisories": {
                                    "advisory": "https://security-tracker.debian.org/t
racker/CVE-2011-3374"
                                },
                                "id": "CVE-2011-3374",
                                "ratings": {
                                    "rating": {
                                        "severity": "Low"
                                    }
                                },
                                "source": {
                                    "-name": "debian:10",
                                    "url": "http://cve.mitre.org/cgi-bin/cvename.cgi?n
ame=CVE-2011-3374"
                                }
                            },
                            {
                                "-ref": "urn:uuid:ebabaa92-2bf9-4d33-8181-595b0fdf55b
d",
                                "advisories": {
                                    "advisory": "https://security-tracker.debian.org/t
racker/CVE-2020-27350"
                                },
                                "id": "CVE-2020-27350",
                                "ratings": {
                                    "rating": {
                                        "severity": "High"
                                    }
                                },
                                "source": {
                                    "-name": "debian:10",
                                    "url": "http://cve.mitre.org/cgi-bin/cvename.cgi?n
ame=CVE-2020-27350"
                                }
```

```
                            },
                            {
                                "-ref": "urn:uuid:07c58c81-1e01-459d-9e9d-0e10456a9bf
0",
                                "advisories": {
                                    "advisory": "https://security-tracker.debian.org/t
racker/CVE-2020-3810"
                                },
                                "id": "CVE-2020-3810",
                                "ratings": {
                                    "rating": {
                                        "severity": "Critical"
                                    }
                                },
                                "source": {
                                    "-name": "debian:10",
                                    "url": "http://cve.mitre.org/cgi-bin/cvename.cgi?n
ame=CVE-2020-3810"
                                }
                            }
                        ]
                    }
                }
            ]
        },
        "metadata": {
            "component": {
                "-type": "container",
                "name": "nginx:1.16",
                "version": "sha256:123"
            },
            "timestamp": "2022-01-28T13:30:43-08:00",
            "tools": {
                "tool": {
                    "name": "grype",
                    "vendor": "anchore",
                    "version": "[not provided]"
                }
            }
        }
    }
}
```

Example input in SPDX JSON format

The example in this section is a modified scan custom resource input, in `.spdx.json`, that contains CVEs at low, medium, high, and critical severities. You can use this example input to evaluate your Rego file.

To troubleshoot using this example output:

1. Paste your Rego file and the example input into the Rego Playground.
2. Evaluate your Rego file against the output and verify that your Rego file detects the intended CVEs. See this Rego example.

```
{
  "id": "SPDXRef-docker-image|nginx",
  "specVersion": "SPDX-3.0",
  "creator": "Organization: Snyk Ltd",
  "created": "2023-03-01T16:10:08Z",
  "profile": [
    "base",
    "vulnerabilities"
  ],
```

```
  "description": "Snyk test result for project docker-image|nginx in SPDX SBOM forma
t",
  "vulnerabilities": [
    {
      "id": "SNYK-DEBIAN10-APT-1049974",
      "name": "SNYK-DEBIAN10-APT-1049974",
      "summary": "Integer Overflow or Wraparound",
      "details": "## NVD Description\n**_Note:_** _Versions mentioned in the descripti
on apply only to the upstream `apt` package and not the `apt` package as distributed b
y `Debian:10`._\n_See `How to fix?` for `Debian:10` relevant fixed versions and statu
s._\n\nAPT had several integer overflows and underflows while parsing .deb packages, a
ka GHSL-2020-168 GHSL-2020-169, in files apt-pkg/contrib/extracttar.cc, apt-pkg/deb/de
bfile.cc, and apt-pkg/contrib/arfile.cc. This issue affects: apt 1.2.32ubuntu0 version
s prior to 1.2.32ubuntu0.2; 1.6.12ubuntu0 versions prior to 1.6.12ubuntu0.2; 2.0.2ubun
tu0 versions prior to 2.0.2ubuntu0.2; 2.1.10ubuntu0 versions prior to 2.1.10ubuntu0.
1;\n## Remediation\nUpgrade `Debian:10` `apt` to version 1.8.2.2 or higher.\n## Refere
nces\n- [ADVISORY](https://security-tracker.debian.org/tracker/CVE-2020-27350)\n- [CON
FIRM](https://bugs.launchpad.net/bugs/1899193)\n- [CONFIRM](https://security.netapp.co
m/advisory/ntap-20210108-0005/)\n- [DEBIAN](https://www.debian.org/security/2020/dsa-4
808)\n- [UBUNTU](https://usn.ubuntu.com/usn/usn-4667-1)\n",
      "relationships": [
        {
          "affect": {
            "to": [
              "docker-image|nginx@1.16",
              "apt/libapt-pkg5.0@1.8.2"
            ],
            "type": "AFFECTS"
          },
          "foundBy": {
            "to": [
              ""
            ],
            "type": "FOUND_BY"
          },
          "suppliedBy": {
            "to": [
              ""
            ],
            "type": "SUPPLIED_BY"
          },
          "ratedBy": {
            "to": [
              ""
            ],
            "type": "RATED_BY",
            "cwes": [
              190
            ],
            "rating": [
              {
                "method": "CVSS_3",
                "score": [
                  {
                    "base": 5.7
                  }
                ],
                "severity": "Medium",
                "vector": "CVSS:3.1/AV:L/AC:L/PR:H/UI:N/S:C/C:L/I:L/A:L"
              }
            ]
          }
        }
      ],
      "externalReferences": [
        {
```

```
          "category": "ADVISORY",
          "locator": "https://security-tracker.debian.org/tracker/CVE-2020-27350"
        },
        {
          "category": "ADVISORY",
          "locator": "https://bugs.launchpad.net/bugs/1899193"
        },
        {
          "category": "ADVISORY",
          "locator": "https://security.netapp.com/advisory/ntap-20210108-0005/"
        },
        {
          "category": "ADVISORY",
          "locator": "https://www.debian.org/security/2020/dsa-4808"
        },
        {
          "category": "ADVISORY",
          "locator": "https://usn.ubuntu.com/usn/usn-4667-1"
        }
      ],
      "modified": "2022-10-29T13:11:02.438923Z",
      "published": "2020-12-10T03:10:23.901831Z"
    },
    {
      "id": "SNYK-DEBIAN10-APT-407502",
      "name": "SNYK-DEBIAN10-APT-407502",
      "summary": "Improper Verification of Cryptographic Signature",
      "details": "## NVD Description\n**_Note:_** _Versions mentioned in the descripti
on apply only to the upstream `apt` package and not the `apt` package as distributed b
y `Debian:10`._\n_See `How to fix?` for `Debian:10` relevant fixed versions and statu
s._\n\nIt was found that apt-key in apt, all versions, do not correctly validate gpg k
eys with the primary keyring, leading to a potential man-in-the-middle attack.\n## Rem
ediation\nThere is no fixed version for `Debian:10` `apt`.\n## References\n- [ADVISOR
Y](https://security-tracker.debian.org/tracker/CVE-2011-3374)\n- [Debian Bug Report](h
ttps://bugs.debian.org/cgi-bin/bugreport.cgi?bug=642480)\n- [MISC](https://people.cano
nical.com/~ubuntu-security/cve/2011/CVE-2011-3374.html)\n- [MISC](https://seclists.or
g/fulldisclosure/2011/Sep/221)\n- [MISC](https://snyk.io/vuln/SNYK-LINUX-APT-116518)\n
- [MISC](https://ubuntu.com/security/CVE-2011-3374)\n- [RedHat CVE Database](https://a
ccess.redhat.com/security/cve/cve-2011-3374)\n",
      "relationships": [
        {
          "affect": {
            "to": [
              "docker-image|nginx@1.16",
              "apt/libapt-pkg5.0@1.8.2"
            ],
            "type": "AFFECTS"
          },
          "foundBy": {
            "to": [
              ""
            ],
            "type": "FOUND_BY"
          },
          "suppliedBy": {
            "to": [
              ""
            ],
            "type": "SUPPLIED_BY"
          },
          "ratedBy": {
            "to": [
              ""
            ],
            "type": "RATED_BY",
            "cwes": [
```

```
              347
            ],
            "rating": [
              {
                "method": "CVSS_3",
                "score": [
                  {
                    "base": 3.7
                  }
                ],
                "severity": "Low",
                "vector": "CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:N"
              }
            ]
          }
        }
      ],
      "externalReferences": [
        {
          "category": "ADVISORY",
          "locator": "https://security-tracker.debian.org/tracker/CVE-2011-3374"
        },
        {
          "category": "ADVISORY",
          "locator": "https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=642480"
        },
        {
          "category": "ADVISORY",
          "locator": "https://people.canonical.com/~ubuntu-security/cve/2011/CVE-2011-
3374.html"
        },
        {
          "category": "ADVISORY",
          "locator": "https://seclists.org/fulldisclosure/2011/Sep/221"
        },
        {
          "category": "ADVISORY",
          "locator": "https://snyk.io/vuln/SNYK-LINUX-APT-116518"
        },
        {
          "category": "ADVISORY",
          "locator": "https://ubuntu.com/security/CVE-2011-3374"
        },
        {
          "category": "ADVISORY",
          "locator": "https://access.redhat.com/security/cve/cve-2011-3374"
        }
      ],
      "modified": "2022-11-01T00:08:27.375895Z",
      "published": "2018-06-27T16:20:45.037549Z"
    },
    {
      "id": "SNYK-DEBIAN10-APT-568926",
      "name": "SNYK-DEBIAN10-APT-568926",
      "summary": "Improper Input Validation",
      "details": "## NVD Description\n**_Note:_** _Versions mentioned in the descripti
on apply only to the upstream `apt` package and not the `apt` package as distributed b
y `Debian:10`._\n_See `How to fix?` for `Debian:10` relevant fixed versions and statu
s._\n\nMissing input validation in the ar/tar implementations of APT before version 2.
1.2 could result in denial of service when processing specially crafted deb files.\n##
Remediation\nUpgrade `Debian:10` `apt` to version 1.8.2.1 or higher.\n## References\n-
[ADVISORY](https://security-tracker.debian.org/tracker/CVE-2020-3810)\n- [FEDORA](http
s://lists.fedoraproject.org/archives/list/package-announce@lists.fedoraproject.org/mes
sage/U4PEH357MZM2SUGKETMEHMSGQS652QHH/)\n- [GitHub Issue](https://github.com/Debian/ap
t/issues/111)\n- [MISC](https://bugs.launchpad.net/bugs/1878177)\n- [MISC](https://lis
ts.debian.org/debian-security-announce/2020/msg00089.html)\n- [MISC](https://salsa.deb
```

```
ian.org/apt-team/apt/-/commit/dceb1e49e4b8e4dadaf056be34088b415939cda6)\n- [MISC](http
s://tracker.debian.org/news/1144109/accepted-apt-212-source-into-unstable/)\n- [UBUNT
U](https://usn.ubuntu.com/4359-2/)\n- [Ubuntu CVE Tracker](http://people.ubuntu.com/~u
buntu-security/cve/CVE-2020-3810)\n- [Ubuntu Security Advisory](https://usn.ubuntu.co
m/4359-1/)\n",
      "relationships": [
        {
          "affect": {
            "to": [
              "docker-image|nginx@1.16",
              "apt/libapt-pkg5.0@1.8.2"
            ],
            "type": "AFFECTS"
          },
          "foundBy": {
            "to": [
              ""
            ],
            "type": "FOUND_BY"
          },
          "suppliedBy": {
            "to": [
              ""
            ],
            "type": "SUPPLIED_BY"
          },
          "ratedBy": {
            "to": [
              ""
            ],
            "type": "RATED_BY",
            "cwes": [
              20
            ],
            "rating": [
              {
                "method": "CVSS_3",
                "score": [
                  {
                    "base": 5.5
                  }
                ],
                "severity": "Medium",
                "vector": "CVSS:3.1/AV:L/AC:L/PR:N/UI:R/S:U/C:N/I:N/A:H"
              }
            ]
          }
        }
      ],
      "externalReferences": [
        {
          "category": "ADVISORY",
          "locator": "https://security-tracker.debian.org/tracker/CVE-2020-3810"
        },
        {
          "category": "ADVISORY",
          "locator": "https://lists.fedoraproject.org/archives/list/package-announce@l
ists.fedoraproject.org/message/U4PEH357MZM2SUGKETMEHMSGQS652QHH/"
        },
        {
          "category": "ADVISORY",
          "locator": "https://github.com/Debian/apt/issues/111"
        },
        {
          "category": "ADVISORY",
          "locator": "https://bugs.launchpad.net/bugs/1878177"
```

```
        },
        {
          "category": "ADVISORY",
          "locator": "https://lists.debian.org/debian-security-announce/2020/msg00089.
html"
        },
        {
          "category": "ADVISORY",
          "locator": "https://salsa.debian.org/apt-team/apt/-/commit/dceb1e49e4b8e4dad
af056be34088b415939cda6"
        },
        {
          "category": "ADVISORY",
          "locator": "https://tracker.debian.org/news/1144109/accepted-apt-212-source-
into-unstable/"
        },
        {
          "category": "ADVISORY",
          "locator": "https://usn.ubuntu.com/4359-2/"
        },
        {
          "category": "ADVISORY",
          "locator": "http://people.ubuntu.com/~ubuntu-security/cve/CVE-2020-3810"
        },
        {
          "category": "ADVISORY",
          "locator": "https://usn.ubuntu.com/4359-1/"
        }
      ],
      "modified": "2022-11-01T00:08:51.907776Z",
      "published": "2020-05-12T14:19:01.052295Z"
    },
    {
      "id": "SNYK-DEBIAN10-APT-1049974",
      "name": "SNYK-DEBIAN10-APT-1049974",
      "summary": "Integer Overflow or Wraparound",
      "details": "## NVD Description\n**_Note:_** _Versions mentioned in the descripti
on apply only to the upstream `apt` package and not the `apt` package as distributed b
y `Debian:10`._\n_See `How to fix?` for `Debian:10` relevant fixed versions and statu
s._\n\nAPT had several integer overflows and underflows while parsing .deb packages, a
ka GHSL-2020-168 GHSL-2020-169, in files apt-pkg/contrib/extracttar.cc, apt-pkg/deb/de
bfile.cc, and apt-pkg/contrib/arfile.cc. This issue affects: apt 1.2.32ubuntu0 version
s prior to 1.2.32ubuntu0.2; 1.6.12ubuntu0 versions prior to 1.6.12ubuntu0.2; 2.0.2ubun
tu0 versions prior to 2.0.2ubuntu0.2; 2.1.10ubuntu0 versions prior to 2.1.10ubuntu0.
1;\n## Remediation\nUpgrade `Debian:10` `apt` to version 1.8.2.2 or higher.\n## Refere
nces\n- [ADVISORY](https://security-tracker.debian.org/tracker/CVE-2020-27350)\n- [CON
FIRM](https://bugs.launchpad.net/bugs/1899193)\n- [CONFIRM](https://security.netapp.co
m/advisory/ntap-20210108-0005/)\n- [DEBIAN](https://www.debian.org/security/2020/dsa-4
808)\n- [UBUNTU](https://usn.ubuntu.com/usn/usn-4667-1)\n",
      "relationships": [
        {
          "affect": {
            "to": [
              "docker-image|nginx@1.16",
              "apt@1.8.2",
              "apt/libapt-pkg5.0@1.8.2"
            ],
            "type": "AFFECTS"
          },
          "foundBy": {
            "to": [
              ""
            ],
            "type": "FOUND_BY"
          },
          "suppliedBy": {
```

```
            "to": [
              ""
            ],
            "type": "SUPPLIED_BY"
          },
          "ratedBy": {
            "to": [
              ""
            ],
            "type": "RATED_BY",
            "cwes": [
              190
            ],
            "rating": [
              {
                "method": "CVSS_3",
                "score": [
                  {
                    "base": 5.7
                  }
                ],
                "severity": "Medium",
                "vector": "CVSS:3.1/AV:L/AC:L/PR:H/UI:N/S:C/C:L/I:L/A:L"
              }
            ]
          }
        }
      ],
      "externalReferences": [
        {
          "category": "ADVISORY",
          "locator": "https://security-tracker.debian.org/tracker/CVE-2020-27350"
        },
        {
          "category": "ADVISORY",
          "locator": "https://bugs.launchpad.net/bugs/1899193"
        },
        {
          "category": "ADVISORY",
          "locator": "https://security.netapp.com/advisory/ntap-20210108-0005/"
        },
        {
          "category": "ADVISORY",
          "locator": "https://www.debian.org/security/2020/dsa-4808"
        },
        {
          "category": "ADVISORY",
          "locator": "https://usn.ubuntu.com/usn/usn-4667-1"
        }
      ],
      "modified": "2022-10-29T13:11:02.438923Z",
      "published": "2020-12-10T03:10:23.901831Z"
    },
    {
      "id": "SNYK-DEBIAN10-APT-407502",
      "name": "SNYK-DEBIAN10-APT-407502",
      "summary": "Improper Verification of Cryptographic Signature",
      "details": "## NVD Description\n**_Note:_** _Versions mentioned in the descripti
on apply only to the upstream `apt` package and not the `apt` package as distributed b
y `Debian:10`._\n_See `How to fix?` for `Debian:10` relevant fixed versions and statu
s._\n\nIt was found that apt-key in apt, all versions, do not correctly validate gpg k
eys with the primary keyring, leading to a potential man-in-the-middle attack.\n## Rem
ediation\nThere is no fixed version for `Debian:10` `apt`.\n## References\n- [ADVISOR
Y](https://security-tracker.debian.org/tracker/CVE-2011-3374)\n- [Debian Bug Report](h
ttps://bugs.debian.org/cgi-bin/bugreport.cgi?bug=642480)\n- [MISC](https://people.cano
nical.com/~ubuntu-security/cve/2011/CVE-2011-3374.html)\n- [MISC](https://seclists.or
```

```
g/fulldisclosure/2011/Sep/221)\n- [MISC](https://snyk.io/vuln/SNYK-LINUX-APT-116518)\n
- [MISC](https://ubuntu.com/security/CVE-2011-3374)\n- [RedHat CVE Database](https://a
ccess.redhat.com/security/cve/cve-2011-3374)\n",
      "relationships": [
        {
          "affect": {
            "to": [
              "docker-image|nginx@1.16",
              "apt@1.8.2",
              "apt/libapt-pkg5.0@1.8.2"
            ],
            "type": "AFFECTS"
          },
          "foundBy": {
            "to": [
              ""
            ],
            "type": "FOUND_BY"
          },
          "suppliedBy": {
            "to": [
              ""
            ],
            "type": "SUPPLIED_BY"
          },
          "ratedBy": {
            "to": [
              ""
            ],
            "type": "RATED_BY",
            "cwes": [
              347
            ],
            "rating": [
              {
                "method": "CVSS_3",
                "score": [
                  {
                    "base": 3.7
                  }
                ],
                "severity": "Low",
                "vector": "CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:N"
              }
            ]
          }
        }
      ],
      "externalReferences": [
        {
          "category": "ADVISORY",
          "locator": "https://security-tracker.debian.org/tracker/CVE-2011-3374"
        },
        {
          "category": "ADVISORY",
          "locator": "https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=642480"
        },
        {
          "category": "ADVISORY",
          "locator": "https://people.canonical.com/~ubuntu-security/cve/2011/CVE-2011-
3374.html"
        },
        {
          "category": "ADVISORY",
          "locator": "https://seclists.org/fulldisclosure/2011/Sep/221"
        },
```

```
        {
          "category": "ADVISORY",
          "locator": "https://snyk.io/vuln/SNYK-LINUX-APT-116518"
        },
        {
          "category": "ADVISORY",
          "locator": "https://ubuntu.com/security/CVE-2011-3374"
        },
        {
          "category": "ADVISORY",
          "locator": "https://access.redhat.com/security/cve/cve-2011-3374"
        }
      ],
      "modified": "2022-11-01T00:08:27.375895Z",
      "published": "2018-06-27T16:20:45.037549Z"
    },
    {
      "id": "SNYK-DEBIAN10-EXPAT-2329087",
      "name": "SNYK-DEBIAN10-EXPAT-2329087",
      "summary": "Incorrect Calculation",
      "details": "## NVD Description\n**_Note:_** _Versions mentioned in the descripti
on apply only to the upstream `expat` package and not the `expat` package as distribut
ed by `Debian:10`._\n_See `How to fix?` for `Debian:10` relevant fixed versions and st
atus._\n\nIn Expat (aka libexpat) before 2.4.3, a left shift by 29 (or more) places in
the storeAtts function in xmlparse.c can lead to realloc misbehavior (e.g., allocating
too few bytes, or only freeing memory).\n## Remediation\nUpgrade `Debian:10` `expat` t
o version 2.2.6-2+deb10u2 or higher.\n## References\n- [ADVISORY](https://security-tra
cker.debian.org/tracker/CVE-2021-45960)\n- [MISC](https://bugzilla.mozilla.org/show_bu
g.cgi?id=1217609)\n- [MISC](https://github.com/libexpat/libexpat/issues/531)\n- [MISC]
(https://github.com/libexpat/libexpat/pull/534)\n- [cve@mitre.org](http://www.openwal
l.com/lists/oss-security/2022/01/17/3)\n- [cve@mitre.org](https://security.netapp.com/
advisory/ntap-20220121-0004/)\n- [cve@mitre.org](https://www.tenable.com/security/tns-
2022-05)\n- [cve@mitre.org](https://www.debian.org/security/2022/dsa-5073)\n- [cve@mit
re.org](https://cert-portal.siemens.com/productcert/pdf/ssa-484086.pdf)\n- [cve@mitre.
org](https://security.gentoo.org/glsa/202209-24)\n",
      "relationships": [
        {
          "affect": {
            "to": [
              "docker-image|nginx@1.16",
              "nginx-module-image-filter@1.16.1-1~buster",
              "libgd2/libgd3@2.2.5-5.2",
              "fontconfig/libfontconfig1@2.13.1-2",
              "expat/libexpat1@2.2.6-2+deb10u1"
            ],
            "type": "AFFECTS"
          },
          "foundBy": {
            "to": [
              ""
            ],
            "type": "FOUND_BY"
          },
          "suppliedBy": {
            "to": [
              ""
            ],
            "type": "SUPPLIED_BY"
          },
          "ratedBy": {
            "to": [
              ""
            ],
            "type": "RATED_BY",
            "cwes": [
              682
```

```
            ],
            "rating": [
              {
                "method": "CVSS_3",
                "score": [
                  {
                    "base": 8.8
                  }
                ],
                "severity": "High",
                "vector": "CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H"
              }
            ]
          }
        }
      ],
      "externalReferences": [
        {
          "category": "ADVISORY",
          "locator": "https://security-tracker.debian.org/tracker/CVE-2021-45960"
        },
        {
          "category": "ADVISORY",
          "locator": "https://bugzilla.mozilla.org/show_bug.cgi?id=1217609"
        },
        {
          "category": "ADVISORY",
          "locator": "https://github.com/libexpat/libexpat/issues/531"
        },
        {
          "category": "ADVISORY",
          "locator": "https://github.com/libexpat/libexpat/pull/534"
        },
        {
          "category": "ADVISORY",
          "locator": "http://www.openwall.com/lists/oss-security/2022/01/17/3"
        },
        {
          "category": "ADVISORY",
          "locator": "https://security.netapp.com/advisory/ntap-20220121-0004/"
        },
        {
          "category": "ADVISORY",
          "locator": "https://www.tenable.com/security/tns-2022-05"
        },
        {
          "category": "ADVISORY",
          "locator": "https://www.debian.org/security/2022/dsa-5073"
        },
        {
          "category": "ADVISORY",
          "locator": "https://cert-portal.siemens.com/productcert/pdf/ssa-484086.pdf"
        },
        {
          "category": "ADVISORY",
          "locator": "https://security.gentoo.org/glsa/202209-24"
        }
      ],
      "modified": "2023-02-14T13:37:37.505975Z",
      "published": "2022-01-02T01:41:26.770663Z"
    },
    {
      "id": "SNYK-DEBIAN10-EXPAT-2331803",
      "name": "SNYK-DEBIAN10-EXPAT-2331803",
      "summary": "Integer Overflow or Wraparound",
      "details": "## NVD Description\n**_Note:_** _Versions mentioned in the descripti
```

```
on apply only to the upstream `expat` package and not the `expat` package as distribut
ed by `Debian:10`._\n_See `How to fix?` for `Debian:10` relevant fixed versions and st
atus._\n\ndefineAttribute in xmlparse.c in Expat (aka libexpat) before 2.4.3 has an in
teger overflow.\n## Remediation\nUpgrade `Debian:10` `expat` to version 2.2.6-2+deb10u
2 or higher.\n## References\n- [ADVISORY](https://security-tracker.debian.org/tracker/
CVE-2022-22824)\n- [cve@mitre.org](https://github.com/libexpat/libexpat/pull/539)\n-
[cve@mitre.org](http://www.openwall.com/lists/oss-security/2022/01/17/3)\n- [cve@mitr
e.org](https://www.tenable.com/security/tns-2022-05)\n- [cve@mitre.org](https://www.de
bian.org/security/2022/dsa-5073)\n- [cve@mitre.org](https://cert-portal.siemens.com/pr
oductcert/pdf/ssa-484086.pdf)\n- [cve@mitre.org](https://security.gentoo.org/glsa/2022
09-24)\n",
      "relationships": [
        {
          "affect": {
            "to": [
              "docker-image|nginx@1.16",
              "nginx-module-image-filter@1.16.1-1~buster",
              "libgd2/libgd3@2.2.5-5.2",
              "fontconfig/libfontconfig1@2.13.1-2",
              "expat/libexpat1@2.2.6-2+deb10u1"
            ],
            "type": "AFFECTS"
          },
          "foundBy": {
            "to": [
              ""
            ],
            "type": "FOUND_BY"
          },
          "suppliedBy": {
            "to": [
              ""
            ],
            "type": "SUPPLIED_BY"
          },
          "ratedBy": {
            "to": [
              ""
            ],
            "type": "RATED_BY",
            "cwes": [
              190
            ],
            "rating": [
              {
                "method": "CVSS_3",
                "score": [
                  {
                    "base": 9.8
                  }
                ],
                "severity": "Critical",
                "vector": "CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H"
              }
            ]
          }
        }
      ],
      "externalReferences": [
        {
          "category": "ADVISORY",
          "locator": "https://security-tracker.debian.org/tracker/CVE-2022-22824"
        },
        {
          "category": "ADVISORY",
          "locator": "https://github.com/libexpat/libexpat/pull/539"
```

```
        },
        {
          "category": "ADVISORY",
          "locator": "http://www.openwall.com/lists/oss-security/2022/01/17/3"
        },
        {
          "category": "ADVISORY",
          "locator": "https://www.tenable.com/security/tns-2022-05"
        },
        {
          "category": "ADVISORY",
          "locator": "https://www.debian.org/security/2022/dsa-5073"
        },
        {
          "category": "ADVISORY",
          "locator": "https://cert-portal.siemens.com/productcert/pdf/ssa-484086.pdf"
        },
        {
          "category": "ADVISORY",
          "locator": "https://security.gentoo.org/glsa/202209-24"
        }
      ],
      "modified": "2023-02-14T13:39:18.516672Z",
      "published": "2022-01-08T13:52:14.479733Z"
    }
  ],
  "name": "docker-image|nginx-sha256:d20aa6d1cae56fd17cd458f4807e0de462caf2336f0b70b5e
eb69fcaaf30dd9c",
  "dataLicense": "CC0-1.0",
  "documentNamespace": "spdx.org/spdxdocs/docker-image|nginx-feb02ce6-cd47-49c2-9a97-2
b4833b4a1f0",
  "relationships": [
    {
      "from": "SPDXRef-docker-image|nginx",
      "to": [
        "SPDXRef-index.docker.io/library/nginx-sha256:d20aa6d1cae56fd17cd458f4807e0de4
62caf2336f0b70b5eeb69fcaaf30dd9c"
      ],
      "type": "DESCRIBES"
    }
  ],
  "packages": [
    {
      "SPDXID": "SPDXRef-index.docker.io/library/nginx-sha256:d20aa6d1cae56fd17cd458f4
807e0de462caf2336f0b70b5eeb69fcaaf30dd9c",
      "versionInfo": "sha256:d20aa6d1cae56fd17cd458f4807e0de462caf2336f0b70b5eeb69fcaa
f30dd9c",
      "id": "SPDXRef-index.docker.io/library/nginx-sha256:d20aa6d1cae56fd17cd458f4807e
0de462caf2336f0b70b5eeb69fcaaf30dd9c",
      "name": "index.docker.io/library/nginx",
      "checksums": [
        {
          "algorithm": "SHA256",
          "checksumValue": "d20aa6d1cae56fd17cd458f4807e0de462caf2336f0b70b5eeb69fcaaf
30dd9c"
        }
      ]
    }
  ]
}
```


Configure code repositories and image artifacts for Supply Chain Security Tools - Scan

This topic describes how you can configure code repositories and image artifacts for SCST - Scan.

Prerequisite

Both the source and image scans require you to define a `ScanTemplate`. Run `kubectl get scantemplates` for the ScanTemplates provided with the scanner installation. For information about how to reference these ScanTemplates, see How to create a ScanTemplate.

Deploy scan custom resources

The scan controller defines two custom resources to create scans:

- SourceScan
- ImageScan

SourceScan

The `SourceScan` custom resource helps you define and trigger a scan for a given repository. You can deploy `SourceScan` with source code existing in a public repository or a private one:

1. Create the `SourceScan` custom resource.

   Example:

   ```
   apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
   kind: SourceScan
   metadata:
     # set the name of the source scan CR
     name: sample-source-scan
   spec:
     # At least one of these fields (blob or git) must be defined.
     blob:
       # location to a file with the source code compressed (supported files: .ta
   r.gz)
       url:
     git:
       # A multiline string defining the known hosts that are going to be used for
   the SSH client on the container
       knownHosts:
       # Branch, tag, or commit digest
       revision:
       # The name of the kubernetes secret containing the private SSH key informat
   ion.
       sshKeySecret:
       # A string containing the repository URL.
       url:
       # The username needed to SSH connection. Default value is “git”
       username:

     # A string defining the name of an existing ScanTemplate custom resource.
     scanTemplate: my-scan-template

      # A string defining the name of an existing ScanPolicy custom resource. See
   "Enforcement Policies (OPA)" section.
     scanPolicy: my-scan-policy
   ```

2. Deploy the `SourceScan` custom resource to the desired namespace on cluster by running:

   ```
   kubectl apply -f <path_to_the_cr>/<custom_resource_filename>.yaml -n <desired_n
   amespace>
   ```

After the scanning completes, the following fields appear in the custom resource and are filled by the scanner:

```
# These fields are populated from the source scan results
status:
  # The source code information as provided in the CycloneDX `bom>metadata>comp
onent>*` fields
  artifact:
    blob:
      url:
    git:
      url:
      revision:

  # An array populated with information about the scanning status
  # and the policy validation. These conditions might change in the lifecycle
  # of the scan, refer to the "View Scan Status and Understanding Conditions" s
ection to learn more.
  conditions: []

  # The URL of the vulnerability scan results in the Metadata Store integratio
n.
  # Only available when the integration is configured.
  metadataUrl:

  # When the CRD is updated to point at new revisions, this lets you know
  # if the status reflects the latest one or not
  observedGeneration: 1
  observedPolicyGeneration: 1
  observedTemplateGeneration: 1

  # The latest datetime when the scanning was successfully finished.
  scannedAt:
  # Information about the scanner that was used for the latest image scan.
  # This information reflects what's in the CycloneDX `bom>metadata>tools>tool>
*` fields.
  scannedBy:
    scanner:
      # The name of the scanner that was used.
      name: my-image-scanner

      # The name of the scanner's development company or team
      vendor: my-image-scanner-provider

      # The version of the scanner used.
      version: 1.0.0
```

ImageScan

The `ImageScan` custom resource helps you define and trigger a scan for a given image. You can deploy `ImageScan` with an image existing in a public or private registry:

1. Create the `ImageScan` custom resource.

   Example:

```
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ImageScan
metadata:
  # set the name of the image scan CR
  name: sample-image-scan
spec:
  registry:
    # Required. A string containing the image name can additionally add its tag
```

```
or its digest
    image: nginx:1.16

    # A string containing the secret needed to pull the image from a private re
gistry.
    # The secret needs to be deployed in the same namespace as the ImageScan
    imagePullSecret: my-image-pull-secret

  # A string defining the name of an existing ScanTemplate custom resource. See
"How To Create a ScanTemplate" section.
  scanTemplate: my-scan-template

  # A string defining the name of an existing ScanPolicy custom resource. See
"Enforcement Policies (OPA)" section.
  scanPolicy: my-scan-policy
```

2. Deploy the `ImageScan` custom resource to the desired namespace on cluster by running:

```
kubectl apply -f <path_to_the_cr>/<custom_resource_filename>.yaml -n <desired_n
amespace>
```

After the scanning completes, the following fields appear in the custom resource and are filled by the scanner:

```
 # These fields are populated from the image scan results
status:
  artifact:
    registry:
      # The image name with its digest as provided in the CycloneDX `bom>metada
ta>component>*` fields
      image:
      imagePullSecret:

  # An array that is populated with information about the scanning status
  # and the policy validation. These conditions might change in the lifecycle
  # of the scan, refer to the "View Scan Status and Understanding Conditions" s
ection to learn more.
  conditions: []

  # The URL of the vulnerability scan results in the Metadata Store integratio
n.
  # Only available when the integration is configured.
  metadataUrl:

  # When the CRD is updated to point at new revisions, this lets you know
  # whether the status reflects the latest one
  observedGeneration: 1
  observedPolicyGeneration: 1
  observedTemplateGeneration: 1

  # The latest datetime when the scanning was successfully finished.
  scannedAt:
  # Information about the scanner used for the latest image scan.
  # This information reflects what's in the CycloneDX `bom>metadata>tools>tool>
*` fields.
  scannedBy:
    scanner:
      # The name of the scanner that was used.
      name: my-image-scanner

      # The name of the scanner's development company or team
      vendor: my-image-scanner-provider
```

```
      # The version of the scanner used.
      version: 1.0.0
```

Configure code repositories and image artifacts for Supply Chain Security Tools - Scan

This topic describes how you can configure code repositories and image artifacts for SCST - Scan.

Prerequisite

Both the source and image scans require you to define a `ScanTemplate`. Run `kubectl get scantemplates` for the ScanTemplates provided with the scanner installation. For information about how to reference these ScanTemplates, see How to create a ScanTemplate.

Deploy scan custom resources

The scan controller defines two custom resources to create scans:

- SourceScan
- ImageScan

SourceScan

The `SourceScan` custom resource helps you define and trigger a scan for a given repository. You can deploy `SourceScan` with source code existing in a public repository or a private one:

1. Create the `SourceScan` custom resource.

   Example:

```
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: SourceScan
metadata:
  # set the name of the source scan CR
  name: sample-source-scan
spec:
  # At least one of these fields (blob or git) must be defined.
  blob:
    # location to a file with the source code compressed (supported files: .ta
r.gz)
    url:
  git:
    # A multiline string defining the known hosts that are going to be used for
the SSH client on the container
    knownHosts:
    # Branch, tag, or commit digest
    revision:
    # The name of the kubernetes secret containing the private SSH key informat
ion.
    sshKeySecret:
    # A string containing the repository URL.
    url:
    # The username needed to SSH connection. Default value is "git"
    username:

  # A string defining the name of an existing ScanTemplate custom resource.
  scanTemplate: my-scan-template

   # A string defining the name of an existing ScanPolicy custom resource. See
```

```
"Enforcement Policies (OPA)" section.
  scanPolicy: my-scan-policy
```

2. Deploy the `SourceScan` custom resource to the desired namespace on cluster by running:

```
kubectl apply -f <path_to_the_cr>/<custom_resource_filename>.yaml -n <desired_n
amespace>
```

After the scanning completes, the following fields appear in the custom resource and are filled by the scanner:

```
# These fields are populated from the source scan results
status:
  # The source code information as provided in the CycloneDX `bom>metadata>comp
onent>*` fields
  artifact:
    blob:
      url:
    git:
      url:
      revision:

  # An array populated with information about the scanning status
  # and the policy validation. These conditions might change in the lifecycle
  # of the scan, refer to the "View Scan Status and Understanding Conditions" s
ection to learn more.
  conditions: []

  # The URL of the vulnerability scan results in the Metadata Store integratio
n.
  # Only available when the integration is configured.
  metadataUrl:

  # When the CRD is updated to point at new revisions, this lets you know
  # if the status reflects the latest one or not
  observedGeneration: 1
  observedPolicyGeneration: 1
  observedTemplateGeneration: 1

  # The latest datetime when the scanning was successfully finished.
  scannedAt:
  # Information about the scanner that was used for the latest image scan.
  # This information reflects what's in the CycloneDX `bom>metadata>tools>tool>
*` fields.
  scannedBy:
    scanner:
      # The name of the scanner that was used.
      name: my-image-scanner

      # The name of the scanner's development company or team
      vendor: my-image-scanner-provider

      # The version of the scanner used.
      version: 1.0.0
```

ImageScan

The `ImageScan` custom resource helps you define and trigger a scan for a given image. You can deploy `ImageScan` with an image existing in a public or private registry:

1. Create the `ImageScan` custom resource.

   Example:

```
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ImageScan
metadata:
  # set the name of the image scan CR
  name: sample-image-scan
spec:
  registry:
    # Required. A string containing the image name can additionally add its tag
or its digest
    image: nginx:1.16

    # A string containing the secret needed to pull the image from a private re
gistry.
    # The secret needs to be deployed in the same namespace as the ImageScan
    imagePullSecret: my-image-pull-secret

  # A string defining the name of an existing ScanTemplate custom resource. See
"How To Create a ScanTemplate" section.
  scanTemplate: my-scan-template

  # A string defining the name of an existing ScanPolicy custom resource. See
"Enforcement Policies (OPA)" section.
  scanPolicy: my-scan-policy
```

2. Deploy the `ImageScan` custom resource to the desired namespace on cluster by running:

```
kubectl apply -f <path_to_the_cr>/<custom_resource_filename>.yaml -n <desired_n
amespace>
```

After the scanning completes, the following fields appear in the custom resource and are filled by the scanner:

```
 # These fields are populated from the image scan results
status:
  artifact:
    registry:
      # The image name with its digest as provided in the CycloneDX `bom>metada
ta>component>*` fields
      image:
      imagePullSecret:

  # An array that is populated with information about the scanning status
  # and the policy validation. These conditions might change in the lifecycle
  # of the scan, refer to the "View Scan Status and Understanding Conditions" s
ection to learn more.
  conditions: []

  # The URL of the vulnerability scan results in the Metadata Store integratio
n.
  # Only available when the integration is configured.
  metadataUrl:

  # When the CRD is updated to point at new revisions, this lets you know
  # whether the status reflects the latest one
  observedGeneration: 1
  observedPolicyGeneration: 1
  observedTemplateGeneration: 1

  # The latest datetime when the scanning was successfully finished.
  scannedAt:
  # Information about the scanner used for the latest image scan.
  # This information reflects what's in the CycloneDX `bom>metadata>tools>tool>
*` fields.
  scannedBy:
```

```
    scanner:
      # The name of the scanner that was used.
      name: my-image-scanner

      # The name of the scanner's development company or team
      vendor: my-image-scanner-provider

      # The version of the scanner used.
      version: 1.0.0
```

Enforce compliance policy using Open Policy Agent

This topic describes how you can use Open Policy Agent to enforce compliance policy for Supply Chain Security Tools - Scan.

Writing a policy template

The Scan Policy custom resource (CR) allows you to define a Rego file for policy enforcement that you can reuse across image scan and source scan CRs.

The Scan Controller supports policy enforcement by using an Open Policy Agent (OPA) engine with Rego files. This allows you to validate scan results for company policy compliance and can prevent source code from being built or images from being deployed.

Rego file contract

To define a Rego file for an image scan or source scan, you must comply with the requirements defined for every Rego file for the policy verification to work. For information about how to write Rego, see Open Policy Agent documentation.

- **Package main:** The Rego file must define a package in its body called `main`. The system looks for this package to verify the scan results compliance.
- **Input match:** The Rego file evaluates one vulnerability match at a time, iterating as many times as the Rego file finds vulnerabilities in the scan. The match structure is accessed in the `input.currentVulnerability` object inside the Rego file and has the CycloneDX format.
- **deny rule:** The Rego file must define a `deny` rule inside its body. `deny` is a set of error messages that are returned to the user. Each rule you write adds to that set of error messages. If the conditions in the body of the `deny` statement are true then the user is handed an error message. If false, the vulnerability is allowed in the Source or Image scan.

Define a Rego file for policy enforcement

Follow these steps to define a Rego file for policy enforcement that you can reuse across image scan and source scan CRs that output in the CycloneDX XML format.

Note

The Snyk Scanner outputs SPDX JSON. For an example of a ScanPolicy formatted for SPDX JSON output, see Sample ScanPolicy for Snyk in SPDX JSON format.

1. Create a scan policy with a Rego file. The following is an example scan policy resource:

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
```

```
metadata:
  name: scan-policy
  labels:
    app.kubernetes.io/part-of: enable-in-gui
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "Unkn
ownSeverity"
    notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]
    ignoreCves := []

    contains(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
      severities := { e | e := match.ratings.rating.severity } | { e | e := mat
ch.ratings.rating[_].severity }
      some i
      fails := contains(notAllowedSeverities, severities[i])
      not fails
    }

    isSafe(match) {
      ignore := contains(ignoreCves, match.id)
      ignore
    }

    deny[msg] {
      comps := { e | e := input.bom.components.component } | { e | e := input.b
om.components.component[_] }
      some i
      comp := comps[i]
      vulns := { e | e := comp.vulnerabilities.vulnerability } | { e | e := com
p.vulnerabilities.vulnerability[_] }
      some j
      vuln := vulns[j]
      ratings := { e | e := vuln.ratings.rating.severity } | { e | e := vuln.ra
tings.rating[_].severity }
      not isSafe(vuln)
      msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
    }
```

You can edit the following text boxes of the Rego file as part of the CVE triage workflow:

- `notAllowedSeverities` contains the categories of CVEs that cause the SourceScan or ImageScan failing policy enforcement. The following example shows an `app-operator` blocking only `Critical`, `High` and `UnknownSeverity` CVEs.

  ```
  ...
  spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible",
  "UnknownSeverity"
    notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]
    ignoreCves := []
  ...
  ```

- `ignoreCves` contains individual ignored CVEs when determining policy enforcement. In the following example, an `app-operator` ignores `CVE-2018-14643` and `GHSA-f2jv-r9rf-7988` if they are false positives. See A Note on Vulnerability Scanners.

```
...
spec:
regoFile: |
  package main

  notAllowedSeverities := []
  ignoreCves := ["CVE-2018-14643", "GHSA-f2jv-r9rf-7988"]
...
```

2. Deploy the scan policy to the cluster:

```
kubectl apply -f <path_to_scan_policy>/<scan_policy_filename>.yaml -n <desired_
namespace>
```

For information about how scan policies are used in the CVE triage workflow, see Triaging and Remediating CVEs.

Further refine the Scan Policy for use

The scan policy earlier demonstrates how vulnerabilities are ignored during a compliance check. It is not possible to audit why a vulnerability is ignored. You might want to allow an exception, where a build with a failing vulnerability is allowed to progress through a supply chain. You can allow this exception for a certain period of time, requiring an expiration date. Vulnerability Exploitability Exchange (VEX) documents are gaining popularity to capture security advisory information pertaining to vulnerabilities. You can use Rego for these use cases.

For example, the following scan policy includes an additional text box to capture comments regarding why the scan ignores a vulnerability. The `notAllowedSeverities` array remains an array of strings, but the `ignoreCves` array updates from an array of strings to an array of objects. This causes a change to the `contains` function, splitting it into separate functions for each array.

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: scan-policy
  labels:
    app.kubernetes.io/part-of: enable-in-gui
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "UnknownSeve
rity"
    notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]

    # List of known vulnerabilities to ignore when deciding whether to fail complianc
e. Example:
    # ignoreCves := [
    #   {
    #     "id": "CVE-2018-14643",
    #     "detail": "Determined affected code is not in the execution path."
    #   }
    # ]
    ignoreCves := []

    containsSeverity(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
```

```
      severities := { e | e := match.ratings.rating.severity } | { e | e := match.rati
ngs.rating[_].severity }
      some i
      fails := containsSeverity(notAllowedSeverities, severities[i])
      not fails
    }

    containsCve(array, elem) = true {
      array[_].id = elem
    } else = false { true }

    isSafe(match) {
      ignore := containsCve(ignoreCves, match.id)
      ignore
    }

    deny[msg] {
      comps := { e | e := input.bom.components.component } | { e | e := input.bom.comp
onents.component[_] }
      some i
      comp := comps[i]
      vulns := { e | e := comp.vulnerabilities.vulnerability } | { e | e := comp.vulne
rabilities.vulnerability[_] }
      some j
      vuln := vulns[j]
      ratings := { e | e := vuln.ratings.rating.severity } | { e | e := vuln.ratings.r
ating[_].severity }
      not isSafe(vuln)
      msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
    }
```

The following example includes an expiration text box and only allows the vulnerability to be ignored for a period of time:

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: scan-policy
  labels:
    app.kubernetes.io/part-of: enable-in-gui
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "UnknownSeve
rity"
    notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]

    # List of known vulnerabilities to ignore when deciding whether to fail complianc
e. Example:
    # ignoreCves := [
    #   {
    #     "id": "CVE-2018-14643",
    #     "detail": "Determined affected code is not in the execution path.",
    #     "expiration": "2022-Dec-31"
    #   }
    # ]
    ignoreCves := []

    containsSeverity(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
```

```
      severities := { e | e := match.ratings.rating.severity } | { e | e := match.rati
ngs.rating[_].severity }
      some i
      fails := containsSeverity(notAllowedSeverities, severities[i])
      not fails
    }

    containsCve(array, elem) = true {
      array[_].id = elem
      curr_time := time.now_ns()
      date_format := "2006-Jan-02"
      expire_time := time.parse_ns(date_format, array[_].expiration)
      curr_time < expire_time
    } else = false { true }

    isSafe(match) {
      ignore := containsCve(ignoreCves, match.id)
      ignore
    }

    deny[msg] {
      comps := { e | e := input.bom.components.component } | { e | e := input.bom.comp
onents.component[_] }
      some i
      comp := comps[i]
      vulns := { e | e := comp.vulnerabilities.vulnerability } | { e | e := comp.vulne
rabilities.vulnerability[_] }
      some j
      vuln := vulns[j]
      ratings := { e | e := vuln.ratings.rating.severity } | { e | e := vuln.ratings.r
ating[_].severity }
      not isSafe(vuln)
      msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
    }
```

Troubleshooting Rego files (Scan Policy)

To troubleshoot or confirm that any modifications made to the rego file in the provided sample scan policy are functioning as intended, see Troubleshooting Rego Files.

Enable Tanzu Developer Portal to view ScanPolicy Resource

For the Tanzu Developer Portal to view the ScanPolicy resource, it must have a matching `kubernetes-label-selector` with a `part-of` prefix.

The following example is portion of a ScanPolicy that is viewable by the Tanzu Developer Portal:

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: scan-policy
  labels:
    app.kubernetes.io/part-of: enable-in-gui
spec:
  regoFile: |
    ...
```

Note

Anything van be a value for the label. The Tanzu Application Platform GUI is looking for the existence of the `part-of` prefix string and doesn't match for anything else specific.

Deprecated Rego file Definition

Before Scan Controller v1.2.0, you must use the following format where the rego file differences are:

- The package name must be `package policies` instead of `package main`.
- The deny rule is a Boolean `isCompliant` instead of `deny[msg]`.
    - **isCompliant rule:** The Rego file must define inside its body an `isCompliant` rule. This must be a Boolean type containing the result whether the vulnerability violates the security policy or not. If `isCompliant` is `true`, the vulnerability is allowed in the Source or Image scan. Otherwise, `false` is considered. Any scan that finds at least one vulnerability that evaluates to `isCompliant=false` makes the `PolicySucceeded` condition set to false.

The following is an example scan policy resource:

```
apiVersion: scanning.apps.tanzu.vmware.com/v1alpha1
kind: ScanPolicy
metadata:
  name: v1alpha1-scan-policy
  labels:
    app.kubernetes.io/part-of: enable-in-gui
spec:
  regoFile: |
    package policies

    default isCompliant = false

    ignoreSeverities := ["Critical", "High"]

    contains(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isCompliant {
      ignore := contains(ignoreSeverities, input.currentVulnerability.Ratings.Rating
[_].Severity)
      ignore
    }
```

Create a ScanTemplate with Supply Chain Security Tools - Scan

This topic describes how to create a ScanTemplate with Supply Chain Security Tools - Scan.

Overview

The `ScanTemplate` custom resource (CR) defines how the scan Pod fulfills the task of vulnerability scanning. There are default `ScanTemplates` provided out of the box using the Tanzu Application Platform default scanner, `Anchore Grype`. One or more `initContainers` run to complete the scan and must save results to a shared `volume`. After the `initContainers` completes, a single container

on the scan Pod called `summary` combines the result of the initContainers so that the `Scan CR` status is updated.

A customized ScanTemplate is created by editing or replacing `initContainer` definitions and reusing the `summary` container from the `grype` package. A container can read the `out.yaml` from an earlier step to locate relevant inputs.

Output Model

Each initContainer can create a subdirectory in `/workspace` to use as a scratch space. Before terminating the container must create an `out.yaml` file in the subdirectory containing the relevant subset of fields from the output model:

```
fetch:
  git:
    url:
    revision:
    path:
  blob:
    url:
    revision:
    path:
  image:
    url:
    revision:
    path:
sbom:
    packageCount:
    reports: []
scan:
  cveCount:
    critical:
    high:
    medium:
    low:
    unknown:
  scanner:
    name:
    vendor:
    version:
    db:
      version:
  reports: []
eval:
  violations: []
store:
  locations: []
```

The `scan` portion of the earlier output is required and if missing the scan controller fails to properly update the final status of the `Scan CR`. Other portions of the output, including those of `store` and `policy evaluation`, are optional and can be omitted if not applicable in a custom supply chain setup.

ScanTemplate Structure

```
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanTemplate
spec:
    template: # a core/v1 PodSpec
      # Here are list volumes mounted for writing to or
      # reading from during different stages of the scan
```

```
      volumes:
        # required the results of different scan stages
        # should be saved in files digestible by the scan
        # controller in this volume
        - name: workspace
        emptyDir: { }
      # different steps required for a scanning can be staged
      # in sequential stages through initContainers.
      initContainers:
      # Summary container will take results of initContainers
      # and will let Controller to update Scan CR status.
      containers:
        - name: summary
```

Note: You cannot name a container `sleep` because there is already a container named `sleep` which comes from the scan-link controller.

Sample Outputs

```
# example for a typical git clone (source scan fetch stage)
# saved at: /workspace/git-clone/out.yaml
fetch:
  git:
    url: github.com/my/repo
    revision: aee9f8
    path: /workspace/git-clone/cloned-repository
```

```
# an example of typical scan stage
# saved at: /workspace/grype-scan/out.yaml
scan:
  cveCount:
    critical: 0
    high: 1
    medium: 3
    low: 25
    unknown: 0
  scanner:
    name: grype
    vendor: Anchore
    version: 0.33.0
    db:
      version: 2022-04-13
  reports:
  - /workspace/grype-scan/repo.cyclonedx.xml
  - /workspace/grype-scan/app.cyclonedx.xml
  - /workspace/grype-scan/base.cyclonedx.xml
```

```
# example of a typical evaluation stage
# saved at: /workspace/policy-eval/out.yaml
eval:
  violations:
    - banned package log4j
    - critical CVE 2022-01-01-3333
    - number of critical CVEs over threshold
```

```
# example of a typical upload to store stage
# saved at: /workspace/upload-to-store/out.yaml
store:
  locations:
    - http://metadata-store.cluster.local:8080/reports/3
```

View scan status conditions for Supply Chain Security Tools - Scan

This topic explains how you can view scan status conditions for Supply Chain Security Tools - Scan.

Viewing scan status

You can view the scan status by using `kubectl describe` on a `SourceScan` or `ImageScan`. You can see information about the scan status under the Status field for each scan CR.

Overview of conditions

The `Status.Conditions` array is populated with the scan status information during and after scanning execution, and the policy validation (if defined for the scan) after the results are available.

Condition types for the scans

Scanning

The Condition with type `Scanning` indicates running the scanning TaskRun. The Status field indicates whether the scan is running or has already finished. For example, if `Status: True`, the scan TaskRun is still running and if `Status: False`, the scan is done.

The Reason field is `JobStarted` while the scan is running and `JobFinished` when it is done.

The Message fieldcan either be `The scan job is running` or `The scan job terminated` depending on the current Status and Reason.

Succeeded

The Condition with type `Succeeded` indicates the scanning TaskRun result. The Status field indicates whether the scan finished successfully or if it encountered an error. For example, the status is `Status: True` if it completed successfully or `Status: False` otherwise.

The Reason field is `JobFinished` if the scanning was successful or `Error` if otherwise.

The Message and Error fields have more information about the last seen status of the scan TaskRun.

SendingResults

The condition with type `SendingResults` indicates sending the scan results to the metadata store. In addition to a successful process of sending the results, the condition can also indicate that the metadata store integration has not been configured or that there was an error sending. An error is usually a misconfigured metadata store URL or that the metadata store is inaccessible. Verify the installation steps to ensure that the configuration is correct regarding secrets being set within the `scan-link-system` namespace.

PolicySucceeded

The Condition with type `PolicySucceeded` indicates the compliance of the scanning results against the defined policies. See Code Compliance Policy Enforcement using Open Policy Agent (OPA). The Status field indicates whether the results are compliant or not (`Status: True` or `Status: False` respectively) or `Status: Unknown` in case an error occurred during the policy verification.

The Reason field is `EvaluationPassed` if the scan complies with the defined policies. The Reason field is `EvaluationFailed` if the scan is not compliant, or `Error` if something went wrong.

The Message and Error fields are populated with `An error has occurred` and an error message if something went wrong during policy verification. Otherwise, the Message field displays `No CVEs were found that violated the policy` if there are no non-compliant vulnerabilities found or `Policy violated because of X CVEs` indicating the count of unique vulnerabilities found.

Overview of CVECount

The `status.CVECount` is populated with the number of CVEs in each category (CRITICAL, HIGH, MEDIUM, LOW, UNKNOWN) and the total (CVETOTAL).

Note

You can also view scan CVE summary in print columns with `kubectl get` on a `SourceScan` or `ImageScan`.

Overview of MetadataURL

The `status.metadataURL` is populated with the URL of the vulnerability scan results in> the metadata store integration. This is only available when the integration is configured.

Overview of Phase

The `status.phase` field is populated with the current phase of the scan. The phases are: Pending, Scanning, Completed, Failed, and Error.

- `Pending`: initial phase of the scan.
- `Scanning`: execution of the scan TaskRun is running.
- `Completed`: scan completed and no CVEs were found that violated the scan policy.
- `Failed`: scan completed but CVEs were found that violated the scan policy.
- `Error`: indication of an error (e.g., an invalid scantemplate or scan policy).

Note

The PHASE print column also shows this with `kubectl get` on a `SourceScan` or `ImageScan`.

Overview of ScannedBy

The `status.scannedBy` field is populated with the name, vendor, and scanner version that generates the security assessment report.

Overview of ScannedAt

The `status.scannedAt` field is populated with the latest date when the scanning finishes.

Troubleshoot Rego files with a scan policy for Supply Chain Security Tools - Scan

This topic describes how you can use an example output to troubleshoot your Rego file for SCST - Scan. You use a Rego file in a scan policy custom resource. See Enforce compliance policy using

Open Policy Agent.

For information about how to write Rego, see Open Policy Agent documentation.

Using the Rego playground

Use the Rego Playground, to evaluate your Rego file against an input. In this example, use the example output of an image or source scan custom resource.

Sample input in CycloneDX's XML re-encoded as JSON format

The following is an example scan custom resource output in CycloneDX's XML structure re-encoded as JSON. This example output contains CVEs at low, medium, high, and critical severities.

To troubleshoot using this example output:

1. Paste your Rego file and the example output into the Rego Playground.
2. Evaluate your Rego file against the example output and verify that your Rego file detects the intended CVEs. See this Rego example.


```
{
    "bom": {
        "-serialNumber": "urn:uuid:123",
        "-v": "http://cyclonedx.org/schema/ext/vulnerability/1.0",
        "-version": "1",
        "-xmlns": "http://cyclonedx.org/schema/bom/1.2",
        "components": {
            "component": [
                {
                    "-type": "library",
                    "licenses": {
                        "license": {
                            "name": "GPL-2"
                        }
                    },
                    "name": "adduser",
                    "version": "3.118",
                    "vulnerabilities": {
                        "vulnerability": [
                            {
                                "-ref": "urn:uuid:3d7c61c6-9cfa-494c-858a-9668a318ff2
3",
                                "advisories": {
                                    "advisory": "https://security-tracker.debian.org/t
racker/CVE-2011-3374"
                                },
                                "id": "CVE-2011-3374-a",
                                "ratings": {
                                    "rating": {
                                        "severity": "Low"
                                    }
                                },
                                "source": {
                                    "-name": "debian:10",
                                    "url": "http://cve.mitre.org/cgi-bin/cvename.cgi?n
ame=CVE-2011-3374"
                                }
                            },
                            {
                                "-ref": "urn:uuid:ebabaa92-2bf9-4d33-8181-595b0fdf55b
d",
                                "advisories": {
                                    "advisory": "https://security-tracker.debian.org/t
```

```
racker/CVE-2020-27350"
                                },
                                "id": "CVE-2020-27350-a",
                                "ratings": {
                                    "rating": {
                                        "severity": "Medium"
                                    }
                                },
                                "source": {
                                    "-name": "debian:10",
                                    "url": "http://cve.mitre.org/cgi-bin/cvename.cgi?n
ame=CVE-2020-27350"
                                }
                            },
                            {
                                "-ref": "urn:uuid:07c58c81-1e01-459d-9e9d-0e10456a9bf
0",
                                "advisories": {
                                    "advisory": "https://security-tracker.debian.org/t
racker/CVE-2020-3810"
                                },
                                "id": "CVE-2020-3810-a",
                                "ratings": {
                                    "rating": {
                                        "severity": "Medium"
                                    }
                                },
                                "source": {
                                    "-name": "debian:10",
                                    "url": "http://cve.mitre.org/cgi-bin/cvename.cgi?n
ame=CVE-2020-3810"
                                }
                            }
                        ]
                    }
                },
                {
                    "-type": "library",
                    "licenses": {
                        "license": [
                            {
                                "name": "GPL-2"
                            },
                            {
                                "name": "GPLv2+"
                            }
                        ]
                    },
                    "name": "apt",
                    "version": "1.8.2",
                    "vulnerabilities": {
                        "vulnerability": [
                            {
                                "-ref": "urn:uuid:3d7c61c6-9cfa-494c-858a-9668a318ff2
3",
                                "advisories": {
                                    "advisory": "https://security-tracker.debian.org/t
racker/CVE-2011-3374"
                                },
                                "id": "CVE-2011-3374",
                                "ratings": {
                                    "rating": {
                                        "severity": "Low"
                                    }
                                },
                                "source": {
```

```
                                    "-name": "debian:10",
                                    "url": "http://cve.mitre.org/cgi-bin/cvename.cgi?n
ame=CVE-2011-3374"
                                }
                            },
                            {
                                "-ref": "urn:uuid:ebabaa92-2bf9-4d33-8181-595b0fdf55b
d",
                                "advisories": {
                                    "advisory": "https://security-tracker.debian.org/t
racker/CVE-2020-27350"
                                },
                                "id": "CVE-2020-27350",
                                "ratings": {
                                    "rating": {
                                        "severity": "High"
                                    }
                                },
                                "source": {
                                    "-name": "debian:10",
                                    "url": "http://cve.mitre.org/cgi-bin/cvename.cgi?n
ame=CVE-2020-27350"
                                }
                            },
                            {
                                "-ref": "urn:uuid:07c58c81-1e01-459d-9e9d-0e10456a9bf
0",
                                "advisories": {
                                    "advisory": "https://security-tracker.debian.org/t
racker/CVE-2020-3810"
                                },
                                "id": "CVE-2020-3810",
                                "ratings": {
                                    "rating": {
                                        "severity": "Critical"
                                    }
                                },
                                "source": {
                                    "-name": "debian:10",
                                    "url": "http://cve.mitre.org/cgi-bin/cvename.cgi?n
ame=CVE-2020-3810"
                                }
                            }
                        ]
                    }
                }
            ]
        },
        "metadata": {
            "component": {
                "-type": "container",
                "name": "nginx:1.16",
                "version": "sha256:123"
            },
            "timestamp": "2022-01-28T13:30:43-08:00",
            "tools": {
                "tool": {
                    "name": "grype",
                    "vendor": "anchore",
                    "version": "[not provided]"
                }
            }
        }
    }
}
```

Example input in SPDX JSON format

The example in this section is a modified scan custom resource input, in `.spdx.json`, that contains CVEs at low, medium, high, and critical severities. You can use this example input to evaluate your Rego file.

To troubleshoot using this example output:

1. Paste your Rego file and the example input into the Rego Playground.
2. Evaluate your Rego file against the output and verify that your Rego file detects the intended CVEs. See this Rego example.


```
{
  "id": "SPDXRef-docker-image|nginx",
  "specVersion": "SPDX-3.0",
  "creator": "Organization: Snyk Ltd",
  "created": "2023-03-01T16:10:08Z",
  "profile": [
    "base",
    "vulnerabilities"
  ],
  "description": "Snyk test result for project docker-image|nginx in SPDX SBOM forma
t",
  "vulnerabilities": [
    {
      "id": "SNYK-DEBIAN10-APT-1049974",
      "name": "SNYK-DEBIAN10-APT-1049974",
      "summary": "Integer Overflow or Wraparound",
      "details": "## NVD Description\n**_Note:_** _Versions mentioned in the descripti
on apply only to the upstream `apt` package and not the `apt` package as distributed b
y `Debian:10`._\n_See `How to fix?` for `Debian:10` relevant fixed versions and statu
s._\n\nAPT had several integer overflows and underflows while parsing .deb packages, a
ka GHSL-2020-168 GHSL-2020-169, in files apt-pkg/contrib/extracttar.cc, apt-pkg/deb/de
bfile.cc, and apt-pkg/contrib/arfile.cc. This issue affects: apt 1.2.32ubuntu0 version
s prior to 1.2.32ubuntu0.2; 1.6.12ubuntu0 versions prior to 1.6.12ubuntu0.2; 2.0.2ubun
tu0 versions prior to 2.0.2ubuntu0.2; 2.1.10ubuntu0 versions prior to 2.1.10ubuntu0.
1;\n## Remediation\nUpgrade `Debian:10` `apt` to version 1.8.2.2 or higher.\n## Refere
nces\n- [ADVISORY](https://security-tracker.debian.org/tracker/CVE-2020-27350)\n- [CON
FIRM](https://bugs.launchpad.net/bugs/1899193)\n- [CONFIRM](https://security.netapp.co
m/advisory/ntap-20210108-0005/)\n- [DEBIAN](https://www.debian.org/security/2020/dsa-4
808)\n- [UBUNTU](https://usn.ubuntu.com/usn/usn-4667-1)\n",
      "relationships": [
        {
          "affect": {
            "to": [
              "docker-image|nginx@1.16",
              "apt/libapt-pkg5.0@1.8.2"
            ],
            "type": "AFFECTS"
          },
          "foundBy": {
            "to": [
              ""
            ],
            "type": "FOUND_BY"
          },
          "suppliedBy": {
            "to": [
              ""
            ],
            "type": "SUPPLIED_BY"
          },
          "ratedBy": {
            "to": [
              ""
```

```
            ],
            "type": "RATED_BY",
            "cwes": [
              190
            ],
            "rating": [
              {
                "method": "CVSS_3",
                "score": [
                  {
                    "base": 5.7
                  }
                ],
                "severity": "Medium",
                "vector": "CVSS:3.1/AV:L/AC:L/PR:H/UI:N/S:C/C:L/I:L/A:L"
              }
            ]
          }
        }
      ],
      "externalReferences": [
        {
          "category": "ADVISORY",
          "locator": "https://security-tracker.debian.org/tracker/CVE-2020-27350"
        },
        {
          "category": "ADVISORY",
          "locator": "https://bugs.launchpad.net/bugs/1899193"
        },
        {
          "category": "ADVISORY",
          "locator": "https://security.netapp.com/advisory/ntap-20210108-0005/"
        },
        {
          "category": "ADVISORY",
          "locator": "https://www.debian.org/security/2020/dsa-4808"
        },
        {
          "category": "ADVISORY",
          "locator": "https://usn.ubuntu.com/usn/usn-4667-1"
        }
      ],
      "modified": "2022-10-29T13:11:02.438923Z",
      "published": "2020-12-10T03:10:23.901831Z"
    },
    {
      "id": "SNYK-DEBIAN10-APT-407502",
      "name": "SNYK-DEBIAN10-APT-407502",
      "summary": "Improper Verification of Cryptographic Signature",
      "details": "## NVD Description\n**_Note:_** _Versions mentioned in the descripti
on apply only to the upstream `apt` package and not the `apt` package as distributed b
y `Debian:10`._\n_See `How to fix?` for `Debian:10` relevant fixed versions and statu
s._\n\nIt was found that apt-key in apt, all versions, do not correctly validate gpg k
eys with the primary keyring, leading to a potential man-in-the-middle attack.\n## Rem
ediation\nThere is no fixed version for `Debian:10` `apt`.\n## References\n- [ADVISOR
Y](https://security-tracker.debian.org/tracker/CVE-2011-3374)\n- [Debian Bug Report](h
ttps://bugs.debian.org/cgi-bin/bugreport.cgi?bug=642480)\n- [MISC](https://people.cano
nical.com/~ubuntu-security/cve/2011/CVE-2011-3374.html)\n- [MISC](https://seclists.or
g/fulldisclosure/2011/Sep/221)\n- [MISC](https://snyk.io/vuln/SNYK-LINUX-APT-116518)\n
- [MISC](https://ubuntu.com/security/CVE-2011-3374)\n- [RedHat CVE Database](https://a
ccess.redhat.com/security/cve/cve-2011-3374)\n",
      "relationships": [
        {
          "affect": {
            "to": [
              "docker-image|nginx@1.16",
```

```
              "apt/libapt-pkg5.0@1.8.2"
            ],
            "type": "AFFECTS"
          },
          "foundBy": {
            "to": [
              ""
            ],
            "type": "FOUND_BY"
          },
          "suppliedBy": {
            "to": [
              ""
            ],
            "type": "SUPPLIED_BY"
          },
          "ratedBy": {
            "to": [
              ""
            ],
            "type": "RATED_BY",
            "cwes": [
              347
            ],
            "rating": [
              {
                "method": "CVSS_3",
                "score": [
                  {
                    "base": 3.7
                  }
                ],
                "severity": "Low",
                "vector": "CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:N"
              }
            ]
          }
        }
      ],
      "externalReferences": [
        {
          "category": "ADVISORY",
          "locator": "https://security-tracker.debian.org/tracker/CVE-2011-3374"
        },
        {
          "category": "ADVISORY",
          "locator": "https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=642480"
        },
        {
          "category": "ADVISORY",
          "locator": "https://people.canonical.com/~ubuntu-security/cve/2011/CVE-2011-
3374.html"
        },
        {
          "category": "ADVISORY",
          "locator": "https://seclists.org/fulldisclosure/2011/Sep/221"
        },
        {
          "category": "ADVISORY",
          "locator": "https://snyk.io/vuln/SNYK-LINUX-APT-116518"
        },
        {
          "category": "ADVISORY",
          "locator": "https://ubuntu.com/security/CVE-2011-3374"
        },
        {
```

```
          "category": "ADVISORY",
          "locator": "https://access.redhat.com/security/cve/cve-2011-3374"
        }
      ],
      "modified": "2022-11-01T00:08:27.375895Z",
      "published": "2018-06-27T16:20:45.037549Z"
    },
    {
      "id": "SNYK-DEBIAN10-APT-568926",
      "name": "SNYK-DEBIAN10-APT-568926",
      "summary": "Improper Input Validation",
      "details": "## NVD Description\n**_Note:_** _Versions mentioned in the descripti
on apply only to the upstream `apt` package and not the `apt` package as distributed b
y `Debian:10`._\n_See `How to fix?` for `Debian:10` relevant fixed versions and statu
s._\n\nMissing input validation in the ar/tar implementations of APT before version 2.
1.2 could result in denial of service when processing specially crafted deb files.\n##
Remediation\nUpgrade `Debian:10` `apt` to version 1.8.2.1 or higher.\n## References\n-
[ADVISORY](https://security-tracker.debian.org/tracker/CVE-2020-3810)\n- [FEDORA](http
s://lists.fedoraproject.org/archives/list/package-announce@lists.fedoraproject.org/mes
sage/U4PEH357MZM2SUGKETMEHMSGQS652QHH/)\n- [GitHub Issue](https://github.com/Debian/ap
t/issues/111)\n- [MISC](https://bugs.launchpad.net/bugs/1878177)\n- [MISC](https://lis
ts.debian.org/debian-security-announce/2020/msg00089.html)\n- [MISC](https://salsa.deb
ian.org/apt-team/apt/-/commit/dceb1e49e4b8e4dadaf056be34088b415939cda6)\n- [MISC](http
s://tracker.debian.org/news/1144109/accepted-apt-212-source-into-unstable/)\n- [UBUNT
U](https://usn.ubuntu.com/4359-2/)\n- [Ubuntu CVE Tracker](http://people.ubuntu.com/~u
buntu-security/cve/CVE-2020-3810)\n- [Ubuntu Security Advisory](https://usn.ubuntu.co
m/4359-1/)\n",
      "relationships": [
        {
          "affect": {
            "to": [
              "docker-image|nginx@1.16",
              "apt/libapt-pkg5.0@1.8.2"
            ],
            "type": "AFFECTS"
          },
          "foundBy": {
            "to": [
              ""
            ],
            "type": "FOUND_BY"
          },
          "suppliedBy": {
            "to": [
              ""
            ],
            "type": "SUPPLIED_BY"
          },
          "ratedBy": {
            "to": [
              ""
            ],
            "type": "RATED_BY",
            "cwes": [
              20
            ],
            "rating": [
              {
                "method": "CVSS_3",
                "score": [
                  {
                    "base": 5.5
                  }
                ],
                "severity": "Medium",
                "vector": "CVSS:3.1/AV:L/AC:L/PR:N/UI:R/S:U/C:N/I:N/A:H"
```

```
              }
            ]
          }
        }
      ],
      "externalReferences": [
        {
          "category": "ADVISORY",
          "locator": "https://security-tracker.debian.org/tracker/CVE-2020-3810"
        },
        {
          "category": "ADVISORY",
          "locator": "https://lists.fedoraproject.org/archives/list/package-announce@l
ists.fedoraproject.org/message/U4PEH357MZM2SUGKETMEHMSGQS652QHH/"
        },
        {
          "category": "ADVISORY",
          "locator": "https://github.com/Debian/apt/issues/111"
        },
        {
          "category": "ADVISORY",
          "locator": "https://bugs.launchpad.net/bugs/1878177"
        },
        {
          "category": "ADVISORY",
          "locator": "https://lists.debian.org/debian-security-announce/2020/msg00089.
html"
        },
        {
          "category": "ADVISORY",
          "locator": "https://salsa.debian.org/apt-team/apt/-/commit/dceb1e49e4b8e4dad
af056be34088b415939cda6"
        },
        {
          "category": "ADVISORY",
          "locator": "https://tracker.debian.org/news/1144109/accepted-apt-212-source-
into-unstable/"
        },
        {
          "category": "ADVISORY",
          "locator": "https://usn.ubuntu.com/4359-2/"
        },
        {
          "category": "ADVISORY",
          "locator": "http://people.ubuntu.com/~ubuntu-security/cve/CVE-2020-3810"
        },
        {
          "category": "ADVISORY",
          "locator": "https://usn.ubuntu.com/4359-1/"
        }
      ],
      "modified": "2022-11-01T00:08:51.907776Z",
      "published": "2020-05-12T14:19:01.052295Z"
    },
    {
      "id": "SNYK-DEBIAN10-APT-1049974",
      "name": "SNYK-DEBIAN10-APT-1049974",
      "summary": "Integer Overflow or Wraparound",
      "details": "## NVD Description\n**_Note:_** _Versions mentioned in the descripti
on apply only to the upstream `apt` package and not the `apt` package as distributed b
y `Debian:10`._\n_See `How to fix?` for `Debian:10` relevant fixed versions and statu
s._\n\nAPT had several integer overflows and underflows while parsing .deb packages, a
ka GHSL-2020-168 GHSL-2020-169, in files apt-pkg/contrib/extracttar.cc, apt-pkg/deb/de
bfile.cc, and apt-pkg/contrib/arfile.cc. This issue affects: apt 1.2.32ubuntu0 version
s prior to 1.2.32ubuntu0.2; 1.6.12ubuntu0 versions prior to 1.6.12ubuntu0.2; 2.0.2ubun
tu0 versions prior to 2.0.2ubuntu0.2; 2.1.10ubuntu0 versions prior to 2.1.10ubuntu0.
```

```
1;\n## Remediation\nUpgrade `Debian:10` `apt` to version 1.8.2.2 or higher.\n## Refere
nces\n- [ADVISORY](https://security-tracker.debian.org/tracker/CVE-2020-27350)\n- [CON
FIRM](https://bugs.launchpad.net/bugs/1899193)\n- [CONFIRM](https://security.netapp.co
m/advisory/ntap-20210108-0005/)\n- [DEBIAN](https://www.debian.org/security/2020/dsa-4
808)\n- [UBUNTU](https://usn.ubuntu.com/usn/usn-4667-1)\n",
      "relationships": [
        {
          "affect": {
            "to": [
              "docker-image|nginx@1.16",
              "apt@1.8.2",
              "apt/libapt-pkg5.0@1.8.2"
            ],
            "type": "AFFECTS"
          },
          "foundBy": {
            "to": [
              ""
            ],
            "type": "FOUND_BY"
          },
          "suppliedBy": {
            "to": [
              ""
            ],
            "type": "SUPPLIED_BY"
          },
          "ratedBy": {
            "to": [
              ""
            ],
            "type": "RATED_BY",
            "cwes": [
              190
            ],
            "rating": [
              {
                "method": "CVSS_3",
                "score": [
                  {
                    "base": 5.7
                  }
                ],
                "severity": "Medium",
                "vector": "CVSS:3.1/AV:L/AC:L/PR:H/UI:N/S:C/C:L/I:L/A:L"
              }
            ]
          }
        }
      ],
      "externalReferences": [
        {
          "category": "ADVISORY",
          "locator": "https://security-tracker.debian.org/tracker/CVE-2020-27350"
        },
        {
          "category": "ADVISORY",
          "locator": "https://bugs.launchpad.net/bugs/1899193"
        },
        {
          "category": "ADVISORY",
          "locator": "https://security.netapp.com/advisory/ntap-20210108-0005/"
        },
        {
          "category": "ADVISORY",
          "locator": "https://www.debian.org/security/2020/dsa-4808"
```

```
        },
        {
          "category": "ADVISORY",
          "locator": "https://usn.ubuntu.com/usn/usn-4667-1"
        }
      ],
      "modified": "2022-10-29T13:11:02.438923Z",
      "published": "2020-12-10T03:10:23.901831Z"
    },
    {
      "id": "SNYK-DEBIAN10-APT-407502",
      "name": "SNYK-DEBIAN10-APT-407502",
      "summary": "Improper Verification of Cryptographic Signature",
      "details": "## NVD Description\n**_Note:_** _Versions mentioned in the descripti
on apply only to the upstream `apt` package and not the `apt` package as distributed b
y `Debian:10`._\n_See `How to fix?` for `Debian:10` relevant fixed versions and statu
s._\n\nIt was found that apt-key in apt, all versions, do not correctly validate gpg k
eys with the primary keyring, leading to a potential man-in-the-middle attack.\n## Rem
ediation\nThere is no fixed version for `Debian:10` `apt`.\n## References\n- [ADVISOR
Y](https://security-tracker.debian.org/tracker/CVE-2011-3374)\n- [Debian Bug Report](h
ttps://bugs.debian.org/cgi-bin/bugreport.cgi?bug=642480)\n- [MISC](https://people.cano
nical.com/~ubuntu-security/cve/2011/CVE-2011-3374.html)\n- [MISC](https://seclists.or
g/fulldisclosure/2011/Sep/221)\n- [MISC](https://snyk.io/vuln/SNYK-LINUX-APT-116518)\n
- [MISC](https://ubuntu.com/security/CVE-2011-3374)\n- [RedHat CVE Database](https://a
ccess.redhat.com/security/cve/cve-2011-3374)\n",
      "relationships": [
        {
          "affect": {
            "to": [
              "docker-image|nginx@1.16",
              "apt@1.8.2",
              "apt/libapt-pkg5.0@1.8.2"
            ],
            "type": "AFFECTS"
          },
          "foundBy": {
            "to": [
              ""
            ],
            "type": "FOUND_BY"
          },
          "suppliedBy": {
            "to": [
              ""
            ],
            "type": "SUPPLIED_BY"
          },
          "ratedBy": {
            "to": [
              ""
            ],
            "type": "RATED_BY",
            "cwes": [
              347
            ],
            "rating": [
              {
                "method": "CVSS_3",
                "score": [
                  {
                    "base": 3.7
                  }
                ],
                "severity": "Low",
                "vector": "CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:N"
              }
```

```
            ]
          }
        }
      ],
      "externalReferences": [
        {
          "category": "ADVISORY",
          "locator": "https://security-tracker.debian.org/tracker/CVE-2011-3374"
        },
        {
          "category": "ADVISORY",
          "locator": "https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=642480"
        },
        {
          "category": "ADVISORY",
          "locator": "https://people.canonical.com/~ubuntu-security/cve/2011/CVE-2011-
3374.html"
        },
        {
          "category": "ADVISORY",
          "locator": "https://seclists.org/fulldisclosure/2011/Sep/221"
        },
        {
          "category": "ADVISORY",
          "locator": "https://snyk.io/vuln/SNYK-LINUX-APT-116518"
        },
        {
          "category": "ADVISORY",
          "locator": "https://ubuntu.com/security/CVE-2011-3374"
        },
        {
          "category": "ADVISORY",
          "locator": "https://access.redhat.com/security/cve/cve-2011-3374"
        }
      ],
      "modified": "2022-11-01T00:08:27.375895Z",
      "published": "2018-06-27T16:20:45.037549Z"
    },
    {
      "id": "SNYK-DEBIAN10-EXPAT-2329087",
      "name": "SNYK-DEBIAN10-EXPAT-2329087",
      "summary": "Incorrect Calculation",
      "details": "## NVD Description\n**_Note:_** _Versions mentioned in the descripti
on apply only to the upstream `expat` package and not the `expat` package as distribut
ed by `Debian:10`._\n_See `How to fix?` for `Debian:10` relevant fixed versions and st
atus._\n\nIn Expat (aka libexpat) before 2.4.3, a left shift by 29 (or more) places in
the storeAtts function in xmlparse.c can lead to realloc misbehavior (e.g., allocating
too few bytes, or only freeing memory).\n## Remediation\nUpgrade `Debian:10` `expat` t
o version 2.2.6-2+deb10u2 or higher.\n## References\n- [ADVISORY](https://security-tra
cker.debian.org/tracker/CVE-2021-45960)\n- [MISC](https://bugzilla.mozilla.org/show_bu
g.cgi?id=1217609)\n- [MISC](https://github.com/libexpat/libexpat/issues/531)\n- [MISC]
(https://github.com/libexpat/libexpat/pull/534)\n- [cve@mitre.org](http://www.openwal
l.com/lists/oss-security/2022/01/17/3)\n- [cve@mitre.org](https://security.netapp.com/
advisory/ntap-20220121-0004/)\n- [cve@mitre.org](https://www.tenable.com/security/tns-
2022-05)\n- [cve@mitre.org](https://www.debian.org/security/2022/dsa-5073)\n- [cve@mit
re.org](https://cert-portal.siemens.com/productcert/pdf/ssa-484086.pdf)\n- [cve@mitre.
org](https://security.gentoo.org/glsa/202209-24)\n",
      "relationships": [
        {
          "affect": {
            "to": [
              "docker-image|nginx@1.16",
              "nginx-module-image-filter@1.16.1-1~buster",
              "libgd2/libgd3@2.2.5-5.2",
              "fontconfig/libfontconfig1@2.13.1-2",
              "expat/libexpat1@2.2.6-2+deb10u1"
```

```
            ],
            "type": "AFFECTS"
          },
          "foundBy": {
            "to": [
              ""
            ],
            "type": "FOUND_BY"
          },
          "suppliedBy": {
            "to": [
              ""
            ],
            "type": "SUPPLIED_BY"
          },
          "ratedBy": {
            "to": [
              ""
            ],
            "type": "RATED_BY",
            "cwes": [
              682
            ],
            "rating": [
              {
                "method": "CVSS_3",
                "score": [
                  {
                    "base": 8.8
                  }
                ],
                "severity": "High",
                "vector": "CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H"
              }
            ]
          }
        }
      ],
      "externalReferences": [
        {
          "category": "ADVISORY",
          "locator": "https://security-tracker.debian.org/tracker/CVE-2021-45960"
        },
        {
          "category": "ADVISORY",
          "locator": "https://bugzilla.mozilla.org/show_bug.cgi?id=1217609"
        },
        {
          "category": "ADVISORY",
          "locator": "https://github.com/libexpat/libexpat/issues/531"
        },
        {
          "category": "ADVISORY",
          "locator": "https://github.com/libexpat/libexpat/pull/534"
        },
        {
          "category": "ADVISORY",
          "locator": "http://www.openwall.com/lists/oss-security/2022/01/17/3"
        },
        {
          "category": "ADVISORY",
          "locator": "https://security.netapp.com/advisory/ntap-20220121-0004/"
        },
        {
          "category": "ADVISORY",
          "locator": "https://www.tenable.com/security/tns-2022-05"
```

```
        },
        {
          "category": "ADVISORY",
          "locator": "https://www.debian.org/security/2022/dsa-5073"
        },
        {
          "category": "ADVISORY",
          "locator": "https://cert-portal.siemens.com/productcert/pdf/ssa-484086.pdf"
        },
        {
          "category": "ADVISORY",
          "locator": "https://security.gentoo.org/glsa/202209-24"
        }
      ],
      "modified": "2023-02-14T13:37:37.505975Z",
      "published": "2022-01-02T01:41:26.770663Z"
    },
    {
      "id": "SNYK-DEBIAN10-EXPAT-2331803",
      "name": "SNYK-DEBIAN10-EXPAT-2331803",
      "summary": "Integer Overflow or Wraparound",
      "details": "## NVD Description\n**_Note:_** _Versions mentioned in the descripti
on apply only to the upstream `expat` package and not the `expat` package as distribut
ed by `Debian:10`._\n_See `How to fix?` for `Debian:10` relevant fixed versions and st
atus._\n\ndefineAttribute in xmlparse.c in Expat (aka libexpat) before 2.4.3 has an in
teger overflow.\n## Remediation\nUpgrade `Debian:10` `expat` to version 2.2.6-2+deb10u
2 or higher.\n## References\n- [ADVISORY](https://security-tracker.debian.org/tracker/
CVE-2022-22824)\n- [cve@mitre.org](https://github.com/libexpat/libexpat/pull/539)\n-
[cve@mitre.org](http://www.openwall.com/lists/oss-security/2022/01/17/3)\n- [cve@mitr
e.org](https://www.tenable.com/security/tns-2022-05)\n- [cve@mitre.org](https://www.de
bian.org/security/2022/dsa-5073)\n- [cve@mitre.org](https://cert-portal.siemens.com/pr
oductcert/pdf/ssa-484086.pdf)\n- [cve@mitre.org](https://security.gentoo.org/glsa/2022
09-24)\n",
      "relationships": [
        {
          "affect": {
            "to": [
              "docker-image|nginx@1.16",
              "nginx-module-image-filter@1.16.1-1~buster",
              "libgd2/libgd3@2.2.5-5.2",
              "fontconfig/libfontconfig1@2.13.1-2",
              "expat/libexpat1@2.2.6-2+deb10u1"
            ],
            "type": "AFFECTS"
          },
          "foundBy": {
            "to": [
              ""
            ],
            "type": "FOUND_BY"
          },
          "suppliedBy": {
            "to": [
              ""
            ],
            "type": "SUPPLIED_BY"
          },
          "ratedBy": {
            "to": [
              ""
            ],
            "type": "RATED_BY",
            "cwes": [
              190
            ],
            "rating": [
```

```
              {
                "method": "CVSS_3",
                "score": [
                  {
                    "base": 9.8
                  }
                ],
                "severity": "Critical",
                "vector": "CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H"
              }
            ]
          }
        }
      ],
      "externalReferences": [
        {
          "category": "ADVISORY",
          "locator": "https://security-tracker.debian.org/tracker/CVE-2022-22824"
        },
        {
          "category": "ADVISORY",
          "locator": "https://github.com/libexpat/libexpat/pull/539"
        },
        {
          "category": "ADVISORY",
          "locator": "http://www.openwall.com/lists/oss-security/2022/01/17/3"
        },
        {
          "category": "ADVISORY",
          "locator": "https://www.tenable.com/security/tns-2022-05"
        },
        {
          "category": "ADVISORY",
          "locator": "https://www.debian.org/security/2022/dsa-5073"
        },
        {
          "category": "ADVISORY",
          "locator": "https://cert-portal.siemens.com/productcert/pdf/ssa-484086.pdf"
        },
        {
          "category": "ADVISORY",
          "locator": "https://security.gentoo.org/glsa/202209-24"
        }
      ],
      "modified": "2023-02-14T13:39:18.516672Z",
      "published": "2022-01-08T13:52:14.479733Z"
    }
  ],
  "name": "docker-image|nginx-sha256:d20aa6d1cae56fd17cd458f4807e0de462caf2336f0b70b5e
eb69fcaaf30dd9c",
  "dataLicense": "CC0-1.0",
  "documentNamespace": "spdx.org/spdxdocs/docker-image|nginx-feb02ce6-cd47-49c2-9a97-2
b4833b4a1f0",
  "relationships": [
    {
      "from": "SPDXRef-docker-image|nginx",
      "to": [
        "SPDXRef-index.docker.io/library/nginx-sha256:d20aa6d1cae56fd17cd458f4807e0de4
62caf2336f0b70b5eeb69fcaaf30dd9c"
      ],
      "type": "DESCRIBES"
    }
  ],
  "packages": [
    {
      "SPDXID": "SPDXRef-index.docker.io/library/nginx-sha256:d20aa6d1cae56fd17cd458f4
```

```
807e0de462caf2336f0b70b5eeb69fcaaf30dd9c",
      "versionInfo": "sha256:d20aa6d1cae56fd17cd458f4807e0de462caf2336f0b70b5eeb69fcaa
f30dd9c",
      "id": "SPDXRef-index.docker.io/library/nginx-sha256:d20aa6d1cae56fd17cd458f4807e
0de462caf2336f0b70b5eeb69fcaaf30dd9c",
      "name": "index.docker.io/library/nginx",
      "checksums": [
        {
          "algorithm": "SHA256",
          "checksumValue": "d20aa6d1cae56fd17cd458f4807e0de462caf2336f0b70b5eeb69fcaaf
30dd9c"
        }
      ]
    }
  ]
}
```

Supply Chain Security Tools - Scan 2.0

This topic gives you an overview of Supply Chain Security Tools (SCST) - Scan 2.0. Include this component in your software supply chain to help identify vulnerabilities earlier in the development life cycle. This helps to increase the security posture of your application.

Overview

SCST - Scan 2.0 is the next-generation scanning framework for Tanzu Application Platform. It will supersede the SCST - Scan component in a future release. It provides a framework to scan workload components.

With SCST - Scan 2.0 you can:

- Scan container images for a workload that is built by the supply chain or provided as part of a workload definition for known Common Vulnerabilities and Exposures (CVEs).
- Use your preferred container image scan solution such as Anchore's Grype, Aqua's Trivy, Palo Alto's Prisma, and VMware Carbon Black Cloud.
- Post the scan results in an industry-standard format, such as CycloneDX or SPDX, as an OCI artifact to an OCI-compliant container image registry. By using industry standards, you do not need to be familiar with proprietary Tanzu Application Platform configurations.

AMR Observer

Downstream services such as Tanzu CLI Insight plug-in, Supply Chain Choreographer, and Security Analysis dashboards in the Tanzu Developer Portal depend on data being in the SCST - Store component. Because pushing scan results to the proprietary store endpoint is decoupled from the scan framework in SCST - Scan 2.0, AMR Observer observes results pushed to a registry, parses the results, and pushes them to the SCST - Store component.

For information about AMR observer, see Overview of Supply Chain Security Tools for Tanzu – Store.

Integrating into a supply chain

The SCST - Scan 2.0 component defines how to scan a container image with a scan solution using the generic Kubernetes custom resource `ImageVulnerabilityScan`. This provides a generic interface that allows you to declare how the Tanzu Application Platform executes a scan on a container image for a container image scan solution. For Cartographer to stamp out an `ImageVulnerabilityScan` custom resource as part of a supply chain execution, the

`ImageVulnerabilityScan` must be wrapped in a `ClusterImageTemplate` custom resource. This custom resource tells Cartographer not only how to stamp out the `ImageVulnerabilityScan` template, but also what configurations are passed to it.

Getting started with SCST - Scan 2.0

To use the SCST - Scan 2.0 component, see Getting Started with Supply Chain Security Tools - Scan 2.0.

Getting Started with Supply Chain Security Tools - Scan 2.0

Get started with Supply Chain Security Tools (SCST) - Scan 2.0 by using the following topics:

Install Supply Chain Security Tools - Scan 2.0 in a cluster

Enable App Scanning for default Test and Scan supply chains

Bring your own scanner with Supply Chain Security Tools - Scan 2.0

Verifying Scanning with Supply Chain Integration

Install Supply Chain Security Tools - Scan 2.0 in a cluster

This topic describes how to install Supply Chain Security Tools (SCST) - Scan 2.0 if are not using a profile. By default, SCST - Scan 2.0 is installed in the build and full profiles.

Note

Follow the steps in this topic if you do not want to use a profile to install SCST - Scan 2.0. For more information about profiles, see Installation profiles in Tanzu Application Platform v1.9.

Prerequisites

SCST - Scan 2.0 requires the following prerequisites:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- Install the Tekton component.
- To store results, install AMR, and AMR Observer. Downstream Tanzu Application Platform services, such as Tanzu Developer Portal and Tanzu CLI, depend on scan results stored in SCST - Store to display correctly. For more information, see Artifact Metadata Repository Observer for Supply Chain Security Tools - Store.

Note

If you are installing SCST - Scan 2.0 in a cluster with restricted Kubernetes Pod Security Standards, you must update configurations for the Tekton Pipelines package. See Scanning in a cluster with restricted Kubernetes Pod Security Standards.

Configure properties

When you install SCST - Scan 2.0, you can configure the following optional properties:

Key	Default	Type	Description
caCertData	""	string	The custom certificates trusted by the scan's connections
docker.import	true	Boolean	Import `docker.pullSecret` from another namespace (requires secretgen-controller). Set to false if the secret is already present.
docker.pullSecret	registries-credentials	string	Name of a Docker pull secret in the deployment namespace to pull the scanner images
scans.maxConcurrentScans	10	integer	Maximum number of scans running at one time. Set to 0 to have no limit.
scans.priorityClassName	""	string	Name of a predefined PriorityClass to apply to scan pods
workspace.storageSize	2Gi	string	Size of the PersistentVolume that the Tekton pipelineruns uses
workspace.storageClass	""	string	Name of the storage class to use while creating the PersistentVolume claims used by Tekton pipelineruns

Note

If the StorageClass you select does not have a node limit but uses the node storage, such as hostpath, the nodes must have large enough disks. For example, if a scan creates a 2Gi volume on a hostpath type storage class, `2Gi * number of AMR images` indicates how much storage this cluster needs overall. `2Gi * number of AMR images / number of nodes` indicates how much storage each node needs.

Install

To install SCST - Scan 2.0:

1. List version information for the package:

   ```
   tanzu package available list app-scanning.apps.tanzu.vmware.com --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package available list app-scanning.apps.tanzu.vmware.com --namespace tap-install
   - Retrieving package versions for app-scanning.apps.tanzu.vmware.com...
     NAME                                VERSION              RELEASED-AT
     app-scanning.apps.tanzu.vmware.com  0.1.0              2023-03-01 20:00:00 -0400 EDT
   ```

2. (Optional) Make changes to the default installation settings:

   Retrieve the configurable settings:

   ```
   tanzu package available get app-scanning.apps.tanzu.vmware.com/VERSION --values-schema --namespace tap-install
   ```

   Where `VERSION` is your package version number. For example, `0.1.0`.

   For example:

```
tanzu package available get app-scanning.apps.tanzu.vmware.com/0.1.0 --values-s
chema --namespace tap-install
| Retrieving package details for app-scanning.apps.tanzu.vmware.com/0.1.0...

  KEY                     DEFAULT                 TYPE     DESCRIPTION
  docker.import           true                    boolean  Import `docker.pullS
ecret` from another namespace (requires
                                                           secretgen-controlle
r). Set to false if the secret will already be present.
  docker.pullSecret       registries-credentials  string   Name of a Docker pul
l secret in the deployment namespace to pull the scanner
                                                           images.
  workspace.storageSize   2Gi                     string   Size of the Persiste
nt Volume to be used by the tekton pipelineruns
  workspace.storageClass                          string   Name of the storage
class to use while creating the Persistent Volume Claims
                                                           used by tekton pipel
ineruns
  caCertData                                      string   The custom certifica
tes to be trusted by the scan's connections
```

To edit any of the default installation settings, create an `app-scanning-values-file.yaml` and append the key-value pairs to be modified to the file. For example:

```
scans:
  workspace:
    storageSize: 200Mi
```

3. Install the package. If you did not edit the default installation settings, you do not need to specify the `--values-file` flag.

```
tanzu package install app-scanning --package-name app-scanning.apps.tanzu.vmwar
e.com \
    --version VERSION \
    --namespace tap-install \
    --values-file app-scanning-values-file.yaml
```

Where `VERSION` is your package version number. For example, `0.1.0`.

For example:

```
tanzu package install app-scanning --package app-scanning.apps.tanzu.vmware.com
\
    --version 0.1.0 \
    --namespace tap-install \
    --values-file app-scanning-values-file.yaml

    Installing package 'app-scanning.apps.tanzu.vmware.com'
    Getting package metadata for 'app-scanning.apps.tanzu.vmware.com'
    Creating service account 'app-scanning-default-sa'
    Creating cluster admin role 'app-scanning-default-cluster-role'
    Creating cluster role binding 'app-scanning-default-cluster-rolebinding'
    Creating package resource
    Waiting for 'PackageInstall' reconciliation for 'app-scanning'
    'PackageInstall' resource install status: Reconciling
    'PackageInstall' resource install status: ReconcileSucceeded
```

Configure service accounts and registry credentials

This section contains instructions for running a standalone `ImageVulnerabilityScan` or using multiple registries.

If the image that you are scanning using a default scanner or your own scanner, and your vulnerability scanner image are located in private registries different from the `Tanzu Application Platform bundles registry`, you must edit your scanner service account to include registry credentials for these registries.

Important

If your use case is listed below, skip this topic and proceed to Enable App Scanning for default Test and Scan supply chains.

- You are running an `ImageVulnerabilityScan` in the context of a supply chain.
- You used the Namespace Provisioner to provision your developer namespace. For more information, see the Namespace Provisioner documentation.

To configure service accounts and registry credentials, SCST - Scan 2.0 requires the following access:

Registry	Permission	Service Account	Example
Tanzu Application Platform bundles registry	Read	scanner	`registry.tanzu.vmware.com`
Target image registry	Read	scanner	`your-registry.io`
Vulnerability scanner image registry	Read	scanner	`your-registry.io`
Scan results location registry	Write	publisher	`your-registry.io`

Where:

- `Tanzu Application Platform bundles registry` is the registry containing the Tanzu Application Platform bundles. This is the registry from the Relocate images to a registry step or `registry.tanzu.vmware.com`.
- `Target image registry` is the registry containing the image to scan. This registry credential is required if you are scanning a private image. The `image to scan is` called the `target image` or `TARGET-IMAGE`.
- `Vulnerability scanner image registry` is the registry containing your vulnerability scanner image. This is only needed if you are bringing your own scanner and your vulnerability scanner image is located in a private registry different from the `Tanzu Application Platform bundles registry`.
- `Scan results location registry` is the registry where scan results are published.

To configure service accounts and registry credentials:

1. Create a secret `scanning-tap-component-read-creds` with read access to the registry containing the Tanzu Application Platform bundles. This pulls the SCST - Scan 2.0 images. If you previously relocated the Tanzu Application Platform bundles to your own registry, you can also place your vulnerability scanner image in this registry.

```
read -s TAP_REGISTRY_PASSWORD
kubectl create secret docker-registry scanning-tap-component-read-creds \
  --docker-username=TAP-REGISTRY-USERNAME \
  --docker-password=$TAP_REGISTRY_PASSWORD \
  --docker-server=TAP-REGISTRY-URL \
  -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where scanning occurs.

2. If you are scanning a private target image, create a secret `target-image-read-creds` with read access to the registry containing that target image.

Important

If you followed the directions for Install Tanzu Application Platform, you can skip this step and use the `targetImagePullSecret` secret with your service account as referenced in your `tap-values.yaml`. See Full profile.

```
read -s REGISTRY_PASSWORD
kubectl create secret docker-registry target-image-read-creds \
  --docker-username=REGISTRY-USERNAME \
  --docker-password=$REGISTRY_PASSWORD \
  --docker-server=REGISTRY-URL \
  -n DEV-NAMESPACE
```

3. Create a secret `write-creds` with write access to the registry for the scanner to upload the scan results to.

```
read -s WRITE_PASSWORD
kubectl create secret docker-registry write-creds \
  --docker-username=WRITE-USERNAME \
  --docker-password=$WRITE_PASSWORD \
  --docker-server=DESTINATION-REGISTRY-URL \
  -n DEV-NAMESPACE
```

4. (Optional) If you are bringing your own vulnerability scanner and your vulnerability scanner image is located in a private registry different from the registry containing your Tanzu Application Platform bundles, you must create a secret `vulnerability-scanner-image-read-creds` with read access to the registry.

```
read -s WRITE_PASSWORD
kubectl create secret docker-registry vulnerability-scanner-image-read-creds \
  --docker-username=WRITE-USERNAME \
  --docker-password=$WRITE_PASSWORD \
  --docker-server=REGISTRY-URL \
  -n DEV-NAMESPACE
```

5. Create a `scanner-sa.yaml` file containing the service account `scanner` which enables SCST - Scan 2.0 to pull both the vulnerability scanner image and target image. Attach the one or more read secrets created earlier pulling the Tanzu Application Platform bundles, and optionally, your vulnerability scanner image under `imagePullSecrets`. Attach the read secret created earlier for your target image under `secrets`.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: scanner
  namespace: DEV-NAMESPACE
imagePullSecrets:
- name: scanning-tap-component-read-creds
- name: vulnerability-scanner-image-read-creds # optional
secrets:
- name: target-image-read-creds
```

Where:

- `imagePullSecrets.name` includes the name of the secret used to pull the scan component from the registry. If you are bringing your own vulnerability scanner and the vulnerability scanner image is located in a separate private registry, you must also include the name of the secret with those registry credentials.
- `secrets.name` is the name of the secret used to pull the target image to scan. This is required if the image you are scanning is private.

6. Apply the service account to your developer namespace by running:

```
kubectl apply -f scanner-sa.yaml
```

7. Create a `publisher-sa.yaml` file containing the service account `publisher` which enables SCST - Scan 2.0 to push the scan results to a user specified registry.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: publisher
  namespace: DEV-NAMESPACE
imagePullSecrets:
- name: scanning-tap-component-read-creds
secrets:
- name: write-creds
```

Where:

- `imagePullSecrets.name` is the name of the secret used to pull the scan component image from the registry.
- `secrets.name` is the name of the secret used to publish the scan results.

8. Apply the service account to your developer namespace by running:

```
kubectl apply -f publisher-sa.yaml
```

(Optional) Set up your registry retention policy

Although Tanzu Application Platform ingests scan artifacts into the Metadata Store, and stores information such as packages and parsed vulnerabilities, only a pointer to the original SBOM location is stored. The original SBOM generated by the scan is not preserved within the Metadata Store. VMware recommends that you keep these original artifacts according to your organization's archival requirements.

If the registry specified to push scan results to support retention policies, you can configure the registry to delete old scan results automatically, depending on your archival requirements. Scan result artifacts accumulate over time and can quickly consume hard disk space.

For information about configuring Harbor tag retention rules, see the Harbor documentation. For example, you can configure Harbor to retain the most recently pushed # artifacts or retain the artifacts pushed within the last # days.

Retention policy setup differs between registry providers. Confirm with your specific registry's documentation about the configuration options.

Enable SCST - Scan 2.0 for default Test and Scan supply chains

This topic tells you how to enable Supply Chain Security Tools (SCST) - Scan 2.0 and an included container image scanner for Out of the Box Supply Chain with Testing and Scanning. The default configuration for Out of the Box Supply Chain with Testing and Scanning is SCST - Scan 1.0.

Overview

SCST - Scan 2.0 includes two integrations for container image scanners:

Container Image Scanner	Documentation	Cluster Image Template Name	Status
Aqua Trivy	Link	`image-vulnerability-scan-trivy`	Recommended scanner for SCST - Scan 2.0
Anchore Grype	Link	`image-vulnerability-scan-grype`	Alternative to Trivy that is used in SCST - Scan 1.0

VMware recommends using Aqua Trivy scanner with Tanzu Application Platform for container image scanning. If you want to remain consistent with the default scanner in SCST - Scan 1.0, Anchore Grype is included as an open-source alternative. Additionally, you can build an integration for additional scanners. For more information, see Bring your own scanner with Supply Chain Security Tools - Scan 2.0.

Enable with OOTB supply chain

To enable SCST - Scan 2.0 with an OOTB supply chain using the Trivy scanner:

1. Update your `tap-values.yaml` file to specify the Trivy `ClusterImageTemplate`. For example:

   ```
   ootb_supply_chain_testing_scanning:
     image_scanner_template_name: image-vulnerability-scan-trivy
   ```

2. (Optional) In Tanzu Application Platform v1.8 if you are using a vulnerability scanner other than Trivy, you must specify the registry location of the container image to use for scanning. For example, if you are using the Grype template, set `ootb_supply_chain_testing_scanning.image_scanning_cli` to the Grype image, for example,

   ```
   ootb_supply_chain_testing_scanning:
     image_scanner_template_name: image-vulnerability-scan-grype
     image_scanning_cli:
       image: registry.tanzu.vmware.com/tanzu-application-platform/tap-packages@sh
   a256:feb1cdbd5c918aae7a89bdb2aa39d486bf6ffc81000764b522842e5934578497
   ```

   This example uses the packaged Grype image, but you can also use images from public repositories such as `anchore/grype:latest`.

3. Update your Tanzu Application Platform installation by running:

   ```
   tanzu package installed update tap -p tap.tanzu.vmware.com -v TAP-VERSION  --va
   lues-file tap-values.yaml -n tap-install
   ```

   Where `TAP-VERSION` is the version of Tanzu Application Platform installed.

4. Verify the scan capability is working as expected by creating a workload. See Verify scanning with Supply Chain integration.

Bring your own scanner with Supply Chain Security Tools - Scan 2.0

This topic tells you how to bring your own scanner to use with Supply Chain Security Tools (SCST) - Scan 2.0.

Overview

Supply Chain Security Tools (SCST) - Scan 2.0 includes integrations with Trivy and Grype and examples for the following container image scanning tools:

- Trivy
- Prisma
- Snyk
- Carbon Black Cloud

You might have an existing investment in a scan solution that VMware does not have a published integration with. You can build an integration to your existing scan solution with SCST - Scan 2.0. To use your own scanner:

1. Create an ImageVulnerabilityScan template that tells Tanzu Application Platform how to run your scanner. See Customize an ImageVulnerabilityScan.
2. Verify that your ImageVulnerabilityScan is working correctly so that downstream Tanzu Application Platform services work correctly. See Verify an ImageVulnerabilityScan.
3. Wrap your ImageVulnerabilityScan in a ClusterImageTemplate. The ClusterImageTemplate wraps the ImageVulnerabilityScan and allows the Tanzu Application Platform supply chain to run the scan job. See Author a ClusterImageTemplate for Supply Chain integration.

Prerequisites

You must have the following prerequisites:

- Provide a Vulnerability Scanner Image with one of the following methods:
    - Use a publicly available image that contains the scanner CLI. For example, the official Aqua image for Trivy from Dockerhub.
    - Build your own image with the scanner CLI, which allows for a more customizable scanning experience. For example, you can create an image with the scanner CLI with any dependencies required to run the scanner CLI and manage your image to meet the Tanzu Application Platform compliance standards.
- Know how your preferred scanner works. For example, what commands to use to call scan results.

Bring your own scanner using an ImageVulnerabilityScan

This topic tells you how to bring your own scanner using an `ImageVulnerabilityScan`. An `ImageVulnerabilityScan` allows you to scan with any scanner by defining your scan as a Tekton step. For more information, see the Tekton documentation.

Customize an ImageVulnerabilityScan

To customize an ImageVulnerabilityScan to use your scanner:

1. Create a file named `image-vulnerability-scan.yaml`.

    ```
    apiVersion: app-scanning.apps.tanzu.vmware.com/v1alpha1
    kind: ImageVulnerabilityScan
    ```

```
metadata:
  name: generic-image-scan
  namespace: DEV-NAMESPACE
  annotations:
    app-scanning.apps.tanzu.vmware.com/scanner-name: SCANNER-NAME
spec:
  image: TARGET-IMAGE
  scanResults:
    location: registry/project/scan-results
  serviceAccountNames:
    scanner: scanner
    publisher: publisher
  steps:
  - name: scan
    image: SCANNER-IMAGE
    command: ["SCANNER-CLI-COMMAND"]
    args:
    ...
```

Where:

- `DEV-NAMESPACE` is the developer namespace where scanning occurs.
- `spec.image` is the image that you are scanning. You must specify the digest. See Retrieving an image digest.
- `scanResults.location` is the registry URL where results are uploaded. For example, `my.registry/scan-results`.
- `serviceAccountNames` includes:
  - `scanner` is the service account that runs the scan. It must have read access to `image`.
  - `publisher` is the service account that uploads results. It must have write access to `scanResults.location`.
- `SCANNER-IMAGE` is your vulnerability scanner image, such as the image containing the scanner of your choice.
- `SCANNER-CLI-COMMAND` is the scanner's CLI command.
- `SCANNER-NAME` is the scanner image name that is reported in the Tanzu Developer Portal, formerly Tanzu Application Platform GUI.

Important

Do not define `write-certs` or `cred-helper` as step names. These names are already used during scanning.

2. Configure the `scan` step. You must input your scanner specific `image`, `command`, and `args`. For example:

```
- name: scan
  image: anchore/grype:latest
  command: ["grype"]
  args:
  - registry:$(params.image)
  - -o
  - cyclonedx
  - --file
  - $(params.scan-results-path)/scan.cdx
```

To pass `spec.image` and `scanResults.location` to `args`, you can use `$(params.image)` and `$(params.scan-results-path)`.

Because volumes on a Tekton pipeline are shared amongst steps, files created by one step are consumable by the other steps. The scan controller applies the following security context to `pipelinerun.spec.podTemplate`:

```
runAsUser: 65534
fsGroup: 65534
runAsGroup: 65534
```

Note

If you populate any of the following fields in the `securityContext`, you must populate all the other fields or you might see a runtime error in the `ImageVulnerabilityScan` controller:

```
allowPrivilegeEscalation
runAsNonRoot
seccompProfile
capabilities
```

The `SCANNER-IMAGE` runs and manipulates files with user and group ids of `65534`.

Configuration options

This section lists optional and required ImageVulnerabilityScan specifications fields.

Required fields:

- `image` is the registry URL and digest of the target image. For example, `nginx@sha256:aa0afebbb3cfa473099a62c4b32e9b3fb73ed23f2a75a65ce1d4b4f55a5c2ef2`.
- `scanResults.location` is the registry URL where results are uploaded. For example, `my.registry/scan-results`.

Optional fields:

- `activeKeychains` is an array of enabled credential helpers to authenticate against registries using workload identity mechanisms.

  ```
  activeKeychains:
  - name: acr  # Azure Container Registry
  - name: ecr  # Elastic Container Registry
  - name: gcr  # Google Container Registry
  - name: ghcr # Github Container Registry
  ```

- `serviceAccountNames` includes:
  - `scanner` is the service account that runs the scan. It must have read access to `image`.
  - `publisher` is the service account that uploads results. It must have write access to `scanResults.location`.
- `workspace` includes:
  - `size` is size of the PersistentVolumeClaim the scan uses to download the image and vulnerability database.
  - `bindings` are additional array of secrets, ConfigMaps, or EmptyDir volumes to mount to the running scan. The `name` is used as the mount path.

```
bindings:
- name: additionalconfig
  configMap:
    name: my-configmap
- name: additionalsecret
  secret:
    secretName: my-secret
- name: scratch
  emptyDir: {}
```

For information about workspace bindings, see Using other types of volume sources. Only Secrets, ConfigMaps, and EmptyDirs are supported.

Default environment

The following describes the default environment for Tekton workspaces:

- `/home/app-scanning` is a memory-backed EmptyDir mount that contains service account credentials loaded by Tekton.
- `/cred-helper` is a memory-backed EmptyDir mount containing:
    - config.json combines static credentials with workload identity credentials when `activeKeychains` is enabled.
    - `trusted-cas.crt` when SCST - Scan 2.0 is deployed with `caCertData`
- `/workspace` is a PersistentVolumeClaim to hold scan artifacts and results.
    - The working directory for all Steps is by default located at `/workspace/scan-results`.

Environment variables

If undefined by your `step` definition the environment uses the following default variables:

- HOME=/home/app-scanning
- DOCKER_CONFIG=/cred-helper
- XDG_CACHE_HOME=/workspace/.cache
- TMPDIR=/workspace/tmp
- SSL_CERT_DIR=/etc/ssl/certs:/cred-helper

Tekton pipeline parameters:

These parameters are populated after creating the GrypeImageVulnerabilityScan. For information about parameters, see the Tekton documentation.

Parameters	Default	Type	Description
image	""	string	The scanned image
scan-results-path	/workspace/scan-results	string	Location to save scanner output
trusted-ca-certs	""	string	PEM data from the installation's `caCertData`

Note

The `publisher` service account uploads any files, such as scanner output, in the `scan-results-path` directory to the registry of your choice. For information about configuring the registry URL where the `publisher` service account uploads scan results, see Configure your custom ImageVulnerabilityScan samples.

Retrieving an image digest

SCST - Scan 2.0 custom resources require the digest form of the URL. For example, `nginx@sha256:aa0afebbb3cfa473099a62c4b32e9b3fb73ed23f2a75a65ce1d4b4f55a5c2ef2`.

Use the Docker documentation to pull and inspect an image digest:

```
docker pull nginx:latest
docker inspect --format='{{index .RepoDigests 0}}' nginx:latest
```

Alternatively, you can install krane to retrieve the digest without pulling the image:

```
krane digest nginx:latest
```

Verifying an ImageVulnerabilityScan

This topic tells you how to verify an ImageVulnerabilityScan without Supply Chain integration.

Overview

After you build an ImageVulnerabilityScan to bring your own scanner, you can validate the capabilities to verify the integration.

Ensure that the scan integration is working correctly so that downstream servers such as AMR Observer, Tanzu Developer Portal, and the insight CLI can use scan results.

To verify scanning:

1. Verify that a triggered scan is completed.
2. Retrieve the scan results from the registry.
3. Verify that the scan results are in a supported format.

Trigger and observe scanning

To verify that you can scan an image using your ImageVulnerabilityScan:

1. Deploy your ImageVulnerabilityScan to the cluster by running:

   ```
   kubectl apply -f image-vulnerability-scan.yaml -n DEV-NAMESPACE
   ```

   Where `DEV-NAMESPACE` is the name of the developer namespace you want to use.

2. Child resources are created. View the child PipelineRun, TaskRuns, and pods:

   ```
   kubectl get -l imagevulnerabilityscan pipelinerun,taskrun,pod -n DEV-NAMESPACE
   ```

3. When the scanning completes, the status is shown. Specify `-o wide` to see the digest of the image scanned and the location of the published results.

   ```
   kubectl get imagevulnerabilityscans -n DEV-NAMESPACE -o wide
   ```

   The following is an example of expected output:

   ```
   NAME                 SCANRESULT                            SCANNEDIMAGE
   SUCCEEDED   REASON
   generic-image-scan   registry/project/scan-results@digest nginx:latest@digest
   True        Succeeded
   ```

Retrieve scan results

Scan results are uploaded to the container image registry as an imgpkg bundle. To retrieve a vulnerability report:

1. Retrieve the result location from the ImageVulnerabilityScan CR Status:

```
SCAN_RESULT_URL=$(kubectl get imagevulnerabilityscan my-scan -n DEV-NAMESPACE -
o jsonpath='{.status.scanResult}')
```

2. Download the bundle to a local directory and list the content:

```
imgpkg pull -b $SCAN_RESULT_URL -o scan-results/
ls scan-results/
```

Validating scan format

After retrieving the scan results, you must verify that the scan results are in a format that downstream Tanzu Application Platform services such as AMR Observer support. AMR Observer supports the following SBOM formats and versions.

Verifying an ImageVulnerabilityScan

SBOM Formats	Versions
CycloneDX	1.2, 1.3, 1.4, 1.5
SPDX	2.2

VMware recommends validating the scan results by using this CycloneDX tool, sbom-utility. This tool validates CycloneDX and SPDX BOMs against versioned schemas.

Note The output of the scan must be valid in accordance with SPDX or CycloneDX specifications. If not, although it might be parsed correctly, VMware cannot ensure that the information is parsed correctly, and results might not be displayed accurately in Tanzu Developer Portal and Tanzu Application s CLI.

To validate a scan format with sbom:

1. Setup and install sbom. See the sbom documentation.
2. Run the `sbom-utility` CLI with the subcommand validate to validate the scan report against its declared format, such as SPDX, CycloneDX, and version.

```
./sbom-utility validate -i SCAN-REPORT-FILE-NAME
```

Where `SCAN-REPORT-FILE-NAME` is the name of the scan report.

For example:

```
sbom-utility-v0.11.0-darwin-amd64 % ./sbom-utility validate -i scan-results/sca
n.json
Welcome to the sbom-utility! Version `v0.11.0` (sbom-utility) (darwin/amd64)
================================================================================
[INFO] Loading license policy config file: `license.json`...
[WARN] Invalid flag for command: `output-file` (`o`). Ignoring...
[INFO] Attempting to load and unmarshal file `/Users/lrobin/go/src/gitlab/app-s
canning/scan-results-grype-cyclonedx-json/scan.json`...
[INFO] Successfully unmarshalled data from: `/Users/lrobin/go/src/gitlab/app-sc
anning/scan-results-grype-cyclonedx-json/scan.json`
[INFO] Determining file's SBOM format and version...
[INFO] Determined SBOM format, version (variant): `CycloneDX`, `1.4` (latest)
```

```
[INFO] Matching SBOM schema (for validation): schema/cyclonedx/1.4/bom-1.4.sche
ma.json
[INFO] Loading schema `schema/cyclonedx/1.4/bom-1.4.schema.json`...
[INFO] Schema `schema/cyclonedx/1.4/bom-1.4.schema.json` loaded.
[INFO] Validating `/Users/lrobin/go/src/gitlab/app-scanning/scan-results-grype-
cyclonedx-json/scan.json`...
[INFO] SBOM valid against JSON schema: `true`
```

Important

The `sbom-utility` only accepts JSON as input. Your scan report must be a JSON file to use this tool.

Author a ClusterImageTemplate for Supply Chain integration

This topic tells you how to create your own ClusterImageTemplate and customize the embedded ImageVulnerabilityScan to use the scanner of your choice.

Prerequisites

The following prerequisite is required to author a ClusterImageTemplate for Supply Chain integration:

- You create your own ImageVulnerabilityScan or configured one of the samples provided in Configure your custom ImageVulnerabilityScan.
- For more information, see ClusterImageTemplate in the Cartographer documentation.
- Understand `ytt` templates. See Carvel `ytt` docs for documentation and playground samples.

Create a ClusterImageTemplate

This section describes how to create a ClusterImageTemplate using an ImageVulnerabilityScan with Trivy. To use a different scanner, replace the embedded ImageVulnerabilityScan with your own.

ClusterImageTemplate parameters and values:

- The `spec.params` in this sample YAML define default values for fields within the ImageVulnerabilityScan. For more information, see `params` in the Cartographer documentation.
    - `#@ data.values.params` in the `ytt` template block and in the ImageVulnerabilityScan both reference `spec.params` fields.
- The values in the `data.values.workload` such as `metadata`, `labels`, `annotations`, and `spec` come from the supply chain workload.
- The `data.values.image` is the container image built from Buildpacks in the supply chain step that scans for vulnerabilities.
- The example YAML in this topic uses `ytt` to define a resource template written in `ytt` for the ImageVulnerabilityScan Custom Resource. For more information, see `ytt` in the Cartographer documentation. For example, inside the `ytt` template are defined functions that you can use within the ImageVulnerabilityScan.

To create a ClusterImageTemplate:

1. Create a YAML file with the following content and name it `custom-ivs-template.yaml`.

```
apiVersion: carto.run/v1alpha1
kind: ClusterImageTemplate
metadata:
  name: image-vulnerability-scan-custom # input name of your ClusterImageTempla
te
spec:
  imagePath: .status.scannedImage
  retentionPolicy:
    maxFailedRuns: 10
    maxSuccessfulRuns: 10
  lifecycle: immutable

  healthRule:
    multiMatch:
      healthy:
        matchConditions:
          - status: "True"
            type: ScanCompleted
          - status: "True"
            type: Succeeded
      unhealthy:
        matchConditions:
          - status: "False"
            type: ScanCompleted
          - status: "False"
            type: Succeeded

  params:
    - name: image_scanning_workspace_size
      default: 4Gi
    - name: image_scanning_service_account_scanner
      default: scanner
    - name: image_scanning_service_account_publisher
      default: publisher
    - name: image_scanning_active_keychains
      default: []
    - name: image_scanning_workspace_bindings
      default: []
    - name: image_scanning_steps_env_vars
      default: []
    - name: trivy_db_repository
      default: ghcr.io/aquasecurity/trivy-db
    - name: trivy_java_db_repository
      default: ghcr.io/aquasecurity/trivy-java-db
    - name: registry
      default:
        server: REGISTRY-SERVER    # input your registry server
        repository: REGISTRY-REPOSITORY    # input your registry repository

  ytt: |
    #@ load("@ytt:data", "data")
    #@ load("@ytt:template", "template")

    #@ def merge_labels(fixed_values):
    #@   labels = {}
    #@   if hasattr(data.values.workload.metadata, "labels"):
    #@     exclusions = ["kapp.k14s.io/app", "kapp.k14s.io/association"]
    #@     for k,v in dict(data.values.workload.metadata.labels).items():
    #@       if k not in exclusions:
    #@         labels[k] = v
    #@       end
    #@     end
    #@   end
    #@   labels.update(fixed_values)
    #@   return labels
    #@ end
```

```
    #@ def scanResultsLocation():
    #@   return "/".join([
    #@    data.values.params.registry.server,
    #@    data.values.params.registry.repository,
    #@    "-".join([
    #@      data.values.workload.metadata.name,
    #@      data.values.workload.metadata.namespace,
    #@      "scan-results",
    #@    ])
    #@   ]) + ":" + data.values.workload.metadata.uid
    #@ end

    #@ def param(key):
    #@   if not key in data.values.params:
    #@     return None
    #@   end
    #@   return data.values.params[key]
    #@ end

    #@ def maven_param(key):
    #@   if not key in data.values.params["maven"]:
    #@     return None
    #@   end
    #@   return data.values.params["maven"][key]
    #@ end

    #@ def maven_repository_url():
    #@   if maven_param("repository") and "url" in maven_param("repository"):
    #@     return maven_param("repository")["url"]
    #@   elif param("maven_repository_url"):
    #@     return param("maven_repository_url")
    #@   else:
    #@     return None
    #@   end
    #@ end

    #@ def correlationId():
    #@   if hasattr(data.values.workload, "annotations") and hasattr(data.value
s.workload.annotations, "apps.tanzu.vmware.com/correlationid"):
    #@     return data.values.workload.annotations["apps.tanzu.vmware.com/corre
lationid"]
    #@   end
    #@   url = ""
    #@   if hasattr(data.values.workload.spec, "source"):
    #@     if hasattr(data.values.workload.spec.source, "git"):
    #@       url = data.values.workload.spec.source.git.url
    #@     elif hasattr(data.values.workload.spec.source, "image"):
    #@       url = data.values.workload.spec.source.image.split("@")[0]
    #@     end
    #@     url = url + "?sub_path=" + getattr(data.values.workload.spec.source,
"subPath", "/")
    #@   end
    #@   if param("maven"):
    #@     url = maven_repository_url() + "/" + maven_param("groupId").replace
(".", "/") + "/" + maven_param("artifactId")
    #@   end
    #@   if hasattr(data.values.workload.spec, "image"):
    #@     url = data.values.workload.spec.image.split("@",1)[0]
    #@     url = url.split(":",1)[0]
    #@   end
    #@   return url
    #@ end

    ---
    apiVersion: app-scanning.apps.tanzu.vmware.com/v1alpha1
```

```
    kind: ImageVulnerabilityScan
    metadata:
      labels: #@ merge_labels({ "app.kubernetes.io/component": "image-scan" })
      annotations:
        apps.tanzu.vmware.com/correlationid: #@ correlationId()
        app-scanning.apps.tanzu.vmware.com/scanner-name: Trivy
      generateName: #@ data.values.workload.metadata.name + "-trivy-scan-"
    spec:
      image: #@ data.values.image
      activeKeychains: #@ data.values.params.image_scanning_active_keychains
      scanResults:
        location: #@ scanResultsLocation()
      workspace:
        size: #@ data.values.params.image_scanning_workspace_size
        #@ if/end data.values.params.image_scanning_workspace_bindings:
        bindings: #@ data.values.params.image_scanning_workspace_bindings
      serviceAccountNames:
        scanner: #@ data.values.params.image_scanning_service_account_scanner
        publisher: #@ data.values.params.image_scanning_service_account_publish
er
      steps:
      - name: trivy-generate-report
        image: TRIVY-SCANNER-IMAGE
        env:
        - name: TRIVY_DB_REPOSITORY
          value: #@ data.values.params.trivy_db_repository
        - name: TRIVY_JAVA_DB_REPOSITORY
          value: #@ data.values.params.trivy_java_db_repository
        - name: TRIVY_CACHE_DIR
          value: /workspace/trivy-cache
        - name: XDG_CACHE_HOME
          value: /workspace/.cache
        - name: TMPDIR
          value: /workspace
        - #@ template.replace(data.values.params.image_scanning_steps_env_vars)
        args:
        - image
        - $(params.image)
        - --exit-code=0
        - --no-progress
        - --scanners=vuln
        - --format=cyclonedx
        - --output=$(params.scan-results-path)/scan.cdx.json
```

Note

`apps.tanzu.vmware.com/correlationid` contains the metadata of the mapping to the source of the scanned resource.

2. Edit the following in your `custom-ivs-template.yaml` file:
    - `.metadata.name` is the name of your ClusterImageTemplate. Ensure that it does not conflict with the names of packaged templates. See Author your supply chains.
    - `REGISTRY-SERVER` is the registry server used for the scan results location.
    - `REGISTRY-REPOSITORY` is the registry repository used for the scan results location.
    - `TRIVY-SCANNER-IMAGE` is the location of your Trivy scanner CLI image
    - `.metadata.annotations.'app-scanning.apps.tanzu.vmware.com/scanner-name'` is the scanner image name reported in the Tanzu Developer Portal, formerly Tanzu Application Platform GUI.

3. Create the ClusterImageTemplate:

```
kubectl apply -f custom-ivs-template.yaml
```

4. After you create your custom ClusterImageTemplate, you can integrate it with SCST - Scan 2.0. See Add App Scanning to default Test and Scan supply chains.

Configure your custom ImageVulnerabilityScan samples for Supply Chain Security Tools - Scan

This topic gives you sample ImageVulnerabilityScans for various scanners, and any associated secrets.

ImageVulnerabilityScan samples

This section includes ImageVulnerabilityScans (IVS) for various scanners. To use them, copy the YAML content for the scanner you want to use in the following topics:

- Carbon Black
- Snyk
- Prisma
- Trivy
- Grype

Use custom ImageVulnerabilityScan samples

To use a custom ImageVulnerabilityScan sample:

1. Copy the sample YAML into a file named `custom-ivs.yaml`. Some scanners, such as Carbon Black, Snyk, and Prisma Scanner, require specific credentials that you must specify in the secret.
2. Obtain the one or more necessary images. For example, an image containing the scanner.
3. Edit these fields of your ImageVulnerabilityScan:
    - `spec.image` is the image that you are scanning. See Retrieving an image digest.
    - `scanResults.location` is the registry URL where the `publisher` service account uploads the scan results. For example, `my.registry/scan-results`.
    - `serviceAccountNames` includes:
        - `scanner` is the service account that runs the scan. It must have read access to `image`.
        - `publisher` is the service account that uploads results. It must have write access to `scanResults.location`.
4. Complete any scanner specific changes specified on the sample ImageVulnerabilityScan page.
5. You can either incorporate your custom ImageVulnerabilityScan into a ClusterImageTemplate or run a standalone scan using:

    ```
    kubectl apply -f custom-ivs.yaml -n DEV-NAMESPACE
    ```

    Where `DEV-NAMESPACE` is the name of the developer namespace where scanning occurs.

Retrieving an image digest

SCST - Scan 2.0 custom resources require the digest form of the URL. For example, `nginx@sha256:aa0afebbb3cfa473099a62c4b32e9b3fb73ed23f2a75a65ce1d4b4f55a5c2ef2`.

Use the Docker documentation to pull and inspect an image digest:

```
docker pull nginx:latest
docker inspect --format='{{index .RepoDigests 0}}' nginx:latest
```

Alternatively, you can install krane to retrieve the digest without pulling the image:

```
krane digest nginx:latest
```

Configure a ImageVulnerabilityScan for Carbon Black

This topic gives you an example of how to configure a secret and ImageVulnerabilityScan (IVS) for Carbon Black.

Example secret

This section contains a sample secret containing the Carbon Black credentials inside the `~/.cbctl/cbctl.yaml` config file. These credentials are used to authenticate your Carbon Black account. You can find these credentials in the Carbon Black console. See the Carbon Black documentation. You must apply this once to your developer namespace.

```
apiVersion: v1
kind: Secret
metadata:
  name: cbctl-creds
stringData:
  cbctl: |
    cb_api_id: CB-API-ID
    cb_api_key: CB-API-KEY
    org_key: ORG-KEY
    saas_url: SAAS-URL
```

Where:

- `CB-API-ID` is the API ID obtained from Carbon Black Cloud.
- `CB-API-KEY` is the API Key obtained from Carbon Black.
- `ORG-KEY` is the Org Key for your Carbon Black organization.
- `SAAS-URL` is the Carbon Black Backend URL.

Example ImageVulnerabilityScan

This section contains a sample IVS that uses Carbon Black to scan a targeted image and push the results to the specified registry location. For information about the IVS specification, see Configuration Options.

Set the tekton-pipelines feature-flags configmap `enable-api-fields` to `alpha`. This lets you use the `stdoutConfig` which is needed to output the scan report as a file.

```
apiVersion: app-scanning.apps.tanzu.vmware.com/v1alpha1
kind: ImageVulnerabilityScan
metadata:
  name: carbon-black-ivs
```

```
  annotations:
    app-scanning.apps.tanzu.vmware.com/scanner-name: Carbon-Black
spec:
  image: nginx@sha256:... # The image to be scanned. Digest must be specified.
  scanResults:
    location: registry/project/scan-results
  serviceAccountNames:
    publisher: publisher
    scanner: scanner
  workspace:
    bindings:
    - name: cbctl
      secret:
        secretName: cbctl-creds
        items:
          - key: cbctl
            path: .cbctl.yaml
  steps:
  - name: carbon-black
    image: CARBON-BLACK-SCANNER-IMAGE
    imagePullPolicy: IfNotPresent
    command:
    - cbctl
    - image
    - scan
    - --force=true
    - $(params.image)
    - --config
    - /cbctl/.cbctl.yaml
    - -ocyclonedx
    stdoutConfig:
      path: /workspace/scan-results/scan-results.cdx.xml
```

Where:

- `CARBON-BLACK-SCANNER-IMAGE` is the Carbon Black scanner image. For example, `cbartifactory/cbctl:latest`. For information about publicly available Carbon Black images, see DockerHub. For more information about using the Carbon Black Scanner CLI, see the Carbon Black documentation.

The Carbon Black `cbctl-creds` secret is mounted as a workspace binding and the credentials are inserted into a `cbctl.yaml` config file that the Carbon Black CLI uses.

`stdoutConfig.path` is specified to take the output stream of the step to a file where you can publish it to the registry. For more information, see the Tekton documentation.

Disclaimer

For the publicly available Carbon Black scanner CLI image, CLI commands and parameters used are accurate at the time of documentation.

Configure an ImageVulnerabilityScan for Snyk

This topic gives you an example of how to configure a secret and ImageVulnerabilityScan (IVS) for Snyk.

Example Secret

Important

> For the publicly available Snyk scanner CLI image, CLI commands and parameters used are accurate at the time of documentation.

This section contains a sample secret containing the Snyk API token, which authenticates your Snyk account. You must apply this once to your developer namespace.

```
apiVersion: v1
kind: Secret
metadata:
  name: snyk-token
stringData:
  snyk: |
    {"api": "SNYK-API-TOKEN"}
```

Where:

- `SNYK-API-TOKEN` is your Snyk API token obtained by following the instructions in the Snyk documentation. Do not base64 encode this value.

Example ImageVulnerabilityScan

This section contains a sample IVS that uses Snyk to scan a targeted image and push the results to the specified registry location. For information about the IVS specification, see Configuration Options.

```
apiVersion: app-scanning.apps.tanzu.vmware.com/v1alpha1
kind: ImageVulnerabilityScan
metadata:
  name: snyk-ivs
  annotations:
    app-scanning.apps.tanzu.vmware.com/scanner-name: Snyk
spec:
  image: TARGET-IMAGE
  scanResults:
    location: registry/project/scan-results
  serviceAccountNames:
    publisher: publisher
    scanner: scanner
  workspace:
    bindings:
    - name: snyk
      secret:
        secretName: snyk-token
        items:
          - key: snyk
            path: configstore/snyk.json
  steps:
  - name: snyk
    image: SNYK-SCANNER-IMAGE
    env:
    - name: XDG-CONFIG-HOME
      value: /snyk
    command: ["snyk","container","test",$(params.image),"--json-file-output=$(params.s
can-results-path)/scan.json"]
    onError: continue
  - name: snyk2spdx # You will need to create your own image. See explanation below.
    image: SNYK2SPDX-IMAGE
    command: ["/bin/bash"]
    args:
      - "-c"
      - |
        set -e
```

```
        cat $(params.scan-results-path)/scan.json | /app/bin/snyk2spdx --output=$(para
ms.scan-results-path)/scan.spdx.json
```

Where:

- `TARGET-IMAGE` is the image to be scanned. You must specify the digest.
- `SNYK-SCANNER-IMAGE` is the image containing the Snyk CLI. For example, `snyk/snyk:golang`. For information about publicly available Snyk images, see DockerHub. For more information about using the Snyk CLI, see the Snyk documentation.
- `XDG-CONFIG-HOME` is the directory that contains your Snyk CLI config file, `configstore/snyk.json`, which is populated using the snyk-token `Secret` you created. For more information, see the Snyk Config documentation.
- `SNYK2SPDX-IMAGE` is the image used to convert the Snyk CLI output `scan.json` in the `snyk` step to SPDX format and have its missing `DOCUMENT DESCRIBES` relation inserted. See the Snyk snyk2spdx repository in GitHub. To do this:
    1. Clone the snyk2spdx repository.
    2. Add the following Dockerfile to the root of the repository:

        ```
        FROM node AS build

        RUN npm install -g typescript
        RUN npm install -g ts-node

        WORKDIR /build-dir
        ADD . .

        RUN npm install --legacy-peer-deps
        RUN npm run build && npm prune --json --omit=dev --legacy-peer-deps
        RUN npx nexe@3.3.7 dist/index.js  -r './dist/**/*.js' -t linux-x64-12.16.
        2 -o snyk2spdx-linux

        FROM paketobuildpacks/builder-jammy-base AS run

        COPY --from=build /build-dir/dist/ /app/dist/
        COPY --from=build /build-dir/node_modules /app/node_modules
        COPY --from=build /build-dir/snyk2spdx-linux /app/bin/snyk2spdx

        ENTRYPOINT ["/app/bin/snyk2spdx"]
        CMD ["/app/bin/snyk2spdx"]
        ```

    3. Build and push the image to a registry. Replace `SNYK2SPDX-IMAGE` with the new image you built.

Note

The `snyk2spdx` output does not conform to the verification process. Although the results might be ingested to the Tanzu Application Platform metadata store, VMware does not ensure the accuracy of the results.

Note

After detecting vulnerabilities, the Snyk image exits with Exit Code 1 and causes a failed scan task. You can ignore the step error by setting `onError` and handling the error in a subsequent step. For instructions, see the Tekton documentation.

For information about setting up scanner credentials, see the Snyk CLI documentation.

Configure an ImageVulnerabilityScan for Prisma

This topic gives you an example of how to configure a secret and ImageVulnerabilityScan (IVS) for Prisma.

Example secret

This section contains a sample secret containing the Prisma Cloud account access key, secret key, and console URL which are used to authenticate your Prisma account. You must apply this once to your developer namespace.

```
apiVersion: v1
kind: Secret
metadata:
  name: prisma-auth
stringData:
  username: USERNAME
  password: PASSWORD
  address: ADDRESS
```

Where:

- `USERNAME` is the access Key from the Prisma Cloud account.
- `PASSWORD` is the secret Key from the Prisma Cloud account.
- `ADDRESS` is the URL for Console from the Prisma Cloud account.

Example ImageVulnerabilityScan

This section contains a sample IVS that uses Prisma to scan a targeted image and push the results to the specified registry location. For information about the IVS specification, see Configuration Options.

```
apiVersion: app-scanning.apps.tanzu.vmware.com/v1alpha1
kind: ImageVulnerabilityScan
metadata:
  name: prisma-ivs
  annotations:
    app-scanning.apps.tanzu.vmware.com/scanner-name: Prisma
spec:
  image: TARGET-IMAGE
  scanResults:
    location: registry/project/scan-results
  serviceAccountNames:
    publisher: publisher
    scanner: scanner
  steps:
  - name: prisma
    image: PRISMA-SCANNER-IMAGE
    imagePullPolicy: IfNotPresent
    workingDir: /workspace
    securityContext:
      privileged: true
      allowPrivilegeEscalation: true
      seccompProfile:
        type: RuntimeDefault
      capabilities:
        drop:
          - ALL
```

```
    script: |
      #!/bin/bash
      # Alternative method of making twistcli available in the container
      curl --output ./twistcli --write-out "%{http_code}" -s -L -k -u $USER_NAME:$PASS
WORD $ADDRESS/api/v1/util/twistcli
      chmod +x /workspace/twistcli

      if [[ ! -e /cred-helper/config.json ]]; then
      echo "{}" > /cred-helper/config.json
      fi

      podman pull $IMAGE
      twistcli images scan --podman-path /usr/bin/podman --address $ADDRESS --user $US
ER_NAME --password $PASSWORD $IMAGE --output-file ./twist-scan.json --containerized

      # Input commands used for your Prisma summary report conversion. See below for m
ore detail.
      twistoutput=./twist-scan.json
      twistversion=$(./twistcli -v | sed "s/twistcli version //")
      IMAGE_NAME=$(echo $IMAGE | cut -d'@' -f1)
      IMAGE_DIGEST=$(echo $IMAGE | cut -d'@' -f2)
      /prismaconverter image $twistversion $twistoutput "./" "./scan-results/twist-sca
n-cdx.json" $IMAGE_NAME $IMAGE_DIGEST
    env:
    - name: USER_NAME
      valueFrom:
        secretKeyRef:
          key: username
          name: prisma-auth
          optional: false
    - name: PASSWORD
      valueFrom:
        secretKeyRef:
          key: password
          name: prisma-auth
          optional: false
    - name: ADDRESS
      valueFrom:
        secretKeyRef:
          key: address
          name: prisma-auth
          optional: false
    - name: IMAGE
      value: $(params.image)
  workspace: {}
```

Where:

- `TARGET-IMAGE` is the image to scan. You must specify the digest.
- `PRISMA-SCANNER-IMAGE` is the image containing the Prisma Cloud twistcli, podman, and a utility to convert the Prisma summary report in JSON format to a CycloneDX SBOM in XML format. The Prisma scanner produces a proprietary output instead of community standard CycloneDX or SPDX. Because of this, you cannot use the scan report summary produced by Prisma as is and you must convert it. Contact your account team if you want to use Prisma and need this utility.
- The `securityContext` grants root access as Prisma requires root access to run. If permission is not given, you might encounter a `cannot clone: Operation not permitted` error message. For information about the `securityContext`, see Customize an ImageVulnerabilityScan. Due to needing root access, you cannot run Prisma scans in clusters with restricted pod Security Standards.

For information about using the CLI, see the Prisma twistcli docs.

Configure an ImageVulnerabilityScan for Trivy

This topic gives you an example of how to configure an ImageVulnerabilityScan (IVS) for Trivy.

Example ImageVulnerabilityScan

This section gives you an example IVS that uses Trivy to scan a targeted image and push the results to the specified registry location. For information about the IVS specification, see Configuration Options.

```
apiVersion: app-scanning.apps.tanzu.vmware.com/v1alpha1
kind: ImageVulnerabilityScan
metadata:
  name: trivy-ivs
  annotations:
    app-scanning.apps.tanzu.vmware.com/scanner-name: Trivy
spec:
  image: TARGET-IMAGE
  scanResults:
    location: registry/project/scan-results
  serviceAccountNames:
    publisher: publisher
    scanner: scanner
  steps:
  - name: trivy
    image: TRIVY-SCANNER-IMAGE
    command: ["trivy"]
    args:
    - image
    - $(params.image)
    - --exit-code=0
    - --no-progress
    - --scanners=vuln
    - --format=cyclonedx
    - --output=$(params.scan-results-path)/scan.cdx.json
```

Where:

- `TARGET-IMAGE` is the image to be scanned. Digest must be specified.
- `TRIVY-SCANNER-IMAGE` is the image containing the Trivy CLI. For example, `aquasec/trivy:0.42.1`. For information about publicly available Trivy images, see DockerHub. For more information about using the Trivy CLI, see the Trivy documentation.

Note

Trivy versions greater than 0.42.1 are not supported as they output CycloneDX 1.5 which is not supported for ingestion.

Trivy database size requirement

The recommended `storageSize` for Trivy scans is 4Gi due to the size of the Trivy database. If the `storageSize` is not sufficient, you might encounter a `no space left on device` error when initializing the database in the scan pod.

Update the `app-scanning-values-file.yaml` for the `app-scanning.apps.tanzu.vmware.com` package to change the default `storageSize`. See installation documentation.

```
scans:
  workspace:
    storageSize: 4Gi
```

If you do not want to set a default `storageSize` by updating the `app-scanning-values-file.yaml`, you must specify the `spec.workspace.size` when creating each standalone ImageVulnerabilityScan or embedded ImageVulnerabilityScan in a ClusterImageTemplate.

Warning As a publicly maintained image that is built outside of VMware build systems, the image might not meet the security standards VMware has established. Ensure that you review the image before use to ensure that it meets your organizations security and compliance policies. For the publicly available Trivy scanner CLI image, CLI commands and parameters used are accurate at the time of documentation.

Configure a ImageVulnerabilityScan for Grype

This topic gives you an example of how to configure an ImageVulnerabilityScan (IVS) for Grype.

Example ImageVulnerabilityScan

This section contains a sample IVS that uses Grype to scan a targeted image and push the results to the specified registry location. For information about the IVS specification, see Configuration Options.

```
apiVersion: app-scanning.apps.tanzu.vmware.com/v1alpha1
kind: ImageVulnerabilityScan
metadata:
  name: grype-ivs
  annotations:
    app-scanning.apps.tanzu.vmware.com/scanner-name: Grype
spec:
  image: TARGET-IMAGE
  scanResults:
    location: registry/project/scan-results
  serviceAccountNames:
    publisher: publisher
    scanner: scanner
  steps:
  - name: grype
    image: GRYPE-SCANNER-IMAGE
    args:
    - -o
    - cyclonedx-json
    - registry:$(params.image)
    - --file
    - /workspace/scan-results/scan.cdx.json
    env:
    - name: GRYPE_ADD_CPES_IF_NONE
      value: "false"
    - name: GRYPE_EXCLUDE
    - name: GRYPE_SCOPE
```

Where:

- `TARGET-IMAGE` is the image to scan. You must specify digest.
- `GRYPE-SCANNER-IMAGE` is the image containing the Grype CLI. For example, `anchore/grype:latest`. For information about publicly available Grype images, see DockerHub. For more information about using the Grype CLI, see the Grype documentation.

Grype database size requirement

The recommended `storageSize` for Grype scans is 4Gi due to the size of the Grype database. If the `storageSize` is not sufficient, you might encounter an error indicating insufficient space when initializing the database in the scan pod.

Update the `app-scanning-values-file.yaml` for the `app-scanning.apps.tanzu.vmware.com` package to change the default `storageSize`. See installation documentation.

```
scans:
  workspace:
    storageSize: 4Gi
```

If you do not want to set a default `storageSize` by updating the `app-scanning-values-file.yaml`, you must specify the `spec.workspace.size` when creating each standalone ImageVulnerabilityScan or embedded the ImageVulnerabilityScan in a ClusterImageTemplate.

Warning As a publicly maintained image that is built outside of VMware build systems, the image might not meet the security standards VMware established. Ensure that you review the image before use to ensure that it meets your organizations security and compliance policies. For the publicly available Grype scanner CLI image, CLI commands and parameters used are accurate at the time of documentation.

Verify scanning with Supply Chain integration

This topic tells you how to verify scanning with Supply Chains.

Create a workload

Create a sample workload with a pre-built image by using the `tanzu apps workload create` command:

```
tanzu apps workload create WORKLOAD-NAME \
  --app APP-NAME \
  --git-repo GIT-REPO \
  --git-branch GIT-BRANCH \
  --type TYPE \
  --namespace DEV-NAMESPACE
```

Where:

- `WORKLOAD-NAME` is the name you choose for your workload.
- `APP-NAME` is the name of your app.
- `GIT-REPO` is the Git repository from which the workload is created.
- `GIT-BRANCH` is the branch in a Git repository from where the workload is created.
- `TYPE` is the type of your app.
- `DEV-NAMESPACE` is the name of the developer namespace where scanning occurs.

Note For information about how to use the Tanzu CLI workload creation, see Create a Workload.

Retrieve scan results

Scan results are uploaded to the container image registry as an imgpkg bundle. To retrieve a vulnerability report:

1. Retrieve the result location from the ImageVulnerabilityScan CR Status:

```
SCAN_RESULT_URL=$(kubectl get imagevulnerabilityscan my-scan -n DEV-NAMESPACE -
o jsonpath='{.status.scanResult}')
```

2. Download the bundle to a local directory and list the content:

```
imgpkg pull -b $SCAN_RESULT_URL -o scan-results/
ls scan-results/
```

Set up recurring scanning

This topic describes how to set up recurring scans using Supply Chain Security Tools (SCST) - Scan 2.0.

Overview

Scan workloads with updated vulnerability databases frequently. Use the scanner of your choice to scan images produced by your software supply chain, and any container images running in your Tanzu Application Platform clusters.

Use SCST - Scan 2.0 to schedule container image scans with the following capabilities:

- Detect vulnerabilities without a supply chain run for the following container image sources:
    - Images produced by the supply chain using Tanzu Build Service or kaniko.
    - Images defined in a workload when using a prebuilt container image in your supply chain.
    - Images running in a Tanzu Application Platform cluster that were not produced by a software supply chain.
- Customize how far back you want to scan container images based on:
    - When the container image was created in the supply chain.
    - When the container image entered a running state on your Tanzu Application Platform cluster.
- Use one of the `ImageVulnerabilityScan` samples or create your own.
    - For more information, see ImageVulnerabilityScan samples or Bring your own scanner using an ImageVulnerabilityScan.
- View discovered vulnerabilities from your most recent image built for a workload in the Tanzu Developer Portal Supply Chain Choreographer plug-in.
- Use when either SCST - Scan 1.0 or SCST - Scan 2.0 is used to scan a recently built container in your supply chain.

Set up recurring scanning

Unlike scans that occur as part of the software supply chain, scans invoked by recurring scanning run as a centralized scan job in a single namespace. The scan job gathers images from all namespaces. To set up recurring scanning, you must create a `RecurringImageVulnerabilityScan`. A `RecurringImageVulnerabilityScan` defines:

- The interval in which to scan container images in crontab format.
- How many images, created using the supply chain, to scan as determined by the selected number of days in the past.

- How many images, started in your Tanzu Application Platform clusters, to scan as determined by the selected number of days in the past.
- The steps from the IVS template to use to scan your images, which define which scanner to use.
- The OCI-compliant registry to push the recurring scan results to.

Prerequisites

Before you define your `RecurringImageVulnerabilityScan` template, you must have:

- A repository created on an OCI compliant registry that scan results are pushed to.
- A namespace in which to create the recurring scan template. This is where all the recurring scan jobs will run.
- A service account within the namespace that can push an OCI artifact to the results repository.
- Credentials for any registry the scanner must pull images from to scan.

Recurring scanning uses the SCST - Scan 2.0 component, which is in the `Full` and `Build Profiles`.

Note

Pay special attention to the service accounts and credentials that are needed. For a simplified setup of the namespace, VMware recommends that you use Namespace Provisioner to create a namespace. Namespace Provisioner automatically creates the service accounts and secrets needed for images created by supply chains. The examples in this topic use a namespace created by Namespace Provisioner. For more information, see Namespace Provisioner.

Example RecurringImageVulnerabilityScan template

The following sample template provides an explanation of the input variables for the `RecurringImageVulnerabilityScan` CR. Use the Grype and Trivy samples in a namespace created by Namespace Provisioner. The Grype and Trivy examples are a subset of this template. Add additional configurations from this template to the Grype and Trivy samples for more advanced configurations.

```
apiVersion: app-scanning.apps.tanzu.vmware.com/v1alpha1
kind: RecurringImageVulnerabilityScan
metadata:
  name: recurring-scan-template
spec:
  images:
    ranWithinDays: RAN-INTERVAL
    createdWithinDays: CREATED-INTERVAL
  retentionPolicy:
    maxFailedScans: FAILED-RETENTION
    maxSuccessfulScans: SUCESSFUL-RETENTION
  schedule:
    cron: CRON-SCHEDULE
    startingDeadline: START-DEADLINE
  scan:
    activeKeychains:
    # Only enable keychains for registries you are using
    - name: acr  # Azure Container Registry
    - name: ecr  # Elastic Container Registry
    - name: gcr  # Google Container Registry
```

```
    - name: ghcr # Github Container Registry
    workspace:
      size: WORKSPACE-SIZE
    scanResults:
      location: RESULTS-REPOSITORY
    steps:
      STEPS-FROM-IVS-TEMPLATE
```

Where:

- `RAN-INTERVAL`: How many prior days of images from pods that have started on the Tanzu Application Platform clusters to scan.
- `CREATED-INTERVAL` How many prior days of images from supply chains to scan.
- `FAILED-RETENTION`: The number of failed recurring scan executions to keep in Kubernetes.
- `CRON-SCHEDULE`: The schedule in which to invoke recurring scans in crontab format. For example, to execute a scan daily at 3:00 AM, the value is `0 3 * * *`
- `START-DEADLINE`: The period of time beyond the scheduled start time when scans can be started if they did not start on time. If this period elapses, the scheduled scan is skipped.
- `SUCESSFUL-RETENTION`: The number of successful recurring scan executions to keep in Kubernetes.
- `WORKSPACE-SIZE`: The size of the workspace used when scanning images. This is created as a Kubernetes PVC. This depends mostly on the size of the vulnerability database, the number of images to be scanned, and the output of the vulnerability scanner. `10Gi` is the recommended starting point.
- `RESULTS-REPOSITORY`: The registry URL where results are uploaded. For example, `my.registry/scan-results`.
- `STEPS-FROM-IVS-TEMPLATE`: The steps to execute to scan the list of the container images. See ImageVulnerabilityScan samples for commonly used samples. To pass `spec.image` and `scanResults.location` to `args`, you can use `{image}` and `{output}` respectively. These interpolation variables differ from the ones currently used in the `ImageVulnerabilityScan` (`$(params.image)` and `$(params.scan-results-path)`).

Note

The scan step should be the last step in the `steps` property.

Pay special attention to the scanning runtime, the scan frequency, and the configured deadline. Setting `RAN-INTERVAL` and `CREATED-INTERVAL` to a large number of days can increase the number of images scanned, and in turn, the scanning runtime. If the scan runtime is greater than the `CRON-SCHEDULE` frequency, the scans start to queue up. VMware recommends running the scan once every 24 hours at a time when your cluster load is low.

Grype RecurringImageVulnerabilityScan template

The SCST - Scan 1.0 default scanner is Grype, and this template works with the SCST - Scan 1.0 default configuration.

Note

You must match the scanner and version to the scanner and version used in the software supply chain. Using different types of scanners between build time and

recurring scans is not supported and results in vulnerabilities being double counted in the Security Analysis plug-in in Tanzu Developer Portal.

To apply this configuration, save it to a file and apply it to the namespace created for recurring scans:

```
kubectl apply -f grype-recurring-scan.yaml  --namespace NAMESPACE
```

```
apiVersion: app-scanning.apps.tanzu.vmware.com/v1alpha1
kind: RecurringImageVulnerabilityScan
metadata:
  name: grype-recurring-scan
spec:
  images:
    createdWithinDays: 180
    ranWithinDays: 0
  retentionPolicy:
    maxFailedScans: 1
    maxSuccessfulScans: 1
  schedule:
    cron: 0 3 * * *
    startingDeadline: 60m
  scan:
    workspace:
      size: 10Gi
    scanResults:
      location: RESULTS-REPOSITORY
    steps:
    - name: update-db
      image: anchore/grype:latest
      command: [/grype]
      args:
      - db
      - update
      env:
      - name: GRYPE_DB_CACHE_DIR
        value: /workspace/grype-cache
    - name: grype-scan
      image: anchore/grype:latest
      command: [/grype]
      args:
      - "{image}"
      - "-o"
      - "cyclonedx-json"
      - "--file"
      - "{output}"
      env:
      - name: GRYPE_DB_AUTO_UPDATE
        value: "false"
      - name: GRYPE_CHECK_FOR_APP_UPDATE
        value: "false"
      - name: GRYPE_DB_CACHE_DIR
        value: /workspace/grype-cache
```

This sample configuration, downloads the latest Grype vulnerability database and then scans with the stored database. This prevents multiple database updates while running concurrent scans.

The placeholder `{image}` is replaced by each one of the container images that are being scanned. This is equivalent to the `$(params.image)` placeholder used in the `ImageVulnerabilityScan`.

The placeholder `{output}` is replaced by an auto-generated path to store the results for the scanned image. This is equivalent to the `$(params.scan-results-path)` placeholder used in the

`ImageVulnerabilityScan`.

Trivy RecurringImageVulnerabilityScan template

The SCST - Scan 2.0 default scanner is Trivy, and this template works with the Scan 2.0 default configuration.

Note

You must match the scanner and version to the scanner and version used in the software supply chain. Using different types of scanners between build time and recurring scans is not supported and results in vulnerabilities being double counted in the Security Analysis plug-in in Tanzu Developer Portal.

To apply this configuration, save it to a file and apply it to the namespace created for recurring scans:

```
kubectl apply -f trivy-recurring-scan.yaml --namespace NAMESPACE
```

```
apiVersion: app-scanning.apps.tanzu.vmware.com/v1alpha1
kind: RecurringImageVulnerabilityScan
metadata:
  name: trivy-recurring-scan
spec:
  images:
    createdWithinDays: 180
    ranWithinDays: 0
  retentionPolicy:
    maxFailedScans: 1
    maxSuccessfulScans: 1
  schedule:
    cron: 0 3 * * *
    startingDeadline: 60m
  scan:
    workspace:
      size: 10Gi
    scanResults:
      location: RESULTS-REPOSITORY
    steps:
    - name: update-db
      image: aquasec/trivy:0.48.3
      command: [/usr/local/bin/trivy]
      args:
      - image
      - --download-db-only
      - --no-progress
      - --cache-dir=/workspace/trivy-cache
    - name: update-java-db
      image: aquasec/trivy:0.48.3
      command: [/usr/local/bin/trivy]
      args:
      - image
      - --download-java-db-only
      - --no-progress
      - --cache-dir=/workspace/trivy-cache
    - name: trivy-generate-report
      image: aquasec/trivy:0.48.3
      command: [/usr/local/bin/trivy]
      args:
      - image
      - '{image}'
```

```
    - --exit-code=0
    - --no-progress
    - --scanners=vuln
    - --format=cyclonedx
    - --output={output}
    - --cache-dir=/workspace/trivy-cache
    - --skip-db-update
    - --skip-java-db-update
```

This sample configuration downloads the latest Trivy vulnerability and Java databases and then scans with the stored databases. This prevents multiple database updates while running concurrent scans.

The placeholder `{image}` is replaced by each one of the container images that are being scanned. This is equivalent to the `$(params.image)` placeholder used in the `ImageVulnerabilityScan`.

The placeholder `{output}` is replaced by an auto-generated path to store the results for the scanned image. This is equivalent to the `$(params.scan-results-path)` placeholder used in the `ImageVulnerabilityScan`.

Note

Do not enclose the `{output}` interpolation value in quotes for a Trivy scan.

Enable policy enforcement with Scan 2.0 in a supply chain

This topic tells you how to enable policy enforcement in a supply chain with Supply Chain Security Tools (SCST) - Scan 2.0.

Policy enforcement is not built into SCST - Scan 2.0 and is not in any of the Out of the Box supply chains. You must author a custom supply chain to enable policy enforcement.

To enforce a policy in a supply chain you need a `ClusterImageTemplate` that uses a Tekton `TaskRun` to evaluate the vulnerabilities and enforces the set policy. This topic provides a sample `TaskRun` and a sample `ClusterImageTemplate` for you to edit.

Prepare for the `TaskRun`

The `TaskRun` queries the Metadata Store to get the list of vulnerabilities for the image. Authenticate with the Metadata Store API by obtaining an access token and a certificate:

1. Build an image that contains `curl` and `jq` for the `TaskRun` to use.
2. Get the access token from the Metadata Store by running:

   ```
   ACCESS-TOKEN=$(kubectl get secrets -n metadata-store  metadata-store-read-write
   -client -o yaml \
   | yq .data.token | base64 -d)
   ```

3. Create the secret `metadata-store-access-token-secret.yaml` and include the access token:

   ```
   kind: Secret
   apiVersion: v1
   metadata:
     name: metadata-store-access-token
   stringData:
     accessToken: ACCESS-TOKEN
   ```

   Where `ACCESS-TOKEN` is the access token

4. Get the Certificate Authority (CA) certificate from the Metadata Store by running:

```
MDS_CA_CERT=$(kubectl get secret -n metadata-store ingress-cert -o json | jq -r
".data.\"ca.crt\"" \
| base64 -d)
```

5. Create the secret `metadata-store-cert-secret.yaml` and include the certificate:

```
kind: Secret
apiVersion: v1
metadata:
  name: metadata-store-cert
stringData:
  caCrt: |
    METADATA-STORE-CA-CERT
```

   Where `METADATA-STORE-CA-CERT` is the Metadata Store CA certificate

6. Apply the created secrets in the developer namespace:

```
kubectl apply -f metadata-store-access-token-secret.yaml -n DEVELOPER-NAMESPACE
kubectl apply -f metadata-store-cert-secret.yaml -n DEVELOPER-NAMESPACE
```

Edit the `TaskRun` sample to enforce the policy

The following sample `TaskRun` waits until the vulnerability data is available for the image in the Metadata Store. When the data is available, the vulnerabilities are aggregated by severity. The policy that `GATE` sets determines whether `TaskRun` succeeds or fails.

Edit the sample for your needs:

```
apiVersion: tekton.dev/v1
kind: TaskRun
metadata:
  generateName: enforce-policy-task-run-
spec:
  serviceAccountName: SERVICE-ACCOUNT-THAT-CAN-READ-TAP-IMAGES
  params:
    - name: image
      value: IMAGE-WITH-DIGEST-THAT-WAS-SCANNED
  taskSpec:
    steps:
    - name: enforce-policy
      image: TASK-RUN-IMAGE-WITH-CURL-AND-JQ
      script: |
        if [ ${GATE} -eq "none" ]; then
            exit 0
        fi

        IMAGE_DIGEST=$(echo $(params.image) | cut -d "@" -f 2)
        while true; do
          response_code=$(curl https://$METADATA_STORE_URL/api/v1/images/${IMAGE_DIGES
T} -H "Authorization: Bearer ${ACCESS_TOKEN}" -H 'accept: application/json' --cacert /
var/cert/caCrt -o vulnerabilities.json -w "%{http_code}")
          if [ $response_code -eq 200 ]; then
            echo "Vulnerabilities data is available in AMR"
            break
          else
            echo "Vulnerabilities data is not available in AMR, waiting..."
            sleep 10
          fi
        done
```

```
        critical=$(cat vulnerabilities.json | jq '[.Packages[].Vulnerabilities[] | sel
ect(.Ratings[].Severity | contains("Critical"))] | unique | length')
        high=$(cat vulnerabilities.json | jq '[.Packages[].Vulnerabilities[] | select
(.Ratings[].Severity | contains("High"))] | unique | length')
        medium=$(cat vulnerabilities.json | jq '[.Packages[].Vulnerabilities[] | selec
t(.Ratings[].Severity | contains("Medium"))] | unique | length')
        low=$(cat vulnerabilities.json | jq '[.Packages[].Vulnerabilities[] | select(.
Ratings[].Severity | contains("Low"))] | unique | length')
        vulnerabilitiesCount=0
        case $GATE in
          low)
            vulnerabilitiesCount=$(($low+$medium+$high+$critical))
            ;;
          medium)
            vulnerabilitiesCount=$(($medium+$high+$critical))
            ;;

          high)
            vulnerabilitiesCount=$(($high+$critical))
            ;;

          critical)
            vulnerabilitiesCount=$(($critical))
            ;;
        esac
        echo "Vulnerabilities: ${vulnerabilitiesCount}"
        if [ ${vulnerabilitiesCount} -gt 0 ];then
          exit 1
        fi
      env:
      - name: GATE
        value: "critical"
      - name: METADATA_STORE_URL
        value: METADATA-STORE-URL-VALUE
      - name: ACCESS_TOKEN
        valueFrom:
          secretKeyRef:
            name: METADATA-STORE-ACCESS-TOKEN
            key: accessToken
      volumeMounts:
      - name: cert
        mountPath: /var/cert
    volumes:
    - name: cert
      secret:
        secretName: SECRET-CONTAINING-METADATA-STORE-CERT
```

Where:

- `GATE` is an environment variable that sets the threshold for severity in the policy. The accepted values are `low`, `medium`, `high`, and `critical`. For example, if the `GATE` is set to `high` then the `TaskRun` fails if it finds `high` or `critical` vulnerabilities for the image. If the `GATE` is set to `none`, no policy is enforced and `TaskRun` succeeds.
- `METADATA-STORE-URL-VALUE` is the URL for reaching the Metadata Store. In a single-cluster deployment, the value is `metadata-store-app.metadata-store.svc.cluster.local:8443`. In a multicluster deployment, the value is `metadata-store.VIEW-CLUSTER-INGRESS-DOMAIN`.
- `IMAGE-WITH-DIGEST-THAT-WAS-SCANNED` is the image that was built and scanned in the previous steps of the supply chain. The environment variable value is set to `#@ data.values.image` while embedding the `TaskRun` in the `ClusterImageTemplate`. Template this value to enable it be passed through the supply chain.
- `SERVICE-ACCOUNT-THAT-CAN-READ-TAP-IMAGES` is a service account in the developer namespace that has access to read Tanzu Application Platform images. `TaskRun` uses this

service account to pull images for its execution steps.

- `TASK-RUN-IMAGE-WITH-CURL-AND-JQ` is any image that contains `curl` and `jq` commands.
- `METADATA-STORE-ACCESS-TOKEN` is the secret that contains the Metadata Store read access token.
- `SECRET-CONTAINING-METADATA-STORE-CERT` is the name of the secret that contains the Metadata Store certificate.

Include the policy `ClusterImageTemplate` in the newly authored supply chain

To include the policy `ClusterImageTemplate` in the newly authored supply chain:

1. Create `policy-cluster-image-template.yaml` by embedding the updated `TaskRun` in a `ClusterImageTemplate`. Set the values for environment variables and properties you used when editing the TaskRun sample to enforce the policy:

```
#@ load("@ytt:data", "data")
---
apiVersion: carto.run/v1alpha1
kind: ClusterImageTemplate
metadata:
  name: scan-policy-template
spec:
  imagePath: spec.params[?(@.name=="image")].value
  healthRule:
    multiMatch:
      healthy:
        matchConditions:
          - type: Succeeded
            status: "True"
      unhealthy:
        matchConditions:
          - type: Succeeded
            status: "False"
  ytt: |
    #@ load("@ytt:data", "data")
    #@ load("@ytt:template", "template")
    ---
    apiVersion: tekton.dev/v1
    kind: TaskRun
    metadata:
      name: enforce-policy
    spec:
      serviceAccountName: scanner
      taskSpec:
        steps:
        - name: enforce-policy
          image: dev.local:5000/taskrun-image:latest
          script: |
          ....
            IMAGE_DIGEST=$(echo ${IMAGE} | cut -d "@" -f 2)
          ....
          env:
          ...
          - name: IMAGE
            value: #@ data.values.image # <------ template the image so that it
can flow through the supply chain
          ....
```

2. Create a new supply chain named `custom-supply-chain.yaml` by following the steps in Author your supply chains.

   The scan step generates the vulnerability data that the policy step queries the Metadata Store for. Therefore the policy step must be after the scan step and must take the image produced by the scan step as an input image. You can place any step that requires an image as an input after the policy step.

```
---
apiVersion: carto.run/v1alpha1
kind: ClusterSupplyChain
metadata:
  name: CUSTOM-SUPPLY-CHAIN-NAME
spec:
  selectorMatchExpressions:
    - key: 'apps.tanzu.vmware.com/workload-type'
      operator: In
      values:
      - web
      - server
      - worker
  selectorMatchFields:
    - key: spec.image
      operator: Exists

  params:
    - name: image_scanning_service_account_publisher
      value: scanner
    - name: image_scanning_service_account_scanner
      default: report-publisher
    - name: image_scanning_workspace_size
      default: 4Gi

  resources:
  - name: image-provider
    templateRef:
      kind: ClusterImageTemplate
      name: image-provider-template
    params:
      - name: serviceAccount
        default: scanner

  - name: image-scanner
    templateRef:
      kind: ClusterImageTemplate
      name: image-vulnerability-scan-trivy
    params:
      - name: registry
        default:
          server: dev.registry.tanzu.vmware.com
          repository: tanzu-image-signing/test-policy
    images:
      - resource: image-provider
        name: image

  - name: policy # policy cluster image template
    templateRef:
      kind: ClusterImageTemplate
      name: scan-policy-template
    params:
      - name: serviceAccount
        default: scanner
    images:
    - resource: image-scanner
```

```
      name: image
    ... # supply chain continues
```

Apply the template, supply chain, and workload

To apply the template, supply chain, and workload:

1. Apply the generated `ClusterImageTemplate` and `ClusterSupplyChain` by running:

   ```
   kubectl apply -f policy-cluster-image-template.yaml
   kubectl apply -f custom-supply-chain.yaml
   ```

2. Create a workload in the developer namespace that can trigger the supply chain. For more information about selectors and how to use them with a custom supply chain, see Providing your own supply chain.

Troubleshoot the policy

If the policy is still waiting to find the vulnerabilities data for the image in the Metadata Store:

1. Verify that the observer is healthy and running.
2. Verify that the observer registered the controller to monitor the `ImageVulnerabilityScan` kind. For more information about troubleshooting the observer, see Troubleshooting Artifact Metadata Repository.

Supply Chain Security Tools - Scan 2.0 Observability

This topic guides you through observing Supply Chain Security Tools (SCST) - Scan 2.0. This helps you understand each step of scanning.

Scanning Steps

This section describes each of the scanning steps and corresponding observability methods.

- To watch the status of the scanning custom resources and child resources:

  ```
  kubectl get -l imagevulnerabilityscan pipelinerun,taskrun,pod
  kubectl get imagevulnerabilityscan
  ```

- View the status, reason, and urls:

  ```
  kubectl get imagevulnerabilityscan -o wide
  ```

- View the complete status and events of scanning custom resources:

  ```
  kubectl describe imagevulnerabilityscan IMAGE-VULNERABILITY-SCAN-NAME
  ```

  Where `IMAGE-VULNERABILITY-SCAN-NAME` is the name of an `ImageVulnerabilityScan` resource you want to inspect.

- List the child resources of a scan:

  ```
  kubectl get -l imagevulnerabilityscan=$NAME pipelinerun,taskrun,pod
  ```

- Get the logs of the controller:

```
kubectl logs -f deployment/app-scanning-controller-manager -n app-scanning-syst
em -c manager
```

Troubleshooting Supply Chain Security Tools - Scan 2.0

This topic helps you troubleshoot Supply Chain Security Tools (SCST) - Scan 2.0.

Overview

When Scan 2.0 creates an ImageVulnerabilityScan, the following resources are also created:

- Tekton `PipelineRun` with the following `Tasks`:
  - workspace-setup-task
  - scan-task
  - publish-task
- Tekton `TaskRun` corresponding to each `Task`
- `Pod` corresponding to each `TaskRun`

Viewing resources

- To view all resources:

  ```
  kubectl get imagevulnerabilityscans,pipelineruns,taskruns,pods -n DEV-NAMESPACE
  ```

  Where `DEV-NAMESPACE` is the name of your developer namespace.

- To verify which resources are failing, proceed to the following debugging sections:

  ```
  NAME                                                                SUCCEEDED
  REASON
  imagevulnerabilityscan.app-scanning.apps.tanzu.vmware.com/my-scan   False
  Failed

  NAME                                  SUCCEEDED   REASON      STARTTIME   COMP
  LETIONTIME
  pipelinerun.tekton.dev/my-scan-5kllf   False       Failed      2m10s       85s

  NAME                                                    SUCCEEDED   REASON
  STARTTIME   COMPLETIONTIME
  taskrun.tekton.dev/my-scan-5kllf-publish-task           False       Failed
  94s         85s
  taskrun.tekton.dev/my-scan-5kllf-scan-task              True        Succeeded
  2m1s        94s
  taskrun.tekton.dev/my-scan-5kllf-workspace-setup-task   True        Succeeded
  2m9s        2m1s

  NAME                                        READY   STATUS      RESTARTS   AGE
  pod/my-scan-5kllf-publish-task-pod          0/4     Completed   1          94s
  pod/my-scan-5kllf-scan-task-pod             0/4     Completed   1          2m
  pod/my-scan-5kllf-workspace-setup-task-pod  0/2     Completed   1          2m1
  0s
  ```

Debugging commands

The following sections describe commands you run to get logs and details about scanning errors.

Debugging resources

If a resource fails or has errors, inspect the resource. If multiple resources are involved, inspecting them all can provide a broader understanding. For example, inspecting the corresponding `TaskRun` to a failed `Pod`.

To get status conditions on a resource:

```
kubectl describe RESOURCE RESOURCE-NAME -n DEV-NAMESPACE
```

Where:

- `RESOURCE` is one of the following: `ImageVulnerabilityScan`, `PipelineRun`, `TaskRun`, or `Pod`.
- `RESOURCE-NAME` is the name of the `RESOURCE`.
- `DEV-NAMESPACE` is the name of your developer namespace.

Debugging scan pods

You can use the following methods to debug scan pods:

- To get error logs from a pod when scan pods fail:

  ```
  kubectl logs SCAN-POD-NAME -n DEV-NAMESPACE
  ```

  Where `SCAN-POD-NAME` is the name of the scan pod.

  For information about debugging Kubernetes pods, see the Kubernetes documentation.

  A scan run that has an error can indicate that one of the following step containers has a failure:

  - `step-workspace-setup`
  - `step-write-certs`
  - `step-cred-helper`
  - `step-SCANNER`
  - `step-publisher`
  - `sidecar-sleep`

  Where `step-SCANNER` is your scanner step.

- To verify which step container had a failed exit code:

  ```
  kubectl get taskrun TASKRUN-NAME -o json | jq .status
  ```

  Where `TASKRUN-NAME` is the name of the TaskRun.

- To inspect a specific step container in a pod:

  ```
  kubectl logs scan-pod-name -n DEV-NAMESPACE -c step-container-name
  ```

  Where `DEV-NAMESPACE` is your developer namespace.

  For information about debugging a TaskRun, see the Tekton documentation.

- To debug inside of the scan-task pod: Add an additional step with a `sleep` command after your scanner step in the ImageVulnerabilityScan. For example:

  ```
  ...
  spec:
  ```

```
  ...
  steps:
  - name: SCANNER-STEP
    ...
  - name: view
    image: busybox:latest
    args:
    - -c
    - sleep 6000
```

This keeps the pod in a running state so that you can exec into it. Re-run the scan and then exec into the pod:

```
    kubectl exec SCAN-TASK-POD-NAME -n DEV-NAMESPACE -c step-view --stdin --tty
-- sh
```

Where `SCAN-TASK-POD-NAME` is the name of your scan-task pod.

Viewing the Scan-Controller manager logs

You can run these commands to view the Scan-Controller manager logs:

- Retrieve scan-controller manager logs:

  ```
  kubectl logs deployment/app-scanning-controller-manager -n app-scanning-system
  ```

- Tail scan-controller manager logs:

  ```
  kubectl logs -f deployment/app-scanning-controller-manager -n app-scanning-syst
  em
  ```

Troubleshooting issues

Volume permission error

If you encounter a permission error for accessing, opening, and writing to the files inside cluster volume, such as:

```
unsuccessful cred copy: ".git-credentials" from "/tekton/creds" to "/home/app-scannin
g": unable to open destination: open /home/app-scanning/.git-credentials: permission d
enied
```

Ensure that the problematic step runs with the proper user and group ids.

Incompatible Tekton version

Tanzu Application Platform `v1.7.0` or later includes `app-scanning.apps.tanzu.vmware.com` version `0.2.0` and Tekton Pipelines version `0.50.1`. The `app-scanning.apps.tanzu.vmware.com` package is incompatible with previous versions of Tekton Pipelines as v1 CRDs were not enabled. You must upgrade Tanzu Application Platform to `v1.7.0` or later before upgrading `app-scanning.apps.tanzu.vmware.com`.

If you did not upgrade Tanzu Application Platform before upgrading `app-scanning.apps.tanzu.vmware.com`, you can encounter ImageVulnerabilityScans not progressing:

```
NAME      SUCCEEDED   REASON
my-scan
```

To resolve this issue:

1. Confirm that the issue is due to installing an incompatible Tekton version by viewing the controller manager logs by running:

   ```
   kubectl -n app-scanning-system logs -f deployment/app-scanning-controller-manag
   er -c manager
   ```

   If you encounter the following error, proceed to the next step:

   ```
   ERROR  controller-runtime.source.EventHandler  failed to get informer from cach
   e  {"error": "failed to get API group resources: unable to retrieve the complet
   e list of server APIs: tekton.dev/v1: the server could not find the requested r
   esource"}
   ```

2. Upgrade Tanzu Application Platform to `v1.7.0` or later. See Upgrade your Tanzu Application Platform.

Scan results empty

The publish-task task fails if the `scan-results-path` (default value of `/workspace/scan-result`) is empty. To confirm, view the logs of the publish-task pod:

```
kubectl logs PUBLISH-TASK-POD-NAME -c step-publisher -n DEV-NAMESPACE
```

Where `PUBLISH-TASK-POD-NAME` is the name of your publish-task pod.

```
2023/08/22 17:09:49 results folder /workspace/scan-results is empty
```

To resolve this issue, you can debug within the scan-task pod by following the instructions under Debugging scan pods. You must use an image with both a shell and your scanner CLI image to run the `sleep` command and troubleshoot your scanner commands from within the container.

Scanning in a cluster with restricted Kubernetes Pod Security Standards

As part of compliance with the restricted profile for Kubernetes Pod Security Standards, you must set the `securityContext` of containers and initContainers. When a pod does not meet pod Security Standards, it is not created and vulnerability scanning cannot proceed. For more information, see the Kubernetes documentation.

You might see an error message similar to the following when describing the TaskRun:

```
pods "trivy-ivs-abcd-scan-task-pod" is forbidden: violates PodSecurity "restricted:lat
est": allowPrivilegeEscalation != false (container "prepare" must set securityContext.
allowPrivilegeEscalation=false), unrestricted capabilities (container "prepare" must s
et securityContext.capabilities.drop=["ALL"]), seccompProfile (pod or container "prepa
re" must set securityContext.seccompProfile.type to "RuntimeDefault" or "Localhost").
Maybe invalid TaskSpec. ScanPodError PodNotFound: no pod found
```

To resolve this issue:

1. Update your Tekton Pipelines package configuration in your `tap-values.yaml` with the following changes:

   ```
   tekton_pipelines:
       feature_flags:
           set_security_context: "true"
   ```

   Setting the `securityContext` resolves the `prepare` initContainer violation.

2. Update your Tanzu Application Platform installation by running:

```
tanzu package installed update tap -p tap.tanzu.vmware.com -v TAP-VERSION  --va
lues-file tap-values.yaml -n tap-install
```

Where `TAP-VERSION` is the version of Tanzu Application Platform installed.

3. Re-run the scan.

Supply Chain Security Tools - Scan 2.0 with Tanzu Supply Chain (Beta)

This topic provides an overview of using Supply Chain Security Tools (SCST) - Scan 2.0 with Tanzu Supply Chain (Beta). This section is only applicable if you are evaluating the beta release of Tanzu Supply Chain.

Integrating into a Tanzu Supply Chain

The SCST - Scan 2.0 component defines how to scan a container image with a scan solution by using the generic Kubernetes custom resource `ImageVulnerabilityScan`. This provides a generic interface that allows you to declare how Tanzu Application Platform executes a scan on a container image for a container image scan solution.

To run an `ImageVulnerabilityScan` in a Tanzu Supply Chain:

- Construct a Supply Chain using the Tanzu CLI.
- Create a Workload.

Getting started with SCST - Scan 2.0 with Tanzu Supply Chain

To use SCST - Scan 2.0 in a Tanzu Supply Chain, see Create Tanzu Supply Chain with SCST - Scan 2.0.

Create Tanzu Supply Chain with SCST - Scan 2.0

This section tells you how to create a Tanzu Supply Chain with SCST - Scan 2.0.

- Install SCST - Scan 2.0
- Set up the Supply Chain Component
- Create a Supply Chain that uses SCST - Scan 2.0 with a Component
- Create a workload from the Supply Chain

Install Supply Chain Security Tools - Scan 2.0 in a cluster

This topic describes how to install Supply Chain Security Tools (SCST) - Scan 2.0 if are not using a profile. By default, SCST - Scan 2.0 is installed in the build and full profiles.

Note

Follow the steps in this topic if you do not want to use a profile to install SCST - Scan 2.0. For more information about profiles, see Installation profiles in Tanzu Application Platform v1.9.

Prerequisites

SCST - Scan 2.0 requires the following prerequisites:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- Install the Tekton component.
- To store results, install AMR, and AMR Observer. Downstream Tanzu Application Platform services, such as Tanzu Developer Portal and Tanzu CLI, depend on scan results stored in SCST - Store to display correctly. For more information, see Artifact Metadata Repository Observer for Supply Chain Security Tools - Store.

Note

If you are installing SCST - Scan 2.0 in a cluster with restricted Kubernetes Pod Security Standards, you must update configurations for the Tekton Pipelines package. See Scanning in a cluster with restricted Kubernetes Pod Security Standards.

Configure properties

When you install SCST - Scan 2.0, you can configure the following optional properties:

Key	Default	Type	Description
caCertData	""	string	The custom certificates trusted by the scan's connections
docker.import	true	Boolean	Import `docker.pullSecret` from another namespace (requires secretgen-controller). Set to false if the secret is already present.
docker.pullSecret	registries-credentials	string	Name of a Docker pull secret in the deployment namespace to pull the scanner images
scans.maxConcurrentScans	10	integer	Maximum number of scans running at one time. Set to 0 to have no limit.
scans.priorityClassName	""	string	Name of a predefined PriorityClass to apply to scan pods
workspace.storageSize	2Gi	string	Size of the PersistentVolume that the Tekton pipelineruns uses
workspace.storageClass	""	string	Name of the storage class to use while creating the PersistentVolume claims used by Tekton pipelineruns

Note

If the StorageClass you select does not have a node limit but uses the node storage, such as hostpath, the nodes must have large enough disks. For example, if a scan creates a 2Gi volume on a hostpath type storage class, `2Gi * number of AMR images` indicates how much storage this cluster needs overall. `2Gi * number of AMR images / number of nodes` indicates how much storage each node needs.

Install

To install SCST - Scan 2.0:

1. List version information for the package:

```
tanzu package available list app-scanning.apps.tanzu.vmware.com --namespace tap
-install
```

For example:

```
$ tanzu package available list app-scanning.apps.tanzu.vmware.com --namespace t
ap-install
- Retrieving package versions for app-scanning.apps.tanzu.vmware.com...
    NAME                                VERSION              RELEASED-AT
    app-scanning.apps.tanzu.vmware.com  0.1.0              2023-03-01 20:00:00 -
0400 EDT
```

2. (Optional) Make changes to the default installation settings:

Retrieve the configurable settings:

```
tanzu package available get app-scanning.apps.tanzu.vmware.com/VERSION --values
-schema --namespace tap-install
```

Where `VERSION` is your package version number. For example, `0.1.0`.

For example:

```
tanzu package available get app-scanning.apps.tanzu.vmware.com/0.1.0 --values-s
chema --namespace tap-install
| Retrieving package details for app-scanning.apps.tanzu.vmware.com/0.1.0...

  KEY                     DEFAULT                 TYPE     DESCRIPTION
  docker.import           true                    boolean  Import `docker.pullS
ecret` from another namespace (requires
                                                           secretgen-controlle
r). Set to false if the secret will already be present.
  docker.pullSecret       registries-credentials  string   Name of a Docker pul
l secret in the deployment namespace to pull the scanner
                                                           images.
  workspace.storageSize   2Gi                     string   Size of the Persiste
nt Volume to be used by the tekton pipelineruns
  workspace.storageClass                          string   Name of the storage
class to use while creating the Persistent Volume Claims
                                                           used by tekton pipel
ineruns
  caCertData                                      string   The custom certifica
tes to be trusted by the scan's connections
```

To edit any of the default installation settings, create an `app-scanning-values-file.yaml` and append the key-value pairs to be modified to the file. For example:

```
scans:
  workspace:
    storageSize: 200Mi
```

3. Install the package. If you did not edit the default installation settings, you do not need to specify the `--values-file` flag.

```
tanzu package install app-scanning --package-name app-scanning.apps.tanzu.vmwar
e.com \
    --version VERSION \
    --namespace tap-install \
    --values-file app-scanning-values-file.yaml
```

Where `VERSION` is your package version number. For example, `0.1.0`.

For example:

```
tanzu package install app-scanning --package app-scanning.apps.tanzu.vmware.com
\
    --version 0.1.0 \
    --namespace tap-install \
    --values-file app-scanning-values-file.yaml

    Installing package 'app-scanning.apps.tanzu.vmware.com'
    Getting package metadata for 'app-scanning.apps.tanzu.vmware.com'
    Creating service account 'app-scanning-default-sa'
    Creating cluster admin role 'app-scanning-default-cluster-role'
    Creating cluster role binding 'app-scanning-default-cluster-rolebinding'
    Creating package resource
    Waiting for 'PackageInstall' reconciliation for 'app-scanning'
    'PackageInstall' resource install status: Reconciling
    'PackageInstall' resource install status: ReconcileSucceeded
```

Configure service accounts and registry credentials

This section contains instructions for running a standalone `ImageVulnerabilityScan` or using multiple registries.

If the image that you are scanning using a default scanner or your own scanner, and your vulnerability scanner image are located in private registries different from the `Tanzu Application Platform bundles registry`, you must edit your scanner service account to include registry credentials for these registries.

Important

If your use case is listed below, skip this topic and proceed to Enable App Scanning for default Test and Scan supply chains.

- You are running an `ImageVulnerabilityScan` in the context of a supply chain.
- You used the Namespace Provisioner to provision your developer namespace. For more information, see the Namespace Provisioner documentation.

To configure service accounts and registry credentials, SCST - Scan 2.0 requires the following access:

Registry	Permission	Service Account	Example
Tanzu Application Platform bundles registry	Read	scanner	`registry.tanzu.vmware.com`
Target image registry	Read	scanner	`your-registry.io`
Vulnerability scanner image registry	Read	scanner	`your-registry.io`
Scan results location registry	Write	publisher	`your-registry.io`

Where:

- `Tanzu Application Platform bundles registry` is the registry containing the Tanzu Application Platform bundles. This is the registry from the Relocate images to a registry step or `registry.tanzu.vmware.com`.
- `Target image registry` is the registry containing the image to scan. This registry credential is required if you are scanning a private image. The `image to scan is` called the `target`

`image` or `TARGET-IMAGE`.

- `Vulnerability scanner image registry` is the registry containing your vulnerability scanner image. This is only needed if you are bringing your own scanner and your vulnerability scanner image is located in a private registry different from the `Tanzu Application Platform bundles registry`.
- `Scan results location registry` is the registry where scan results are published.

To configure service accounts and registry credentials:

1. Create a secret `scanning-tap-component-read-creds` with read access to the registry containing the Tanzu Application Platform bundles. This pulls the SCST - Scan 2.0 images. If you previously relocated the Tanzu Application Platform bundles to your own registry, you can also place your vulnerability scanner image in this registry.

   ```
   read -s TAP_REGISTRY_PASSWORD
   kubectl create secret docker-registry scanning-tap-component-read-creds \
     --docker-username=TAP-REGISTRY-USERNAME \
     --docker-password=$TAP_REGISTRY_PASSWORD \
     --docker-server=TAP-REGISTRY-URL \
     -n DEV-NAMESPACE
   ```

   Where `DEV-NAMESPACE` is the developer namespace where scanning occurs.

2. If you are scanning a private target image, create a secret `target-image-read-creds` with read access to the registry containing that target image.

   Important

   If you followed the directions for Install Tanzu Application Platform, you can skip this step and use the `targetImagePullSecret` secret with your service account as referenced in your `tap-values.yaml`. See Full profile.

   ```
   read -s REGISTRY_PASSWORD
   kubectl create secret docker-registry target-image-read-creds \
     --docker-username=REGISTRY-USERNAME \
     --docker-password=$REGISTRY_PASSWORD \
     --docker-server=REGISTRY-URL \
     -n DEV-NAMESPACE
   ```

3. Create a secret `write-creds` with write access to the registry for the scanner to upload the scan results to.

   ```
   read -s WRITE_PASSWORD
   kubectl create secret docker-registry write-creds \
     --docker-username=WRITE-USERNAME \
     --docker-password=$WRITE_PASSWORD \
     --docker-server=DESTINATION-REGISTRY-URL \
     -n DEV-NAMESPACE
   ```

4. (Optional) If you are bringing your own vulnerability scanner and your vulnerability scanner image is located in a private registry different from the registry containing your Tanzu Application Platform bundles, you must create a secret `vulnerability-scanner-image-read-creds` with read access to the registry.

   ```
   read -s WRITE_PASSWORD
   kubectl create secret docker-registry vulnerability-scanner-image-read-creds \
     --docker-username=WRITE-USERNAME \
     --docker-password=$WRITE_PASSWORD \
   ```

```
  --docker-server=REGISTRY-URL \
  -n DEV-NAMESPACE
```

5. Create a `scanner-sa.yaml` file containing the service account `scanner` which enables SCST - Scan 2.0 to pull both the vulnerability scanner image and target image. Attach the one or more read secrets created earlier pulling the Tanzu Application Platform bundles, and optionally, your vulnerability scanner image under `imagePullSecrets`. Attach the read secret created earlier for your target image under `secrets`.

   ```
   apiVersion: v1
   kind: ServiceAccount
   metadata:
     name: scanner
     namespace: DEV-NAMESPACE
   imagePullSecrets:
   - name: scanning-tap-component-read-creds
   - name: vulnerability-scanner-image-read-creds # optional
   secrets:
   - name: target-image-read-creds
   ```

   Where:

   - `imagePullSecrets.name` includes the name of the secret used to pull the scan component from the registry. If you are bringing your own vulnerability scanner and the vulnerability scanner image is located in a separate private registry, you must also include the name of the secret with those registry credentials.
   - `secrets.name` is the name of the secret used to pull the target image to scan. This is required if the image you are scanning is private.

6. Apply the service account to your developer namespace by running:

   ```
   kubectl apply -f scanner-sa.yaml
   ```

7. Create a `publisher-sa.yaml` file containing the service account `publisher` which enables SCST - Scan 2.0 to push the scan results to a user specified registry.

   ```
   apiVersion: v1
   kind: ServiceAccount
   metadata:
     name: publisher
     namespace: DEV-NAMESPACE
   imagePullSecrets:
   - name: scanning-tap-component-read-creds
   secrets:
   - name: write-creds
   ```

   Where:

   - `imagePullSecrets.name` is the name of the secret used to pull the scan component image from the registry.
   - `secrets.name` is the name of the secret used to publish the scan results.

8. Apply the service account to your developer namespace by running:

   ```
   kubectl apply -f publisher-sa.yaml
   ```

(Optional) Set up your registry retention policy

Although Tanzu Application Platform ingests scan artifacts into the Metadata Store, and stores information such as packages and parsed vulnerabilities, only a pointer to the original SBOM location is stored. The original SBOM generated by the scan is not preserved within the Metadata Store. VMware recommends that you keep these original artifacts according to your organization's archival requirements.

If the registry specified to push scan results to support retention policies, you can configure the registry to delete old scan results automatically, depending on your archival requirements. Scan result artifacts accumulate over time and can quickly consume hard disk space.

For information about configuring Harbor tag retention rules, see the Harbor documentation. For example, you can configure Harbor to retain the most recently pushed # artifacts or retain the artifacts pushed within the last # days.

Retention policy setup differs between registry providers. Confirm with your specific registry's documentation about the configuration options.

Set up the Supply Chain Component

This topic describes tells you how to install the Trivy Supply Chain Component, and create a Custom Supply Chain Component by using SCST - Scan 2.0.

Install Trivy Supply Chain Component

This section describes how to install the Trivy Supply Chain Component that uses SCST - Scan 2.0.

1. List version information for the Trivy Supply Chain Component package by running.

   ```
   tanzu package available list trivy.app-scanning.component.apps.tanzu.vmware.com
   --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package available list trivy.app-scanning.component.apps.tanzu.vmware.c
   om --namespace tap-install

   NAME                                                VERSION
   RELEASED-AT
   trivy.app-scanning.component.apps.tanzu.vmware.com  TRIVY-COMPONENT-VERSION
   2024-01-26 12:35:39 -0500 EST
   ```

2. Install the Trivy Supply Chain Component package by running.

   ```
   tanzu package install trivy-app-scanning-component -p trivy.app-scanning.compon
   ent.apps.tanzu.vmware.com \
       --version TRIVY-COMPONENT-VERSION \
       --namespace tap-install
   ```

Create a Custom Scanning Component

This section describes how to create a Custom Scanning Supply Chain Component that uses SCST - Scan 2.0. For more details about how to create a Component, see Tanzu Supply Chain docs.

1. Retrieve the component YAML of the Trivy Supply Chain Component by running.

   ```
   kubectl get component trivy-image-scan-1.0.0 -n trivy-app-scanning-catalog -o y
   aml > component.yaml
   ```

2. Edit the following lines in `component.yaml`:

```
apiVersion: supply-chain.apps.tanzu.vmware.com/v1alpha1
kind: Component
metadata:
  name: trivy-image-scan-1.0.0 # change to SCANNER-image-scan-1.0.0
  namespace: trivy-app-scanning-catalog # change to DEV-NAMESPACE
...
  description: Performs a trivy image scan using the scan 2.0 components # chan
ge trivy to SCANNER
  ...
    - name: image-scanning-cli
      value: # change to registry url of scanner image
    ...
    - name: image-scanning-steps-env-vars
      value: '[{"name":"<Name of Env var>","value":"<value of Env var>"}]' # in
sert env vars inside nested {}
    ...
    pipelineRef:
      name: trivy-image-scan-v2 # SCANNER-image-scan-v2. Replace with the name
of the pipeline created in the next step.
```

3. Remove the following fields from `component.yaml`:

```
metadata:
  annotations:
    ...
  creationTimestamp:
  generation:
  labels:
    ...
  resourceVersion:
  uid:
```

4. Customize a pipeline by retrieving the YAML of the Trivy Supply Chain Component's Pipeline.

    1. Retrieve pipeline YAML by running:

    ```
    kubectl get pipeline trivy-image-scan-v2 -n trivy-app-scanning-catalog -o
    yaml > pipeline.yaml
    ```

    2. Edit the following lines in the `pipeline.yaml`:

    ```
    apiVersion: tekton.dev/v1
    kind: Pipeline
    metadata:
      name: trivy-image-scan-v2 # change to SCANNER-image-scan-v2
    spec:
      description: Scans your image for vulnerabilities using Trivy # change
    Trivy to SCANNER
    ...
            resourceSpec:
              apiVersion: app-scanning.apps.tanzu.vmware.com/v1alpha1
              kind: ImageVulnerabilityScan
              metadata:
                annotations:
                  app-scanning.apps.tanzu.vmware.com/scanner-name: Trivy # ch
    ange to SCANNER
                ...
                steps:
                - args:
                  - # insert array of steps to run for SCANNER
                ...
    ```

```
              name: trivy-generate-report # change to any SCANNER
            ...
```

Where `SCANNER` is the name of the scanner from the scanning component

3. Remove the following fields from the `pipeline.yaml`:

```
metadata:
  annotations:
    ...
  creationTimestamp:
  generation:
  labels:
    ...
  resourceVersion:
  uid:
```

5. Apply the custom component and pipeline by running:

```
kubectl apply -f component.yaml -n DEV-NAMESPACE
kubectl apply -f pipeline.yaml -n DEV-NAMESPACE
```

6. (Optional) If you created your own component, it requires the following label so that it can be observed by Tanzu Supply Chain:

```
labels:
  supply-chain.apps.tanzu.vmware.com/catalog: tanzu
```

View components

This section tells you how to view the available components that were installed or applied.

```
kubectl get components -A -l "supply-chain.apps.tanzu.vmware.com/catalog=tanzu"
```

Example output:

```
$ kubectl get components -A -l "supply-chain.apps.tanzu.vmware.com/catalog=tanzu"

NAMESPACE                    NAME                               RESUMPTIONS   READY   R
EASON   AGE
alm-catalog                  aggregator-1.0.0                   False         True    R
eady    3d
alm-catalog                  app-config-server-1.0.0            False         True    R
eady    3d
alm-catalog                  app-config-web-1.0.0               False         True    R
eady    3d
alm-catalog                  app-config-worker-1.0.0            False         True    R
eady    3d
alm-catalog                  carvel-package-1.0.0               False         True    R
eady    3d
alm-catalog                  deployer-1.0.0                     False         True    R
eady    3d
alm-catalog                  source-package-translator-1.0.0    False         True    R
eady    3d
conventions-component        conventions-1.0.0                  False         True    R
eady    3d
git-writer-catalog           git-writer-1.0.0                   False         True    R
eady    3d
git-writer-catalog           git-writer-pr-1.0.0                False         True    R
eady    3d
grype-app-scanning-catalog   grype-image-scan-1.0.0            False         True    R
eady    10h
```

```
source-provider               source-git-provider-1.0.0          True          True    R
eady    3d
tbs-catalog                   buildpack-build-1.0.0              True          True    R
eady    3d
trivy-app-scanning-catalog    trivy-image-scan-1.0.0             False         True    R
eady    3d
```

Create a Supply Chain that uses SCST - Scan 2.0 with a Component

This topic tels you how to create a Tanzu Supply Chain with SCST - Scan 2.0, which replaces the previous solution for Supply Chain Choreographer for Tanzu.

Prerequisites

- Tanzu Supply Chain packages
    - Tanzu Supply Chain packages
    - Managed Resource Controller

  For more information, see Packages.
- Tanzu Supply Chain components
    - Source
    - Buildpack
    - Trivy Scanning
- Tanzu Application Platform packages
    - SCST - Scan 2.0
    - Tekton
- Tanzu CLI plug-ins
    - Tanzu Workload CLI plug-in
    - Tanzu Supply Chain CLI plug-in

  For more information see, Install the Tanzu Supply Chain CLI plug-ins.

Create a Supply Chain with SCST - Scan 2.0 and a Component

This section tells you how to create a supply chain with SCST - Scan 2.0 by using either a Trivy Supply Chain Component or a Custom Scanning Component.

Using Trivy Supply Chain Component
Complete the following steps:

1. Initialize Tanzu Supply Chain by running:

   ```
   tanzu supplychain init --group example.com
   ```

   Example output:

   ```
   $ tanzu supplychain init --group example.com
   Initializing group example.com
   Creating directory structure
   ```

```
├─ supplychains/
├─ components/
├─ pipelines/
├─ tasks/
└─ config.yaml

Writing group configuration to config.yaml
```

2. Generate a Supply Chain by running:

```
tanzu supplychain generate --kind TrivySC \
--description Trivy \
--component source-git-provider-1.0.0 \
--component buildpack-build-1.0.0 \
--component trivy-image-scan-1.0.0
```

Example output:

```
$ tanzu supplychain generate --kind TrivySC \
 --description Trivy \
 --component source-git-provider-1.0.0 \
 --component buildpack-build-1.0.0 \
 --component trivy-image-scan-1.0.0

✓ Successfully fetched all component dependencies
Created file supplychains/trivysc.yaml
Created file components/buildpack-build-1.0.0.yaml
Created file components/source-git-provider-1.0.0.yaml
Created file components/trivy-image-scan-1.0.0.yaml
Created file pipelines/buildpack-build.yaml
Created file pipelines/source-git-provider.yaml
Created file pipelines/trivy-image-scan-v2.yaml
Created file tasks/calculate-digest.yaml
Created file tasks/check-builders.yaml
Created file tasks/prepare-build.yaml
Created file tasks/source-git-check.yaml
Created file tasks/source-git-clone.yaml
Created file tasks/store-content-oci.yaml
```

Using Custom Scanning Component

Complete the following steps:

1. Initialize Tanzu Supply Chain by running:

```
tanzu supplychain init --group example.com
```

Example output:

```
$ tanzu supplychain init --group example.com
Initializing group example.com
Creating directory structure
├─ supplychains/
├─ components/
├─ pipelines/
├─ tasks/
└─ config.yaml

Writing group configuration to config.yaml
```

2. Generate a Supply Chain by running:

```
tanzu supplychain generate --kind CUSTOM-KIND-WORKLOAD \
--description DESCRIPTION-OF-SCANNER \
```

```
--component source-git-provider-1.0.0 \
--component buildpack-build-1.0.0 \
--component SCANNING-COMPONENT-NAME
```

Where `SCANNING-COMPONENT-NAME` is the name of the Custom Scanning Component.

For more information about how to construct a Supply Chain by using the Tanzu CLI, see Construct a Supply Chain using the CLI.

For more details about how to create a Supply Chain, see Build your first Supply Chain.

Apply Supply Chain

Generating the supply chain created the following directory structure:

```
├─ supplychains/
├─ components/
├─ pipelines/
├─ tasks/
```

Apply these directories to the namespace where the workload will run by running.

```
kubectl apply -R -f components -f supplychains -f tasks -f pipelines -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the same namespace where the intended workload will be.

Create a Workload from the Supply Chain

This topic tells you how to create and apply a workload from a Supply Chain, how to observe a workload, and how to verify the scanning performed in a workload.

Define a workload

This section tells you how to create a workload from an existing Supply Chain that was created by using SCST - Scan 2.0 and one of the following:

- Trivy Supply Chain Component
- Custom Scanning Component

You can define a workload in YAML or use the Tanzu Workload CLI plug-in.

Using YAML

Run:

```
kind: KIND
apiVersion: API-VERSION
metadata:
  name: WORKLOAD-NAME
spec:
  registry:
    repository: REGISTRY-REPOSITORY
    server: REGISTRY-SERVER
  source:
    git:
      branch: GIT-BRANCH
      url: GIT-URL
    subPath: GIT-SUBPATH
```

Where:

- `KIND` is the kind defined in the Trivy Supply Chain Component or Custom Scanning Component. The kind can be found in the supplychain YAML in the supplychains directory.
- `API-VERSION` is defined in the Create a Supply Chain with SCST - Scan 2.0 and Trivy Supply Chain Component or Create Supply Chain with SCST - Scan 2.0 and Custom Scanning Component. `API-VERSION` is the `group` and `version` found in the supplychain YAML in the supplychains directory.
- `REGISTRY-REPOSITORY` is the registry repository used for the scan results location.
- `REGISTRY-SERVER` is the registry server used for the scan results location.
- `GIT-URL` is the Git repository URL to clone from for the source component.
- `GIT-BRANCH` is the Git branch reference to watch for the new source.
- `GIT-SUBPATH` is the path where the source code is located.

For more information about any of the `GIT-*` values, see Source Git Provider.

Using Tanzu Workload CLI plug-in

Run:

```
tanzu workload generate NAME --kind KIND
```

Where `KIND` is the kind API resource defined in the Trivy Supply Chain Component or Custom Scanning Component.

This renders a sample workload YAML that you can configure and put in a `workload.yaml`.

Create a workload

Using the `workload.yaml` created in the previous section, create the workload:

```
tanzu workload create --file workload.yaml --namespace DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the same namespace where the intended workload will be.

Observe a workload

This section shows you how to use the Tanzu Workload CLI plug-in to observe a workload.

1. List workloads in the cluster by running:

   ```
   tanzu workload list -n DEV-NAMESPACE
   ```

   Where `DEV-NAMESPACE` is the namespace where the workload is.

   Example output:

   ```
   $ tanzu workload list -n grype-app-scanning-catalog

   Listing workloads from all namespaces

     NAMESPACE                   NAME                          KIND
   VERSION   AGE
     grype-app-scanning-catalog  golang-app-grype-test  grypescs.example.com  v1al
   pha1  35m
   ```

```
  To see more details about a workload, use 'tanzu workload get workload-name --kind workload-kind'
```

2. View workload details by running:

   ```
   tanzu workload get Sample WORKLOAD-NAME
   ```

   Example output:

   ```
   $ tanzu workload get golang-app-grype-test  -n grype-app-scanning-catalog
     Overview
     name:       golang-app-grype-test
     kind:       grypescs.example.com/golang-app-grype-test
     namespace:  grype-app-scanning-catalog
     age:        37m

     Runs:
     ID                                    STATUS     DURATION  AGE
     golang-app-grype-test-run-btlvx  Succeeded  4m13s     37m

     To view a run information, use 'tanzu workload run get run-id'
   ```

   For more information, How to observe the Runs of your Workload.

Check the workload scan results

Get the `ImageVulnerabilityScan` name by looking in the namespace it was created in:

```
kubectl get ivs -n DEV-NAMESPACE
```

Example output:

```
NAMESPACE                   NAME                          SUCCEEDED   REASON      AGE
dev-namespace               golang-app-test-123-bbrpz     True        Succeeded   4m52s
```

For information about how to retrieve scan results by using the `ImageVulnerabilityScan` name, see Retrieve scan results.

For more information about how to create a workload, see Work with Workloads.

Supply Chain Security Tools - Scan 2.0 Observability

This topic guides you through observing Supply Chain Security Tools (SCST) - Scan 2.0. This helps you understand each step of scanning.

Scanning Steps

This section describes each of the scanning steps and corresponding observability methods.

- To watch the status of the scanning custom resources and child resources:

  ```
  kubectl get -l imagevulnerabilityscan pipelinerun,taskrun,pod
  kubectl get imagevulnerabilityscan
  ```

- View the status, reason, and urls:

  ```
  kubectl get imagevulnerabilityscan -o wide
  ```

- View the complete status and events of scanning custom resources:

```
kubectl describe imagevulnerabilityscan IMAGE-VULNERABILITY-SCAN-NAME
```

Where `IMAGE-VULNERABILITY-SCAN-NAME` is the name of an `ImageVulnerabilityScan` resource you want to inspect.

- List the child resources of a scan:

```
kubectl get -l imagevulnerabilityscan=$NAME pipelinerun,taskrun,pod
```

- Get the logs of the controller:

```
kubectl logs -f deployment/app-scanning-controller-manager -n app-scanning-syst
em -c manager
```

Troubleshooting Supply Chain Security Tools - Scan 2.0

This topic helps you troubleshoot Supply Chain Security Tools (SCST) - Scan 2.0.

Overview

When Scan 2.0 creates an ImageVulnerabilityScan, the following resources are also created:

- Tekton `PipelineRun` with the following `Tasks`:
  - workspace-setup-task
  - scan-task
  - publish-task
- Tekton `TaskRun` corresponding to each `Task`
- `Pod` corresponding to each `TaskRun`

Viewing resources

- To view all resources:

```
kubectl get imagevulnerabilityscans,pipelineruns,taskruns,pods -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the name of your developer namespace.

- To verify which resources are failing, proceed to the following debugging sections:

```
NAME                                                                SUCCEEDED
REASON
imagevulnerabilityscan.app-scanning.apps.tanzu.vmware.com/my-scan   False
Failed

NAME                                    SUCCEEDED   REASON      STARTTIME   COMP
LETIONTIME
pipelinerun.tekton.dev/my-scan-5kllf    False       Failed      2m10s       85s

NAME                                                    SUCCEEDED   REASON
STARTTIME   COMPLETIONTIME
taskrun.tekton.dev/my-scan-5kllf-publish-task           False       Failed
94s         85s
taskrun.tekton.dev/my-scan-5kllf-scan-task              True        Succeeded
2m1s        94s
taskrun.tekton.dev/my-scan-5kllf-workspace-setup-task   True        Succeeded
2m9s        2m1s
```

```
NAME                                          READY   STATUS      RESTARTS   AGE
pod/my-scan-5kllf-publish-task-pod            0/4     Completed   1          94s
pod/my-scan-5kllf-scan-task-pod               0/4     Completed   1          2m
pod/my-scan-5kllf-workspace-setup-task-pod    0/2     Completed   1          2m10s
```

Debugging commands

The following sections describe commands you run to get logs and details about scanning errors.

Debugging resources

If a resource fails or has errors, inspect the resource. If multiple resources are involved, inspecting them all can provide a broader understanding. For example, inspecting the corresponding `TaskRun` to a failed `Pod`.

To get status conditions on a resource:

```
kubectl describe RESOURCE RESOURCE-NAME -n DEV-NAMESPACE
```

Where:

- `RESOURCE` is one of the following: `ImageVulnerabilityScan`, `PipelineRun`, `TaskRun`, or `Pod`.
- `RESOURCE-NAME` is the name of the `RESOURCE`.
- `DEV-NAMESPACE` is the name of your developer namespace.

Debugging scan pods

You can use the following methods to debug scan pods:

- To get error logs from a pod when scan pods fail:

  ```
  kubectl logs SCAN-POD-NAME -n DEV-NAMESPACE
  ```

  Where `SCAN-POD-NAME` is the name of the scan pod.

  For information about debugging Kubernetes pods, see the Kubernetes documentation.

  A scan run that has an error can indicate that one of the following step containers has a failure:

  - `step-workspace-setup`
  - `step-write-certs`
  - `step-cred-helper`
  - `step-SCANNER`
  - `step-publisher`
  - `sidecar-sleep`

  Where `step-SCANNER` is your scanner step.

- To verify which step container had a failed exit code:

  ```
  kubectl get taskrun TASKRUN-NAME -o json | jq .status
  ```

  Where `TASKRUN-NAME` is the name of the TaskRun.

- To inspect a specific step container in a pod:

```
kubectl logs scan-pod-name -n DEV-NAMESPACE -c step-container-name
```

Where `DEV-NAMESPACE` is your developer namespace.

For information about debugging a TaskRun, see the Tekton documentation.

- To debug inside of the scan-task pod: Add an additional step with a `sleep` command after your scanner step in the ImageVulnerabilityScan. For example:

```
...
spec:
  ...
  steps:
  - name: SCANNER-STEP
    ...
  - name: view
    image: busybox:latest
    args:
    - -c
    - sleep 6000
```

This keeps the pod in a running state so that you can exec into it. Re-run the scan and then exec into the pod:

```
    kubectl exec SCAN-TASK-POD-NAME -n DEV-NAMESPACE -c step-view --stdin --tty
-- sh
```

Where `SCAN-TASK-POD-NAME` is the name of your scan-task pod.

Viewing the Scan-Controller manager logs

You can run these commands to view the Scan-Controller manager logs:

- Retrieve scan-controller manager logs:

```
kubectl logs deployment/app-scanning-controller-manager -n app-scanning-system
```

- Tail scan-controller manager logs:

```
kubectl logs -f deployment/app-scanning-controller-manager -n app-scanning-syst
em
```

Troubleshooting issues

Volume permission error

If you encounter a permission error for accessing, opening, and writing to the files inside cluster volume, such as:

```
unsuccessful cred copy: ".git-credentials" from "/tekton/creds" to "/home/app-scannin
g": unable to open destination: open /home/app-scanning/.git-credentials: permission d
enied
```

Ensure that the problematic step runs with the proper user and group ids.

Incompatible Tekton version

Tanzu Application Platform `v1.7.0` or later includes `app-scanning.apps.tanzu.vmware.com` version `0.2.0` and Tekton Pipelines version `0.50.1`. The `app-scanning.apps.tanzu.vmware.com` package is incompatible with previous versions of Tekton Pipelines as v1 CRDs were not enabled. You must

upgrade Tanzu Application Platform to `v1.7.0` or later before upgrading `app-scanning.apps.tanzu.vmware.com`.

If you did not upgrade Tanzu Application Platform before upgrading `app-scanning.apps.tanzu.vmware.com`, you can encounter ImageVulnerabilityScans not progressing:

```
NAME      SUCCEEDED   REASON
my-scan
```

To resolve this issue:

1. Confirm that the issue is due to installing an incompatible Tekton version by viewing the controller manager logs by running:

   ```
   kubectl -n app-scanning-system logs -f deployment/app-scanning-controller-manag
   er -c manager
   ```

   If you encounter the following error, proceed to the next step:

   ```
   ERROR  controller-runtime.source.EventHandler  failed to get informer from cach
   e  {"error": "failed to get API group resources: unable to retrieve the complet
   e list of server APIs: tekton.dev/v1: the server could not find the requested r
   esource"}
   ```

2. Upgrade Tanzu Application Platform to `v1.7.0` or later. See Upgrade your Tanzu Application Platform.

Scan results empty

The publish-task task fails if the `scan-results-path` (default value of `/workspace/scan-result`) is empty. To confirm, view the logs of the publish-task pod:

```
kubectl logs PUBLISH-TASK-POD-NAME -c step-publisher -n DEV-NAMESPACE
```

Where `PUBLISH-TASK-POD-NAME` is the name of your publish-task pod.

```
2023/08/22 17:09:49 results folder /workspace/scan-results is empty
```

To resolve this issue, you can debug within the scan-task pod by following the instructions under Debugging scan pods. You must use an image with both a shell and your scanner CLI image to run the `sleep` command and troubleshoot your scanner commands from within the container.

Scanning in a cluster with restricted Kubernetes Pod Security Standards

As part of compliance with the restricted profile for Kubernetes Pod Security Standards, you must set the `securityContext` of containers and initContainers. When a pod does not meet pod Security Standards, it is not created and vulnerability scanning cannot proceed. For more information, see the Kubernetes documentation.

You might see an error message similar to the following when describing the TaskRun:

```
pods "trivy-ivs-abcd-scan-task-pod" is forbidden: violates PodSecurity "restricted:lat
est": allowPrivilegeEscalation != false (container "prepare" must set securityContext.
allowPrivilegeEscalation=false), unrestricted capabilities (container "prepare" must s
et securityContext.capabilities.drop=["ALL"]), seccompProfile (pod or container "prepa
re" must set securityContext.seccompProfile.type to "RuntimeDefault" or "Localhost").
Maybe invalid TaskSpec. ScanPodError PodNotFound: no pod found
```

To resolve this issue:

1. Update your Tekton Pipelines package configuration in your `tap-values.yaml` with the following changes:

   ```
   tekton_pipelines:
       feature_flags:
           set_security_context: "true"
   ```

   Setting the `securityContext` resolves the `prepare` initContainer violation.

2. Update your Tanzu Application Platform installation by running:

   ```
   tanzu package installed update tap -p tap.tanzu.vmware.com -v TAP-VERSION  --va
   lues-file tap-values.yaml -n tap-install
   ```

   Where `TAP-VERSION` is the version of Tanzu Application Platform installed.

3. Re-run the scan.

Overview of Supply Chain Security Tools for Tanzu – Store

This topic gives you an overview of Supply Chain Security Tools (SCST) – Store.

SCST - Store stores the metadata generated by supplies chains in Tanzu Application Platform. It is made up of two services: Artifact Metadata Repository (AMR) and the Metadata Store.

Artifact Metadata Repository

AMR includes both AMR Observer and AMR CloudEvent Handler.

AMR Observer

AMR Observer is a set of managed controllers that watches for relevant updates on resources of interest. When relevant events are observed, a CloudEvent is generated and sent to AMR CloudEvent Handler. By design, you can deploy this component on Full, Build, and Run profile clusters.

AMR CloudEvent Handler

AMR CloudEvent Handler receives CloudEvents and stores relevant information in Artifact Metadata Repository or the Metadata Store. By design, you can deploy this component on Full and View profile clusters.

Additional resources

- Artifact Metadata Repository architecture
- Configure Artifact Metadata Repository
- AMR GraphQL Query

The Metadata Store

For more information about the Metadata Store, see Overview of the Metadata Store.

Overview of Supply Chain Security Tools for Tanzu – Store

This topic gives you an overview of Supply Chain Security Tools (SCST) – Store.

SCST - Store stores the metadata generated by supplies chains in Tanzu Application Platform. It is made up of two services: Artifact Metadata Repository (AMR) and the Metadata Store.

Artifact Metadata Repository

AMR includes both AMR Observer and AMR CloudEvent Handler.

AMR Observer

AMR Observer is a set of managed controllers that watches for relevant updates on resources of interest. When relevant events are observed, a CloudEvent is generated and sent to AMR CloudEvent Handler. By design, you can deploy this component on Full, Build, and Run profile clusters.

AMR CloudEvent Handler

AMR CloudEvent Handler receives CloudEvents and stores relevant information in Artifact Metadata Repository or the Metadata Store. By design, you can deploy this component on Full and View profile clusters.

Additional resources

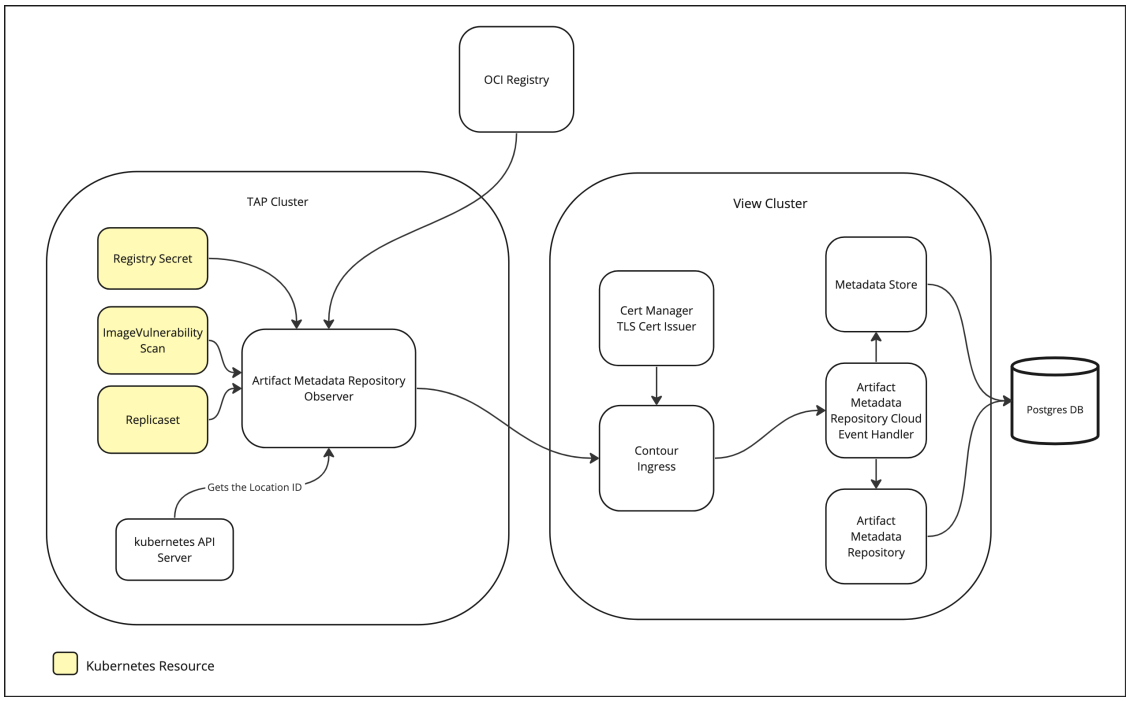
- Artifact Metadata Repository architecture
- Configure Artifact Metadata Repository
- AMR GraphQL Query

The Metadata Store

For more information about the Metadata Store, see Overview of the Metadata Store.

Artifact Metadata Repository architecture

This topic gives you an overview of the Artifact Metadata Repository (AMR) architecture.



AMR Observer

The full, build, and run clusters include AMR Observer. AMR Observer communicates with the Kubernetes API Server to obtain the cluster's location ID, which is the globally unique identifier (GUID) of the `kube-system` namespace. After AMR Observer retrieves the location ID, AMR Observer emits a CloudEvent to AMR CloudEvent Handler, including any operator-defined metadata. This CloudEvent registers the location, and subsequent CloudEvents from that cluster use the same location reference in the source field. This mechanism helps AMR track artifacts with their associated location.

Watched resources

AMR Observer consists of managed controllers that watch resources. In Tanzu Application Platform v1.9, AMR Observer watches for:

AMR Observer Supported Resources	**-**
ImageVulnerabilityScans	app-scanning.apps.tanzu.vmware.com/v1alpha1
Pods	v1
GitRepository	source.toolkit.fluxcd.io/v1
OCIRepository	source.toolkit.fluxcd.io/v1beta2
ImageRepository	source.apps.tanzu.vmware.com/v1alpha1
MavenArtifacts	source.apps.tanzu.vmware.com/v1alpha1
Build	kpack.io/v1alpha2
TaskRun	tekton.dev/v1beta1

ImageVulnerabilityScans

AMR Observer watches the `ImageVulnerabilityScan` Custom Resources for completed scans. When a scan finishes, AMR Observer uses the `registry secret` and the location information from the `ImageVulnerabilityScan` Custom Resources to fetch the Software Bill of Materials (SBOM) report. After obtaining the SBOM report, AMR Observer wraps it in a CloudEvent and emits it to the AMR CloudEvent Handler. The AMR CloudEvent Handler persists this event in the Metadata Store.

Configure Artifact Metadata Repository

This topic tells you how to configure Artifact Metadata Repository (AMR).

AMR Observer

You can obtain the Tanzu Application Platform values schema by running:

```
tanzu package available get amr-observer.apps.tanzu.vmware.com/${VERSION} --values-sch
ema --namespace tap-install
```

The following example is an AMR Observer configuration, located under the `amr` key in the Tanzu Application Platform values file:

```
amr:
  observer:
    location: |
```

```
      labels:
      - key: env
        value: prod
    resync_period: "10h"
    ca_cert_data: |
      -----BEGIN CERTIFICATE-----
      Custom CA certificate for AMR CloudEvent Handler's HTTPProxy with custom TLS cer
ts
      -----END CERTIFICATE-----
    cloudevent_handler:
      endpoint: "https://amr-cloudevent-handler.DOMAIN"
      liveness_period_seconds: 10
    auth:
      kubernetes_service_accounts:
        enable: true
        autoconfigure: true
        secret:
          ref: "amr-observer-edit-token"
          value: ""
    max_concurrent_reconciles:
      image_vulnerability_scans: 1
```

Where `DOMAIN` is the domain you want to target.

Configuration options:

- `amr.observer.location`
    - Default: ""
    - Location is the multiline string configuration for the location content.
    - The YAML string can contain a single field:
    - `labels`: Consists of an array for key and value pairing. Useful for adding searchable and identifiable metadata. For enabling DORA functionality, it is important to have a label named `env`. For more information, see DORA metrics in Tanzu Developer Portal.
- `amr.observer.resync_period`
    - Default: “10h”
    - `resync_period` decides the minimum frequency at which watched resources reconcile. A lower period corrects entropy more quickly, but reduce responsiveness to change if there are many watched resources. Change this value only if you know what you are doing. Defaults to 10 hours if unset.
- `amr.observer.ca_cert_data` or `shared.ca_cert_data`
    - Default: ""
    - `ca_cert_data` adds certificates to the truststore that amr-observer uses.

    ```
    kubectl -n metadata-store get secrets/amr-cloudevent-handler-ingress-cert -o js
    onpath='{.data."crt.ca"}' | base64 -d
    ```

- `amr.observer.cloudevent_handler.endpoint`
    - Default: `http://amr-cloudevent-handler.metadata-store.svc.cluster.local:80`
    - The URL of the AMR CloudEvent Handler endpoint.
    - On the view or full Tanzu Application Platform profile cluster, obtain the AMR CloudEvent Handler ingress address to configure this property:

```
kubectl -n metadata-store get httpproxies.projectcontour.io amr-cloudevent-hand
ler-ingress -o jsonpath='{.spec.virtualhost.fqdn}'
```

Note

Ensure that you set the correct protocol. If there is TLS, you must prepend `https://`. If there is no TLS, you must prepend `http://`.

- `amr.observer.cloudevent_handler.liveness_period_seconds`
  - Default: `10`
  - The period in seconds between executed health checks to the AMR CloudEvent Handler endpoint.
- `amr.observer.auth.kubernetes_service_accounts`
  - `.enable`
  - Default: `true`
  - Include an Authorization header when communicating with AMR CloudEvent Handler.
  - `.autoconfigure`
  - Default: `true`
  - Delegate creation of authentication token secret to the artifact metadata repository. Only applicable on Full and View clusters.
  - `.secret`
  - The secret with the access token for communicating with the cloudevent-handler
  - `.ref`
    - Default: ""
    - Secret name which contains the access token.
  - `.value`
    - Default: ""
    - Secret as a plain text string. This allows integrating with Tanzu Mission Control (TMC) secret imports.
- `amr.observer.deployed_through_tmc`
  - Default: `null`
  - Tanzu Application Platform multicluster deployment happens through TMC when you set `deployed_through_tmc` to true.
- `amr.observer.max_concurrent_reconciles`
  - Configure maximum concurrent reconciles for controllers.
  - `.image_vulnerability_scans`
  - Default: `1`
  - Maximum concurrent reconciles for observing ImageVulnerabilityScans.
- When deploying with TMC, `MultiClusterPropertyCollector` overwrites existing Observer package configuration values. For the workaround, see the known issue.

AMR GraphQL

- `amr.graphql.auth.kubernetes_service_accounts``
  - `.enable`
  - Default: true
  - Enable authentication for artifact metadata repository GraphQL server. By default it is set to true.
  - `.autoconfigure`
  - Default: `true`
  - Delegate creation of authentication token secret to the artifact metadata repository. By default it is set to true.

AMR CloudEvent Handler

- `amr.cloudevent_handler.auth.kubernetes_service_accounts`
  - `.enable`
  - Default: true
  - Enable authentication and authorization for services accessing Artifact Metadata Repository.
  - `.autoconfigure`
  - Default: `true`
  - Delegate creation of authentication token secret to the artifact metadata repository. By default it is set to true.

Authentication and authorization

This topic tells you how to authenticate and authorize the Artifact Metadata Repository (AMR).

Overview

The following are included in authentication and authorization:

- High-level design
- Kubernetes service account automatic configuration
- User-defined kubernetes service account configuration
- Cloudevent user-defined service account
- Configuring AMR Observer with the CloudEvent Handler service account access token

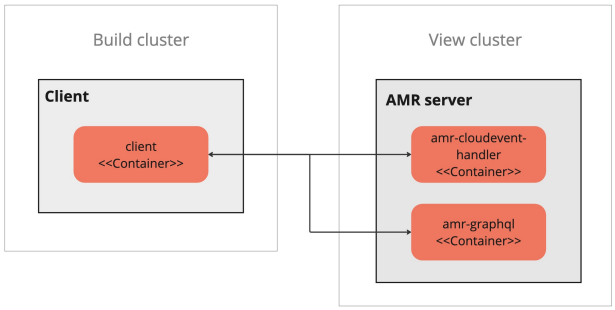
High-level design

The Artifact Metadata Repository (AMR) deploys the following Kubernetes services which expose http endpoints:

- Cloudevent-handler
- GraphQL

Both Cloudevent-handler and GraphQL are in the same cluster. In a multicluster Tanzu Application Platform deployment, they're in the view cluster and the clients can be from any cluster. This topic

shows the client in the build cluster in our examples.

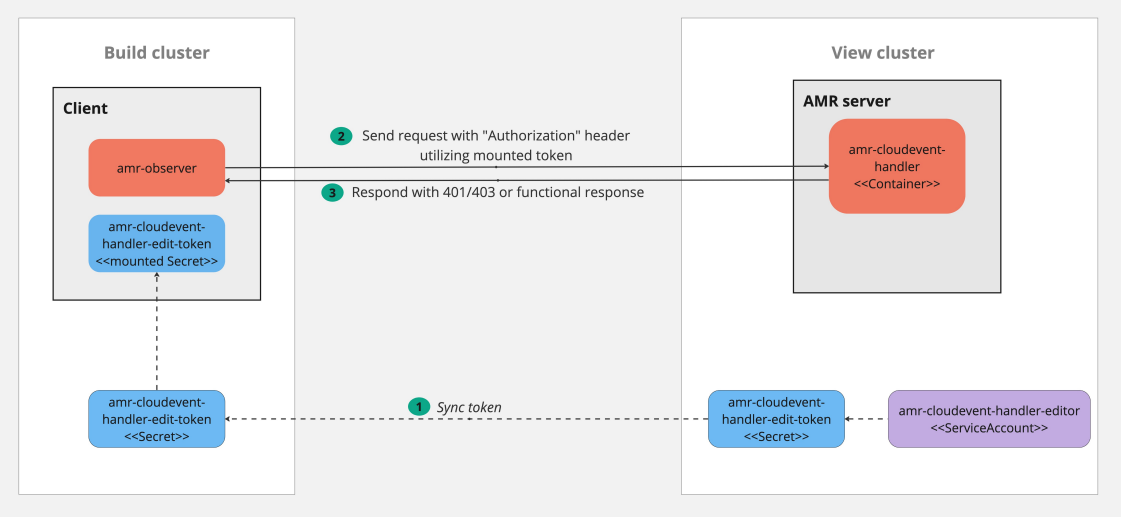


The client sends requests to either service depending on their current task. The cloudevent-handler ingests events from the client and stores it in a database. The GraphQL server answers queries from the client and returns data from the database. Other than those points, the two are treated the same in this design. They both use the same authentication and authorization solution. This topic simplifies the explanation by only showing the cloudevent-handler.

Kubernetes RBAC

The server implements support for authentication using Kubernetes RBAC. This includes requiring the client to send a token from a Kubernetes service account token bound to a Kubernetes role.

1. The administrator creates a service account, role/clusterrole, and role binding in the cluster where the cloudevent-handler is deployed in the View cluster. The role declares what permissions the client has:
    - For the AMR **Observer** the supported permissions are `update`, resource `*`, group `cloudevents.amr.apps.tanzu.vmware.com`. No resourceNames are supported. That translates to “write for all resources” for the CloudEvents API.
    - For **GraphQL** service the supported permissions are `get`, resource `*` and group `graphql.amr.apps.tanzu.vmware.com`. No resourceNames are supported. That translates to “read all” from the GraphQL API.
2. The administrator copies the service account token and puts it in the client cluster where the client container can read it. For example, the client can get the token mounted as a Kubernetes secret.
3. The client sends a request to AMR, and puts the service account token in the http header `Authorization: Bearer <token>`.
4. The cloudevent-handler reads the token and conduct a **TokenReview**.
5. The cloudevent-handler does a **SubjectAccessReview** using the userinfo returned from TokenReview and the resource information as described in #1.
6. The SubjectAccessReview looks, according to the Kubernetes RBAC system, for any Role/ClusterRole associations by using bindings to find a match between the assigned roles to the specific service account and the requested resource information.



Kubernetes service account automatic configuration

This topic tells you about the resources that you create for Kubernetes service account automatic configuration for AMR authentication and authorization.

Overview

This topic describes which resources play a role in the default configuration and how to troubleshoot. For more information about the resources, for example, if you use means to manage your service accounts or have other requirements related to roles and bindings, see User-defined kubernetes service account configuration

The package-level configuration has a component top level key (TLK) prefix and this topic describes how Tanzu Application Platform configurations can influence this prefix.

Observer

Observer configuration in the Tanzu Application Platform context has the prefix TLK `amr.observer`. For authentication and authorization, the Tanzu Application Platform profiles influence automatic configuration. Observer can only automatically configure itself when co-located with the CloudEvent Handler, which is in the `full` or `view` profile. If this is not the case `auth.kubernetes_service_accounts.autoconfigure` is set to false at installation.

If `auth.kubernetes_service_accounts.enable` and `auth.kubernetes_service_accounts.autoconfigure` are true, the observer package creates the following resources to set up authentication automatically in the `amr-observer-system` namespace:

- a `ServiceAccount` named `amr-observer-editor` that observer uses to send requests to the CloudEvent Handler
- a `Secret` named `amr-observer-edit-token` of type `kubernetes.io/service-account-token` which generates a long-lived token for the service account
- a `ClusterRole` named `tanzu:amr:observer:edit` defining the necessary `update` permissions for all resources in `cloudevents.amr.apps.tanzu.vmware.com`
- a `ClusterRoleBinding` named `tanzu:amr:observer:editor` binding the defined role to the service account

If `auth.kubernetes_service_accounts.autoconfigure` is set to false, you must configure the observer package with all the above resources manually. For information about how to set up the observer, see User-defined Kubernetes service account configuration.

CloudEvent Handler

You can find the CloudEvent Handler configuration in the Tanzu Application Platform context under the TLK `amr.cloudevent_handler`. This prefix is not stripped in this case.

On the package level, if `amr.cloudevent_handler.auth.kubernetes_service_accounts.enable` and `amr.cloudevent_handler.auth.kubernetes_service_accounts.autoconfigure` are true, the package creates the following resources to set up authentication automatically in the `metadata-store` namespace:

- a `ServiceAccount` named `amr-cloudevent-handler-editor` that clients use to send requests to the CloudEvent Handler
- a `Secret` named `amr-cloudevent-handler-edit-token` of type `kubernetes.io/service-account-token` which generates a long-lived token for the service account
- a `ClusterRole` named `tanzu:amr:cloudevent-handler:edit` defining the necessary `update` permissions for all resources in `cloudevents.amr.apps.tanzu.vmware.com`
- a `ClusterRoleBinding` named `tanzu:amr:cloudevent-handler:editor` binding the defined role to the service account

GraphQL handler

You can find the GraphQL configuration in the Tanzu Application Platform context under the TLK `amr.graphql`. This prefix is not stripped in this case.

If `amr.graphql.auth.kubernetes_service_accounts.enable` and `amr.graphql.auth.kubernetes_service_accounts.autoconfigure` are true, the package creates the following resources to set up authentication automatically in the `metadata-store` namespace:

- a `ServiceAccount` named `amr-graphql-viewer` that clients use to send requests to the graphql interface
- a `Secret` named `amr-graphql-view-token` of type `kubernetes.io/service-account-token` which generates a long-lived token for the service account
- a `ClusterRole` named `tanzu:amr:graphql:view` defining the necessary `get` permissions for all resources in `graphql.amr.apps.tanzu.vmware.com`
- a `ClusterRoleBinding` named `tanzu:amr:graphql:viewer` binding the defined role to the service account

User-defined Kubernetes service account configuration

This topic tells you how to configure a user-defined Kubernetes service account.

Overview

To configure a Kubernetes service account authentication for AMR Observer, CloudEvent Handler or GraphQL clients must have the required permissions, resources, and groups in the following sections.

Clients to CloudEvent Handler

Clients to the CloudEvent Handler, such as Observer, must have the permission `update`, resource `*`, and group `cloudevents.amr.apps.tanzu.vmware.com`. `resourceNames` are not supported. That translates to write for all resources for the CloudEvents API.

For example:

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: tanzu:amr:observer:editor
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: tanzu:amr:observer:edit
subjects:
  - kind: ServiceAccount
    name: amr-observer-editor
    namespace: amr-observer-system

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: tanzu:amr:observer:edit
rules:
  - apiGroups: ["cloudevents.amr.apps.tanzu.vmware.com"]
    resources: ["*"]
    verbs: ["update"]

---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: amr-observer-editor
  namespace: amr-observer-system
  annotations:
automountServiceAccountToken: false

---
apiVersion: v1
kind: Secret
type: kubernetes.io/service-account-token
metadata:
  name: amr-observer-editor-token
  namespace: amr-observer-system
  annotations:
    kubernetes.io/service-account.name: amr-observer
```

If you saved this to a `observer-rbac.yaml` file, run `kubectl apply -f observer-rbac.yaml` to set everything up. If you prefer short lived service account tokens, remove the secret from the file beforehand. After creating the resources, run `kubectl create token amr-observer-editor-token -n amr-observer-system` to create a token.

To configure custom service accounts even if automatic configuration is on, you must edit the resource naming.

Clients to GraphQL

Clients to the AMR GraphQL interface must have the permission `get`, resource `*`, and group `graphql.amr.apps.tanzu.vmware.com`. `resourceNames` are not supported. That translates to “get for all resources for the GraphQL API”.

For example:

```
---
apiVersion: rbac.authorization.k8s.io/v1
```

```
kind: ClusterRoleBinding
metadata:
  name: tanzu:amr:graphql:viewer
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: tanzu:amr:graphql:view
subjects:
  - kind: ServiceAccount
    name: amr-graphql-viewer
    namespace: metadata-store
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: tanzu:amr:graphql:view
rules:
  - apiGroups: ["graphql.amr.apps.tanzu.vmware.com"]
    resources: ["*"]
    verbs: ["get"]
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: amr-graphql-viewer
  namespace: metadata-store
  annotations:
    kapp.k14s.io/change-group: amr-graphql-viewer.metadata-store.apps.tanzu.vmware.co
m/service-account
automountServiceAccountToken: false
---
apiVersion: v1
kind: Secret
type: kubernetes.io/service-account-token
metadata:
  name: amr-graphql-view-token
  namespace: metadata-store
  annotations:
    kubernetes.io/service-account.name: amr-graphql-viewer
```

If you saved this to a `graphql-client-rbac.yaml` file, run `kubectl apply -f graphql-client-rbac.yaml` to set everything up. If you prefer short lived service account tokens, remove the secret from the file beforehand. After creating the resources, run `kubectl create token amr-graphql-view-token -n metadata-store` to create a token.

To configure custom service accounts even if automatic configuration is on, you must edit the resource naming.

Run query with GraphQL

This topic tells you how to connect to the GraphQL playground and how to run some queries.

Connect to AMR GraphQL

There are two ways you can perform GraphQL queries:

- Use the GraphQL playground
- Use curl

Use the GraphQL playground
Complete the following steps to perform GraphQL queries:

1. Enable ingress. VMware recommends enabling ingress. The Supply Chain Security Tools for Tanzu – Store and Artifact Metadata Repository (AMR) packages share the same ingress configuration. For information about enabling ingress, see Ingress support for Supply Chain Security Tools - Store.
2. Retrieve the access token. Run:

   ```
   kubectl -n metadata-store get secret amr-graphql-view-token -o json | jq -r
   ".data.token" | base64 -d
   ```

3. To connect to the AMR GraphQL playground, visit `https://amr-graphql.INGRESS-DOMAIN/play`.

   Where `INGRESS-DOMAIN` is the domain of the ingress you want to use.

4. In the **Headers** tab at the bottom of the query window, add a JSON block containing the following authentication header:

   ```
   {
     "Authorization": "Bearer ACCESS-TOKEN"
   }
   ```

   Where `ACCESS-TOKEN` is the AMR GraphQL access token.

   You can use this to write and execute your own GraphQL queries to fetch data from the AMR.

Use curl

Write and execute GraphQL queries to fetch data from the AMR. This procedure uses curl to query the AMR GraphQL endpoint, but you can use other similar tools to access the endpoint:

1. Enable ingress. VMware recommends enabling ingress. The Supply Chain Security Tools for Tanzu – Store and Artifact Metadata Repository (AMR) packages share the same ingress configuration. For information about enabling ingress, see Ingress support for Supply Chain Security Tools - Store.
2. To connect to the AMR GraphQL by using curl after you enable ingress, you first need the AMR GraphQL access token and its CA certificate. Run:

   ```
   kubectl get secret ingress-cert -n metadata-store -o json | jq -r '.data."ca.
   crt"' | base64 -d > /tmp/graphql-ca.crt
   ```

3. After the token and certificate are retrieved, use curl to perform GraphQL queries. Run:

   ```
   curl "https://amr-graphql.INGRESS-DOMAIN/query" \
     --cacert FILE-LOCATION \
     -H "Authorization: Bearer ACCESS-TOKEN" \
     -H 'accept: application/json' \
     -H 'content-type: application/json' \
     --data-raw '{"query":"query getAppAcceleratorRuns { appAcceleratorRuns(firs
   t: 250){ nodes { guid name namespace timestamp } pageInfo{ endCursor hasNextP
   age } } }"}' | jq .
   ```

   Where:

   - `INGRESS-DOMAIN` is the domain of the ingress you want to use.
   - `ACCESS-TOKEN` is the AMR GraphQL access token
   - `FILE-LOCATION` is the file containing the AMR GraphQL CA certificate, for example, `/tmp/graphql-ca.crt`

Query for AppAcceleratorRuns (alpha)

This section tells you about GraphQL query arguments, and lists the fields available for `AppAcceleratorRuns` and `AppAcceleratorFragments`.

AppAcceleratorRuns query arguments

You can specify the following arguments when querying for `AppAcceleratorRuns`. If you don't specify an argument the query will return all AppAcceleratorRuns. `query` expects an object that specifies additional arguments to query.

Argument	Description	Example
`guid`	String value unique identifier for each AppAcceleratorRuns.	`appAcceleratorRuns(query:{guid: "d2934b09-5d4c-45da-8eb1-e464f218454e"})`
`source`	String representing the client used to run the accelerator. Supported values include `TAP-GUI`, `VSCODE`, and `INTELLIJ`.	`appAcceleratorRuns(query:{source: "TAP-GUI"})`
`username`	String representing the user name of the person who runs the accelerator, as captured by the client UI.	`appAcceleratorRuns(query:{username: "test.user"})`
`namespace` and `name`	Strings representing the accelerator that was used to create an application.	`appAcceleratorRuns(query:{name: "tanzu-java-web-app"})`
`appAcceleratorRepoURL`, `appAcceleratorRevision`, and `appAcceleratorSubpath`	Location in VCS (Version Control System) of the accelerator sources used.	`appAcceleratorRuns(query: {appAcceleratorRepoURL: "https://github.com/vmware-tanzu/application-accelerator-samples.git", appAcceleratorRevision: "v1.6"`
`timestamp`	String representation of the time the accelerator ran. You can query for runs that happened `before` or `after` a particular instant.	`appAcceleratorRuns(query: {timestamp: {after: "2023-10-11T13:40:46.952Z"}})`

AppAcceleratorRuns fields

You can choose the following fields to return in the GraphQL query. You must specify at least one field.

- `guid`: String value unique identifier for each AppAcceleratorRuns.
- `source`: String representing the client used to run the accelerator.
- `username`: String representing the user name of the person who ran the accelerator.
- `namespace` and `name`: Strings representing the accelerator that was used to create an application.
- `appAcceleratorRepoURL`, `appAcceleratorRevision`, and `appAcceleratorSubpath`: Location in VCS of the accelerator sources used.
- `timestamp`: String representation of the time the accelerator ran.
- `appAcceleratorSource`: VCS information of the sources of the accelerator used, but navigable as a commit.
- `appAcceleratorFragments`: A one-to-many container of nodes representing the fragment versions used in each `AppAcceleratorRuns`. Fragment nodes share many of the fields with

`AppAcceleratorRuns`, with the same semantics but applied to the particular fragment. These include:

- `namespace` and `name`: Strings representing the identity of the fragment.
- `appAcceleratorFragmentSourceRepoURL` , `appAcceleratorFragmentSourceRevision`, and `appAcceleratorFragmentSourceSubpath`: Location in VCS of the sources of the fragment used
- `appAcceleratorFragmentSource`: VCS information of the sources of the fragment, but navigable as a commit.

Sample Application Accelerator queries

- Get the list of all Application Accelerator runs, with the fragments used for each.

```
query getAllAcceleratorRuns {
  appAcceleratorRuns {
    nodes {
      name
      appAcceleratorFragments {
        nodes {
          name
        }
      }
    }
  }
}
```

Install Artifact Metadata Repository CloudEvent Handler

This topic tells you how to install the Artifact Metadata Repository (AMR) CloudEvent Handler.

Switching Context

If AMR CloudEvent Handler is installed on a separate cluster, such as with a view profile cluster, it is important that you target the correct cluster when updating. Ensure that the correct cluster is targeted before updating package values.

```
# 1. Switch context to view profile cluster
kubectl config use-context VIEW-CLUSTER-NAME

# 2. Update the tap-values.yaml in an editor according to the desired configuration

# 3. Update the installed TAP package on the cluster
tanzu package installed update tap --values-file tap-values.yaml -n tap-install
```

Where `VIEW-CLUSTER-NAME` is the name of the view profile cluster you want to use.

Install

The AMR CloudEvent Handler is installed by default in the Tanzu Application Platform Full and View profiles.

Artifact Metadata Repository Observer for Supply Chain Security Tools - Store

This topic tells you how to install Artifact Metadata Repository (AMR) Observer for Supply Chain Security Tools (SCST) - Store.

Prerequisites

The AMR Observer is deployed by default on the Tanzu Application Platform Full, Build and Run profiles.

Switching Context

If Artifact Metadata Repository Observer is installed on a separate cluster from AMR CloudEvent Handler, you must ensure that the correct cluster is targeted before updating package values.

```
# 1. Switch context to cluster with AMR Observer
kubectl config use-context OBSERVER-CLUSTER-NAME

# 2. Update the tap-values.yaml in an editor according to the desired configuration

# 3. Update the installed TAP package on the cluster
tanzu package installed update tap --values-file tap-values.yaml -n tap-install
```

Where `OBSERVER-CLUSTER-NAME` is the name of the cluster you want to use.

Configuring AMR Observer in a multicluster deployment

When you install AMR Observer on a different cluster from the AMR CloudEvent Handler, the following values are required:

- `amr.observer.cloudevent_handler.endpoint` is required for the Observer to send to the AMR CloudEvent Handler.
- `amr.observer.cloudevent_handler.ca_cert_data` or `shared.ca_cert_data` are required for AMR CloudEvent Handlers that use Custom CA certificates to generate the associated TLS certificate for the ingress endpoint.

Note

If SCST - Scan 2.0 is installed after AMR Observer has already been deployed, a deployment you must restart AMR Observer to observe the new ImageVulnerabilityScan Custom Resource that was installed with SCST - Scan 2.0.

```
kubectl -n amr-observer-system rollout restart deployment amr-observer-controller-mana
ger
```

The following log appears if the AMR Observer is observing the ImageVulnerabilityScan Custom Resource:

```
2023-06-28T17:56:43Z  INFO  Starting Controller  {"controller": "imagevulnerabilitysca
n", "controllerGroup": "app-scanning.apps.tanzu.vmware.com", "controllerKind": "ImageV
ulnerabilityScan"}
```

For information about logging, see Troubleshoot - AMR Observer Logs.

See Configuration - AMR Observer.

Installing Artifact Metadata Repository Observer Standalone

1. To install AMR Observer standalone from a Tanzu Application Platform profile, verify the available version:

```
$ tanzu package available list amr-observer.apps.tanzu.vmware.com -n tap-instal
l

NAME                                VERSION        RELEASED-AT
amr-observer.apps.tanzu.vmware.com  0.2.0          2023-10-05 16:17:22 -0400 ED
T
```

2. Look at the package values-schema to create the values file. For more information, see Configuration.

```
$ tanzu package available get amr-observer.apps.tanzu.vmware.com/0.2.0 --values
-schema --namespace tap-install

KEY                                                                    DEFAULT
TYPE     DESCRIPTION
deployed_through_tmc                                                   false
boolean  TAP Multi Cluster deployment will happen through Tanzu Mission Control
when `deployed_through_tmc` is set to true
observer.auth.kubernetes_service_accounts.autoconfigure                true
boolean  Delegate creation of auth token secret to AMR Observer. By default it
is set to true.
observer.auth.kubernetes_service_accounts.enable                       true
boolean  Include Authorization header when communicating with AMR CloudEvent Ha
ndler.
observer.auth.kubernetes_service_accounts.secret.ref                   amr-obse
rver-edit-token                                                string   Secret
name which contains the access token.
observer.auth.kubernetes_service_accounts.secret.value                 ""
string   Secret as a plain text string. This allows integrating with Tanzu Miss
ion Control secret imports.
observer.ca_cert_data                                                  ""
string   ca_cert_data is used to add certificates to the truststore that is use
d by the amr-observer.
observer.cloudevent_handler.endpoint                                   "http://
amr-cloudevent-handler.metadata-store.svc.cluster.local:80"    string   The URL
of the CloudEvent Handler endpoint.
observer.cloudevent_handler.liveness_period_seconds                    10
integer  The period in seconds between executed health checks to the CloudEvent
Handler endpoint.
observer.location                                                      ""
string   location is the multiline string configuration for the location.conf c
ontent.
observer.max_concurrent_reconciles.image_vulnerability_scans           1
integer  Max concurrent reconciles for observing ImageVulnerabilityScans.
observer.resync_period                                                 10h
string   resync_period determines the minimum frequency at which watched resour
ces are reconciled.

A lower period will correct entropy more quickly, but reduce

responsiveness to change if there are many watched resources. Change this value
only if you

know what you are doing. Defaults to 10 hours if unset.
```

3. Install the package using `tanzu package install`.

Troubleshooting Artifact Metadata Repository (AMR)

This topic tells you how to troubleshoot issues with Artifact Metadata Repository (AMR).

Debug AMR

1. Pause reconciliation of the package

```
tanzu package installed pause amr-observer -n tap-install
# OR
kctrl package installed pause -i amr-observer -n tap-install
```

2. Change the zap log level.

```
kubectl -n amr-observer-system \
patch deployment amr-observer-controller-manager \
  --type='json' \
  -p='[{"op": "add",
        "path": "/spec/template/spec/containers/1/args/-",
        "value": "--zap-log-level=3"
      }]'
```

Logs now show `LEVEL(-3)` with the example patch above.

```
$ kubectl -n amr-observer-system logs deployments/amr-observer-controller-manag
er
2023-06-20T15:42:39Z  LEVEL(-3)  httpclient.circuitbreaker  AMR CloudEventHandl
er  {"availability": true, "State": "closed"}
```

3. Unpause reconciliation of the package:

```
tanzu package installed kick amr-observer -n tap-install
# OR
kctrl package installed kick -i amr-observer -n tap-install
```

Health Check

AMR Observer does not send events to AMR CloudEvent Handler if either the AMRCloudEvent Handler, AMR, or MDS isn't working.

Sample log in AMR Observer when HealthCheck is working:

```
2023-06-20T15:51:50Z  INFO  httpclient.circuitbreaker  Received response with status
{"status": "204 No Content"}
2023-06-20T15:51:50Z  INFO  httpclient.circuitbreaker  Successfully sent CloudEvent
```

Sample log in AMR Observer when HealthCheck is failing:

```
2023-06-20T19:01:58Z  INFO  httpclient.circuitbreaker  Received response with status
{"status": "503 Service Unavailable"}
2023-06-20T19:01:58Z  ERROR  httpclient    {"error": "request failed with status 503"}
gitlab.eng.vmware.com/tanzu-image-signing/amr-observer/internal/cloud/event/client.(*C
loudEventClient).logAndReturnError
  /workspace/internal/cloud/event/client/client.go:53
gitlab.eng.vmware.com/tanzu-image-signing/amr-observer/internal/cloud/event/client.(*C
loudEventClient).do
  /workspace/internal/cloud/event/client/client.go:102
gitlab.eng.vmware.com/tanzu-image-signing/amr-observer/internal/cloud/event/client.(*C
loudEventClient).checkCloudEventHandlerAvailability.func1
  /workspace/internal/cloud/event/client/client.go:123
github.com/sony/gobreaker.(*CircuitBreaker).Execute
  /go/pkg/mod/github.com/sony/gobreaker@v0.5.0/gobreaker.go:242
gitlab.eng.vmware.com/tanzu-image-signing/amr-observer/internal/cloud/event/client.(*C
loudEventClient).checkCloudEventHandlerAvailability
  /workspace/internal/cloud/event/client/client.go:117
```

```
gitlab.eng.vmware.com/tanzu-image-signing/amr-observer/internal/cloud/event/client.(*C
loudEventClient).CheckAvailabilityPeriodically
  /workspace/internal/cloud/event/client/client.go:111
```

Sample log in AMR CloudEvent Handler:

```
{"level":"error","ts":"2023-06-20T15:15:04.321672436Z","caller":"amr-persister/main.g
o:99","msg":"AMR is unavailable: Get \"https://amr-graphql-app.metadata-store.svc.clus
ter.local:8443/play\": dial tcp 10.28.116.184:8443: connect: connection refused","stac
ktrace":"<...>"}
```

```
{"level":"error","ts":"2023-06-20T20:13:15.689628865Z","caller":"amr-persister/main.g
o:102","msg":"MDS is unavailable: Get \"https://metadata-store-app.metadata-store.svc.
cluster.local:8443/api/health\": dial tcp 10.28.122.145:8443: connect: connection refu
sed","stacktrace":"..."}
```

Unsupported protocol is used for the `amr.observer.cloudevent_handler.endpoint`.

```
2023-06-28T18:48:31Z  ERROR  httpclient  error sending request to AMR CloudEvent Handl
er  {"error": "Get \"amr-persister.example.com/healthz\": unsupported protocol scheme
\"\""}

...

2023-06-28T18:48:31Z  ERROR  setup  unable to registry location  {"error": "failed to
send, Post \"amr-persister.example.com\": unsupported protocol scheme \"\""}

...

2023-06-28T18:48:31Z  ERROR  httpclient.circuitbreaker  error contacting event handler
{"error": "Get \"amr-persister.example.com/healthz\": unsupported protocol scheme
\"\""}
```

To fix this, use the appropriate `https://` or `http://` prepended protocol.

If the log contains:

```
2023/06/28 19:09:06 No certs appended, using system certs only
2023-06-28T19:09:06Z  INFO  controller-runtime.metrics  Metrics server is starting to
listen  {"addr": "127.0.0.1:8080"}
2023-06-28T19:09:06Z  INFO  setup  Establishing  {"locationId": "d9fa1ee9-ba42-4262-aa
f6-4128f99096b9"}
```

And if the following describe on the pod shows:

```
$ kubectl -n amr-observer-system describe pods "<amr-observer-controller-manager-...>"
```

Sample output:

```
...

Events:
  Type     Reason     Age                    From               Message
  ----     ------     ----                   ----               -------
...

  Warning  Unhealthy  3m35s (x3 over 4m15s)  kubelet            Liveness probe failed:
Get "http://192.168.45.78:8081/healthz": context deadline exceeded (Client.Timeout exc
eeded while awaiting headers)

...
```

```
  Warning  Unhealthy  3m5s (x10 over 4m25s)  kubelet            Readiness probe faile
d: Get "http://192.168.45.78:8081/readyz": context deadline exceeded (Client.Timeout e
xceeded while awaiting headers)
```

This is a symptom of a wrong protocol for `amr.observer.cloudevent_handler.endpoint`. The fix is to use the appropriate `https://` or `http://` prepended protocol depending on the TLS configuration.

AMR Observer Logs

```
kubectl -n amr-observer-system logs deployments/amr-observer-controller-manager
```

The AMR Observer is not observing `ImageVulnerabilityScan` CRD.

```
2023-06-20T15:47:09Z  INFO  ivs.SetupWithManager  Not registering ImageVulnerabilitySc
ans Controller: customresourcedefinitions.apiextensions.k8s.io "imagevulnerabilityscan
s.app-scanning.apps.tanzu.vmware.com" not found"
```

The AMR Observer is observing `ImageVulnerabilityScan` CRD if it is installed.

```
2023-06-28T17:56:43Z  INFO  Starting Controller  {"controller": "imagevulnerabilitysca
n", "controllerGroup": "app-scanning.apps.tanzu.vmware.com", "controllerKind": "ImageV
ulnerabilityScan"}
```

When Observer first starts up, it registers a Location to the Metadata Store. See Configure Artifact Metadata Repository.

```
2023-06-20T15:47:09Z  INFO  setup  Establishing  {"locationId": "f17f073d-dfec-4624-95
3e-a133b694ecad"}
sent: type: vmware.tanzu.apps.location.created.v1
  source: f17f073d-dfec-4624-953e-a133b694ecad
  id: 1667244096281455275
Extensions:
map[]
```

Information about what CA certificates were added to the Observer's HTTP client.

```
2023-06-20T15:47:05Z  INFO  setup  No additional certs read from configured path, cont
inuing with system truststore"  {"path": "/truststore/ca.crt"}
```

No valid CA certificates were found.

```
2023/06/28 18:41:04 No certs appended, using system certs only
```

When the ReplicaSet is observed to be deleted, it attempts to send the result to AMR CloudEvent Handler. There is a known minor issue that even non workload replicaSets are sent during create and delete events. The log shows an error, however, it is a no-op because AMR CloudEvent Handler filters out ReplicaSet CloudEvents without a workload container.

```
result: 204: sent: type: dev.knative.apiserver.resource.delete
  source: f17f073d-dfec-4624-953e-a133b694ecad
2023-06-20T19:01:23Z  INFO  replicaset.sendCloudEvent  received result  {"ceType": "de
v.knative.apiserver.resource.delete", "result": "500: "}
  id: 4565357219078868493
Extensions:
map[kind:ReplicaSet name:amr-graphql-app-7ff9bc58f namespace:metadata-store]
```

There is partial legacy support for Knative ApiServerSource CloudEvents which the Observer leverages which is why there are CloudEvents formatted similarly to Knative APIServerSource being sent.

AMR CloudEvent Handler Logs

```
kubectl -n metadata-store logs deployments/amr-persister
```

When an event is received, there are log entries for an event being handled:

```
{"level":"debug","ts":"2023-06-20T15:47:09.203079201Z","caller":"persister/persisterad
apter.go:76","msg":"Handle single event"}
{"level":"debug","ts":"2023-06-20T15:47:09.203160964Z","caller":"persister/persisterad
apter.go:108","msg":"Handle Event"}
```

When the Observer starts up, it sends the Location CloudEvent and the AMR CloudEvent Handler sends the Location information to the Metadata Store. The log entries contain where the request is sent to, the JSON payload, and the response body returned from the Metadata Store:

```
{"level":"debug","ts":"2023-06-20T15:47:09.203390152Z","caller":"persister/persisterad
apter.go:117","msg":"Registry Location Event"}
{"level":"info","ts":"2023-06-20T15:47:09.20349837Z","caller":"persister/persisteradap
ter.go:279","msg":"Sending request to: https://amr-graphql-app.metadata-store.svc.clus
ter.local:8443/api/v1/locations with payload: {\"alias\":\"f17f073d-dfec-4624-953e-a13
3b694ecad\",\"reference\":\"f17f073d-dfec-4624-953e-a133b694ecad\"}"}
{"level":"info","ts":"2023-06-20T15:47:09.238033374Z","caller":"persister/persisterada
pter.go:299","msg":"status: 200, responseBody: {\"alias\":\"f17f073d-dfec-4624-953e-a1
33b694ecad\",\"reference\":\"f17f073d-dfec-4624-953e-a133b694ecad\",\"labels\":null}
\n"}
```

When there is an error sending Location to the Metadata Store:

```
{"level":"debug","ts":"2023-06-20T15:15:04.294883639Z","caller":"persister/persisterad
apter.go:117","msg":"Registry Location Event"}
{"level":"info","ts":"2023-06-20T15:15:04.295528237Z","caller":"persister/persisterada
pter.go:279","msg":"Sending request to: https://amr-graphql-app.metadata-store.svc.clu
ster.local:8443/api/v1/locations with payload: {\"alias\":\"f17f073d-dfec-4624-953e-a1
33b694ecad\",\"reference\":\"f17f073d-dfec-4624-953e-a133b694ecad\"}"}
{"level":"error","ts":"2023-06-20T15:15:04.304782986Z","caller":"persister/persisterad
apter.go:283","msg":"post error: Post \"https://amr-graphql-app.metadata-store.svc.clu
ster.local:8443/api/v1/locations\": dial tcp 10.28.116.184:8443: connect: connection r
efused","stacktrace":"<...>"}
{"level":"error","ts":"2023-06-20T15:15:04.304878714Z","caller":"amr-persister/main.g
o:72","msg":"Error Handler received error Post \"https://amr-graphql-app.metadata-stor
e.svc.cluster.local:8443/api/v1/locations\": dial tcp 10.28.116.184:8443: connect: con
nection refused","stacktrace":"..."}
```

When the AMR CloudEvent Handler is up and ready to receive CloudEvents:

```
{"level":"info","ts":"2023-06-20T15:14:56.308829574Z","caller":"amr-persister/main.go:
61","msg":"Start Receiving"}
```

When the Artifact Metadata Repository is unavailable:

```
{"level":"error","ts":"2023-06-20T19:03:47.006718745Z","caller":"amr-persister/main.g
o:99","msg":"AMR is unavailable: Get \"https://amr-graphql-app.metadata-store.svc.clus
```

```
ter.local:8443/play\": dial tcp 10.28.116.184:8443: connect: connection refused","stac
ktrace":"..."}
```

When the Metadata Store is unavailable.

```
{"level":"error","ts":"2023-06-20T20:13:15.689628865Z","caller":"amr-persister/main.g
o:102","msg":"MDS is unavailable: Get \"https://metadata-store-app.metadata-store.svc.
cluster.local:8443/api/health\": dial tcp 10.28.122.145:8443: connect: connection refu
sed","stacktrace":"..."}
```

The received ReplicaSet CloudEvent did not contain a “workload” container. AMR only supports ReplicaSets generated during a Workload run and contain a workload named container.

```
{"level":"error","ts":"2023-06-20T20:13:11.607332572","caller":"persister/persisterada
pter.go:218","msg":"generate application payload error: unable to find workload contai
ner","stacktrace":"..."}
```

The report already exists. This occurs when you reach the resync period and the controllers are configured to reconcile the ImageVulnerability custom resource. This error is a non-issue if the previous submission of the report was successful.

```
{"level":"info","ts":"2023-06-21T20:54:32.772037121Z","caller":"mds/handle-event.go:10
7","msg":"Error posting image report: Validation failed: {\"message\":\"a report with
uid 'd1625dd5ad94c2c5f83a8de4c3a3c382e4e4774a77c2e71fcf68a0cd3f953f7b' already exists
\"}\n"}
{"level":"error","ts":"2023-06-21T20:54:32.772060231Z","caller":"amr-persister/main.g
o:72","msg":"Error Handler received error Validation failed: {\"message\":\"a report w
ith uid 'd1625dd5ad94c2c5f83a8de4c3a3c382e4e4774a77c2e71fcf68a0cd3f953f7b' already exi
sts\"}\n","stacktrace":"..."}
```

Overview of the Metadata Store

This topic gives you an overview of the Metadata Store.

Overview

The Metadata Store saves software bills of materials (SBoMs) to a database and allows you to query for image, source code, package, and vulnerability relationships. The Metadata Store integrates with Supply Chain Security Tools - Scan to automatically store the resulting source code and image vulnerability reports. The Metadata Store accepts CycloneDX input and outputs in both human-readable and machine-readable formats, including JSON, text, and CycloneDX.

Using the Tanzu Insight CLI plug-in

The Tanzu Insight CLI plug-in is the primary way to view results from the Supply Chain Security Tools (SCST) - Scan of source code and image files. Use it to query by the source code commit, image digest, and CVE identifier to understand security risks.

For information about installing, configuring, and using `tanzu insight`, see Overview of Tanzu Insight plug-in.

Multicluster configuration

You can configure the Metadata Store in a multicluster setup. For more information, see Multicluster setup for Supply Chain Security Tools - Store.

Integrating with Tanzu Developer Portal

By using Supply Chain Choreographer in Tanzu Developer Portal, you can visualize your supply chain. Supply Chain Choreographer uses the Metadata Store to show the packages and vulnerabilities in your source code and images.

Additional documentation

For more information about the API, deployment details and configuration, AWS Relational Database Service (RDS) configuration, other database backup recommendations, and known issues, see Additional documentation.

Overview of the Metadata Store

This topic gives you an overview of the Metadata Store.

Overview

The Metadata Store saves software bills of materials (SBoMs) to a database and allows you to query for image, source code, package, and vulnerability relationships. The Metadata Store integrates with Supply Chain Security Tools - Scan to automatically store the resulting source code and image vulnerability reports. The Metadata Store accepts CycloneDX input and outputs in both human-readable and machine-readable formats, including JSON, text, and CycloneDX.

Using the Tanzu Insight CLI plug-in

The Tanzu Insight CLI plug-in is the primary way to view results from the Supply Chain Security Tools (SCST) - Scan of source code and image files. Use it to query by the source code commit, image digest, and CVE identifier to understand security risks.

For information about installing, configuring, and using `tanzu insight`, see Overview of Tanzu Insight plug-in.

Multicluster configuration

You can configure the Metadata Store in a multicluster setup. For more information, see Multicluster setup for Supply Chain Security Tools - Store.

Integrating with Tanzu Developer Portal

By using Supply Chain Choreographer in Tanzu Developer Portal, you can visualize your supply chain. Supply Chain Choreographer uses the Metadata Store to show the packages and vulnerabilities in your source code and images.

Additional documentation

For more information about the API, deployment details and configuration, AWS Relational Database Service (RDS) configuration, other database backup recommendations, and known issues, see Additional documentation.

Configure your target endpoint and certificate for Supply Chain Security Tools - Store

This topic describes how you can configure your target endpoint and certificate for Supply Chain Security Tools (SCST) - Store.

Overview

The connection to Supply Chain Security Tools - Store requires TLS encryption, and the configuration depends on the kind of installation.

For a production environment, VMware recommends that SCST - Store is installed with ingress enabled. The following instructions help set up the TLS connection, assuming that you deployed with ingress enabled.

Using `Ingress`

When using an Ingress setup, SCST - Store creates a specific TLS Certificate for HTTPS communications under the `metadata-store` namespace.

Set the endpoint host to `metadata-store.INGRESS-DOMAIN`, such as `metadata-store.example.domain.com`. Where `INGRESS-DOMAIN` isthe value of the `ingress_domain` property in your deployment yaml.

Note In a multi-cluster setup, a DNS record is **required** for the domain. The below instructions for single cluster setup do not apply, skip to Set Target section.

Single Cluster setup

In a single-cluster setup, a DNS record is still recommended. However, if no accessible DNS record exists for the domain, edit the `/etc/hosts` file to add a local record:

```
ENVOY_IP=$(kubectl get svc envoy -n tanzu-system-ingress -o jsonpath="{.status.loadBal
ancer.ingress[0].ip}")

# Replace with your domain
METADATA_STORE_DOMAIN="metadata-store.example.domain.com"

# Delete any previously added entry
sudo sed -i '' "/$METADATA_STORE_DOMAIN/d" /etc/hosts

echo "$ENVOY_IP $METADATA_STORE_DOMAIN" | sudo tee -a /etc/hosts > /dev/null
```

Set Target

To get the certificate, run:

```
kubectl get secret tap-ingress-selfsigned-root-ca -n cert-manager -o json | jq -r '.da
ta."ca.crt"' | base64 -d > insight-ca.crt
```

Set the target by running:

```
tanzu insight config set-target https://$METADATA_STORE_DOMAIN --ca-cert insight-ca.cr
t
```

Important

The `tanzu insight config set-target` does not initiate a test connection. Use `tanzu insight health` to test connecting using the configured endpoint and CA

certificate. Neither commands test whether the access token is correct. For that you must use the plug-in to add data and query data.

Next Step

- Configure access token

Additional Resources

For information about deploying SCST - Store **without** Ingress, see:

- Using LoadBalancer
- Using NodePort

Configure your access tokens for Supply Chain Security Tools - Store

This topic describes how to configure your access tokens for Supply Chain Security Tools - Store.

The access token is a `Bearer` token used in the http request header `Authorization`. For example, `Authorization: Bearer eyJhbGciOiJSUzI1NiIsImtpZCI6IjhMV0....`

Service accounts are required to have associated access tokens. Before Kubernetes 1.24, service accounts generated access tokens automatically. Since Kubernetes 1.24, a secret must be applied manually.

By default, Supply Chain Security Tools - Store includes a `read-write` service account installed with an access token generated. This service account is cluster-wide. If you want to create your own service accounts, see Create Service Accounts.

Setting the Access Token

When using the `insight` plug-in, you must set the `METADATA_STORE_ACCESS_TOKEN` environment variable, or use the `--access-token` flag. VMware discourages using the `--access-token` flag as the token appears in your shell history.

The following command retrieves the access token from the default `metadata-store-read-write-client` service account and stores it in `METADATA_STORE_ACCESS_TOKEN`:

```
export METADATA_STORE_ACCESS_TOKEN=$(kubectl get secrets metadata-store-read-write-cli
ent -n metadata-store -o jsonpath="{.data.token}" | base64 -d)
```

Additional Resources

- Retrieve access tokens
- Create service accounts
- Create a service account with a custom cluster role

Security details for Supply Chain Security Tools - Store

This topic describes the security details for Supply Chain Security Tools (SCST) - Store.

Application security

TLS encryption

Supply Chain Security Tools - Store requires TLS connection. If certificates are not provided, the application does not start. It supports TLS v1.2 and TLS v1.3. It does not support TLS 1.0, so a downgrade attack cannot happen. TLS 1.0 is prohibited under Payment Card Industry Data Security Standard (PCI DSS).

Cryptographic algorithms

Elliptic Curve:

```
CurveP521
CurveP384
CurveP256
```

Cipher Suites:

```
TLS_AES_128_GCM_SHA256
TLS_AES_256_GCM_SHA384
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
```

Access controls

SCST - Store uses kube-rbac-proxy as the only entry point to its API. Authentication and Authorization must be completed by using the `kube-rbac-proxy` before its API is accessible.

Authentication

The `kube-rbac-proxy` uses Token Review to verify that the token is valid. `Token Review` is a Kubernetes API to ensure that a trusted vendor issued the access token provided by the user. To issue an access token using Kubernetes, the user can create a Kubernetes Service Account and retrieve the corresponding generated secret for the access token.

To create a service account and use its access token, see the Create Service Account Docs.

Authorization

The `kube-rbac-proxy` uses Subject Access Review to ensure that users access certain operations. `Subject Access Review` is a Kubernetes API that uses Kubernetes RBAC to verify that the user can perform specific actions. See Create Service Account Doc.

There are two supported roles:

- `Read Only` cluster role
- `Read and Write` cluster role

These cluster roles are deployed by default. Additionally, a service account is created and bound to the `Read and Write` cluster role by default. If you do not want this service account, set the `add_default_rw_service_account` property to `false` in the `metadata-store-values.yaml` file durring deployment. See Install SCST - Store.

There is no default service account bound to the `Read Only` cluster role. You must create your service account and cluster role binding to bind to the `Read Only` role.

Important

There is no support for roles with access to only specific types of resources For example, images, packages, and vulnerabilities.

Container security

Non-root user

All containers shipped do not use root user accounts or accounts with root access. Using Kubernetes Security Context ensures that applications do not run with root users.

Security Context for the API server:

```
allowPrivilegeEscalation: false
runAsUser: 65532
fsGroup: 65532
```

Security Context for the PostgreSQL database pod:

```
allowPrivilegeEscalation: false
runAsUser: 999
fsGroup: 999
```

Note

`65532` is the UUID for the nobody user. `999` is the UUID for the PostgreSQL user.

Security scanning

There are two types of security scans that are performed before every release.

Static Application Security Testing (SAST)

A Coverity Scan is run on the source code of the API server, CLI, and all their dependencies. There are no high or critical items outstanding at the time of release.

Software Composition Analysis (SCA)

A Black Duck scan is run on the compiled binary to check for vulnerabilities and license data. There are no high or critical items outstanding at the time of release.

A Grype scan is run against the source code and the compiled container for dependencies vulnerabilities. There are no high or critical items outstanding at the time of release.

Additional documentation for Supply Chain Security Tools - Store

This topic describes additional documentation you can use with Supply Chain Security Tools - Store.

Use and operate

- Multicluster setup
- Developer namespace setup
- Failover, redundancy, and backups

Troubleshooting and logging

- Troubleshooting upgrading
- Log configuration and usage
- Connecting to the Postgres Database

Configuration

- Deployment details and configuration

Access control

- Retrieve access tokens
- Create service accounts
- Create a service account with a custom cluster role

Certificates

- Ingress support
- Using LoadBalancer
- Using NodePort
- Custom certificate configuration
- TLS configuration
- Certificate rotation
- Scanners cluster specific configurations

Database

- Use external postgres database
- AWS RDS postgres configuration
- Database backup recommendations

Other

- Install SCST - Store independent from TAP profiles

Additional documentation for Supply Chain Security Tools - Store

This topic describes additional documentation you can use with Supply Chain Security Tools - Store.

Use and operate

- Multicluster setup
- Developer namespace setup
- Failover, redundancy, and backups

Troubleshooting and logging

- Troubleshooting upgrading
- Log configuration and usage
- Connecting to the Postgres Database

Configuration

- Deployment details and configuration

Access control

- Retrieve access tokens
- Create service accounts
- Create a service account with a custom cluster role

Certificates

- Ingress support
- Using LoadBalancer
- Using NodePort
- Custom certificate configuration
- TLS configuration
- Certificate rotation
- Scanners cluster specific configurations

Database

- Use external postgres database
- AWS RDS postgres configuration
- Database backup recommendations

Other

- Install SCST - Store independent from TAP profiles

Connecting to the Postgres Database

To connect to the Postgres Database, you will need the following values: * database name * username * password * database host * database port * database CA certificate

1. To obtain the database name, username, password, and CA cert, run the following commands:

```
db_name=$(kubectl get secret postgres-db-secret -n metadata-store -o json | jq
-r '.data.POSTGRES_DB' | base64 -d)
db_username=$(kubectl get secret postgres-db-secret -n metadata-store -o json |
jq -r '.data.POSTGRES_USER' | base64 -d)
db_password=$(kubectl get secret postgres-db-secret -n metadata-store -o json |
jq -r '.data.POSTGRES_PASSWORD' | base64 -d)

db_ca_dir=$(mktemp -d -t ca-cert-XXXX)
db_ca_path="$db_ca_dir/ca.crt"
```

```
kubectl get secrets postgres-db-tls-cert -n metadata-store -o json | jq -r '.da
ta."ca.crt"' | base64 -d > $db_ca_path
```

If the password was auto-generated, the above `password` command will return an empty string. Instead, run the following command:

```
db_password=$(kubectl get secret postgres-db-password -n metadata-store -o json
| jq -r '.data.DB_PASSWORD' | base64 -d)
```

2. In a separate terminal, run the following command:

```
kubectl port-forward service/metadata-store-db 5432:5432 -n metadata-store
```

Set the database host and port values on the first terminal with the following:

```
db_host="localhost"
db_port=5432
```

To port forward to a different local port number, use the following command template:

```
kubectl port-forward service/metadata-store-db <LOCAL_PORT>:5432 -n metadata-st
ore
```

With this setup, you can now connect to the database and make queries like the following example:

```
psql "host=$db_host port=$db_port user=$db_username dbname=$db_name sslmode=verify-ca
sslrootcert=$db_ca_path" -c "SELECT * FROM images"
```

You could also use GUI clients such as Postico or DBeaver to interact with the database.

Deployment details and configuration for Supply Chain Security Tools - Store

This topic describes how you can deploy and configure your Kubernetes cluster for Supply Chain Security Tools (SCST) - Store.

What is deployed

The installation creates the following in your Kubernetes cluster:

- Four components:
  - `metadata-store` API back end
  - Database
  - AMR API back end. If AMR is deployed, see Deploying AMR.
  - AMR CloudEvent Handler. If AMR is deployed, see Deploying AMR.
- Services for each of the four components, named:
  - `metadata-store-app`
  - `metadata-store-db`
  - `amr-cloudevent-handler`. If AMR is deployed, see Deploying AMR.
  - `amr-graphql-app`. If AMR is deployed, see Deploying AMR.
- A namespace called `metadata-store`.

- Persistent volume claim `postgres-db-pv-claim` in the `metadata-store` namespace.
- A Kubernetes secret in the namespace `tap` is installed to allow pulling SCST - Store images from a registry.
- Two ClusterRoles:
  - `metadata-store-read-write-client` is bound to a service account by default, giving the service account read and write privileges
  - `metadata-store-read-only` isn't bound to any service accounts, you can bind to it if needed. See Service Accounts.
- (Optional) An HTTPProxy object for ingress support.

Deployment configuration

All configurations are nested inside of `metadata_store` in your Tanzu Application Platform values deployment YAML. AMR specific configurations are nested under `amr` in the `metadata_store` section.

Supported Network Configurations

VMware recommends the following connection methods for Tanzu Application Platform:

- For a single or multicluster configuration with Contour, use `Ingress`.
- For a single cluster without Contour and with `LoadBalancer` support configuration, use `LoadBalancer`.
- For a single cluster without Contour and without `LoadBalancer` configuration, use `NodePort`.
- Multicluster without Contour configuration is not supported.

For a production environment, VMware recommends installing SCST - Store with ingress enabled.

App service type

The Metadata Store's app service type is configured inside the `metadata_store` property of the values file.

```
metadata_store:
  app_service_type: "ClusterIP"
```

Supported values include:

- `LoadBalancer`
- `ClusterIP`
- `NodePort`

The `app_service_type` is set to `ClusterIP` by default.

Ingress support

SCST - Store's values file allows you to enable ingress support and to configure a custom domain name to use Contour to provide external access to SCST - Store's API. These ingress configurations are shared for the metadata store and AMR. Enabling ingress for store enables it for both metadata store and AMR.

For example:

```
metadata_store:
  ingress_enabled: "true"
  ingress_domain: "example.com"
  app_service_type: "ClusterIP" # recommended setting when ingress is enabled
```

An HTTPProxy object is installed with `metadata-store.example.com` as the fully qualified domain name. See Ingress.

Note

The `ingress_enabled` property expects a string value of `"true"` or `"false"`, not a Boolean value.

Database configuration

The default database included with the deployment gets you started using the metadata store. The default database deployment does not support many enterprise production requirements, including scaling, redundancy, or fail over. However, it is a secure deployment.

Using AWS RDS PostgreSQL database

Users can also configure the deployment to use their own RDS database instead of the default. See AWS RDS Postgres Configuration.

Using external PostgreSQL database

Users can configure the deployment to use any other PostgreSQL database. See Use external postgres database.

Custom database password

By default, a database password is generated upon deployment. To configure a custom password, use the `metadata_store.db_password` property in the values file

Important

There is a known issue related to changing database passwords Persistent Volume Retains Data.

To configure a custom database password for AMR:

```
metadata_store:
  db_password: "PASSWORD"
```

Where `PASSWORD` is the same password used for both deployments.

Service accounts

By default, a service account with read-write privileges to the metadata store app is installed. This service account is a cluster-wide account that uses ClusterRole. If you don't want the service account and role, set the `add_default_rw_service_account` property to `"false"`. To create a custom service account, see Create Service Account.

The store creates a read-only cluster role, which is bound to a service account by using `ClusterRoleBinding`. To create service accounts to bind to this cluster role, see Create Service

Account.

Exporting certificates

SCST - Store creates a Secret Export for exporting certificates to `Supply Chain Security Tools - Scan` to securely post scan results. These certificates are exported to the namespace where `Supply Chain Security Tools - Scan` is installed.

Configure your AWS RDS PostgreSQL configuration

This topic describes how you can configure your AWS RDS PostgreSQL configuration for Supply Chain Security Tools (SCST) - Store.

Prerequisites

- AWS Account

Setup certificate and configuration

1. Create an Amazon RDS Postgres using the Amazon RDS Getting Started Guide
2. Once the database instance starts, retrieve the following information:
    1. DB Instance Endpoint
    2. Master Username
    3. Master Password
    4. Database Name

    Note

    If the database name is – in the AWS RDS UI, the value is likely `postgres`.
3. Create a security group to allow inbound connections from the cluster to the Postgres DB
4. Retrieve the corresponding CA Certificate that signed the Postgres TLS Certificate using the following link
5. In the `metadata-store-values.yaml` fill the following settings:

```
db_host: "<DB Instance Endpoint>"
db_user: "<Master Username>"
db_password: "<Master Password>"
db_name: "<Database Name>"
db_port: "5432"
db_sslmode: "verify-full"
db_max_open_conns: 10
db_max_idle_conns: 100
db_conn_max_lifetime: 60
db_ca_certificate: |
  <Corresponding CA Certification>
  ...
  ...
  ...
deploy_internal_db: "false"
```

Note

If `deploy_internal_db` is set to `false`, an instance of Postgres will not be deployed in the cluster.

Use external PostgreSQL database for Supply Chain Security Tools - Store

This topic describes how you configure Tanzu Application Platform to use an external database.

For production deployments, VMware recommends you use an external PostgreSQL database rather than the one packaged with Tanzu Application Platform. This allows you to manage the external database using the best practices and processes that your organization has established.

Prerequisites

Gather the following connection information for the external PostgreSQL database:

- Database Instance Connect Endpoint
- CA Certificate for the database instance TLS certificate (if applicable)
- Database credentials
- Name of the database

If migrating from the internal database to an external database, back up the data before proceeding.

Configure Artifact Metadata Repository (AMR) and Metadata Store (MDS) to use the external database

1. Update your `tap-values.yaml` file:

```
metadata_store:
 db_host: "<DB Instance Endpoint>"
 db_user: "<Master Username>"
 db_password: "<Master Password>"
 db_name: "<Database Name>"
 # SEE NOTE BELOW
 # Disable the internal database deployment
 deploy_internal_db: "false"
 db_port: "5432"
 db_max_open_conns: 10
 db_max_idle_conns: 100
 db_conn_max_lifetime: 60
 # If TLS is enabled on PostgreSQL Instance
 db_sslmode: "verify-full"
 # If TLS Certificate is self-signed
 db_ca_certificate: |
   <Corresponding CA Certification>
   ...
   ...
   ...
```

Note

If you initially deployed Tanzu Application Platform with an internal database and are migrating to an external database, be aware that setting `deploy_internal_db` to `false`, removes the internal instance of PostgreSQL. Back up and migrate your data to the database before setting this value to false or it might cause data loss.

2. Apply the new configuration:

```
tanzu package installed update tap -p tap.tanzu.vmware.com -v 1.9.1  --values-f
ile tap-values.yaml -n tap-install
```

3. (Optional) If you are migrating from the internal database to an external database, the reconciliation might fail. You must manually delete the secret the cert-manager created and cycle the AMR and MDS Services so that the new secret is picked up.

```
kubectl delete secret -n metadata_store postgres-db-tls-cert
kubectl rollout restart -n metadata_store deployment metadata_store-app
```

Validation

Use Bitnami PostgreSQL to verify. For more information, see the Bitnami documentation.

Database backup recommendations for Supply Chain Security Tools - Store

This topic describes database backup recommendations for Supply Chain Security Tools - Store.

By default, the metadata store uses a `PersistentVolume` mounted on a Postgres instance, making it a stateful component of Tanzu Application Platform. VMware recommends implementing a regular backup strategy as part of your disaster recovery plan when using the provided Postgres instance.

Backup

You can use Velero to create regular backups.

Note

Backup support for `PersistentVolume` depends on the used `StorageClass` and existing provider plug-ins. See the officially supported plug-ins here.

```
velero install --provider PROVIDER --bucket BUCKET-NAME --plugins PLUGIN-IMAGE-LOCATIO
N --secret-file SECRET-FILE
```

Where:

- `PROVIDER` is the name of the provider you want to use.
- `BUCKET-NAME` is the name of the bucket you want to use.
- `PLUGIN-IMAGE-LOCATION` is the location of the plug ins you want to use.
- `SECRET-FILE` is the file where the secret is located.

Velero CLI can then be used to create a backup of all the resources in the `metadata-store` namespace, including `PersistentVolumeClaim` and `PersistentVolume`.

```
velero backup create metadata-store-$(date '+%s') --include-namespaces=metadata-store
```

Restore

Velero CLI can restore the Store in the same or a different cluster. The same namespace can be used to restore, but may collide with other Supply Chain Security Tools – Store installations. Furthermore, restoring into the same namespace restores a fully functional instance of Supply Chain Security Tools – Store; however, this instance is not managed by Tanzu Application Platform and can cause conflicts with future installations.

```
velero restore create restore-metadata-store-$timestamp --from-backup metadata-store
-$timestamp --namespace-mappings metadata-store:metadata-store
```

Alternatively, a different namespace can be used to restore Supply Chain Security Tools – Store. In this case, Supply Chain Security Tools – Store API is not available due to conflicting definitions in the RBAC proxy configuration, causing all requests to fail with an `Unauthorized` error. In this scenario, the postgres instance is still accessible, and tools such as `pg_dump` can be used to retrieve table contents and restore in a new live installation of Supply Chain Security Tools – Store.

```
velero restore create restore-metadata-store-$timestamp --from-backup metadata-store
-$timestamp --namespace-mappings metadata-store:restored-metadata-store
```

Currently, mounting an existing `PersistentVolume` or `PersistentVolumeClaim` during installation is not supported.

The minimum suggested resources for backups are `PersistentVolume`, `PersistentVolumeClaim` and `Secret`. The database password `Secret` is needed to set up a Postgres instance with the correct password to properly read data from the restored volume.

Log configuration and usage for Supply Chain Security Tools - Store

This topic describes how you can configure Supply Chain Security Tools (SCST) - Store to output and interpret detailed log information.

Verbosity levels

There are six verbosity levels that Supply Chain Security Tools - Store supports.

Level	Description
Trace	Output extended debugging logs.
Debug	Output standard debugging logs.
More	Output more verbose informational logs.
Default	Output standard informational logs.
Less	Outputs less verbose informational logs.
Minimum	Outputs a minimal set of informational logs.

When SCST - Store is deployed at a specific verbosity level, all logs of that level and lower are output to the console. For example, setting the verbosity level to `More` outputs logs from `Minimal` to `More`, while `Debug` and `Trace` logs are muted.

Currently, the application logs output at these levels:

- **Minimum** does not output any logs.
- **Less** outputs a single log line indicating the current verbosity level is configured in Metadata Store when the application starts.
- **Default** outputs API endpoint access information.
- **Debug** outputs API endpoint payload information, both for requests and responses.
- **Trace** outputs verbose debug information about the actual SQL queries for the database.

Other log levels do not output any additional log information and are present for future extensibility.

If you don't specify a verbosity level when the Store is installed, the level is set to `default`.

Slow SQL

A slow SQL statement is logged only when verbosity level is set to `trace` and the SQL query takes more than 200 milliseconds to execute. You can change this default by setting the `db_slow_threshold_ms` value in values.yaml file and redeploying metadata store.

Error logs

Error logs are always output regardless of the verbosity level, even when set to `minimum`.

Obtaining logs

Kubernetes pods emit logs. The deployment has two pods, one for the database and one for the API back end.

Use `kubectl get pods` to obtain the names of the pods:

```
kubectl get pods -n metadata-store
```

For example:

```
$ kubectl get pods -n metadata-store
NAME                                  READY   STATUS    RESTARTS   AGE
metadata-store-app-67659bbc66-2rc6k   2/2     Running   0          4d3h
metadata-store-db-64d5b88587-8dns7    1/1     Running   0          4d3h
```

The database pod has the prefix `metadata-store-db-`. The API backend pod has the prefix `metadata-store-app-`. Use `kubectl logs` to get the logs from the pod you're interested in. For example, to get the logs of the database pod, run:

```
$ kubectl logs metadata-store-db-64d5b88587-8dns7 -n metadata-store
The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.
...
```

The API backend pod has two containers, one for `kube-rbac-proxy`, and one for the API server. Use the `--all-containers` flag to see logs from both containers. For example:

```
$ kubectl logs metadata-store-app-67659bbc66-2rc6k --all-containers -n metadata-store
I1206 18:34:17.686135       1 main.go:150] Reading config file: /etc/kube-rbac-proxy/c
onfig-file.yaml
I1206 18:34:17.784900       1 main.go:180] Valid token audiences:
...
{"level":"info","ts":"2022-05-27T13:47:52.54099339Z","logger":"MetadataStore","msg":"L
```

```
og settings","hostname":"metadata-store-app-5c9d6bccdb-kcrt2","LOG_LEVEL":"default"}
{"level":"info","ts":"2022-05-27T13:47:52.541133699Z","logger":"MetadataStore","ms
g":"Server Settings","hostname":"metadata-store-app-5c9d6bccdb-kcrt2","bindingaddres
s":"localhost:9443"}
{"level":"info","ts":"2022-05-27T13:47:52.541150096Z","logger":"MetadataStore","ms
g":"Database Settings","hostname":"metadata-store-app-5c9d6bccdb-kcrt2","maxopenconnec
tion":10,"maxidleconnection":100,"connectionmaxlifetime":60}
```

Note

The `kube-rbac-proxy` container uses a different log format than the Store. For information about the proxy's container log format, see Logging Formats in GithHub.

API endpoint log output

When an API endpoint handles a request, the Store generates two and five log entries:

- When the endpoint receives a request, it outputs a `Processing request` line. This logline is shown at the `default` verbosity level.
- If the endpoint includes query or path parameters, it outputs a `Request parameters` line. This line logs the parameters passed in the request. This line is shown at the `default` verbosity level.
- If the endpoint takes in a request body, it outputs a `Request body` line. This line outputs the entire request body as a string. This line is shown at the `debug` verbosity level.
- When the endpoint returns a response, it outputs a `Request response` line. This line is shown at the `default` verbosity level.
- If the endpoint returns a response body, it outputs a second `Request response` line with an extra key `payload`, and its value is set to the entire response body. This line is shown at the `debug` verbosity level.

Format

The logs use JSON output format.

When the Store handles a request, it outputs API endpoint access information in the following format:

```
{"level":"info","ts":"2022-05-27T15:41:36.051991749Z","logger":"MetadataStore","ms
g":"Processing request","hostname":"metadata-store-app-c7c8648f7-8dmdl","method":"GE
T","endpoint":"/api/images?digest=sha256%3A20521f76ff3d27f436e03dc666cc97a511bbe71e8e8
495f851d0f4bf57b0bab6"}
```

Key-value pairs

Because JSON output format uses key-value pairs, the tables in the following sections list each key and the meaning of their values.

Common to all logs

The following key-value pairs are common for all logs.

Key	Type	Verbosity Level	Description
level	string	all	The log level of the message. This is either `error` for error messages, or `info` for all other messages.
ts	string	all	The timestamp when the log entry was generated. It uses RFC 3339 format with nanosecond precision and 00:00 offset from UTC, meaning Zulu time.
logger	string	all	Used to identify what produced the log entry. For Store, the name always starts with `MetadataStore`. For log entries that display the raw SQL queries, the name is `MetadataStore.gorm`
msg	string	all	A short description of the logged event
hostname	string	all	The Kubernetes host name of the pod handling the request. This helps identify the specific instance of the Store when you deploy multiple instances on a cluster.
error	string	all	The error message which is only available in error log entries
endpoint	string	default	The API endpoint the Metadata Store attempts to handle the request. This also includes any query and path parameters passed in.
method	string	default	The HTTP verb to access the endpoint. For example, `GET` or `POST`.
code	integer	default	The HTTP response code
response	string	default	The HTTP response in human-readable format. For example, `OK`, `Bad Request`, or `Internal Server Error`.
function	string	debug	The function name that handles the request.

Logging query and path parameter values

Those endpoints that use query or path parameters are logged on the `Request parameters` log entry as key-value pairs. They are appended to all other log entries of the same request as key-value pairs.

The key names are the query or path parameter's name, while the value is set to the value of those parameters in string format.

For example, the following log entry contains the `digest` and `id` key, which represents the respective `digest` and `id` query parameters, and their values:

```
{"level":"info","ts":"2022-05-27T15:41:36.0520631762","logger":"MetadataStore","msg":"Request parameters","hostname":"metadata-store-app-c7c8648f7-8dmdl","method":"GET","endpoint":"/api/images?digest=sha256%3A20521f76ff3d27f436e03dc666cc97a511bbe71e8e8495f851d0f4bf57b0bab6","id":0,"digest":"sha256:20521f76ff3d27f436e03dc666cc97a511bbe71e8e8495f851d0f4bf57b0bab6","name":""}
```

These key-value pairs show up in all subsequent log entries of the same call. For example:

```
{"level":"info","ts":"2022-05-27T15:41:36.0573935192","logger":"MetadataStore","msg":"Request response","hostname":"metadata-store-app-c7c8648f7-8dmdl","method":"GET","endpoint":"/api/images?digest=sha256%3A20521f76ff3d27f436e03dc666cc97a511bbe71e8e8495f851d0f4bf57b0bab6","id":0,"digest":"sha256:20521f76ff3d27f436e03dc666cc97a511bbe71e8e8495f851d0f4bf57b0bab6","name":"","code":200,"response":"OK"}
```

This is done to:

- Ensure that the application interprets the values of the query or path parameters correctly.

- Figure out which log entries are associated with a particular API request. Because there might be several simultaneous endpoint calls, this is a first attempt at grouping logs by specific calls.

API payload log output

As mentioned at the start of this section, by setting the verbosity level to `debug`, the Store logs the body payload data for both the request and response of an API call.

The `debug` verbosity level, instead of the `default`, displays this information instead of `default` because:

- Body payloads can contain full CycloneDX and SBOM information. Moving the payload information at this level helps keep the production log output to a reasonable size.
- Some information in these payloads might be sensitive, and the user might not want them exposed in production environment logs.

GraphQL endpoint log output

Note

This section is only applicable to Artifactory Metadata Repository (AMR) logs.

When an GraphQL endpoint handles a request, the AMR generates following types of logs:

- Every request received produces a `Processing request` log, which includes the name of the operation called and the requested fields.
- Every response will produces a log containing the actual query and the return status
- If the endpoint returns a response body, it outputs a second `Request response` line with an extra key `payload`, and its value is set to the entire response body. This line is shown at the `debug` verbosity level

Format

The logs output are in JSON format.

When the AMR handles a request, it outputs some GraphQL endpoint access information in the following format:


```
{"level":"info","ts":"2023-03-23T13:11:31.161531-06:00","logger":"Artifact Metadata Re
pository","msg":"Processing request","hostname":"xyzp2DMD6R.vmware.com","getAllApp
s":"query getAllApps {\n  apps(latest:true) {\n    \n    timestamp\n    location {\n
alias\n    }\n  }\n}"}
{"level":"info","ts":"2023-03-23T13:11:31.172953-06:00","logger":"Artifact Metadata Re
pository","msg":"Request response","hostname":"xyzp2DMD6R.vmware.com","getAllApps":"qu
ery getAllApps {\n  apps(latest:true) {\n    \n    timestamp\n    location {\n      al
ias\n    }\n  }\n}","code":200,"response":"OK"}
```


Key-value pairs

The following tables list the meaning of each key found in the logs.

Common to all logs

The following key-value pairs are common for all logs.

Key	Type	Verbosity Level	Description
level	string	all	The log level of the message. This is either `error` for error messages, or `info` for all other messages.
ts	string	all	The timestamp when the log entry was generated. It uses RFC 3339 format with nanosecond precision and 00:00 offset from UTC, meaning Zulu time.
logger	string	all	Used to identify what produced the log entry. For Store, the name always starts with `MetadataStore`. For log entries that display the raw SQL queries, the name is `MetadataStore.gorm`.
msg	string	all	A short description of the logged event
hostname	string	all	The Kubernetes host name of the pod handling the request. This helps identify the specific instance of the Store when you deploy multiple instances on a cluster.
error	string	all	The error message which is only available in error log entries
code	integer	default	The HTTP response code
response	string	default	The HTTP response in human-readable format. For example, `OK`, `Bad Request`, or `Internal Server Error`.
query	string	debug	The operation name is the key and value fields that the fields requested.

API payload log output

By setting the verbosity level to `debug`, the AMR logs the body payload data for both the request and response of an API call.

The `debug` verbosity level, instead of the `default`, displays this information instead of `default` because body payloads can be large and some information in these payloads might be sensitive. You might not want the sensitive payloads exposed in production environment logs.

Logs containing payload information might be in the following format:

```
{"level":"info","ts":"2023-03-23T13:11:31.172966-06:00","logger":"Artifact Metadata Repository","msg":"Request response","hostname":"xyzp2DMD6R.vmware.com","getAllApps":"query getAllApps {\n  apps(latest:true) {\n    \n    timestamp\n    location {\n      alias\n    }\n  }\n}","payload":{"apps":[{"timestamp":"2023-03-22T15:09:38.867371-06:00","location":{"alias":"1-Alias"}}]}}
```

Slow SQL query log output

When the verbosity level is set to `trace`, you see log entries containing slow SQL queries.

Note

Some information in these SQL Query `trace` logs might be sensitive, and you might not want them exposed in production environment logs.

SQL Query log output

Slow SQL query logs are displayed in the following format when verbosity level is set to `trace`:

```
{"level":"info","ts":"2023-03-23T12:48:12.337749-06:00","logger":"Artifact Metadata Repository.gorm","msg":"slow sql >= 200ms","hostname":"xyzp2DMD6R.vmware.com","rows":50000,"sql":"SELECT \"artifact_apps\".\"id\",\"artifact_apps\".\"created_at\",\"artifact_
```

```
apps\".\"updated_at\",\"artifact_apps\".\"deleted_at\",\"artifact_apps\".\"location_id
\",\"artifact_apps\".\"correlation_id\",\"artifact_apps\".\"image_url\",\"artifact_app
s\".\"image_digest\",\"artifact_apps\".\"namespace\",\"artifact_apps\".\"name\",\"arti
fact_apps\".\"instances\",\"artifact_apps\".\"status\",\"artifact_apps\".\"timestamp\"
FROM \"artifact_apps\" INNER JOIN (select max(timestamp) as timestamp, name, namespac
e, location_id from artifact_apps group by location_id, name, namespace) as argo on ar
go.timestamp = artifact_apps.timestamp and argo.name = artifact_apps.name and argo.loc
ation_id = artifact_apps.location_id and argo.namespace = artifact_apps.namespace WHER
E \"artifact_apps\".\"deleted_at\" IS NULL"}
```


It is similar to the API endpoint log output format, but uses the following key-value pairs:

Key	Type	Log Level	Description
rows	integer	trace	Indicates the number of rows affected by the SQL query
sql	string	trace	Displays the raw SQL query for the database
data#	string	all	Used in error log entries. You can replace `#` with an integer because multiples of these keys can appear in the same log entry. These keys contain extra information related to the error.

SQL Query log output

Some Store logs display the executed SQL query commands when you set the verbosity level to `trace` or a SQL call fails.

Note

Some information in these SQL Query trace logs might be sensitive, and you might not want them exposed in production environment logs.

Format

When the Store display SQL query logs, it uses the following format:


```
{"level":"info","ts":"2022-05-27T15:37:26.1869603242","logger":"MetadataStore.gorm","m
sg":"sql call","hostname":"metadata-store-app-c7c8648f7-8dmdl","rows":1,"sql":"SELECT
count(*) FROM information_schema.tables WHERE table_schema = CURRENT_SCHEMA() AND tabl
e_name = 'images' AND table_type = 'BASE TABLE'"}
```


It is similar to the API endpoint log output format, but uses the following key-value pairs:

Key	Type	Log Level	Description
rows	integer	trace	Indicates the number of rows affected by the SQL query
sql	string	trace	Displays the raw SQL query for the database
data#	string	all	Used in error log entries. You can replace `#` with an integer because multiples of these keys can appear in the same log entry. These keys contain extra information related to the error.

Connecting to the Postgres Database

To connect to the Postgres Database, you will need the following values: * database name * username * password * database host * database port * database CA certificate

1. To obtain the database name, username, password, and CA cert, run the following commands:

```
db_name=$(kubectl get secret postgres-db-secret -n metadata-store -o json | jq
-r '.data.POSTGRES_DB' | base64 -d)
db_username=$(kubectl get secret postgres-db-secret -n metadata-store -o json |
jq -r '.data.POSTGRES_USER' | base64 -d)
db_password=$(kubectl get secret postgres-db-secret -n metadata-store -o json |
jq -r '.data.POSTGRES_PASSWORD' | base64 -d)

db_ca_dir=$(mktemp -d -t ca-cert-XXXX)
db_ca_path="$db_ca_dir/ca.crt"
kubectl get secrets postgres-db-tls-cert -n metadata-store -o json | jq -r '.da
ta."ca.crt"' | base64 -d > $db_ca_path
```

   If the password was auto-generated, the above `password` command will return an empty string. Instead, run the following command:

```
db_password=$(kubectl get secret postgres-db-password -n metadata-store -o json
| jq -r '.data.DB_PASSWORD' | base64 -d)
```

2. In a separate terminal, run the following command:

```
kubectl port-forward service/metadata-store-db 5432:5432 -n metadata-store
```

   Set the database host and port values on the first terminal with the following:

```
db_host="localhost"
db_port=5432
```

   To port forward to a different local port number, use the following command template:

```
kubectl port-forward service/metadata-store-db <LOCAL_PORT>:5432 -n metadata-st
ore
```

With this setup, you can now connect to the database and make queries like the following example:

```
psql "host=$db_host port=$db_port user=$db_username dbname=$db_name sslmode=verify-ca
sslrootcert=$db_ca_path" -c "SELECT * FROM images"
```

You could also use GUI clients such as Postico or DBeaver to interact with the database.

Troubleshooting Supply Chain Security Tools - Store

This topic contains ways you can troubleshoot known issues for Supply Chain Security Tools (SCST) - Store.

Querying by `insight source` returns zero CVEs even though there are CVEs in the source scan

Symptom

The `insight source get` and other `insight source` commands return zero results.

Solution

You might have to include different combinations of `--repo`, `--org`, `--commit` due to how the scan-controller populates the software bill of materials (SBOM). For more information see Query vulnerabilities, images, and packages.

Persistent volume retains data

Symptom

If SCST - Store is deployed, deleted, redeployed, and the database password is changed during the redeployment, the `metadata-store-db` pod fails to start. The persistent volume used by PostgreSQL retaining old data, even though the retention policy is set to `DELETE`, causes this issue.

Solution

Caution

Changing the database password deletes your SCST - Store data.

To redeploy the app, either use the same database password or follow these steps to erase the data on the volume:

1. Deploy metadata-store app by using `kapp`.
2. Verify that the `metadata-store-db-*` pod fails.
3. Run:

   ```
   kubectl exec -it metadata-store-db-<some-id> -n metadata-store /bin/bash
   ```

   Where `<some-id>` is the ID generated by Kubernetes and appended to the pod name.
4. Run `rm -rf /var/lib/postgresql/data/*` to delete all database data.

   Where `/var/lib/postgresql/data/*` is the path found in `postgres-db-deployment.yaml`.
5. Delete the `metadata-store` app by using `kapp`.
6. Deploy the `metadata-store` app by using `kapp`.

Missing persistent volume

Symptom

After SCST - Store is deployed, `metadata-store-db` pod might fail for missing volume while `postgres-db-pv-claim` pvc is in `PENDING` state.

This is because the cluster where SCST - Store is deployed does not have `storageclass` defined. `storageclass`'s provisioner is responsible for creating the persistent volume after `metadata-store-db` attaches `postgres-db-pv-claim`.

Solution

1. Verify that your cluster has `storageclass` by running `kubectl get storageclass`.
2. Create a `storageclass` in your cluster before deploying SCST - Store. For example:

   ```
   # This is the storageclass that Kind uses
   kubectl apply -f https://raw.githubusercontent.com/rancher/local-path-provision
   er/master/deploy/local-path-storage.yaml
   ```

```
# set the storage class as default
kubectl patch storageclass local-path -p '{"metadata": {"annotations":{"storage
class.kubernetes.io/is-default-class":"true"}}}'
```

Builds fail due to volume errors on EKS running Kubernetes v1.23

Symptom

When installing SCST - Store on or upgrading an existing EKS cluster to Kubernetes v1.23, the satabase pod shows:

```
running PreBind plugin "VolumeBinding": binding volumes: provisioning failed for PVC
"postgres-db-pv-claim"
```

Explanation

This is due to the CSIMigrationAWS in this Kubernetes version which requires users to install the Amazon Elastic Block Store (EBS) CSI Driver to use EBS volumes. For more information, see the AWS documentation.

SCST - Store uses the default storage class which uses EBS volumes by default on EKS.

Solution

Follow the AWS documentation to install the Amazon EBS CSI Driver before installing SCST - Store or before upgrading to Kubernetes v1.23. For more information, see the AWS documentation.

CA Cert expires

Symptom

The Insight CLI or Scan Controller fails to communicate with the SCST - Store and receives an error message containing the following text:

```
tls: failed to verify certificate: x509: certificate has expired or is not yet valid
```

Explanation

The CA certificate expired before the app certificate expires, which causes the error even though the app certificate is still valid. Certmanager will rotate the expired CA certificate, but it doesn't rotate the certificates that were previously created by the expired CA certificate. So this leads to contour being in a bad state and the certificates won't be refreshed properly.

Solution

1. Delete the existing expired cacert

```
kubectl delete secret cacert contourcert envoycert -n projectcontour
```

1. Delete the contour-certgen job

```
kubectl delete job contour-certgen -n projectcontour
```

1. Trigger reconciliation for contour

```
kctrl package installed kick --package-install contour -n tap-install
```

1. Restart envoy pods. First find the name of the envoy pod

```
kubectl get pods -n projectcontour
```

Find the pod that is named in the format `envoy-<some-random-id>`. Use that name and restart the pods by deleting them:

```
kubectl delete pod <envoy-pod-name> -n projectcontour
```

Certificate Expires

Symptom

The Insight CLI or the Scan Controller fails to connect to SCST - Store.

The logs of the metadata-store-app pod show the following error:

```
$ kubectl logs deployment/metadata-store-app -c metadata-store-app -n metadata-store
...
2022/09/12 21:22:07 http: TLS handshake error from 127.0.0.1:35678: write tcp 127.0.0.
1:9443->127.0.0.1:35678: write: broken pipe
...
```

or

The logs of metadata-store-db show the following error:

```
$ kubectl logs statefulset/metadata-store-db -n metadata-store
...
2022-07-20 20:02:51.206 UTC [1] LOG:  database system is ready to accept connections
2022-09-19 18:05:26.576 UTC [13097] LOG:  could not accept SSL connection: sslv3 alert
bad certificate
...
```

Explanation

cert-manager rotates the certificates, but the metadata-store and the PostgreSQL db are unaware of the change, and are using the old certificates.

Solution

If you see `TLS handshake error` in the metadata-store-app logs, delete the metadata-store-app pod and wait for it to come back up.

```
kubectl delete pod metadata-store-app-xxxx -n metadata-store
```

If you see `could not accept SSL connection` in the metadata-store-db logs, delete the metadata-store-db pod and wait for it to come back up.

```
kubectl delete pod metadata-store-db-0 -n metadata-store
```

Database index corruption issue in SCST - Store

Metadata Store unable to reconcile because the metadata store pod complains about potential database index corruption issue.

```
kubectl logs metadata-store-app-pod_name -n metadata-store
```

```
{"level":"error","ts":"2023-08-15T16:38:31.528115988Z","logger":"MetadataStore","ms
g":"unable to check index corruption since user is not a superuser to perform \"CREATE
EXTENSION amcheck\". Please create this extension and check for index corruption using
following sql command \"SELECT bt_index_check(oid) FROM pg_class WHERE relname in (SEL
ECT indexrelid::regclass::text FROM (SELECT indexrelid, indrelid, indcollation[i] coll
FROM pg_index, generate_subscripts(indcollation, 1) g(i)) s JOIN pg_collation c ON col
l=c.oid WHERE collprovider IN ('d', 'c') AND collname NOT IN ('C', 'POSIX'));\"","host
name":"metadata-store-app-77c9fb59c8-qplxt"}
{"level":"error","ts":"2023-08-15T16:38:31.528139637Z","logger":"MetadataStore","ms
g":"Found corrupted database indexes but unable to fix them","hostname":"metadata-stor
e-app-77c9fb59c8-qplxt","error":"unable to check index corruption since user is not a
superuser to perform \"CREATE EXTENSION amcheck\". Please create this extension and ch
eck for index corruption using following sql command \"SELECT bt_index_check(oid) FROM
pg_class WHERE relname in (SELECT indexrelid::regclass::text FROM (SELECT indexrelid,
indrelid, indcollation[i] coll FROM pg_index, generate_subscripts(indcollation, 1) g
(i)) s JOIN pg_collation c ON coll=c.oid WHERE collprovider IN ('d', 'c') AND collname
NOT IN ('C', 'POSIX'));\""}
```

For information about the solution, seePostgres Database Index Corruption.

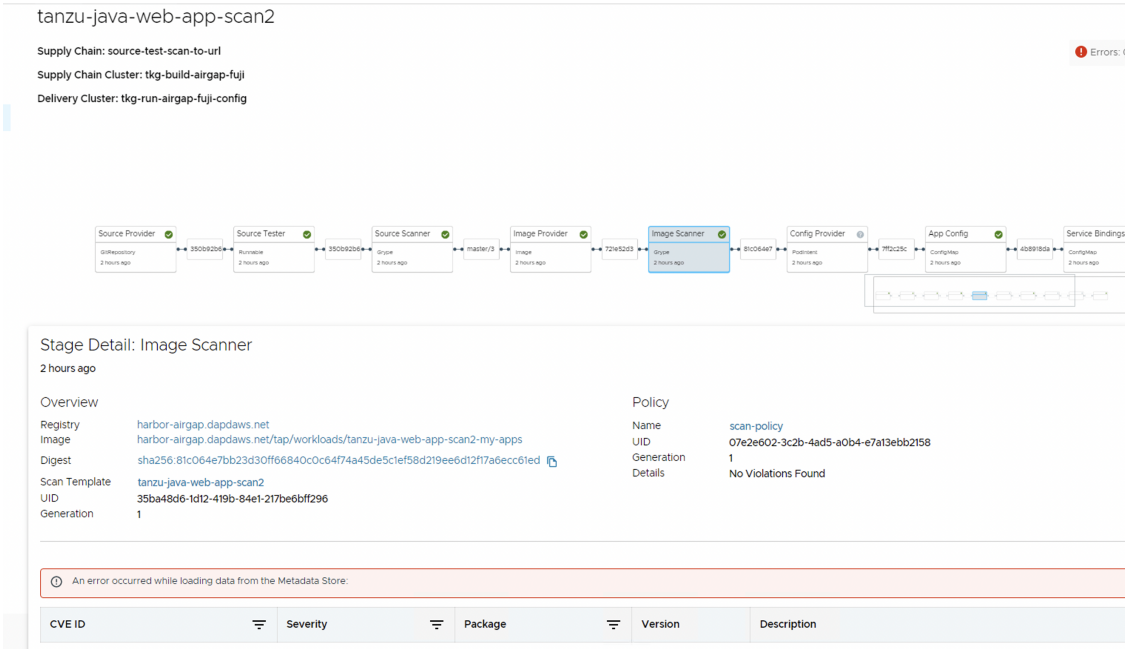
Errors from Tanzu Developer Portal related to SCST - Store

Different Tanzu Developer Portal plug-ins use SCST - Store to display information about vulnerabilities and packages. Some errors visible in Tanzu Developer Portal are related to this connection.

An error occurred while loading data from the Metadata Store

Symptom

In the Supply Chain Choreographer plug-in, you see the error message `An error occurred while loading data from the Metadata Store`.



Cause

There are multiple potential causes. The most common cause is `tap-values.yaml` missing the configuration that enables Tanzu Developer Portal to communicate with Supply Chain Security Tools - Store.

Solution

See Supply Chain Choreographer - Enable CVE scan results for the necessary configuration to add to `tap-values.yaml`. After adding the configuration, update your Tanzu Application Platform deployment or Tanzu Developer Portal deployment with the new values.

Upgrade Supply Chain Security Tools - Store

This topic tells you how to upgrade Supply Chain Security Tools (SCST) - Store and how to troubleshoot upgrade issues.

Upgrading to 1.7 and later

In Tanzu Application Platform v1.7 and later, VMware introduces Artifact Metadata Repository (AMR) to SCST - Store. Tanzu Application Platform installs AMR components by default after upgrading.

How you must configure AMR depends on if:

- Tanzu Application Platform is installed in a single cluster, Full profile, deployment. See Single cluster or Full profile upgrade.
- Tanzu Application Platform is installed in a multicluster deployment. See Multicluster upgrade.

Single cluster or Full profile upgrade

In a Full profile, AMR does not need any additional user configuration. See AMR architecture and deployment details.

Multicluster upgrade

In a multicluster deployment, AMR components are installed on the View, Build and Run clusters. The AMR Observer is installed on the Build and Run clusters, and must be configured to talk to the AMR CloudEvent Handler installed on the View cluster.

Read SCST - Store multicluster setup for the new configuration in detail. The documentation includes instructions for configuring both the new AMR and the existing Metadata Store. The new configurations for 1.7 and later are:

1. Copy AMR CloudEvent Handler CA certificate data from the View cluster
2. Copy AMR CloudEvent Handler edit token from the View cluster
3. Apply the CloudEvent Handler CA certificate data and edit token to the Build and Run clusters

Upgrading AMR Beta to 1.7 and later

This topic is about a special upgrade scenario for AMR: Upgrading from Tanzu Application Platform 1.6 with AMR beta enabled to Tanzu Application Platform v1.7 and later. Because AMR was not enabled by default in Tanzu Application Platform 1.6, most users will not encounter this scenario. See Upgrading from AMR Beta to AMR GA release.

Troubleshoot upgrading

The following sections tell you how to troubleshoot AMR upgrades.

AMR Observer cannot talk to AMR CloudEvent Handler

To see the AMR Observer pod:

```
kubectl get pods -n amr-observer-system
```

To view AMR Observer logs:

```
kubectl logs OBSERVER-POD-NAME -n amr-observer-system
```

Where:

- `OBSERVER-POD-NAME` is the name of the AMR observer pod.

If you encounter errors relating to the authentication token, verify that you configured the AMR Observer correctly. It might be missing the edit token for the AMR CloudEvent Handler. For information about how to configure the edit token, and the CA certificate and endpoint, see SCST - Store multicluster setup.

Database deployment does not exist

To prevent issues with the metadata store database, such as the ones described in this topic, the database deployment is `StatefulSet` in:

- Tanzu Application Platform v1.2 and later
- Metadata Store v1.1 and later

If you have scripts searching for a `metadata-store-db` deployment, edit the scripts to instead search for `StatefulSet`.

Invalid checkpoint record

When you use Tanzu to upgrade SCST - Store, there is occasionally data corruption. For example:

```
PostgreSQL Database directory appears to contain a database; Skipping initialization

2022-01-21 21:53:38.799 UTC [1] LOG:  starting PostgreSQL 13.5 (Ubuntu 13.5-1.pgdg18.0
4+1) on x86_64-pc-linux-gnu, compiled by gcc (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0, 64-b
it
2022-01-21 21:53:38.799 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
2022-01-21 21:53:38.799 UTC [1] LOG:  listening on IPv6 address "::", port 5432
2022-01-21 21:53:38.802 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.
s.PGSQL.5432"
2022-01-21 21:53:38.807 UTC [14] LOG:  database system was shut down at 2022-01-21 21:
21:12 UTC
2022-01-21 21:53:38.807 UTC [14] LOG:  invalid record length at 0/1898BE8: wanted 24,
got 0
2022-01-21 21:53:38.807 UTC [14] LOG:  invalid primary checkpoint record
2022-01-21 21:53:38.807 UTC [14] PANIC:  could not locate a valid checkpoint record
2022-01-21 21:53:39.496 UTC [1] LOG:  startup process (PID 14) was terminated by signa
l 6: Aborted
2022-01-21 21:53:39.496 UTC [1] LOG:  aborting startup due to startup process failure
2022-01-21 21:53:39.507 UTC [1] LOG:  database system is shut down
```

The log shows a database pod in a failure loop. For information about how to fix the issue, see the SysOpsPro documentation.

Upgraded pod stops responding

Because the default access mode in the PVC is `ReadWriteOnce`, if you are deploying in an environment with multiple nodes then each pod might be on a different node. This causes the upgraded pod to spin up but then get stuck initializing because the original pod does not stop. To resolve this issue, find, and delete the original pod so that the new pod can attach to the persistent volume:

1. Discover the name of the app pod that is not in a pending state by running:

   ```
   kubectl get pods -n metadata-store
   ```

2. Delete the pod by running:

   ```
   kubectl delete pod METADATA-STORE-APP-POD-NAME -n metadata-store
   ```

Upgrading from AMR Beta to AMR GA release

This topic tells you how to upgrade from Tanzu Application Platform 1.6 to Tanzu Application Platform 1.7 or later with Artifact Metadata Repository (AMR) beta enabled. Because AMR is not enabled by default in Tanzu Application Platform 1.6, most users will not encounter this scenario. To upgrade without AMR, see Supply Chain Security Tools - Store - Upgrading.

Known issues and workarounds

Because AMR was in beta in Tanzu Application Platform v1.6, there are breaking changes when upgrading to Tanzu Application Platform v1.9. This section lists all the known issues and workarounds.

Configuration Changes

In the AMR Beta release, most of the AMR configurations are in-line with `metadata_store` section inside `values.yaml` file. You must remove `metadata_store.amr` from the `values.yaml` file.

1. Remove `metadata_store.amr` from the values file.

   ```
   metadata_store:
     amr:
       deploy: true
       graphql:
       app_service_type: "ClusterIP"
   ```

2. Remove `amr.deploy_observer: true` from the values file
3. Remove Alias from the `amr.observer.location` configmap

   ```
   location: |
     alias: my-cluster
   ```

Database changes

In the AMR Beta release, the `Alias` field was introduced in the `Location` table. The `Alias` field is removed in Tanzu Application Platform v1.9. To drop this field from Tanzu Application Platform v1.9:

1. Connect to the Postgres database.
2. Run the following SQL command:

```
ALTER TABLE artifact_locations DROP COLUMN IF EXISTS alias;
```

Failover, redundancy, and backups for Supply Chain Security Tools - Store

This topic describes how you can configure and use failover, redundancy, and backups for Supply Chain Security Tools (SCST) - Store.

API Server

By default the API server has 1 replica. If the pod fails, the single instance restarts by normal Kubernetes behavior, but there is downtime. If you upgrade, some downtime is possible.

You can configure the number of replicas by using the `app_replicas` text box in the `scst-store-values.yaml` file.

Database

By default, the database has one replica, and restarts with some downtime if it fails.

Caution

Although you can configure `db_replicas` in `scst-store-values.yaml`, this is discouraged because `db_replicas` is still experimental. The default internal database is not for production use. For production deployments, use an external database.

- Use external postgres database
- AWS RDS postgres configuration

For the default PostgreSQL database deployment, with `deploy_internal_db` set to `true`, you can use `Velero` as the backup method. For information about using `Velero` as the backup method, see Backups.

Custom certificate configuration for Supply Chain Security Tools - Store

This topic describes how you can configure the following certificates for Supply Chain Security Tools (SCST) - Store:

1. Default configuration
2. Custom certificate

Default configuration

By default, SCST - Store creates a self-signed certificate and TLS communication is automatically enabled.

If ingress support is enabled, SCST - Store installation creates an HTTPProxy entry with host routing by using the qualified name `metadata-store.<ingress_domain>`. For example, `metadata-store.example.com`. The created route supports HTTPS communication using the self-signed certificate with the same subject `Alternative Name`.

(Optional) Setting up custom ingress TLS certificate

(Optional) Users can configure TLS to use a custom certificate. To do that:

1. Place the certificates in the secret.
2. Edit the `tap-values.yaml` to use this secret.

Place the certificates in secret

1. Create the certificate secret before deploying SCST - Store.
2. Create a Kubernetes object with kind `Secret` and type `kubernetes.io/tls`.

Update `tap-values.yaml`

1. In the `tap-values.yaml` file, you can configure the metadata store to use the `namespace` and `secretName` from the secret created in the last step.

```
metadata_store:
  tls:
    namespace: "namespace"
    secretName: "secretName"
```

Where:

- `namespace` is the targeted namespace for secret consumption by the HTTPProxy.
- `secretName` is the name of secret for consumption by the HTTPProxy.

Additional resources

- Ingress support
- TLS configuration

TLS configuration for Supply Chain Security Tools - Store

This topic describes how you can configure TLS for Supply Chain Security Tools (SCST) - Store.

Important

SCST - Store only supports TLS v1.2.

Setting up custom ingress TLS ciphers

In the `tap-values.yaml` file, `tls.server.rfcCiphers` are set as shown in the following YAML:

```
metadata_store:
  tls:
    server:
      rfcCiphers:
        - TLS_AES_128_GCM_SHA256
        - TLS_AES_256_GCM_SHA384
        - TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
        - TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
        - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
        - TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
```

Where `tls.server.rfcCiphers` is a list of cipher suites for the server. Values are from the Go TLS package constants. If you omit values, the default Go cipher suites are used. These are the default values:

- `TLS_AES_128_GCM_SHA256`
- `TLS_AES_256_GCM_SHA384`
- `TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256`
- `TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384`
- `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`
- `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`

Example custom TLS settings

The following is a complete example of TLS configuration:

```
metadata_store:
  tls:
    namespace: NAMESPACE
    secretName: SECRET-NAME
    server:
      rfcCiphers:
        - TLS_AES_128_GCM_SHA256
        - TLS_AES_256_GCM_SHA384
        - TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
        - TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
        - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
        - TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
```

Where:

- `NAMESPACE` is the name of the namespace you want to configure TLS with.
- `SECRET-NAME` is the name of the secret you want to configure TLS with.

Additional resources

- Custom certificate configuration
- Ingress support

Certificate rotation for Supply Chain Security Tools - Store

This topic describes how you can rotate TLS certificates for Supply Chain Security Tools (SCST) - Store.

Certificates

By default, the `use_cert_manager` setting is set to `"true"`. When the setting `use_cert_manager` is `"true"` the Store uses `cert-manager` to generate a CA certificate, an API certificate, and a database Certificate.

To see these certificates:

```
$ kubectl get certificate -n metadata-store
NAME                    READY   SECRET                  AGE
app-tls-ca-cert         True    app-tls-ca-cert         38d
```

```
app-tls-cert             True    app-tls-cert             38d
postgres-db-tls-cert     True    postgres-db-tls-cert     38d
```

The earlier certificates are automatically rotated by `cert-manager`.

The Store can run these certificates automatically once `cert-manager` rotates them.

If the environment is a multi-cluster setup, the operator must manually copy over the new CA certificate to the build cluster.

Certificate duration setting

In the `tap-values.yaml` file, `api_cert_duration`, `api_cert_renew_before`, `ca_cert_duration`, and `ca_cert_renew_before` are configurable as shown in the following YAML:

```
metadata_store:
  ca_cert_duration: CA-DURATION
  ca_cert_renew_before: CA-RENEW
  api_cert_duration: API-DURATION
  api_cert_renew_before: API-RENEW
```

Where:

- `CA-DURATION` is the duration that the ca certificate is valid for. Must be given in `h`, `m`, or `s`. Default value is 8760h.
- `CA-RENEW` is how long before the expiry of the ca certificate is renewed. Must be given in `h`, `m`, or `s`. Default value is 1h.
- `API-DURATION` is the duration that the API certificate is valid for. Must be given in `h`, `m`, or `s`. Default value is 2160h.
- `API-RENEW` is how long before the expiry of the API certificate is renewed. Must be given in h, m, or s. Default value is 24h.

Important

The `*_cert_duration` and the corresponding `*_renew_before` settings must not be close. For more information, see the cert-manager documentation. This can lead to a renewal loop. The `*_cert_duration` must be greater than the corresponding `*_renew_before`. The earlier settings only take effect when `use_cert_manager` is `"true"`. If the `use_cert_manager` is not set, it defaults to `"true"`.

Ingress support for Supply Chain Security Tools - Store

This topic describes how to configure ingress for Supply Chain Security Tools (SCST) - Store.

Ingress configuration

Supply Chain Security Tools (SCST) - Store has ingress support by using Contour's HTTPProxy resources. To enable ingress support, a Contour installation must be available in the cluster.

To change ingress configuration, edit your `tap-values.yaml` when you install a Tanzu Application Platform profile. When you configure the `shared.ingress_domain` property, SCST - Store automatically uses that setting.

Alternatively, you can customize SCST - Store's configuration under the `metadata_store` property. Under `metadata_store`, there are two values to configure the proxy:

- `ingress_enabled`
- `ingress_domain`

This is an example snippet in a `tap-values.yaml`:

```
...
metadata_store:
  ingress_enabled: "true"
  ingress_domain: "example.com"
  app_service_type: "ClusterIP"  # Defaults to `LoadBalancer`. If ingress is enabled t
hen this must be set to `ClusterIP`.
...
```

SCST - Store installation creates an HTTPProxy entry with host routing by using the qualified name `METADATA-STORE.INGRESS-DOMAIN`. For example, `metadata-store.example.com`. The route supports HTTPS communication using a certificate. By default, a self-signed certificate is used with the same subject `alternative name`. For more information, see Custom certificate configuration.

Contour and DNS setup are not part of SCST - Store installation. Access to SCST - Store using Contour depends on the correct configuration of these two components.

Make the proper DNS record available to clients to resolve `metadata-store` and set `ingress_domain` to Envoy service's external IP address.

DNS setup example:

```
$ kubectl describe svc envoy -n tanzu-system-ingress
> ...
  Type:                     LoadBalancer
  ...
  LoadBalancer Ingress:     100.2.3.4
  ...
  Port:                     https  443/TCP
  ...

$ nslookup metadata-store.example.com
> Server:    8.8.8.8
  Address:  8.8.8.8#53

  Non-authoritative answer:
  Name:  metadata-store.example.com
  Address: 100.2.3.4

$ curl https://metadata-store.example.com/api/health -k -v
> ...
  < HTTP/2 200
  ...
```

Note

The preceding `curl` example uses the not secure `-k` flag to skip TLS verification because the Store installs a self-signed certificate. The following section shows how to access the CA certificate to enable TLS verification for HTTP clients.

Get the TLS CA certificate

To get SCST - Store's TLS CA certificate, use `kubectl get secret`. In this example, you save the certificate for the environment variable to a file.

```
kubectl get secret CERT-NAME -n metadata-store -o json | jq -r '.data."ca.crt"' | base
64 -d > OUTPUT-FILE
```

Where:

- `CERT-NAME` is the name of the certificate. This must be `ingress-cert` if no custom certificate is used.
- `OUTPUT-FILE` is the file you want to create to store the certificate.

For example:

```
$ kubectl get secret tap-ingress-selfsigned-root-ca -n cert-manager -o json | jq -r '.
data."ca.crt"' | base64 -d > insight-ca.crt
$ cat insight-ca.crt
```

Additional Resources

- Custom certificate configuration
- TLS configuration
- Certificate rotation
- Configure target endpoint and certificate

Use your LoadBalancer with Supply Chain Security Tools - Store

This topic describes how to use your LoadBalancer with Supply Chain Security Tools (SCST) - Store.

Configure LoadBalancer

Note

`LoadBalancer` is not the recommended service type. Consider the recommended configuration of enabling Ingress.

To configure a `LoadBalancer`:

1. Edit `/etc/hosts/` to use the external IP address of the `metadata-store-app` service.

   ```
   METADATA_STORE_IP=$(kubectl get service/metadata-store-app --namespace metadata
   -store -o jsonpath="{.status.loadBalancer.ingress[0].ip}")
   METADATA_STORE_PORT=$(kubectl get service/metadata-store-app --namespace metada
   ta-store -o jsonpath="{.spec.ports[0].port}")
   METADATA_STORE_DOMAIN="metadata-store-app.metadata-store.svc.cluster.local"

   # Delete any previously added entry
   sudo sed -i '' "/$METADATA_STORE_DOMAIN/d" /etc/hosts

   echo "$METADATA_STORE_IP $METADATA_STORE_DOMAIN" | sudo tee -a /etc/hosts > /de
   v/null
   ```

   Note

   On EKS, you must get the IP address for the LoadBalancer. Find the IP address by running something similar to the following: `dig RANDOM-SHA.us-`

east-2.elb.amazonaws.com. Where RANDOM-SHA is the EXTERNAL-IP received for the LoadBalancer.

2. Select one of the IP addresses returned from the dig command and write it to the /etc/hosts file.

Port forwarding

If you want to use port forwarding instead of the external IP address from the LoadBalancer, follow these steps:

Configure port forwarding for the service so the insight plug-in can access SCST - Store. Run:

```
kubectl port-forward service/metadata-store-app 8443:8443 -n metadata-store
```

Note: You must run the port forwarding command in a separate terminal window, or run the command in the background: kubectl port-forward service/metadata-store-app 8443:8443 -n metadata-store &

Edit your /etc/hosts file for Port Forwarding

Use the following script to add a new local entry to /etc/hosts:

```
METADATA_STORE_PORT=$(kubectl get service/metadata-store-app --namespace metadata-stor
e -o jsonpath="{.spec.ports[0].port}")
METADATA_STORE_DOMAIN="metadata-store-app.metadata-store.svc.cluster.local"

# delete any previously added entry
sudo sed -i '' "/$METADATA_STORE_DOMAIN/d" /etc/hosts

echo "127.0.0.1 $METADATA_STORE_DOMAIN" | sudo tee -a /etc/hosts > /dev/null
```

Configure the Insight plug-in

Because you deployed Supply Chain Security Tools (SCST) - Store without using Ingress, you must use the Certificate resource app-tls-cert for HTTPS communication.

To get the CA Certificate:

```
kubectl get secret app-tls-cert -n metadata-store -o json | jq -r '.data."ca.crt"' | b
ase64 -d > insight-ca.crt
```

Set the target by running:

```
tanzu insight config set-target https://$METADATA_STORE_DOMAIN:$METADATA_STORE_PORT --
ca-cert insight-ca.crt
```

Important

The tanzu insight config set-target does not initiate a test connection. Use tanzu insight health to test connecting using the configured endpoint and CA certificate. Neither commands test whether the access token is correct. For that you must use the plug-in to add data and query data.

Use your NodePort with Supply Chain Security Tools - Store

This topic describes how you can use your NodePort with Supply Chain Security Tools (SCST) - Store.

Overview

Note

The recommended service type is Ingress. NodePort is only recommended when the cluster does not support Ingress or the cluster does not support the LoadBalancer service type. `NodePort` is not supported for a multicluster setup, as certificates cannot be modified.

You must use port forwarding when using the `NodePort` configuration.

Configure port forwarding for the service so the insight plug-in can access SCST - Store. Run:

```
kubectl port-forward service/metadata-store-app 8443:8443 -n metadata-store
```

Note: You must run the port forwarding command in a separate terminal window, or run the command in the background: `kubectl port-forward service/metadata-store-app 8443:8443 -n metadata-store &`

Edit your `/etc/hosts` file for Port Forwarding

Use the following script to add a new local entry to `/etc/hosts`:

```
METADATA_STORE_PORT=$(kubectl get service/metadata-store-app --namespace metadata-stor
e -o jsonpath="{.spec.ports[0].port}")
METADATA_STORE_DOMAIN="metadata-store-app.metadata-store.svc.cluster.local"

# delete any previously added entry
sudo sed -i '' "/$METADATA_STORE_DOMAIN/d" /etc/hosts

echo "127.0.0.1 $METADATA_STORE_DOMAIN" | sudo tee -a /etc/hosts > /dev/null
```

Configure the Insight plug-in

Because you deployed Supply Chain Security Tools (SCST) - Store without using Ingress, you must use the Certificate resource `app-tls-cert` for HTTPS communication.

To get the CA Certificate:

```
kubectl get secret app-tls-cert -n metadata-store -o json | jq -r '.data."ca.crt"' | b
ase64 -d > insight-ca.crt
```

Set the target by running:

```
tanzu insight config set-target https://$METADATA_STORE_DOMAIN:$METADATA_STORE_PORT --
ca-cert insight-ca.crt
```

Important

> The `tanzu insight config set-target` does not initiate a test connection. Use `tanzu insight health` to test connecting using the configured endpoint and CA certificate. Neither commands test whether the access token is correct. For that you must use the plug-in to add data and query data.

Developer namespace setup for Supply Chain Security Tools - Store

This topic describes how you can set up your developer namespace for Supply Chain Security Tools (SCST) - Store.

Overview

After you finish the entire Tanzu Application Platform installation process, you are ready to configure the developer namespace. When you configure a developer namespace, you must export the Supply Chain Security Tools (SCST) - Store CA certificate and authentication token to the namespace. This enables SCST - Scan to find the credentials to send scan results to SCST - Store.

There are two ways to deploy Tanzu Application Platform:

- Single cluster, which entails using the Tanzu Application Platform values file
- Multicluster, which entails using `SecretExport`

Single cluster - Using the Tanzu Application Platform values file

When deploy the Tanzu Application Platform Full or Build profile, edit the `tap-values.yaml` file you used to deploy Tanzu Application Platform.

```
metadata_store:
  ns_for_export_app_cert: "DEV-NAMESPACE"
```

Where `DEV-NAMESPACE` is the name of the developer namespace.

The `ns_for_export_app_cert` supports one namespace at a time. If you have multiple namespaces you can replace this value with a `"*"`, but this exports the CA to all namespaces. Consider whether this increased visibility presents a risk.

```
metadata_store:
  ns_for_export_app_cert: "*"
```

Update Tanzu Application Platform to apply the changes by running:

```
$ tanzu package installed update tap -f tap-values.yaml -n tap-install
```

Multicluster - Using `SecretExport`

In a multicluster deployment, follow the steps in Multicluster setup. It describes how to create secrets and export secrets to developer namespaces.

Next steps

If you arrived in this topic from Setting up the Out of the Box Supply Chain with testing and scanning, return to that topic and continue with the instructions.

Retrieve access tokens for Supply Chain Security Tools - Store

This topic describes how you can retrieve access tokens for Supply Chain Security Tools (SCST) - Store.

Overview

When you install Tanzu Application Platform, the Supply Chain Security Tools (SCST) - Store deployment automatically includes a read-write service account. This service account is bound to the `metadata-store-read-write` role.

There are two types of SCST - Store service accounts:

1. Read-write service account - full access to the `POST` and `GET` API requests
2. Read-only service account - can only use `GET` API requests

This topic shows how to retrieve the access token for these service accounts.

Retrieving the read-write access token

To retrieve the read-write access token, run:

```
kubectl get secrets metadata-store-read-write-client -n metadata-store -o jsonpath="{.data.token}" | base64 -d
```

Retrieving the read-only access token

In order retrieve the read-only access token, you must first have a read-only service account. See Create read-only service account.

To retrieve the read-only access token, run:

```
kubectl get secrets metadata-store-read-client -n metadata-store -o jsonpath="{.data.token}" | base64 -d
```

Using an access token

The access token is a Bearer token used in the http request header `Authorization`. For example, `Authorization: Bearer eyJhbGciOiJSUzI1NiIsImtpZCI6IjhMV0....`

Additional Resources

- Create service accounts
- Create a service account with a custom cluster role

Retrieve and create service accounts for Supply Chain Security Tools - Store

This topic explains how you can create service accounts for Supply Chain Security Tools (SCST) - Store.

Overview

When you install Tanzu Application Platform, the Supply Chain Security Tools (SCST) - Store deployment automatically includes a read-write service account. This service account is bound to the `metadata-store-read-write` role.

There are two types of SCST - Store service accounts:

1. Read-write service account - full access to the `POST` and `GET` API requests
2. Read-only service account - can only use `GET` API requests

Create read-write service account

When you install Tanzu Application Platform, the SCST - Store deployment automatically includes a read-write service account. This service account is already bound to the `metadata-store-read-write` role.

To create an additional read-write service account, run the following command. The command creates a service account called `metadata-store-read-write-client`:

```
kubectl apply -f - -o yaml << EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: metadata-store-read-write
  namespace: metadata-store
rules:
- resources: ["all"]
  verbs: ["get", "create", "update"]
  apiGroups: [ "metadata-store/v1" ]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: metadata-store-read-write
  namespace: metadata-store
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: metadata-store-read-write
subjects:
- kind: ServiceAccount
  name: metadata-store-read-write-client
  namespace: metadata-store
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: metadata-store-read-write-client
  namespace: metadata-store
  annotations:
    kapp.k14s.io/change-group: "metadata-store.apps.tanzu.vmware.com/service-account"
automountServiceAccountToken: false
---
apiVersion: v1
kind: Secret
type: kubernetes.io/service-account-token
metadata:
  name: metadata-store-read-write-client
  namespace: metadata-store
  annotations:
    kapp.k14s.io/change-rule: "upsert after upserting metadata-store.apps.tanzu.vmwar
e.com/service-account"
```

```
    kubernetes.io/service-account.name: "metadata-store-read-write-client"
EOF
```

Create a read-only service account

You can create a read-only service account with a default cluster role or with a custom cluster role.

With a default cluster role

During Store installation, the `metadata-store-read-only` cluster role is created by default. This cluster role allows the bound user to have `get` access to all resources. To bind to this cluster role, run the following command:

```
kubectl apply -f - -o yaml << EOF
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: metadata-store-read-only
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: metadata-store-read-only
subjects:
- kind: ServiceAccount
  name: metadata-store-read-client
  namespace: metadata-store
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: metadata-store-read-client
  namespace: metadata-store
  annotations:
    kapp.k14s.io/change-group: "metadata-store.apps.tanzu.vmware.com/service-account"
automountServiceAccountToken: false
---
apiVersion: v1
kind: Secret
type: kubernetes.io/service-account-token
metadata:
  name: metadata-store-read-client
  namespace: metadata-store
  annotations:
    kapp.k14s.io/change-rule: "upsert after upserting metadata-store.apps.tanzu.vmwar
e.com/service-account"
    kubernetes.io/service-account.name: "metadata-store-read-client"
EOF
```

With a custom cluster role

If using the default role is not sufficient, see Create a service account with a custom cluster role.

Additional Resources

- Retrieve access tokens
- Create a service account with a custom cluster role

Create a service account with a custom cluster role for Supply Chain Security Tools - Store

This topic describes how you can create a service account with a custom cluster role for Supply Chain Security Tools (SCST)- Store.

Example service account

If you do not want to bind to the default cluster role, create a read-only role in the `metadata-store` namespace with a service account. The following example creates a service account named `metadata-store-read-client`:

```
kubectl apply -f - -o yaml << EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: metadata-store-ro
  namespace: metadata-store
rules:
- resources: ["all"]
  verbs: ["get"]
  apiGroups: [ "metadata-store/v1" ]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: metadata-store-ro
  namespace: metadata-store
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: metadata-store-ro
subjects:
- kind: ServiceAccount
  name: metadata-store-read-client
  namespace: metadata-store
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: metadata-store-read-client
  namespace: metadata-store
  annotations:
    kapp.k14s.io/change-group: "metadata-store.apps.tanzu.vmware.com/service-account"
automountServiceAccountToken: false
---
apiVersion: v1
kind: Secret
type: kubernetes.io/service-account-token
metadata:
  name: metadata-store-read-client
  namespace: metadata-store
  annotations:
    kapp.k14s.io/change-rule: "upsert after upserting metadata-store.apps.tanzu.vmwar
e.com/service-account"
    kubernetes.io/service-account.name: "metadata-store-read-client"
EOF
```

Additional Resources

- Retrieve access tokens

- Create service accounts

Install Supply Chain Security Tools - Store

This topic describes how you can install Supply Chain Security Tools (SCST) - Store independently from Tanzu Application Platform (commonly known as TAP) profiles.

Note

Follow the steps in this topic if you do not want to use a profile to install Supply Chain Security Tools - Store. For more information about profiles, see Components and installation profiles.

Prerequisites

Before installing SCST - Store from the Tanzu Application Platform package repository:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- Install cert-manager on the cluster. See Install cert-manager.
- See Deployment Details and Configuration to review what resources are deployed. For more information, see the overview.
- Create ClusterIssuer

```
kubectl apply -f - <<EOF
apiVersion: cert-manager.io/v1
  kind: ClusterIssuer
  metadata:
    name: tap-ingress-selfsigned
  spec:
    selfSigned: {}
EOF
```

Install

To install SCST - Store:

1. To use this deployment, the user must have set up the Kubernetes cluster to provision persistent volumes on demand. Ensure that a default storage class is available in your cluster. Verify whether default storage class is set in your cluster using `kubectl get storageClass`.

   ```
   kubectl get storageClass
   ```

   For example:

   ```
   $ kubectl get storageClass
   NAME                 PROVISIONER             RECLAIMPOLICY   VOLUMEBINDINGMODE
   ALLOWVOLUMEEXPANSION   AGE
   standard (default)   rancher.io/local-path   Delete          WaitForFirstConsum
   er   false                  7s
   ```

2. List version information for the package using `tanzu package available list`.

```
tanzu package available list metadata-store.apps.tanzu.vmware.com --namespace t
ap-install
```

For example:

```
$ tanzu package available list metadata-store.apps.tanzu.vmware.com --namespace
tap-install
- Retrieving package versions for metadata-store.apps.tanzu.vmware.com...
  NAME                          VERSION       RELEASED-AT
  metadata-store.apps.tanzu.vmware.com  1.0.2
```

3. (Optional) List all the available deployment configuration options.

```
tanzu package available get metadata-store.apps.tanzu.vmware.com/VERSION --valu
es-schema -n tap-install
```

Where `VERSION` is the your package version number.

For example:

```
$ tanzu package available get metadata-store.apps.tanzu.vmware.com/1.0.2 --valu
es-schema -n tap-install
| Retrieving package details for metadata-store.apps.tanzu.vmware.com/1.0.2...
  KEY                             DEFAULT
TYPE     DESCRIPTION
  pg_limit_memory                 4Gi
string   Memory limit for postgres container in metadata-store-db deployment
  tls.namespace
string   The targeted namespace for secret consumption by the HTTPProxy.
  add_default_rw_service_account  true
string   Adds a read-write service account which can be used to obtain access t
oken to use metadata-store CLI
  api_host                        localhost
string   The internal hostname for the metadata api endpoint. This will be used
by the kube-rbac-proxy sidecar.
  app_replicas                    1
integer  The number of replicas for the metadata-store-app
  ingress_domain
string   Domain to be used by the HTTPProxy ingress object. The "metadata-stor
e" subdomain will be prepended to the value provided. For example: "example.co
m" would become "metadata-store.example.com". Required if ingress_enabled is tr
ue.
  kube_rbac_proxy_limit_cpu       250m
string   CPU limit for kube-rbac-proxy container in the metadata-store-app depl
oyment
  pg_limit_cpu                    2Gi
string   CPU limit for postgres container in metadata-store-db deployment
  tls.server.minTLSVersion        VersionTLS12
string   Minimum TLS version supported. Value must match version names from htt
ps://golang.org/pkg/crypto/tls/#pkg-constants. (default "VersionTLS12")
  db_host                         metadata-store-db
string   The address to the postgres database host that the metadata-store app
uses to connect. The default is set to metadata-store-db which is the postgres
service name. Changing this does not change the postgres service name
  db_replicas                     1
integer  The number of replicas for the metadata-store-db
  pg_req_cpu                      1Gi
string   CPU request for postgres container in metadata-store-db deployment
  priority_class_name
string   If specified, this value is the name of the desired PriorityClass for
the metadata-store-db deployment
  tls.secretName
string   The name of secret for consumption by the HTTPProxy.
  db_ca_certificate
```

```
string   This should only be set in the case when 'deploy_internal_db' is 'fals
e'. Set this to the trusted CA Certificate that signed the Postgres DB TLS Cert
ificate
  db_password
string   The database user password. If no value is provided, a 32 character va
lue will be generated.
  db_port                         5432
string   The database port to use. This is the port to use when connecting to t
he database pod.
  app_limit_cpu                   250m
string   CPU limit for metadata-store-app container
  auth_proxy_host                 0.0.0.0
string   The binding ip address of the kube-rbac-proxy sidecar
  db_max_open_conns               10
integer  Sets the maximum number of open database connections from the Metadata
Store to the database.
  db_name                         metadata-store
string   The name of the database to use.
  db_user                         metadata-store-user
string   The database user to create and use for updating and querying. The met
adata postgres section create this user. The metadata api server uses this user
name to connect to the database.
  kube_rbac_proxy_req_memory      128Mi
string   Memory request for kube-rbac-proxy container in the metadata-store-app
deployment
  auth_proxy_port                 8443
integer  The external port address of the of the kube-rbac-proxy sidecar
  db_conn_max_lifetime            60
integer  Sets the maximum amount of time a database connection may be reused in
seconds.
  ingress_enabled                 false
string   Contour is required to be installed to use this flag. When true, this
creates an HTTPProxy object for the metadata-store. If false, then no ingress i
s configured.
  storage_class_name
string   The storage class name of the persistent volume used by Postgres datab
ase for storing data. The default value will use the default class name defined
on the cluster.
  api_port                        9443
integer  The internal port for the metadata app api endpoint. This will be used
by the kube-rbac-proxy sidecar.
  app_service_type                LoadBalancer
string   The type of service to use for the metadata app service. This can be s
et to 'Nodeport', 'ClusterIP' or 'LoadBalancer'.
  db_sslmode                      verify-full
string   Determines the security connection between API server and Postgres dat
abase. This can be set to 'verify-ca' or 'verify-full'
  use_cert_manager                true
string   Cert manager is required to be installed to use this flag. When true,
this creates certificates object to be signed by cert manager for the API serve
r and Postgres database. If false, the certificate object have to be provided b
y the user.
  app_req_cpu                     100m
string   CPU request for metadata-store-app container
  database_request_storage        10Gi
string   The storage requested of the persistent volume used by Postgres databa
se for storing data.
  deploy_internal_db              true
string   If set to 'true', a postgres deployment will be created. If set to 'fa
lse', db_host and db_port should point to an accessible postgres instance. Post
gres connections require TLS, so the corresponding db_ca_certification must be
provided
  kube_rbac_proxy_req_cpu         100m
string   CPU request for kube-rbac-proxy container in the metadata-store-app de
ployment
  ns_for_export_app_cert          scan-link-system
```

```
string   The namespace where the "Supply Chain Security Tools for VMware Tanzu
- Scan" component is installed in. Certain certificates will be exported to tha
t namespace so that scan reports can be posted to the Metadata Store.
  pg_req_memory                  1Gi
string   Memory request for postgres container in metadata-store-db deployment
  app_limit_memory               512Mi
string   Memory limit for metadata-store-app container
  app_req_memory                 128Mi
string   Memory request for metadata-store-app container
  db_max_idle_conns              100
integer  Sets the maximum number of database connections from the Metadata Stor
e in the idle connection pool.
  kube_rbac_proxy_limit_memory   512Mi
string   Memory limit for kube-rbac-proxy container in the metadata-store-app d
eployment
  kubernetes_distribution
string   Kubernetes platform distribution where the metadata-store is being ins
talled on. Accepted values: ["", "openshift"]
  log_level                      default
string   Sets the log level. This can be set to "minimum", "less", "default",
"more", "debug" or "trace". "minimum" currently does not output logs. "less" ou
tputs log configuration options only. "default" and "more" outputs API endpoint
access information. "debug" and "trade" outputs extended API endpoint

access information(such as body payload) and other debug information.
  tls.server.rfcCiphers          [TLS_AES_128_GCM_SHA256 TLS_AES_256_GCM_SHA38
4 TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA3
84 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384]
array    List of cipher suites for the server. Values are from tls package cons
tants (https://golang.org/pkg/crypto/tls/#pkg-constants). If omitted, the defau
lt Go cipher suites will be used
  ingress_issuer                 tap-ingress-selfsigned
string   tap-ingress-selfsigned is the default value when installed via any TAP
profile. When installing only the metadata-store package, a ClusterIssuer needs
to be installed and its name needs to be specified as this value.
```

4. (Optional) Edit one of the deployment configurations by creating a configuration YAML with the custom configuration values you want. For example, if your environment does not support `LoadBalancer`, and you want to use `ClusterIP`, then create a `metadata-store-values.yaml` and configure the `app_service_type` property.

   ```
   ---
   app_service_type: "ClusterIP"
   ```

   See Deployment details and configuration for more information about configuration options.

   For information about ingress and custom domain name support, see Ingress support.

5. Install the package using `tanzu package install`.

   ```
   tanzu package install metadata-store \
     --package metadata-store.apps.tanzu.vmware.com \
     --version VERSION \
     --namespace tap-install \
     --values-file metadata-store-values.yaml
   ```

   Where:

   - `--values-file` is an optional flag. Only use it to customize the deployment configuration.
   - `VERSION` is the package version number.

   For example:

```
$ tanzu package install metadata-store \
  --package metadata-store.apps.tanzu.vmware.com \
  --version 1.0.2 \
  --namespace tap-install \
  --values-file metadata-store-values.yaml

- Installing package 'metadata-store.apps.tanzu.vmware.com'
/ Getting namespace 'tap-install'
- Getting package metadata for 'metadata-store.apps.tanzu.vmware.com'
/ Creating service account 'metadata-store-tap-install-sa'
/ Creating cluster admin role 'metadata-store-tap-install-cluster-role'
/ Creating cluster role binding 'metadata-store-tap-install-cluster-rolebindin
g'
/ Creating secret 'metadata-store-tap-install-values'
| Creating package resource
- Package install status: Reconciling

Added installed package 'metadata-store' in namespace 'tap-install'
```

Prerequisite for Standalone SCST - Store

The following prerequisites are required to install SCST - Store individually, instead of using a profile:

- Install the SelfSigned Cluster Issuer:

```
# Install `tap-ingress-selfsigned` ClusterIssuer
cat << EOF > /tmp/tap-ingress-selfsigned-cluster-issuer.yaml
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
name: tap-ingress-selfsigned
spec:
selfSigned: {}
EOF

kapp deploy -a issuer -f /tmp/tap-ingress-selfsigned-cluster-issuer.yaml -y
```

- Install `SCST - Store` and update the certificate:

```
cat << EOF > /tmp/ingress-issuer-cert.yaml
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
name: ingress-cert
namespace: metadata-store
spec:
isCA: true
dnsNames:
  - metadata-store.${INGRESS_DOMAIN}
issuerRef:
  kind: ClusterIssuer
  name: tap-ingress-selfsigned
  group: cert-manager.io
secretName: ingress-cert
commonName: metadata-store-ca
EOF
kubectl apply -f /tmp/ingress-issuer-cert.yaml
# `ingress-cert` is deleted. It will be automatically created again with newly
applied ClusterIssuer cert
kubectl delete secret ingress-cert -n metadata-store
```

These steps ensure that you have valid certificate after `SCST-Store` installation is complete.

Supply Chain Security Tools - Store Database Index Corruption

This topic tells you how to troubleshoot the Metadata Store Database index issue after upgrading from v1.5 to a higher version for Supply Chain Security Tools (SCST) - Store.

Overview

The index corruption happened due to a base OS upgrade in our PostgreSQL Database images to address some of the CVE which includes a breaking change in `glibc locale` library.

- Tanzu Application Platform v1.5 and before: `Postgres-bionic-13:1.22.0` uses `ubuntu:18.04` which includes `ubuntu/glibc 2.27-3ubuntu1.6`.
- Tanzu Application Platform v1.6 and later: `Postgres-bionic-13:1.23.0` uses `ubuntu:22.04`, `jammy, jammy-20221101`, which includes `ubuntu/glibc 2.35-0ubuntu3.1`.

`glibc v2.28` includes a major update to the locale data, which can potentially corrupt indexes after upgrade. For more information, see the postgreSQL wiki.

Database Index Corruption issue reported in Metadata Store App Container logs

Symptom

The Metadata Store can't reconcile because the Metadata Store pod complains about a potential database index corruption issue.

```
kubectl logs metadata-store-app-pod_name -n metadata-store
```

Output

```
{"level":"error","ts":"2023-08-15T16:38:31.528115988Z","logger":"MetadataStore","ms
g":"unable to check index corruption since user is not a superuser to perform \"CREATE
EXTENSION amcheck\". Please create this extension and check for index corruption using
following sql command \"SELECT bt_index_check(oid) FROM pg_class WHERE relname in (SEL
ECT indexrelid::regclass::text FROM (SELECT indexrelid, indrelid, indcollation[i] coll
FROM pg_index, generate_subscripts(indcollation, 1) g(i)) s JOIN pg_collation c ON col
l=c.oid WHERE collprovider IN ('d', 'c') AND collname NOT IN ('C', 'POSIX'));\"","host
name":"metadata-store-app-77c9fb59c8-qplxt"}
{"level":"error","ts":"2023-08-15T16:38:31.528139637Z","logger":"MetadataStore","ms
g":"Found corrupted database indexes but unable to fix them","hostname":"metadata-stor
e-app-77c9fb59c8-qplxt","error":"unable to check index corruption since user is not a
superuser to perform \"CREATE EXTENSION amcheck\". Please create this extension and ch
eck for index corruption using following sql command \"SELECT bt_index_check(oid) FROM
pg_class WHERE relname in (SELECT indexrelid::regclass::text FROM (SELECT indexrelid,
indrelid, indcollation[i] coll FROM pg_index, generate_subscripts(indcollation, 1) g
(i)) s JOIN pg_collation c ON coll=c.oid WHERE collprovider IN ('d', 'c') AND collname
NOT IN ('C', 'POSIX'));\""}
```

Solutions

There are several ways to address this issue, including upgrading Tanzu Application Platform with updated schema values to connect to the PostgreSQL database directly and fixing the index manually. The following alternative approaches are below:

- Edit `auto_correct_db_indexes` property to true in `tap-values.yaml`

- Connect to the Metadata Store database with the same account used by the Metadata Store API
- Connect to the Metadata Store database with a superuser account and manually fix the index

Edit `auto_correct_db_indexes` property to true in `tap-values.yaml`

1. To change `auto_correct_db_indexes`, edit your `tap-values.yaml` and upgrade the Tanzu Application Platform profile. Change the property `auto_correct_db_indexes` value to `true`.
2. Upgrade Tanzu Application Platform.

   ```
   tanzu package installed update tap -p tap.tanzu.vmware.com -v {{ vars.tap_versi
   on }}  --values-file tap-values.yaml -n tap-install
   ```

3. Examine the Metadata Store API logs. You should not see any errors.

Connect to the Metadata Store database with same account used by Metadata Store API

1. Connect to the Postgres Database.
2. Run this SQL to re-index individual tables.

   ```
   REINDEX TABLE  "artifact_groups";
   REINDEX TABLE  "artifact_group_images";
   REINDEX TABLE  "images";
   REINDEX TABLE  "sources";
   REINDEX TABLE  "source_images";
   REINDEX TABLE  "packages";
   REINDEX TABLE  "image_packages";
   REINDEX TABLE  "vulnerabilities";
   REINDEX TABLE  "package_vulnerabilities";
   REINDEX TABLE  "source_packages";
   REINDEX TABLE  "artifact_group_sources";
   REINDEX TABLE  "method_types";
   REINDEX TABLE  "ratings";
   REINDEX TABLE  "package_managers";
   REINDEX TABLE  "artifact_group_labels";
   REINDEX TABLE  "analysis_instances";
   REINDEX TABLE  "vulnerability_analyses";
   REINDEX TABLE  "reports";
   REINDEX TABLE  "report_ratings";
   REINDEX TABLE  "report_packages_vulnerabilities";
   REINDEX TABLE  "index_migration_statuses";
   ```

3. Run the following SQL:

   ```
   INSERT INTO index_migration_statuses (created_at, updated_at, "version", statu
   s) VALUES(now(), now(), version(), true)
   ```

4. Examine the Metadata Store API logs. You should not see any errors.

Connect to the Metadata Store database with a superuser account and manually fix the index

1. Connect to the Postgres Database.
2. Install the extension on your PostgreSQL Database.

   ```
   CREATE EXTENSION amcheck
   ```

3. Run the following SQL:

```
SELECT indexrelid::regclass::text FROM (SELECT indexrelid, indrelid, indcollati
on[i] coll FROM pg_index, generate_subscripts(indcollation, 1) g(i)) s JOIN pg_
collation c ON coll=c.oid
WHERE collprovider IN ('d', 'c') AND collname NOT IN ('C', 'POSIX');
```

4. Run the following SQL when you see an error:

```
SELECT indexrelid::regclass::text FROM (SELECT indexrelid, indrelid, indcollati
on[i] coll FROM pg_index, generate_subscripts(indcollation, 1) g(i)) s JOIN pg_
collation c ON coll=c.oid
WHERE collprovider IN ('d', 'c') AND collname NOT IN ('C', 'POSIX');
```

5. Run the following SQL if the previous step causes an error:

```
REINDEX database "metadata-store"
```

6. Run the following SQL:

```
INSERT INTO index_migration_statuses (created_at, updated_at, "version", statu
s) VALUES(now(), now(), version(), true)
```

7. Examine the Metadata Store API logs. You should not see any errors.

Overview of Tanzu Application Platform Telemetry

Tanzu Application Platform Telemetry (commonly known as TAP Telemetry) is a set of objects that collect data about the usage of Tanzu Application Platform (commonly known as TAP) and send it back to VMware for product improvements.

A benefit of remaining enrolled in telemetry and identifying your company during Tanzu Application Platform installation is that VMware can provide your organization with usage reports about Tanzu Application Platform.

For more information about enrolling in telemetry reports, see Tanzu Application Platform usage reports.

For more information about how to install the telemetry component, see Install Tanzu Application Platform Telemetry.

Tanzu Application Platform usage reports

VMware offers the option to enroll in a usage reporting program that offers a summary of usage of your Tanzu Application Platform. These reports provide an overview of various deployments of Tanzu Application Platform and their utilization. They encompass details such as the quantity of workloads and supply chains, and their health status. Additionally, the reports include information about the virtual CPU (vCPU) usage for each Tanzu Application Platform deployment. You can enroll in the program by providing the Entitlement Account Number (EAN). An EAN is a unique ID assigned to all VMware customers. VMware uses EAN to identify data about Tanzu Application Platform.

After locating the EAN, pass the number under the telemetry header in the `tap-values.yaml` file as a value for the `customer_entitlement_account_number` key.

```
tap_telemetry:
  customer_entitlement_account_number: "CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER"
```

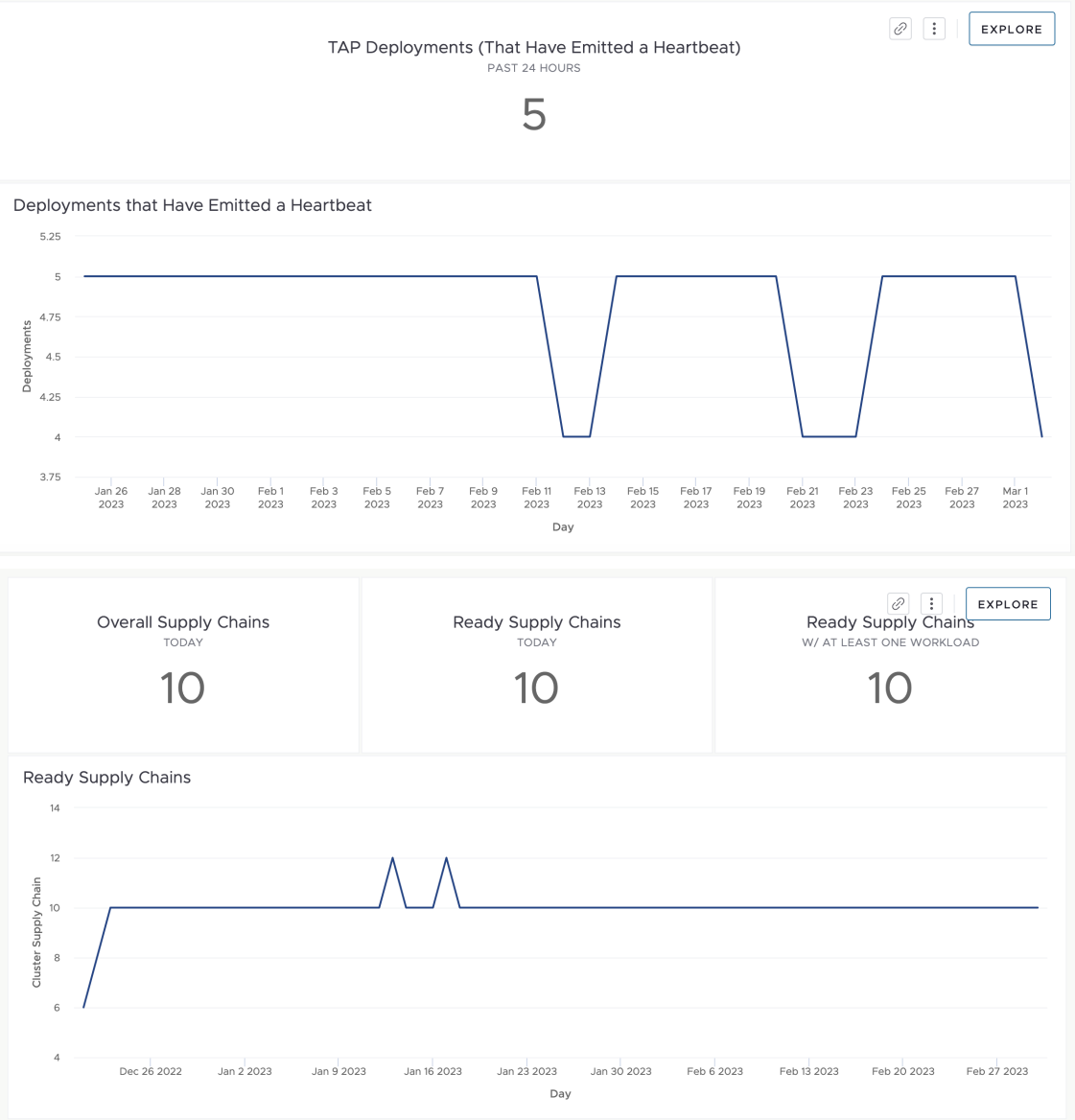
You must repeat the process for each Tanzu Application Platform Cluster included in the telemetry report. For more information, see Full profile.

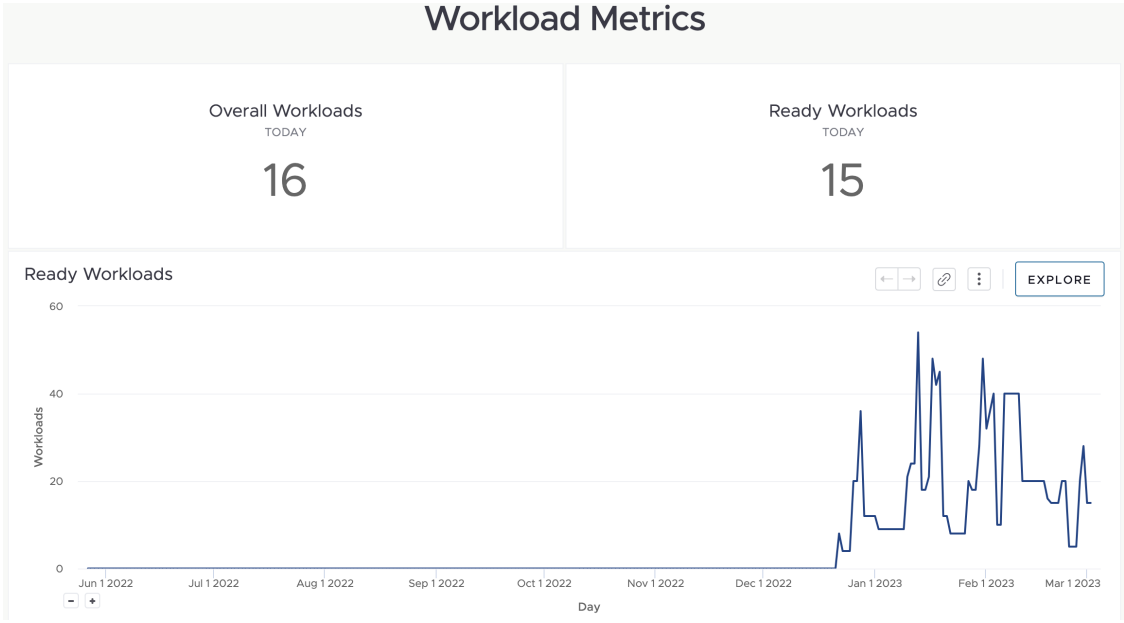
After enrollment, alert your VMware account team that you have configured the EAN field and want telemetry reports. This allows VMware to identify who the newly added EAN belongs to.

Note

Usage report is only supported for non-airgapped deployments of Tanzu Application Platform and the Cluster must participate in Tanzu Application Platform telemetry. You are enrolled in telemetry by default. You can opt out of telemetry collection by following the instructions in Opt out of telemetry collection.

The following screenshots show the sample telemetry reports.





Overview of Tanzu Application Platform Telemetry

Tanzu Application Platform Telemetry (commonly known as TAP Telemetry) is a set of objects that collect data about the usage of Tanzu Application Platform (commonly known as TAP) and send it back to VMware for product improvements.

A benefit of remaining enrolled in telemetry and identifying your company during Tanzu Application Platform installation is that VMware can provide your organization with usage reports about Tanzu Application Platform.

For more information about enrolling in telemetry reports, see Tanzu Application Platform usage reports.

For more information about how to install the telemetry component, see Install Tanzu Application Platform Telemetry.

Tanzu Application Platform usage reports

VMware offers the option to enroll in a usage reporting program that offers a summary of usage of your Tanzu Application Platform. These reports provide an overview of various deployments of Tanzu Application Platform and their utilization. They encompass details such as the quantity of workloads and supply chains, and their health status. Additionally, the reports include information about the virtual CPU (vCPU) usage for each Tanzu Application Platform deployment. You can enroll in the program by providing the Entitlement Account Number (EAN). An EAN is a unique ID assigned to all VMware customers. VMware uses EAN to identify data about Tanzu Application Platform.

After locating the EAN, pass the number under the telemetry header in the `tap-values.yaml` file as a value for the `customer_entitlement_account_number` key.

```
tap_telemetry:
  customer_entitlement_account_number: "CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER"
```

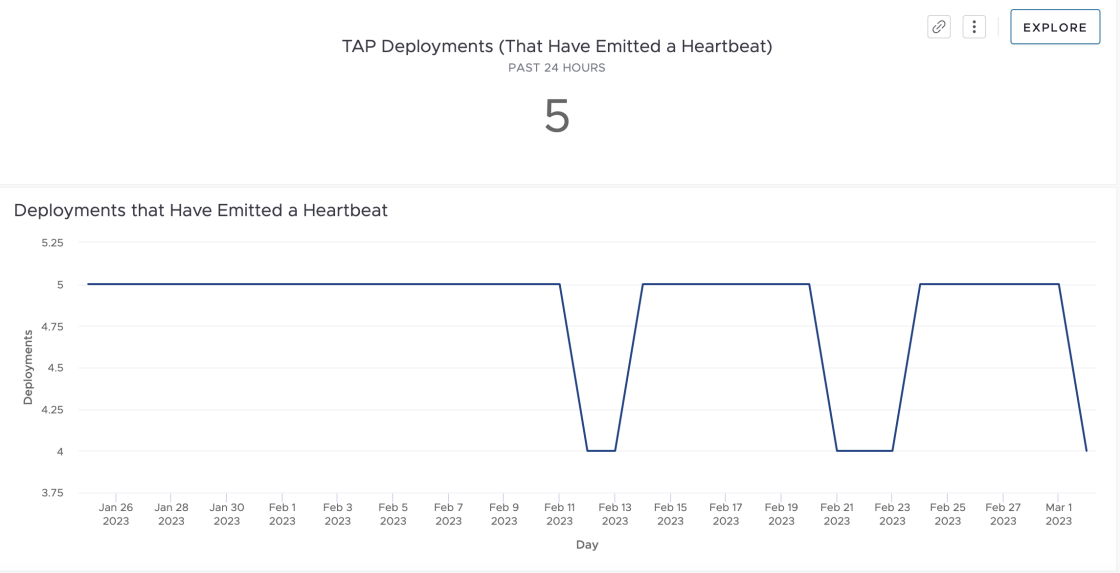
You must repeat the process for each Tanzu Application Platform Cluster included in the telemetry report. For more information, see Full profile.

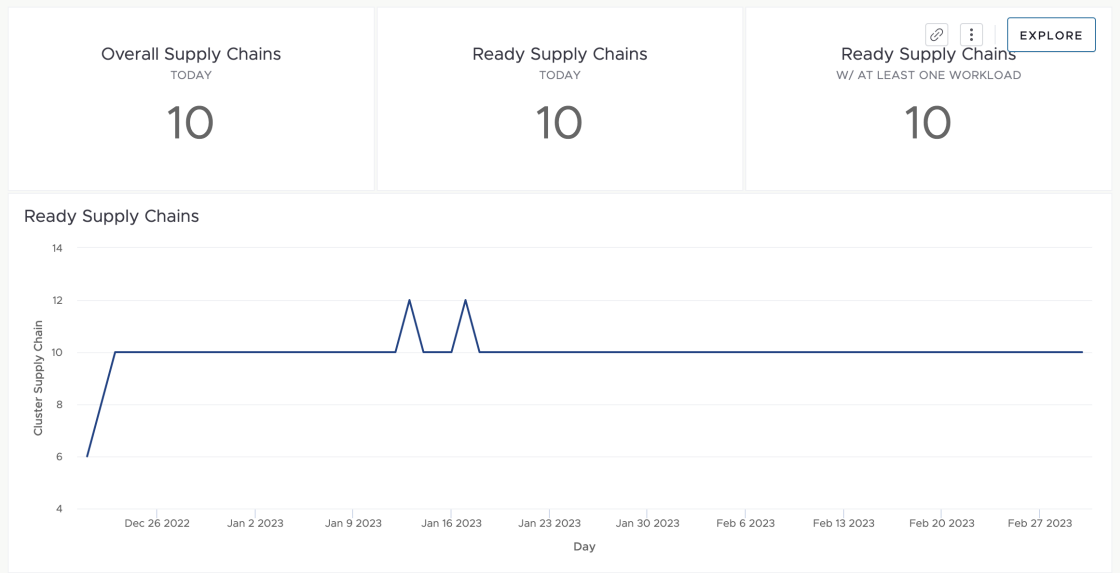
After enrollment, alert your VMware account team that you have configured the EAN field and want telemetry reports. This allows VMware to identify who the newly added EAN belongs to.

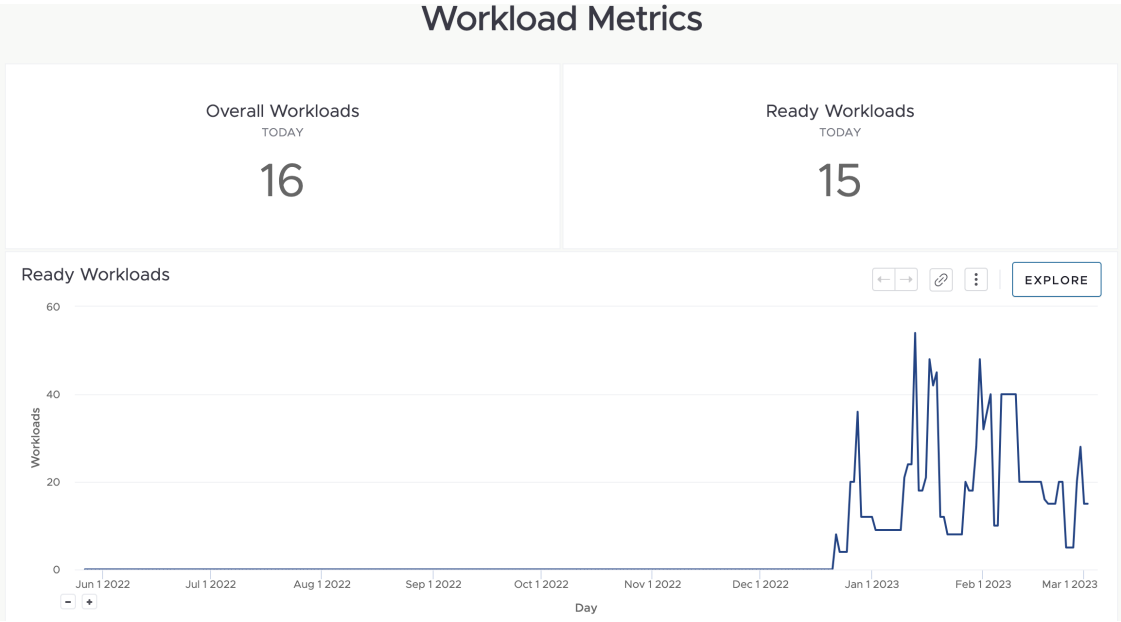
Note

Usage report is only supported for non-airgapped deployments of Tanzu Application Platform and the Cluster must participate in Tanzu Application Platform telemetry. You are enrolled in telemetry by default. You can opt out of telemetry collection by following the instructions in Opt out of telemetry collection.

The following screenshots show the sample telemetry reports.







Install Tanzu Application Platform Telemetry

This topic tells you how to install Tanzu Application Platform Telemetry from the Tanzu Application Platform (commonly known as TAP) package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Telemetry. For more information about profiles, see Components and installation profiles.

Prerequisites

Before installing Tap Telemetry:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- Install cert-manager on the cluster. For more information, see the cert-manager documentation.
- See Deployment Details and Configuration to review what resources will be deployed.

Install

To install Tanzu Application Platform Telemetry:

1. List version information for the package by running:

   ```
   tanzu package available list tap-telemetry.tanzu.vmware.com --namespace tap-ins
   tall
   ```

   For example:

   ```
   $ tanzu package available list tap-telemetry.tanzu.vmware.com --namespace tap-i
   nstall
   - Retrieving package versions for tap-telemetry.tanzu.vmware.com...
   ```

```
  NAME                           VERSION       RELEASED-AT
  tap-telemetry.tanzu.vmware.com  0.3.1
```

2. (Optional) List all the available deployment configuration options:

   ```
   tanzu package available get tap-telemetry.tanzu.vmware.com/VERSION --values-sch
   ema -n tap-install
   ```

   Where `VERSION` is the your package version number. For example, `0.3.1`.

   For example:

   ```
   $ tanzu package available get tap-telemetry.tanzu.vmware.com/0.3.1 --values-sch
   ema -n tap-install
   | Retrieving package details for tap-telemetry.tanzu.vmware.com/0.3.1...
   KEY                                  DEFAULT  TYPE    DESCRIPTION
   kubernetes_distribution                       string  Kubernetes platform flavo
   r where the tap-telemetry is being installed on. Accepted values are ['', 'open
   shift']
   customer_entitlement_account_number           string  Account number used to di
   stinguish data by customer.
   installed_for_vmware_internal_use             string  Indication of if the depl
   oyment is for vmware internal user. Accepted values are ['true', 'false']
   ```

3. (Optional) Modify the deployment configurations by creating a configuration YAML with the desired custom configuration values. For example, if you want to provide your Customer Entitlement Number, create a `tap-telemetry-values.yaml` and configure the `customer_entitlement_account_number` property:

   ```
   ---
   customer_entitlement_account_number: "12345"
   ```

   See Deployment details and configuration for more information about the configuration options.

4. Install the package by running:

   ```
   tanzu package install tap-telmetry \
     --package tap-telemetry.tanzu.vmware.com \
     --version VERSION \
     --namespace tap-install \
     --values-file tap-telemetry-values.yaml
   ```

   Where:

   - `--values-file` is an optional flag. Only use it to customize the deployment configuration.
   - `VERSION` is the package version number. For example, `0.3.1`.

   For example:

   ```
   $ tanzu package install tap-telmetry \
     --package tap-telemetry.tanzu.vmware.com \
     --version 0.3.1 \
     --namespace tap-install \
     --values-file tap-telemetry-values.yaml

     Installing package 'tap-telemetry.tanzu.vmware.com'
     Getting package metadata for 'tap-telemetry.tanzu.vmware.com'
     Creating service account 'tap-telemetry-tap-install-sa'
     Creating cluster admin role 'tap-telemetry-tap-install-cluster-role'
     Creating cluster role binding 'tap-telemetry-tap-install-cluster-rolebinding'
   ```

```
  Creating secret 'tap-telemetry-tap-install-values'
  Creating package resource
  Waiting for 'PackageInstall' reconciliation for 'tap-telemetry'
  'PackageInstall' resource install status: Reconciling
  'PackageInstall' resource install status: ReconcileSucceeded
  'PackageInstall' resource successfully reconciled

Added installed package 'tap-telemetry'
```

Deployment details and configurations of Tanzu Application Platform Telemetry

Use this topic to learn the deployment details and configurations of your Tanzu Application Platform Telemetry (commonly known as TAP Telemetry).

What is deployed

The installation creates the following in your Kubernetes cluster:

- A deployment.
- A pod.
- A namespace `tap-telemetry`.
- A service account with read-write privileges named `informer`, and a corresponding secret for the service account. This secret is bound to a ClusterRole named `tap-telemetry-admin`.
- A Role `tap-telemetry-informer` to retrieve the deployment ID, which is sent as sender ID in heartbeat metrics.
- A RoleBinding `tap-telemetry-informer-admin` that binds the `informer` service account to the `tap-telemetry-informer` role.
- A ClusterRole `tap-telemetry-admin` that has access to each Tanzu Application Platform component to gather information from.
- A ClusterRoleBinding `tap-telemetry-informer-admin` that binds the `informer` service account to the `tap-telemetry-informer` cluster role.

Deployment configuration

`customer_entitlement_account_number` is the unique identifier to differentiate between the data from your cluster and the data from other clusters. You can configure this property in your `tap-telemetry-values.yaml`:

```
customer_entitlement_account_number: "12345"
```

It creates a config map named `vmware-telemetry-identifiers` in the `vmware-system-telemetry` namespace, which is used internally to log your information.

Repeat these steps for the Build, Run, and View Cluster. For more information, see Install multicluster Tanzu Application Platform profiles.

Overview of Tanzu Build Service

This topic provides you with an overview of VMware Tanzu Build Service in Tanzu Application Platform (commonly known as TAP).

Overview

Tanzu Build Service automates container creation, management, and governance at enterprise scale. Tanzu Build Service uses the open-source Cloud Native Buildpacks project to turn application source code into container images. It executes reproducible builds aligned with modern container standards and keeps images up to date.

Use the Tanzu Build Service CLI plug-in to view all the Tanzu Build Service resources on any Kubernetes cluster that has Tanzu Application Platform or Tanzu Build Service installed. For information about how to install the Tanzu CLI and plug-ins, see Install Tanzu CLI and Overview of Tanzu CLI.

For more information about Tanzu Build Service, see the Tanzu Build Service documentation. For more information about Tanzu Buildpacks and their configuration, see the Tanzu Buildpack documentation.

Tanzu Application Platform v1.7 includes Tanzu Build Service v1.12.

Overview of Tanzu Build Service

This topic provides you with an overview of VMware Tanzu Build Service in Tanzu Application Platform (commonly known as TAP).

Overview

Tanzu Build Service automates container creation, management, and governance at enterprise scale. Tanzu Build Service uses the open-source Cloud Native Buildpacks project to turn application source code into container images. It executes reproducible builds aligned with modern container standards and keeps images up to date.

Use the Tanzu Build Service CLI plug-in to view all the Tanzu Build Service resources on any Kubernetes cluster that has Tanzu Application Platform or Tanzu Build Service installed. For information about how to install the Tanzu CLI and plug-ins, see Install Tanzu CLI and Overview of Tanzu CLI.

For more information about Tanzu Build Service, see the Tanzu Build Service documentation. For more information about Tanzu Buildpacks and their configuration, see the Tanzu Buildpack documentation.

Tanzu Application Platform v1.7 includes Tanzu Build Service v1.12.

Install Tanzu Build Service

This topic describes how to install Tanzu Build Service from the Tanzu Application Platform (commonly known as TAP) package repository by using the Tanzu CLI.

Before you begin

Use this topic if you do not want to use a Tanzu Application Platform profile that includes Tanzu Build Service. The Full, Iterate, and Build profiles include Tanzu Build Service. For more information about profiles, see Components and installation profiles.

The following procedure might not include some configurations required for your environment. For advanced information about installing Tanzu Build Service, see the Tanzu Build Service documentation.

Prerequisites

Before installing Tanzu Build Service:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- You must have access to a Docker registry that Tanzu Build Service can use to create builder images. Approximately 10 GB of registry space is required when using the `full` dependencies.
- Your Docker registry must be accessible with user name and password credentials.

Deprecated Features

The Cloud Native Buildpack Bill of Materials (CNB BOM) format has been removed.

Install the Tanzu Build Service package

To install Tanzu Build Service by using the Tanzu CLI:

1. Get the latest version of the Tanzu Build Service package by running:

   ```
   tanzu package available list buildservice.tanzu.vmware.com --namespace tap-inst
   all
   ```

2. Gather the values schema by running:

   ```
   tanzu package available get buildservice.tanzu.vmware.com/VERSION --values-sche
   ma --namespace tap-install
   ```

   Where `VERSION` is the version of the Tanzu Build Service package you retrieved earlier in this procedure.

3. Create the secret for the `kp-default-repository` credentials using the `tanzu` cli:

   ```
   tanzu secret registry add kp-default-repository-creds \
     --server "${REGISTRY-HOSTNAME}" \
     --username "${REGISTRY-USERNAME}" \
     --password "${REGISTRY-PASSWORD}" \
     --namespace tap-install
   ```

   Where: - `REGISTRY-HOST` is the host name for the registry that contains your `kp_default_repository`. For example: - Harbor has the form `server: "my-harbor.io"`. - Docker Hub has the form `server: "index.docker.io"`. - Google Cloud Registry has the form `server: "gcr.io"`. - `REGISTRY-USERNAME` and `REGISTRY-PASSWORD` are the user name and password for the user that can write to the repository used in the following step. For Google Cloud Registry, use `_json_key` as the user name and the contents of the service account JSON file for the password.

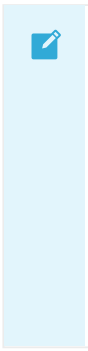
4. Create a `tbs-values.yaml` file using the following template. If `shared.image_registry.project_path` and `shared.image_registry.secret` are configured in the `tap-values.yaml` file, Tanzu Build Service inherits all three values in that section. This can be disabled by setting any of the following three values.

   ```
   ---
   kp_default_repository: "REPO-NAME"
   kp_default_repository_secret:
     name: kp-default-repository-creds
     namespace: tap-install
   ```

Where: - `REPO-NAME` is a writable repository in your registry. Tanzu Build Service dependencies are written to this location. Examples: - Harbor has the form `"my-harbor.io/my-project/build-service"`. - Docker Hub has the form `"my-dockerhub-user/build-service"` or `"index.docker.io/my-user/build-service"`. - Google Cloud Registry has the form `"gcr.io/my-project/build-service"`.

5. If you are running on OpenShift, add `kubernetes_distribution: openshift` to your `tbs-values.yaml` file.

6. (Optional) Under the `ca_cert_data` key in the `tbs-values.yaml` file, provide a PEM-encoded CA certificate for Tanzu Build Service. This certificate is used for accessing the container image registry and is also provided to the build process.

   Note

   If `shared.ca_cert_data` is configured in the `tap-values.yaml` file, Tanzu Build Service inherits that value.

   Configuring `ca_cert_data` key in the `tbs-values.yaml` file adds the CA certificates at build time. To add CA certificates to the built image, see Configure custom CA certificates for a single workload using service bindings.

   For example:

   ```
   ---
   kp_default_repository: "REPO-NAME"
   kp_default_repository_secret:
     name: kp-default-repository-creds
     namespace: tap-install
   ca_cert_data: |
     -----BEGIN CERTIFICATE-----
     ...
     -----END CERTIFICATE-----
   ```

7. (Optional) Tanzu Build Service is bootstrapped with the `lite` set of dependencies. To configure `full` dependencies, add the key-value pair `exclude_dependencies: true` to your `tbs-values.yaml` file. This is to exclude the default `lite` dependencies from the installation. For example:

   ```
   ---
   kp_default_repository: "REPO-NAME"
   kp_default_repository_secret:
     name: kp-default-repository-creds
     namespace: tap-install
   exclude_dependencies: true
   ```

   For more information about the differences between `full` and `lite` dependencies, see About lite and full dependencies.

8. Install the Tanzu Build Service package by running:

   ```
   tanzu package install tbs \
     --package buildservice.tanzu.vmware.com \
     --version VERSION \
     --namespace tap-install \
     --values-file tbs-values.yaml
   ```

   Where `VERSION` is the version of the Tanzu Build Service package you retrieved earlier.

For example:

```
$ tanzu package install tbs \
    --package buildservice.tanzu.vmware.com \
    --version 1.12.4 \
    --namespace tap-install \
    --values-file tbs-values.yaml

| Installing package 'buildservice.tanzu.vmware.com'
| Getting namespace 'tap-install'
| Getting package metadata for 'buildservice.tanzu.vmware.com'
| Creating service account 'tbs-tap-install-sa'
| Creating cluster admin role 'tbs-tap-install-cluster-role'
| Creating cluster role binding 'tbs-tap-install-cluster-rolebinding'
| Creating secret 'tbs-tap-install-values'
- Creating package resource
- Package install status: Reconciling

 Added installed package 'tbs' in namespace 'tap-install'
```

9. (Optional) Verify the cluster builders that the Tanzu Build Service installation created by running:

   ```
   tanzu package installed get tbs -n tap-install
   ```

10. If you configured `full` dependencies in your `tbs-values.yaml` file, install the `full` dependencies by following the procedure in Install full dependencies.

Use AWS IAM authentication for registry credentials

Tanzu Build Service supports using AWS IAM roles to authenticate with Amazon Elastic Container Registry (ECR) on Amazon Elastic Kubernetes Service (EKS) clusters.

To use AWS IAM authentication:

1. Configure an AWS IAM role that has read and write access to the repository in the container image registry used when installing Tanzu Application Platform.
2. Use the following alternative configuration for `tbs-values.yaml`:

   > **Note**
   >
   > if you are installing Tanzu Build Service as part of a Tanzu Application Platform profile, you configure this in your `tap-values.yaml` file under the `buildservice` section.

   ```
   ---
     kp_default_repository: "REPO-NAME"
     kp_default_repository_aws_iam_role_arn: "IAM-ROLE-ARN"
   ```

   Where:

   - `REPO-NAME` is a writable repository in your registry. Tanzu Build Service dependencies are written to this location.
   - `IAM-ROLE-ARN` is the AWS IAM role Amazon Resource Name (ARN) for the role configured earlier in this procedure. For example, `arn:aws:iam::xyz:role/my-install-role`.

3. The developer namespace requires configuration for Tanzu Application Platform to use AWS IAM authentication for ECR. Configure an AWS IAM role that has read and write

access to the registry for storing workload images.

4. Using the supply chain service account, add an annotation including the role ARN configured earlier by running:

   ```
   kubectl annotate serviceaccount -n DEVELOPER-NAMESPACE SERVICE-ACCOUNT-NAME \
     eks.amazonaws.com/role-arn=IAM-ROLE-ARN
   ```

   Where:

   - `DEVELOPER-NAMESPACE` is the namespace where workloads are created.
   - `SERVICE-ACCOUNT-NAME` is the supply chain service account. This is `default` if unset.
   - `IAM-ROLE-ARN` is the AWS IAM role ARN for the role configured earlier. For example, `arn:aws:iam::xyz:role/my-developer-role`.

5. Apply this configuration by continuing the steps in Install the Tanzu Build Service package.

Install full dependencies

If you configured `full` dependencies in your `tbs-values.yaml` file, you must install the `full` dependencies package. For a more information about `lite` and `full` dependencies, see About lite and full dependencies.

To install `full` Tanzu Build Service dependencies:

1. Get the latest version of the Tanzu Application Platform package by running:

   ```
   tanzu package available list tap.tanzu.vmware.com --namespace tap-install
   ```

2. If you have not done so already, you must exclude the default dependencies by adding the key-value pair `exclude_dependencies: true` to your `tap-values.yaml` file under the `buildservice` section. For example:

   ```
   buildservice:
     exclude_dependencies: true
   ```

3. If you have not updated your Tanzu Application Platform package installation after adding the key-value pair `exclude_dependencies: true` to your values file, perform the update by running:

   ```
   tanzu package installed update tap --namespace tap-install --values-file VALUES
   -FILE
   ```

   Where `VALUES-FILE` is the path to the `tap-values.yaml` file you edited earlier.

4. Relocate the Tanzu Build Service `full` dependencies package repository by doing one of the following:

   - Relocate the images directly for online installation:

     ```
     imgpkg copy \
       -b registry.tanzu.vmware.com/tanzu-application-platform/full-deps-packa
     ge-repo:VERSION \
       --to-repo ${INSTALL_REGISTRY_HOSTNAME}/full-deps-package-repo
     ```

     Where `VERSION` is the version of the Tanzu Application Platform package you retrieved earlier.

   - Relocate the images to an external storage device and then to the registry in the air-gapped environment:

```
imgpkg copy \
  -b registry.tanzu.vmware.com/tanzu-application-platform/full-deps-packa
ge-repo:VERSION \
  --to-tar=full-deps-package-repo.tar

# move full-deps-package-repo.tar to environment with registry access
imgpkg copy \
  --tar full-deps-package-repo.tar \
  --to-repo=INSTALL-REGISTRY-HOSTNAME/TARGET-REPOSITORY/full-deps-package
-repo
```

Where:

- `VERSION` is the version of the Tanzu Application Platform package you retrieved earlier.
- `INSTALL-REGISTRY-HOSTNAME` is your container registry.
- `TARGET-REPOSITORY` is your target repository.

5. Add the Tanzu Build Service `full` dependencies package repository by running:

```
tanzu package repository add full-deps-package-repo \
  --url INSTALL-REGISTRY-HOSTNAME/TARGET-REPOSITORY/full-deps-package-repo:VERS
ION \
  --namespace tap-install
```

Where:

- `INSTALL-REGISTRY-HOSTNAME` is your container registry.
- `TARGET-REPOSITORY` is your target repository.
- `VERSION` is the version of the Tanzu Application Platform package you retrieved earlier.

6. Create a new `tbs-full-deps-values.yaml` and copy the `kp_default_repository` key-value pair from your `tap-values.yaml` or `tbs-values.yaml`:

```
---
 kp_default_repository: "REPO-NAME"
 kp_default_repository_secret:
   name: kp-default-repository-creds
   namespace: tap-install
```

Where `REPO-NAME` is copied from the `buildservice.kp_default_repository` field in your `tap-values.yaml` or `tbs-values.yaml`.

1. (Optional) Install the UBI builder.

   The UBI builder uses Red Hat Universal Base Image (UBI) v8 for both build and run images. This builder only supports Java and Node.js. To install the UBI builder, add the key-value pair `enable_ubi_builder: true` to your `tbs-full-deps-values.yaml`.

   ```
   ---
   enable_ubi_builder: true
   ```

2. (Optional) Install the Static builder.

   The Static builder uses Ubuntu Jammy for both build images and a minimal static run image. This builder only supports Golang. To install the Static builder, add the key-value pair `enable_static_builder: true` to your `tbs-full-deps-values.yaml`.

```
---
enable_static_builder: true
```

7. Install the `full` dependencies package by running:

```
tanzu package install full-deps \
  --package full-deps.buildservice.tanzu.vmware.com \
  --version "> 0.0.0" \
  --namespace tap-install \
  --values-file VALUES-FILE
```

   Where `VALUES-FILE` is the path to the `tbs-full-deps-values.yaml` you created earlier.

Install Tanzu Build Service on an air-gapped environment

This topic tells you how to install Tanzu Build Service on a Kubernetes cluster and registry that are air-gapped from external traffic.

Before you begin

Use this topic if you do not want to use a Tanzu Application Platform profile that includes Tanzu Build Service.

The Full, Iterate, and Build profiles include Tanzu Build Service. For more information about profiles, see Components and installation profiles.

Prerequisites

Before installing Tanzu Build Service:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- You must have access to a Docker registry that Tanzu Build Service can use to create builder images. Approximately 10 GB of registry space is required when using the `full` dependencies.
- Your Docker registry must be accessible with user name and password credentials.

Deprecated Features

The Cloud Native Buildpack Bill of Materials (CNB BOM) format has been removed.

Install the Tanzu Build Service package

These steps assume that you have installed the Tanzu Application Platform packages in your air-gapped environment.

To install the Tanzu Build Service package on an air-gapped environment:

1. Get the latest version of the Tanzu Build Service package by running:

```
tanzu package available list buildservice.tanzu.vmware.com --namespace tap-inst
all
```

2. Gather the values schema by running:

```
tanzu package available get buildservice.tanzu.vmware.com/VERSION --values-sche
```

```
ma --namespace tap-install
```

Where `VERSION` is the version of the Tanzu Build Service package you retrieved in the previous step.

3. Create a `tbs-values.yaml` file. The required fields for an air-gapped installation are as follows:

```
---
kp_default_repository: REPO-NAME
kp_default_repository_secret:
  name: "SECRET_NAME"
  namespace: "SECRET_NAMESPACE"
ca_cert_data: CA-CERT-CONTENTS
exclude_dependencies: true
```

Where:

- `REPO-NAME` is the fully qualified path to a writeable repository in your internal registry. Tanzu Build Service dependencies are written to this location. For example:
  - For Harbor: `harbor.io/my-project/build-service`
  - For Artifactory: `artifactory.com/my-project/build-service`
- `SECRET_NAME` is the name of the secret containing credentials that can write to `REPO-NAME`.
- `SECRET_NAMESPACE` is the namespace of the secret containing credentials that can write to `REPO-NAME`.
- `CA-CERT-CONTENTS` are the contents of the PEM-encoded CA certificate for the internal registry.

4. Install the package by running:

```
tanzu package install tbs \
  --package buildservice.tanzu.vmware.com \
  --version VERSION \
  --namespace tap-install \
  --values-file tbs-values.yaml
```

Where `VERSION` is the version of the Tanzu Build Service package you retrieved earlier.

For example:

```
$ tanzu package install tbs \
    --package buildservice.tanzu.vmware.com \
    --version 1.12.4 \
    --namespace tap-install \
    --values-file tbs-values.yaml

| Installing package 'buildservice.tanzu.vmware.com'
| Getting namespace 'tap-install'
| Getting package metadata for 'buildservice.tanzu.vmware.com'
| Creating service account 'tbs-tap-install-sa'
| Creating cluster admin role 'tbs-tap-install-cluster-role'
| Creating cluster role binding 'tbs-tap-install-cluster-rolebinding'
| Creating secret 'tbs-tap-install-values'
- Creating package resource
- Package install status: Reconciling
 Added installed package 'tbs' in namespace 'tap-install'
```

Install the Tanzu Build Service dependencies

Important

By default, Tanzu Build Service is installed with the `lite` dependencies.

When installing Tanzu Build Service in an air-gapped environment, the `lite` dependencies are not available because they require Internet access. You must install the `full` dependencies.

To install `full` dependencies:

1. Get the latest version of the Tanzu Application Platform package by running:

   ```
   tanzu package available list tap.tanzu.vmware.com --namespace tap-install
   ```

2. If you have not done so already, you must exclude the default dependencies by adding the key-value pair `exclude_dependencies: true` to your `tap-values.yaml` file under the `buildservice` section. For example:

   ```
   buildservice:
     exclude_dependencies: true
   ```

3. If you have not updated your Tanzu Application Platform package installation after adding the key-value pair `exclude_dependencies: true` to your values file, perform the update by running:

   ```
   tanzu package installed update tap --namespace tap-install --values-file VALUES
   -FILE
   ```

   Where `VALUES-FILE` is the path to the `tap-values.yaml` file you edited earlier.

4. Relocate the Tanzu Build Service `full` dependencies package repository by doing one of the following:

   - Relocate the images directly for online installation:

     ```
     imgpkg copy \
       -b registry.tanzu.vmware.com/tanzu-application-platform/full-deps-packa
     ge-repo:VERSION \
       --to-repo ${INSTALL_REGISTRY_HOSTNAME}/full-deps-package-repo
     ```

     Where `VERSION` is the version of the Tanzu Application Platform package you retrieved earlier.

   - Relocate the images to an external storage device and then to the registry in the air-gapped environment:

     ```
     imgpkg copy \
       -b registry.tanzu.vmware.com/tanzu-application-platform/full-deps-packa
     ge-repo:VERSION \
       --to-tar=full-deps-package-repo.tar

     # move full-deps-package-repo.tar to environment with registry access
     imgpkg copy \
       --tar full-deps-package-repo.tar \
       --to-repo=INSTALL-REGISTRY-HOSTNAME/TARGET-REPOSITORY/full-deps-package
     -repo
     ```

     Where:

     - `VERSION` is the version of the Tanzu Application Platform package you retrieved earlier.

- `INSTALL-REGISTRY-HOSTNAME` is your container registry.
- `TARGET-REPOSITORY` is your target repository.

5. Add the Tanzu Build Service `full` dependencies package repository by running:

```
tanzu package repository add full-deps-package-repo \
  --url INSTALL-REGISTRY-HOSTNAME/TARGET-REPOSITORY/full-deps-package-repo:VERS
ION \
  --namespace tap-install
```

Where:

- `INSTALL-REGISTRY-HOSTNAME` is your container registry.
- `TARGET-REPOSITORY` is your target repository.
- `VERSION` is the version of the Tanzu Application Platform package you retrieved earlier.

6. Create a new `tbs-full-deps-values.yaml` and copy the `kp_default_repository` key-value pair from your `tap-values.yaml` or `tbs-values.yaml`:

```
---
 kp_default_repository: "REPO-NAME"
 kp_default_repository_secret:
   name: kp-default-repository-creds
   namespace: tap-install
```

Where `REPO-NAME` is copied from the `buildservice.kp_default_repository` field in your `tap-values.yaml` or `tbs-values.yaml`.

1. (Optional) Install the UBI builder.

   The UBI builder uses Red Hat Universal Base Image (UBI) v8 for both build and run images. This builder only supports Java and Node.js. To install the UBI builder, add the key-value pair `enable_ubi_builder: true` to your `tbs-full-deps-values.yaml`.

   ```
   ---
   enable_ubi_builder: true
   ```

2. (Optional) Install the Static builder.

   The Static builder uses Ubuntu Jammy for both build images and a minimal static run image. This builder only supports Golang. To install the Static builder, add the key-value pair `enable_static_builder: true` to your `tbs-full-deps-values.yaml`.

   ```
   ---
   enable_static_builder: true
   ```

7. Install the `full` dependencies package by running:

```
tanzu package install full-deps \
  --package full-deps.buildservice.tanzu.vmware.com \
  --version "> 0.0.0" \
  --namespace tap-install \
  --values-file VALUES-FILE
```

Where `VALUES-FILE` is the path to the `tbs-full-deps-values.yaml` you created earlier.

Configure Tanzu Build Service properties on a workload

This topic tells you how to configure your workload with Tanzu Build Service properties.

Overview

Tanzu Build Service builds registry images from source code for Tanzu Application Platform. You can configure these build configurations by using a workload.

Tanzu Build Service is only applicable to the build process. Configurations, such as environment variables and service bindings, might require a different process for runtime.

Configure build-time service bindings

You can configure build-time service bindings for Tanzu Build Service.

Tanzu Build Service supports using the Service Binding Specification for Kubernetes for application builds. For more information, see the service binding specification for Kubernetes in GitHub.

Service binding configuration is specific to the buildpack that is used to build the app. For more information about configuring buildpack service bindings for the buildpack you are using, see the VMware Tanzu Buildpacks documentation.

To configure a service binding for a Tanzu Application Platform workload, follow these steps:

1. Create a YAML file named `service-binding-secret.yaml`
    - A Maven secret for example:

      ```
      apiVersion: v1
      kind: Secret
      metadata:
        name: settings-xml
        namespace: DEVELOPER-NAMESPACE
      type: service.binding/maven
      stringData:
        type: maven
        provider: sample
        settings.xml: |
        MY-SETTINGS
      ```

      Where: - `DEVELOPER-NAMESPACE` is the namespace where workloads are created. - `MY-SETTINGS` is the contents of your service bindings file.

    - A NuGet secret for example:

      ```
      apiVersion: v1
      kind: Secret
      metadata:
        name: nuget-config
        namespace: DEVELOPER-NAMESPACE
      type: service.binding/nugetconfig
      stringData:
        type: nugetconfig
        nuget.config: |
        MY-SETTINGS
      ```

      Where: - `DEVELOPER-NAMESPACE` is the namespace where workloads are created. - `MY-SETTINGS` is the contents of your service bindings file.

    - A Git secret for example:

      ```
      apiVersion: v1
      kind: Secret
      metadata:
        name: git-credentials
        namespace: DEVELOPER-NAMESPACE
      ```

```
type: service.binding/git-credentials
stringData:
  type: git-credentials
  context: CREDENTIAL-CONTEXT
  credentials: |
  MY-CREDENTIALS
```

Where:

- `DEVELOPER-NAMESPACE` is the namespace where workloads are created.
- `CREDENTIAL-CONTEXT` is the URL context of the credential. If the workload only has one git-credential service binding, this field can be omitted.
- `MY-CREDENTIALS` is the credential defined in the git credential format.

2. Apply the YAML file by running:

```
kubectl apply -f service-binding-secret.yaml
```

3. Create the workload with `buildServiceBindings` configured by running:

```
tanzu apps workload create WORKLOAD-NAME \
  --param-yaml buildServiceBindings='[{"name": "settings-xml", "kind": "Secre
t"}]' \
  ...
```

Where `WORKLOAD-NAME` is the name of the workload you want to configure.

Configure environment variables

If you have build-time environment variable dependencies, you can set environment variables that are available at build-time.

You can also configure buildpacks with environment variables. Buildpack configuration depends on the specific buildpack being used. For more information about configuring environment variables for the buildpack you are using, see the VMware Tanzu Buildpacks documentation.

For example:

```
tanzu apps workload create WORKLOAD-NAME \
  --build-env "ENV_NAME=ENV_VALUE" \
  --build-env "BP_MAVEN_BUILD_ARGUMENTS=-Dmaven.test.skip=true"
```

Where `WORKLOAD-NAME` is the name of the workload you want to configure.

Configure the service account

Using the Tanzu CLI, you can configure the service account used during builds. This service account is the one configured for the developer namespace. If unset, `default` is used.

To configure the service account used during builds, run:

```
tanzu apps workload create WORKLOAD-NAME \
  --param serviceAccount=SERVICE-ACCOUNT-NAME \
```

Where:

- `WORKLOAD-NAME` is the name of the workload you want to configure.
- `SERVICE-ACCOUNT-NAME` is the name of the service account you want to use during builds.

Configure the cluster builder

To configure the ClusterBuilder used during builds:

1. View the available ClusterBuilds by running:

   ```
   kubectl get clusterbuilder
   ```

2. Set the ClusterBuilder used during builds by running:

   ```
   tanzu apps workload create WORKLOAD-NAME \
     --param clusterBuilder=CLUSTER-BUILDER-NAME \
   ```

   Where:

   - `WORKLOAD-NAME` is the name of the workload you want to configure.
   - `CLUSTER-BUILDER-NAME` is the ClusterBuilder you want to use.

Configure the workload container image registry

Using the Tanzu CLI, you can configure the registry where workload images are saved. The service account used for this workload must have read and write access to this registry location.

To configure the registry where workload images are saved, run:

```
tanzu apps workload create WORKLOAD-NAME \
  --param-yaml registry={"server": SERVER-NAME, "repository": REPO-NAME}
```

Where:

- `SERVER-NAME` is the host name of the registry server. Examples:
  - Harbor has the form `"my-harbor.io"`.
  - Docker Hub has the form `"index.docker.io"`.
  - Google Cloud Registry has the form `"gcr.io"`.
- `REPO-NAME` is where workload images are stored in the registry. Images are written to `SERVER-NAME/REPO-NAME/workload-name`. Examples:
  - Harbor has the form `"my-project/supply-chain"`.
  - Docker Hub has the form `"my-dockerhub-user"`.
  - Google Cloud Registry has the form `"my-project/supply-chain"`.

Configure custom CA certificates for a single workload using service bindings

If the language family buildpack you are using includes the Paketo CA certificates buildpack, you can use a service binding to provide custom certificates during the build and run process. For more information about language family buildpacks, see the Tanzu Buildpacks documentation.

To create a service binding to provide custom CA certificates for a workload:

1. Create a YAML file named `service-binding-ca-cert.yaml` for a secret as follows:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: my-ca-certs
   data:
   ```

```
  type: ca-certificates
  provider: sample
  CA-CERT-FILENAME: |
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
```

Where `CA-CERT-FILENAME` is the name of your PEM encoded CA certificate file. For example, `arbitrary-file-name.pem`.

2. Apply the YAML file by running:

```
kubectl apply -f service-binding-ca-cert.yaml
```

3. To build with the custom certificate, create the workload with `--param-yaml buildServiceBindings` flag:

```
tanzu apps workload create WORKLOAD-NAME \
  --param-yaml buildServiceBindings='[{"apiVersion": "v1", "kind": "Secret", "n
ame": "my-ca-certs"}]' \
  ...
```

Where `WORKLOAD-NAME` is the name of the workload you want to create.

4. To deploy with the custom certificate, create the workload with the `--service-ref` flag:

```
tanzu apps workload create WORKLOAD-NAME \
  --service-ref my-ca-certs=v1:Secret:my-ca-certs \
  ...
```

Where `WORKLOAD-NAME` is the name of the workload you want to create.

Using custom CA certificates for all workloads

To provide custom CA certificates to the build process for all workloads, see the optional step to add the `ca_cert_data` key Install the Tanzu Build Service package.

Create a signed container image with Tanzu Build Service

This topic tells you how to create a Tanzu Build Service image resource that builds a container image from source code signed with Cosign.

Prerequisites

Before you can configure Tanzu Build Service to sign your image builds, you must:

- Install Tanzu Build Service. The Full, Iterate, and Build profiles include Tanzu Build Service by default. If you have not installed Tanzu Application Platform with one of these profiles, see Installing Tanzu Build Service.
- Install Cosign. For instructions, see the Cosign documentation.
- Have a Builder or ClusterBuilder resource configured.
- Have an image resource configured.
- Review the kpack tutorial. This topic builds upon the steps in this tutorial.

Configure Tanzu Build Service to sign your image builds

To configure Tanzu Build Service to sign your image builds:

1. Ensure that you are in a Kubernetes context where you are authenticated and authorized to create and edit secret and service account resources.
2. Generate a Cosign keypair and store it as a Kubernetes secret by running:

   ```
   cosign generate-key-pair k8s://NAMESPACE/COSIGN-KEYPAIR-NAME
   ```

   Where:

   - `NAMESPACE` is the namespace to store the Kubernetes secret in.
   - `COSIGN-KEYPAIR-NAME` is the name of the Kubernetes secret.

   For example:

   ```
   cosign generate-key-pair k8s://default/tutorial-cosign-key-pair
   ```

3. Enter a password for the private key. Enter any password you want. After the command has completed, you will see the following output:

   ```
   Successfully created secret tutorial-cosign-key-pair in namespace default
   Public key written to cosign.pub
   ```

   You will also see a `cosign.pub` file in your current directory. Keep this file as you will need it to verify the signature of the images that are built.

4. If you are using Docker Hub or a registry that does not support OCI media types, add the annotation `kpack.io/cosign.docker-media-types: "1"` to the Cosign secret as follows:

   ```
   apiVersion: v1
   kind: Secret
   type: Opaque
   metadata:
     name: tutorial-cosign-key-pair
     namespace: default
     annotations:
       kpack.io/cosign.docker-media-types: "1"
   data:
     cosign.key: PRIVATE-KEY-DATA
     cosign.password: COSIGN-PASSWORD
     cosign.pub: PUBLIC-KEY-DATA
   ```

   For more information about configuring Cosign key pairs, see Image signing with cosign in the Tanzu Build Service documentation.

5. To enable Cosign signing, create or edit the service account resource that is referenced in the image resource so that it includes the Cosign keypair secret created earlier. The service account is in the same namespace as the image resource and is directly referenced by the image or default if there isn't one.

   ```
   apiVersion: v1
   kind: ServiceAccount
   metadata:
     name: SERVICE-ACCOUNT-NAME
     namespace: default
   secrets:
   - name: REGISTRY-CREDENTIALS
   - name: COSIGN-KEYPAIR-NAME
   imagePullSecrets:
   - name: REGISTRY-CREDENTIALS
   ```

Where:

- `SERVICE-ACCOUNT-NAME` is the name of your service account resource. For example, `tutorial-cosign-service-account`.
- `COSIGN-KEYPAIR-NAME` is the name of the Cosign keypair secret generated earlier. For example, `tutorial-cosign-key-pair`.
- `REGISTRY-CREDENTIALS` is the secret that provides credentials for the container registry where application container images are pushed to.

6. Apply the service account resource to the cluster by running:

```
kubectl apply -f cosign-service-account.yaml
```

7. Create an image resource file named `image-cosign.yaml`. For example:

```
apiVersion: kpack.io/v1alpha2
kind: Image
metadata:
  name: tutorial-cosign-image
  namespace: default
spec:
  tag: IMAGE-REGISTRY
  serviceAccountName: tutorial-cosign-service-account
  builder:
    name: my-builder
    kind: Builder
  source:
    git:
      url: https://github.com/spring-projects/spring-petclinic
      revision: 82cb521d636b282340378d80a6307a08e3d4a4c4
```

Where:

- `IMAGE-REGISTRY` with a writable repository in your registry. The secret referenced in the service account is a secret providing credentials for the registry where application container images are pushed to. For example:
  - Harbor has the form `"my-harbor.io/my-project/my-repo"`
  - Docker Hub has the form `"my-dockerhub-user/my-repo"` or `"index.docker.io/my-user/my-repo"`
  - Google Cloud Registry has the form `"gcr.io/my-project/my-repo"`

8. If you are using Out of the Box Supply Chains, edit the respective `ClusterImageTemplate` to enable signing in your supply chain. For more information, see Authoring supply chains.

Important

VMware discourages referencing the service account using the `service_account` value when installing the Out of the Box Supply Chain. This is because it gives your run cluster access to the private signing key.

9. Apply the image resource to the cluster by running:

```
kubectl apply -f image-cosign.yaml
```

10. After the image resource finishes building, you can get the fully resolved and built OCI image by running:

```
kubectl -n default get image tutorial-cosign-image
```

Example output:

```
NAME                  LATESTIMAGE                                          READY
tutorial-cosign-image index.docker.io/your-project/app@sha256:6744b...    True
```

11. Verify image signature by running:

```
cosign verify --insecure-ignore-tlog --key cosign.pub LATEST-IMAGE-WITH-DIGEST
```

Where `LATEST-IMAGE-WITH-DIGEST` is the value of `LATESTIMAGE` you retrieved in the previous step. For example, `index.docker.io/your-project/app@sha256:6744b...`.

Expected output:

```
Verification for index.docker.io/your-project/app@sha256:6744b... --
The following checks were performed on each of these signatures:
- The cosign claims were validated
- The signatures were verified against the specified public key
- Any certificates were verified against the Fulcio roots.
```

Note

You must use the `--insecure-ignore-tlog` flag because the supply chain does not write the signature attestation to a transparency log.

12. Configure Supply Chain Security Tools for VMware Tanzu - Policy Controller to ensure that only signed images are allowed in your cluster. For more information, see the Supply Chain Security Tools for VMware Tanzu - Policy Controller documentation.

Generate Supply-chain Levels for Software Artifacts attestations with Tanzu Build Service

This topic tells you how to configure Tanzu Build Service to generate Supply-chain Levels for Software Artifacts (SLSA) attestations for image resources. It also describes how you can view the attestation and verify the signature.

About SLSA security levels in Tanzu Build Service

Tanzu Build Service supports generating SLSA v1 attestations. Tanzu Build Service can achieve SLSA security levels up to `L3` depending on how the installation and workload is configured. This section documents how Tanzu Build Service adheres to each SLSA security level.

For more technical details about how kpack handles SLSA, see the kpack documentation in GitHub.

Build L0

Specification: Build L0: No guarantees

This is the default level in Tanzu Build Service and Tanzu Application Platform because attestations are not generated unless configured for your installation.

Build L1

Specification: Build L1: Provenance exists

This is the level Tanzu Build Service and Tanzu Application Platform achieve if SLSA generation is enabled.

- The build process is consistent. You can see an example app with expected output in Deploy an app on Tanzu Application Platform.
- The provenance generated is complete, but it is unsigned.
- To view the generated provenance, see View the attestation later in this topic.

Build L2

Specification: Build L2: Hosted build platform

This is the level Tanzu Build Service and Tanzu Application Platform achieve if SLSA generation is enabled and a signing key is attached to the service account of the workload.

- All builds executed by Tanzu Build Service occur on a dedicated Kubernetes cluster, not the user's workstation.
- To verify the attestation signature, see Verify the attestation signature.

Build L3

Specification: Build L3: Hardened builds

This is the level Tanzu Build Service and Tanzu Application Platform achieve if SLSA generation is enabled and a signing key is attached to the service account of the workload.

- Builds are isolated from each other by using a different Kubernetes pod per build process.
- Signing keys are only readable by Tanzu Build Service system resources. No user defined build steps, such as custom Buildpacks, custom Stacks, and custom Builders, can access the signing keys.
- You can use Kubernetes Role-Based Access Control (RBAC) to secure the signing keys at the cluster level.

Prerequisites

Before you can configure Tanzu Build Service to sign your image builds, you must:

- Install Tanzu Application Platform.
- If you did not install Tanzu Application Platform by using the Full, Iterate, or Build profile, install the Tanzu Build Service package. See Installing Tanzu Build Service.
- Install jq. For instructions, see the jq documentation.
- To view and verify signed attestations, install Cosign. For instructions, see the Cosign documentation.
- To view unsigned attestations, install Crane. For instructions, see the Crane documentation.
- Ensure that you have access to the `base64` and `awk` commands. These commands are pre-installed on every macOS and Linux machine.
- Have a Builder or ClusterBuilder resource configured. For instructions, see Manage builders for Tanzu Build Service.

Enable SLSA attestation

Tanzu Build Service does not generate SLSA attestation by default. To enable this behavior:

1. Set the `generate_slsa_attestation` key to `true` in your `tap-values.yaml`:

```
buildservice:
    generate_slsa_attestation: true
```

2. Apply the updated configuration by running:

```
tanzu package installed update tap \
  --package tap.tanzu.vmware.com \
  --version TAP-VERSION \
  --namespace tap-install \
  --values-file tap-values.yaml
```

Where `TAP-VERSION` is the version of Tanzu Application Platform installed.

Create unsigned attestations (SLSA L0)

If Tanzu Build Service has SLSA attestation enabled, every app image built by Tanzu Build Service generates a second image where the attestation is stored.

Build the image

No action is required to configure the Image or Workload resources. To learn how to create an app, see Configure Tanzu Build Service properties on a workload.

View the attestation

To view the attestation, run:

```
crane export ATTESTATION-TAG | jq --raw-output '.payload' | base64 --decode | jq
```

Where `ATTESTATION-TAG` is the tag derived from the app image's digest. For more information, see Location of the attestation later in this topic.

For more information about the attestation, see the kpack documentation.

For an example of expected output, see the kpack documentation.

Create signed attestations (SLSA L3)

If the service account of the workload has a secret with a signing key attached, Tanzu Build Service automatically signs the generated attestation.

Generate and save the signing key

To generate and save the signing key:

1. Use the cosign CLI to generate the Kubernetes secret:

```
cosign generate-key-pair k8s://NAMESPACE/COSIGN-KEYPAIR-NAME
```

Where:

- `NAMESPACE` is the namespace to store the Kubernetes secret in.
- `COSIGN-KEYPAIR-NAME` is the name of the Kubernetes secret.

You will see a `cosign.pub` file in your current directory. Keep this file as you need it to verify the signature of the images that are built.

2. Annotate the secret to use it for signing SLSA attestations by running:

```
kubectl annotate secret COSIGN-KEYPAIR-NAME 'kpack.io/slsa='
```

Tanzu Build Service only considers secrets with the annotation `kpack.io/slsa: ""` for signing.

3. Attach the secret to the service account by running:

```
kubectl patch serviceaccount SERVICE-ACCOUNT-NAME -p '{"secrets":[{"name":"COSI
GN-KEYPAIR-NAME"}]}'
```

Where `SERVICE-ACCOUNT-NAME` is the name of the service account that the workload will use.

Build the image

No action is required to configure the Image or Workload resources. To learn how to create an app, see Configure Tanzu Build Service properties on a workload.

Verify the attestation signature

Because Cosign operates on the tag-based discovery, you can only view the latest attestation for a reproducible build. For more information, see Reproducible builds later in this topic.

1. To verify the signature of the attestation, run:

```
cosign verify-attestation \
  --insecure-ignore-tlog=true \
  --key PUBLIC-KEY \
  --type=slsaprovenance1 APP-IMAGE-DIGEST > /dev/null
```

Where:

- `PUBLIC-KEY` is either `k8s://NAMESPACE/COSIGN-KEYPAIR-NAME` from Generate and save the signing key earlier or the path to the `cosign.pub` that was generated.
- `APP-IMAGE-DIGEST` is digest of the image that the workload built.

Example output:

```
Verification for index.docker.io/your-project/app@sha256:1234... --
The following checks were performed on each of these signatures:
  - The cosign claims were validated
  - The signatures were verified against the specified public key
```

2. (Optional) To view the generated attestation, run:

```
cosign verify-attestation \
  --insecure-ignore-tlog=true \
  --key PUBLIC-KEY \
  --type=slsaprovenance1 APP-IMAGE-DIGEST | jq \
  --raw-output '.payload' | base64 --decode | jq
```

Where:

- `PUBLIC-KEY` is either `k8s://NAMESPACE/COSIGN-KEYPAIR-NAME` from Generate and save the signing key earlier or the path to the `cosign.pub` that was generated.
- `APP-IMAGE-DIGEST` is the digest of the image that the workload built.

For more information about the attestation, see the kpack documentation.

For an example of expected output, see the kpack documentation.

Location of the attestation

The tag of the attestation image is predictable from the digest of the app image.

If the digest of your app image is `index.docker.io/your-project/app@sha256:1234`, the tag of your attestation image is `index.docker.io/your-project/app:sha256-1234.att`.

Reproducible builds

Tanzu Build Service supports reproducible builds. This means that it's possible for a build to generate the exact same bit-for-bit image as a previous run. This is most likely to occur if a build is manually re-triggered, but might also automatically occur as part of a dependency upgrade.

In these cases, because the image is bit-for-bit identical and has the same digest as the previous image, the tag of the attestation image overlaps. In this scenario, the newer attestation overwrites the older attestation.

To view the older attestation, the digest of the attestation image is exposed in the status of the Build resource under the key `.status.latestAttestationImage`. You can retrieve this by running:

```
kubectl get builds.kpack.io BUILD-NAME -o yaml
```

Where `BUILD-NAME` is the name of the Build resource. It is usually in the format `IMAGE-NAME-build-N`.

Expected output:

```
apiVersion: kpack.io/v1alpha2
kind: Build
...
status:
  ...
  latestAttestationImage: index.docker.io/your-project/app@sha256:1234
```

Tanzu Build Service Dependencies

This topic tells you about Tanzu Build Service dependencies.

Important

Ubuntu Bionic stack is deprecated and will be removed in a future release. VMware recommends that you migrate builds to Jammy stacks. For Tanzu Application Platform v1.5 and later, the default stack for Tanzu Build Service is Jammy.

To build OCI images, Tanzu Build Service has the following dependencies: Cloud Native Buildpacks, Stacks, and Lifecycles.

How dependencies are installed

When Tanzu Application Platform is installed with Tanzu Build Service, it is bootstrapped with a set of dependencies. No extra configuration is required. Each version of Tanzu Application Platform and Tanzu Build Service contains new dependencies.

When Tanzu Application Platform is upgraded, new dependencies are installed which might cause workload images to rebuild. To ensure dependency compatibility, Tanzu Build Service only releases patches for dependencies in patch versions of Tanzu Application Platform. For upgrade instructions, see Upgrade the full dependencies package.

By default, Tanzu Build Service is installed with the `lite` set of dependencies, which have a smaller-footprint and contain a subset of the buildpacks and stacks in the `full` set of dependencies. For a comparison of `lite` and `full` dependencies, see Dependency comparison.

View installed dependencies

To view the set of dependencies installed with Tanzu Build Service, inspect the status of the cluster builders by running:

```
kubectl get clusterbuilder -o yaml
```

Cluster builders contain stack and buildpack metadata.

Bionic and Jammy stacks

Tanzu Application Platform v1.3 and later supports Ubuntu v22.04 (Jammy) based builds and will default to it from Tanzu Application Platform v1.5 and later.

Ubuntu Bionic will stop receiving support in April 2023. VMware recommends that you migrate builds to Jammy.

For more information about support for Jammy stacks, see About lite and full dependencies later in this topic.

Note

While upgrading apps to a later stack, you might encounter the build platform erroneously reusing the old build cache. If you encounter this issue, delete and recreate the workload in Tanzu Application Platform, or delete and recreate the image in Tanzu Build Service.

About lite and full dependencies

Each version of Tanzu Application Platform is released with two types of Tanzu Build Service dependencies: `lite` and `full`. These dependencies consist of the buildpacks and stacks required for application builds. Each type serves different use cases. Both types are suitable for production workloads.

By default, Tanzu Build Service is installed with `lite` dependencies, which do not contain all buildpacks and stacks. To use all buildpacks and stacks, you must install the `full` dependencies. For instructions about installing `full` dependencies, see Install full dependencies.

For a table comparing the differences between `full` and `lite` dependencies, see Dependency comparison.

Lite dependencies

The `lite` dependencies are the default set installed with Tanzu Build Service.

`lite` dependencies contain a smaller footprint to speed up installation time, but do not support all workload types. For example, `lite` dependencies do not contain the PHP buildpack and cannot be used to build PHP workloads.

Lite dependencies: stacks

The `lite` dependencies contain the following stacks:

- `base-jammy` (Ubuntu Jammy)
- `default` (identical to `base-jammy`)

For more information, see Stacks in the VMware Tanzu Buildpacks documentation

Lite dependencies: buildpacks

The `lite` dependencies contain the following buildpacks in Tanzu Application Platform v1.9:

Buildpack	Version	Supported Stacks
Java Buildpack for VMware Tanzu (Lite)	9.16.0	Bionic, Jammy, UBI
Java Native Image Buildpack for Tanzu (Lite)	7.14.0	Bionic, Jammy
.NET Core Buildpack for VMware Tanzu (Lite)	2.13.0	Bionic, Jammy, UBI
Node.js Buildpack for VMware Tanzu (Lite)	2.7.1	Bionic, Jammy, UBI
Python Buildpack for VMware Tanzu (Lite)	2.9.1	Bionic, Jammy
Go Buildpack for VMware Tanzu (Lite)	3.3.1	Bionic, Jammy, Jammy Static
Web Servers Buildpack for VMware Tanzu (Lite)	0.19.0	Bionic, Jammy, UBI
Ruby Buildpack for VMware Tanzu (Lite)	2.12.1	Bionic, Jammy

And the following components:

Component	Version	Supported Stacks
CNB Lifecycle	0.16.0	Bionic, Jammy
Base Stack of Ubuntu Jammy for VMware Tanzu	0.1.88	Jammy

Full dependencies

The Tanzu Build Service `full` set of dependencies contain more buildpacks and stacks, which allows for more workload types.

The dependencies are pre-packaged, so builds do not have to download them from the Internet. This can speed up build times and allows builds to occur in air-gapped environments. Due to the larger footprint of `full`, installations might take longer.

The `full` dependencies are not installed with Tanzu Build Service by default, you must install them. For instructions for installing `full` dependencies, see Install Tanzu Build Service with full dependencies.

Full dependencies: stacks

The `full` dependencies contain the following stacks, which support different use cases:

- `base-jammy` (Ubuntu Jammy)
- `full-jammy` (Ubuntu Jammy)
- `tiny-jammy` (Ubuntu Jammy)
- `default` (identical to `base-jammy`)

For more information, see Stacks in the VMware Tanzu Buildpacks documentation.

Full dependencies: buildpacks

The `full` dependencies contain the following buildpacks in Tanzu Application Platform v1.7:

Buildpack	Version	Supported Stacks
Java Buildpack for VMware Tanzu	9.16.0	Bionic, Jammy, UBI
Java Native Image Buildpack for Tanzu	7.14.0	Bionic, Jammy
.NET Core Buildpack for VMware Tanzu	2.13.0	Bionic, Jammy, UBI
Node.js Buildpack for VMware Tanzu	2.7.1	Bionic, Jammy, UBI
Python Buildpack for VMware Tanzu	2.9.1	Bionic, Jammy
Ruby Buildpack for VMware Tanzu	2.12.1	Bionic, Jammy
Go Buildpack for VMware Tanzu	3.3.1	Bionic, Jammy, Jammy Static
PHP Buildpack for VMware Tanzu	2.12.0	Bionic, Jammy
Web Servers Buildpack for VMware Tanzu	0.19.0	Bionic, Jammy, UBI
Procfile Buildpack for VMware Tanzu	5.7.0	Bionic, Jammy

And the following components:

Component	Version	Supported Stacks
CNB Lifecycle	0.16.0	Bionic, Jammy
Tiny Stack of Ubuntu Jammy for VMware Tanzu	0.1.92	Jammy
Base Stack of Ubuntu Jammy for VMware Tanzu	0.1.88	Jammy
Full Stack of Ubuntu Jammy for VMware Tanzu	0.1.148	Jammy
Standard Stack of UBI 8 for VMware Tanzu	0.0.13	UBI 8
Static Stack of Ubuntu Jammy for VMware Tanzu	0.1.28	Jammy

Dependency comparison

The following table compares the contents of the `lite` and `full` dependencies.

Feature	lite	full
Faster installation time	Yes	No
Dependencies pre-packaged (faster builds)	No	Yes
Supports air-gapped installation	No	Yes
Contains base stack	Yes	Yes
Contains full stack	No	Yes
Contains tiny stack	No	Yes
Contains Jammy stack	Yes	Yes
Supports Java workloads	Yes	Yes
Supports Node.js workloads	Yes	Yes
Supports Go workloads	Yes	Yes
Supports Python workloads	Yes	Yes
Supports Ruby workloads	Yes	Yes
Supports .NET Core workloads	Yes	Yes

Feature	lite	full
Supports PHP workloads	No	Yes
Supports static workloads	Yes	Yes
Supports binary workloads	Yes	Yes
Supports web servers buildpack	Yes	Yes

Update dependencies in band with Tanzu Application Platform releases

New versions of dependencies such as buildpacks, and stacks are available in new versions of Tanzu Application Platform. To update dependencies, VMware recommends that you update to the latest patch version of Tanzu Application Platform.

- If you are using `lite` or `full` dependencies, upgrade to the latest patch version of Tanzu Application Platform to update your dependencies.
- If you are using `full` dependencies, you must complete some extra steps after you upgrade to the latest patch. For more information, see Upgrading the full dependencies package.

Note

When Tanzu Application Platform is upgraded, new dependencies are installed which might cause workload images to rebuild.

Update dependencies out of band with Tanzu Application Platform releases

To update dependencies between Tanzu Application Platform releases, you can either use automatic dependency updates or you can update dependencies manually.

Automatic dependency updates

Tanzu Build Service dependencies might be upgraded between Tanzu Application Platform releases, for example, if a CVE is discovered in the OS (stack update) or language (buildpack update).

Automatic dependency updates enable your cluster to consume the stack and buildpack updates immediately instead of waiting for the next Tanzu Application Platform patch release to pull in the updated dependencies.

- Updates are provided through a separate package repository with available version lines for all supported Tanzu Application Platform minor versions.
- Within a version line, only patch versions are incremented to avoid breaking changes.
- You can customize the packages that you want the automatic dependency updater to update through your `tap-values.yaml` file or your full dependencies values.

Prerequisites: These steps assume a registry secret already exists in the cluster for accessing `registry.tanzu.vmware.com` and your registry.

To enable automatic dependency updates:

1. Add the following to your `tap-values.yaml` file:

```
buildservice:
  dependency_updates:
    allow: true
    scope: SCOPE
    include_packages: [""]
    exclude_packages: [""]
```

Where:

- `SCOPE` is the list of dependencies you want updated. The options are:
  - `stacks-only` (default): Only stacks and builders are updated. This addresses CVEs in the base image or operating system.
  - `all`: Stacks, builders, and buildpacks are updated. This addresses CVEs in the base image or operating system and CVEs in the language toolchain such as compilers, interpreters, and standard libraries.
  - `custom`: This list is empty by default. Use the `include_packages` key to add packages to be updated.

Note

You must update the Tanzu Application Platform package install and the Full Dependencies package install after changing the `tap-values.yaml`.

2. Add the Tanzu Build Service Dependency Updates package repository by running:

```
kubectl apply -f - <<EOF
  apiVersion: packaging.carvel.dev/v1alpha1
  kind: PackageRepository
  metadata:
    name: tbs-dependencies-package-repository
    namespace: tap-install
  spec:
    fetch:
      imgpkgBundle:
        image: DEPENDENCY-UPDATER-PACKAGE-REPO
        tagSelection:
          semver:
            constraints: VERSION-CONSTRAINT
EOF
```

Where:

- `DEPENDENCY-UPDATER-PACKAGE-REPO` is the location of the package repository. This is `registry.tanzu.vmware.com/build-service-dependency-updater/package-repo` for online installs and the internal container image registry for air-gapped installs.
- `VERSION-CONSTRAINT` is the Tanzu Application Platform version in the form of `MAJOR.MINOR.x`. For example, `1.8.x`.

After completing this configuration, the repository you set with `DEPENDENCY-UPDATER-PACKAGE-REPO` will be polled for updates and any new releases will automatically be made available to the cluster.

Manual dependency updates

You can update buildpack dependencies outside of upgrading Tanzu Application Platform, but VMware recommends that you upgrade Tanzu Application Platform when possible instead. Each Tanzu Application Platform release version includes a tested set of buildpacks. If you consume a

buildpack from VMware Tanzu Network that is not packaged and tested in a Tanzu Application Platform release, it might introduce errors.

1. Sign into VMware Tanzu Network so that the image can be retrieved from the registry.
2. Select the required buildpack in the Tanzu Buildpacks documentation. Select `full` or `lite` dependencies. Scroll to the Docker command, and copy the buildpack image URL for use in the next step.
3. Run:

   ```
   imgpkg copy -b BUILDPACK-IMAGE-URL --to-repo ${INSTALL_REGISTRY_HOSTNAME}/${INS
   TALL_REPO}/tbs-deps/BUILDPACK-LANGUAGE
   ```

   Where `BUILDPACK-IMAGE-URL` is the buildpack image URL copied from the Docker command in the previous step
4. Create a `ClusterBuildpack` resource referencing the copied buildpack image:

   ```
   apiVersion: kpack.io/v1alpha2
   kind: ClusterBuildpack
   metadata:
     name: out-of-band-LANGUAGE-NAME-BUILDPACK-VERSION
   spec:
     image: RELOCATED-BUILDPACKIMAGE
     serviceAccountRef:
       name: dependencies-pull-serviceaccount
       namespace: build-service
   ```

   Where `RELOCATED-BUILDPACKIMAGE` is the URL of the relocated buildpack image from previous step.

   To avoid naming collisions, follow the name conventions specified in `metadata.name`. The name can follow any convention that allows the Cluster Operator to distinguish this `ClusterBuildpack` from others installed by Tanzu Application Platform.
5. Apply the YAML from the previous step to the Tanzu Application Platform cluster:

   ```
   kubectl apply -f FILE-FROM-PREVIOUS-STEP
   ```

The `ClusterBuildpack` is now deployed. Tanzu Build Service uses the latest available version to run builds. All images that were built with older versions of the buildpack will now be rebuilt.

When you upgrade Tanzu Application Platform, new buildpacks with later versions are installed. After an upgrade, the `ClusterBuildpack` created in this procedure is not needed and can be removed.

Security context constraint for OpenShift

This topic tells you about running Tanzu Build Service on OpenShift clusters.

On OpenShift clusters Tanzu Build Service must run with a custom Security Context Constraint (SCC) to enable compliance. Tanzu Application Platform configures the following SCC for Tanzu Build Service when you configure the `kubernetes_distribution: openshift` key in the `tap-values.yaml` file.

```
---
kind: SecurityContextConstraints
apiVersion: security.openshift.io/v1
metadata:
  name: tbs-restricted-scc-with-seccomp
allowHostDirVolumePlugin: false
```

```
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: false
allowPrivilegedContainer: false
allowedCapabilities:
  - NET_BIND_SERVICE
defaultAddCapabilities: null
fsGroup:
  type: RunAsAny
groups: []
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities:
  - ALL
runAsUser:
  type: MustRunAsNonRoot
seLinuxContext:
  type: MustRunAs
seccompProfiles:
  - runtime/default
supplementalGroups:
  type: RunAsAny
users: []
volumes:
  - configMap
  - downwardAPI
  - emptyDir
  - persistentVolumeClaim
  - projected
  - secret
```

It also applies the following RBAC to allow Tanzu Build Service services to use the SCC:

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    apps.tanzu.vmware.com/aggregate-to-workload: "true"
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  name: system:tbs:scc:restricted-with-seccomp
rules:
  - apiGroups:
      - security.openshift.io
    resourceNames:
      - tbs-restricted-scc-with-seccomp
    resources:
      - securitycontextconstraints
    verbs:
      - use
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: system:tbs:scc:restricted-with-seccomp
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:tbs:scc:restricted-with-seccomp
subjects:
  - kind: ServiceAccount
    namespace: build-service
    name: dependency-updater-serviceaccount
```

```
  - kind: ServiceAccount
    namespace: build-service
    name: dependency-updater-controller-serviceaccount
  - kind: ServiceAccount
    namespace: build-service
    name: secret-syncer-service-account
  - kind: ServiceAccount
    namespace: build-service
    name: warmer-service-account
  - kind: ServiceAccount
    namespace: build-service
    name: build-service-daemonset-serviceaccount
  - kind: ServiceAccount
    namespace: cert-injection-webhook
    name: cert-injection-webhook-sa
  - kind: ServiceAccount
    namespace: kpack
    name: kp-default-repository-serviceaccount
  - kind: ServiceAccount
    namespace: kpack
    name: kpack-pull-lifecycle-serviceaccount
  - kind: ServiceAccount
    namespace: kpack
    name: controller
  - kind: ServiceAccount
    namespace: kpack
    name: webhook
  - kind: ServiceAccount
    namespace: stacks-operator-system
    name: controller-manager
```

Troubleshoot Tanzu Build Service

This topic tells you how to troubleshoot Tanzu Build Service when used with Tanzu Application Platform (commonly known as TAP).

Builds fail due to volume errors on EKS running Kubernetes v1.23

Symptom

After installing Tanzu Application Platform on or upgrading an existing Amazon Elastic Kubernetes Service (EKS) cluster to Kubernetes v1.23, build pods show:

```
'running PreBind plugin "VolumeBinding": binding volumes: timed out waiting
 for the condition'
```

Cause

This is due to the CSIMigrationAWS in this Kubernetes version, which requires users to install the Amazon EBS CSI driver to use AWS Elastic Block Store (EBS) volumes. See the Amazon documentation. For more information about EKS support for Kubernetes v1.23, see the Amazon blog post.

Tanzu Application Platform uses the default storage class which uses EBS volumes by default on EKS.

Solution

Follow the AWS documentation to install the Amazon EBS CSI driver before installing Tanzu Application Platform, or before upgrading to Kubernetes v1.23. See

---

Smart-warmer-image-fetcher reports ErrImagePull due to dockerd's layer depth limitation

Symptom

When using dockerd as the cluster's container runtime, you might see the `smart-warmer-image-fetcher` pods report a status of `ErrImagePull`.

Cause

This error might be due to dockerd's layer depth limitation, in which the maximum supported image layer depth is 125.

To verify that the `ErrImagePull` status is due to dockerd's maximum supported image layer depth, check for event messages containing the words `max depth exceeded`. For example:

```
$ kubectl get events -A | grep "max depth exceeded"
  build-service        73s         Warning     Failed         pod/smart-warmer-image-f
etcher-wxtr8     Failed to pull image
  "harbor.somewhere.com/aws-repo/build-service:clusterbuilder-full@sha256:065bb361fd91
4a3970ad3dd93c603241e69cca214707feaa6
  d8617019e20b65e":  rpc error: code = Unknown desc = failed to register layer: max de
pth exceeded
```

Solution

To work around this issue, configure your cluster to use containerd or CRI-O as its default container runtime. For instructions, see the following documentation for your Kubernetes cluster provider.

For AWS, see:

- The Amazon blog
- The eksctl CLI documentation

For AKS, see:

- The Microsoft Azure documentation
- The Microsoft Azure blog

For GKE, see:

- The GKE documentation

For OpenShift, see:

- The Red Hat Hybrid Cloud blog
- The Red Hat OpenShift documentation

Nodes fail due to "trying to send message larger than max" error

Symptom

You see the following error, or similar, in a node status:

```
Warning ContainerGCFailed 119s (x2523 over 42h) kubelet rpc error: code = ResourceExha
usted desc = grpc: trying to send message larger than max (16779959 vs. 16777216)
```

Cause

This is due to the way that the container runtime interface (CRI) handles garbage collection for unused images and containers.

Solution

Do not use Docker as the CRI because it is not supported. Some versions of EKS default to Docker as the runtime.

Build platform uses the old build cache after upgrade to new stack

Symptom

While upgrading apps to a later stack, you might encounter the build platform erroneously reusing the old build cache.

Solution

If you encounter this issue, delete, and recreate the workload in Tanzu Application Platform, or delete and recreate the image in Tanzu Build Service.

Switching from `buildservice.kp_default_repository` to `shared.image_registry`

Symptom

After switching to using the `shared.image_registry` fields in `tap-values.yaml`, your workloads might start failing with a `TemplateRejectedByAPIServer` error, with the error message: `admission webhook "validation.webhook.kpack.io" denied the request: validation failed: Immutable field changed: spec.tag`.

Cause

Tanzu Application Platform automatically appends `/buildservice` to the end of the repository specified in `shared.image_registry.project_path`. This updates the existing workload image tags, which is not allowed by Tanzu Build Service.

Solution

Delete the `images.kpack.io`, it has the same name as the workload. The workload then recreates it with valid values.

Alternatively, re-add the `buildservice.kp_default_repository_*` fields in the `tap-values.yaml`. You must set both the repository and the authentication fields to override the shared values. Set `kp_default_repository`, `kp_default_repository_secret.name`, and `kp_default_repository_secret.namespace`.

Failing builds during an upgrade

Symptom

During upgrades a large number of builds might be created due to buildpack and stack updates. Some of these builds might fail causing the workload to be in an unhealthy state.

Cause

Builds fail due to transient network issues.

Solution

This resolves itself on subsequent builds after a code change and does not affect the running application.

If you do not want to wait for subsequent builds to run, you can use either the Tanzu Build Service plug-in for the Tanzu CLI or the open source kpack CLI to trigger a build manually.

If using the Tanzu CLI, manually trigger a build as follows:

1. List the image resources in the developer namespace by running:

   ```
   tanzu build-service image list -n DEVELOPER-NAMESPACE
   ```

2. Manually trigger the image resources to re-run builds for each failing image by running:

   ```
   tanzu build-service image trigger IMAGE-NAME -n DEVELOPER-NAMESPACE
   ```

If using the kpack CLI, manually trigger a build as follows:

1. List the image resources in the developer namespace by running:

   ```
   kp image list -n DEVELOPER-NAMESPACE
   ```

2. Manually trigger the image resources to re-run builds for each failing image by running:

   ```
   kp image trigger IMAGE-NAME -n DEVELOPER-NAMESPACE
   ```

Create a GitHub build action (Alpha)

This topic tells you how to use a GitHub action to create a Tanzu Build Service build on a cluster.

Important

Alpha features are experimental and are not ready for production use. Configuration and behavior is likely to change, and functionality might be removed in a future release.

Prerequisites

- Ensure that Tanzu Application Platform is installed.

Procedure

Developer namespace

1. Create a developer namespace where the build resource will be created.

```
kubectl create namespace DEVELOPER-NAMESPACE
```

2. Create a service account in the `DEVELOPER-NAMESPACE` that has access to the registry credentials. This service account name will be used in the action.

Access to Kubernetes API server

The GitHub action talks directly to the Kubernetes API server, if you are running this on github.com with the default action runners, ensure that your API server is accessible from GitHub's IP ranges. Alternatively, it might be possible to run the action on a custom runner within your firewall (with access to the Tanzu Application Platform cluster).

Permissions Required

These are the minimum permissions required on the Tanzu Build Service cluster:

```
```bash
ClusterRole
 └ kpack.io
   └ clusterbuilders verbs=[get]
Role (DEVELOPER NAMESPACE)
 ├ ''
 │ ├ pods verbs=[get watch list] ✔
 │ └ pods/log verbs=[get] ✔
 └ kpack.io
   └ builds verbs=[get watch list create delete] ✔
```
```

This example contains the minimum required permissions:

```
```yaml
apiVersion: v1
kind: Namespace
metadata:
   name: DEVELOPER_NAMESPACE
---
apiVersion: v1
kind: ServiceAccount
metadata:
   namespace: DEVELOPER_NAMESPACE
   name: github-actions
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
   name: github-actions
subjects:
   - kind: ServiceAccount
     namespace: DEVELOPER_NAMESPACE
     name: github-actions
roleRef:
   kind: ClusterRole
   name: github-actions
   apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
   name: github-actions
   namespace: DEVELOPER_NAMESPACE
subjects:
   - kind: ServiceAccount
```
```

```
     namespace: DEVELOPER_NAMESPACE
     name: github-actions
roleRef:
   kind: Role
   name: github-actions
   apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
   name: github-actions
rules:
   - apiGroups: ['kpack.io']
     resources:
        - clusterbuilders
     verbs: ['get']
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
   name: github-actions
   namespace: DEVELOPER_NAMESPACE
rules:
   - apiGroups: ['']
     resources: ['pods']
     verbs: ['get', 'watch', 'list']
   - apiGroups: ['']
     resources: ['pods/log']
     verbs: ['get']
   - apiGroups: ['kpack.io']
     resources:
        - builds
     verbs: ['get', 'watch', 'list', 'create', 'delete']
```

To access the values on Google Kubernetes Engine (steps might vary on other IaaS providers):

```
  ```console
  DEV_NAMESPACE=DEVELOPER_NAMESPACE
  SECRET=$(kubectl get sa github-actions -oyaml -n $DEV_NAMESPACE | yq '.secrets[0].na
me')

  CA_CERT=$(kubectl get secret $SECRET -oyaml -n $DEV_NAMESPACE | yq '.data."ca.crt"')
  NAMESPACE=$(kubectl get secret $SECRET -oyaml -n $DEV_NAMESPACE | yq .data.namespace
| base64 -d)
  TOKEN=$(kubectl get secret $SECRET -oyaml -n $DEV_NAMESPACE | yq .data.token | base6
4 -d)
  SERVER=$(kubectl config view --minify | yq '.clusters[0].cluster.server')
  ```
```

Create the required secrets on the repository through GitHub.com or through the `gh` CLI:

```
```bash
gh secret set CA_CERT --app actions --body "$CA_CERT"
gh secret set NAMESPACE --app actions --body "$NAMESPACE"
gh secret set TOKEN --app actions --body "$TOKEN"
gh secret set SERVER --app actions --body "$SERVER"
```
```

Use the action

1. To use the action in a workflow, run the following YAML:

```
- uses: vmware-tanzu/build-image-action@v1-alpha
  with:
    ## Authorization
    # Host of the API server
    server: `${{ secrets.SERVER }}`
    # CA Certificate of the API Server
    ca_cert: `${{ secrets.CA_CERT }}`
    # Service Account token to access Kubernetes
    token: `${{ secrets.TOKEN }}`
    # _(required)_ The namespace to create the build resource in
    namespace: `${{ secrets.NAMESPACE }}`

    ## Image configuration
    # _(required)_ Destination for the built image
    # Example: gcr.io/<my-project>/<my-image>
    destination: ''
    # Optional list of build time environment variables
    env: ''
    # Name of the service account in the namespace, defaults to `default`
    serviceAccountName: ''
    # Name of the cluster builder to use, defaults to `default`
    clusterBuilder: ''
    # Max active time that the pod can run for in seconds, defaults to 3600
    timeout:
```

For example:

```
- name: Build Image
  id: build
  uses: vmware-tanzu/build-image-action@v1-alpha
  with:
    # Authorization
    server: ${{ secrets.SERVER }}
    token: ${{ secrets.TOKEN }}
    ca_cert: ${{ secrets.CA_CERT }}
    namespace: ${{ secrets.NAMESPACE }}
    # Image configuration
    destination: gcr.io/project-id/name-for-image
    serviceAccountName: my-sa-that-has-access-to-reg-credentials
    env: |
      BP_JAVA_VERSION=17
```

2. The previous step should output the full name, including the SHA of the built image. To use the output in a subsequent step:

```
- name: Do something with image
  run:
    echo "${{ steps.build.outputs.name }}"
```

Debugging

To run this action in “debug” mode, add a secret called `ACTIONS_STEP_DEBUG` with the value set to `true` as documented in the GitHub Action Docs.

Overview of Tanzu Buildpacks

This topic describes how to use Tanzu Buildpacks in Tanzu Application Platform (commonly known as TAP).

Tanzu Buildpacks provide framework and runtime support for applications. Buildpacks examine your applications to determine what dependencies to download, install, and how to configure the

applications to communicate with bound services.

Tanzu Buildpacks use open-source Paketo Buildpacks to allow Tanzu Application Platform users to turn their application source code into container images.

In Tanzu Application Platform v1.6 and later, builders, stacks, and buildpacks are packaged separately from Tanzu Build Service. They are included in the Full, Iterate, and Build installation profiles. All buildpacks follow the package name format `*.buildpacks.tanzu.vmware.com`.

For information about how to install buildpacks and stacks and the versions available in the `lite` and `full` profiles, see About lite and full dependencies.

For information about how to configure Tanzu Buildpacks features and components, see Tanzu Buildpack Documentation.

Overview of Tanzu Buildpacks

This topic describes how to use Tanzu Buildpacks in Tanzu Application Platform (commonly known as TAP).

Tanzu Buildpacks provide framework and runtime support for applications. Buildpacks examine your applications to determine what dependencies to download, install, and how to configure the applications to communicate with bound services.

Tanzu Buildpacks use open-source Paketo Buildpacks to allow Tanzu Application Platform users to turn their application source code into container images.

In Tanzu Application Platform v1.6 and later, builders, stacks, and buildpacks are packaged separately from Tanzu Build Service. They are included in the Full, Iterate, and Build installation profiles. All buildpacks follow the package name format `*.buildpacks.tanzu.vmware.com`.

For information about how to install buildpacks and stacks and the versions available in the `lite` and `full` profiles, see About lite and full dependencies.

For information about how to configure Tanzu Buildpacks features and components, see Tanzu Buildpack Documentation.

Overview of Tanzu Developer Portal

Tanzu Developer Portal is a tool for your developers to view your applications and services running for your organization. This portal provides a central location in which you can view dependencies, relationships, technical documentation, and the service status.

Tanzu Developer Portal is built from the Cloud Native Computing Foundation's project Backstage.

Tanzu Developer Portal consists of the following components:

- **Your organization catalog:**

  The catalog serves as the primary visual representation of your running services (components) and applications (systems).

- **Tanzu Developer Portal plug-ins:**

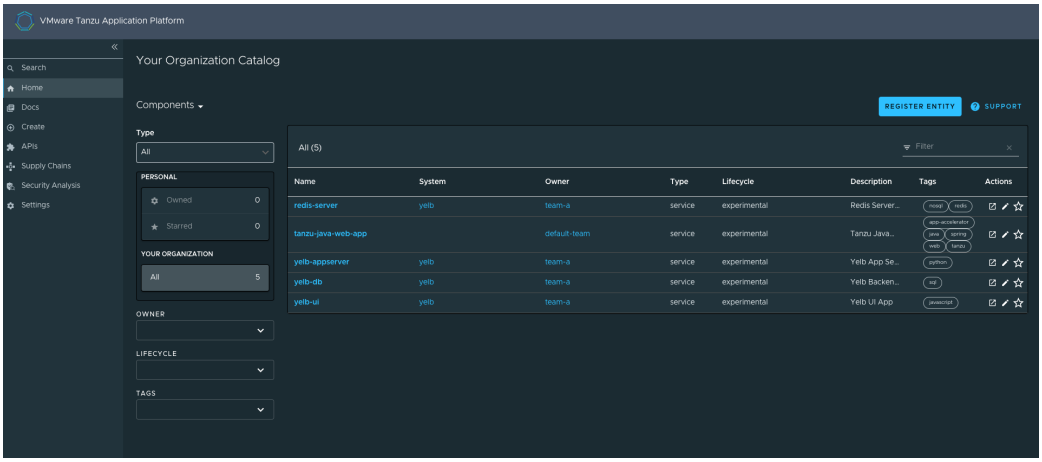
  These plug-ins expose capabilities regarding specific Tanzu Application Platform tools. Initially the included plug-ins are:

  - Runtime Resources Visibility
  - Application Live View
  - Application Accelerator
  - API Documentation

    - Supply Chain Choreographer

- **TechDocs:**

  This plug-in enables you to store your technical documentation in Markdown format in a source-code repository and display it alongside the relevant catalog entries.



- **Search:**

  This plug-in enables you to search your organization's catalog, including domains, systems, components, APIs, accelerators, and TechDocs.

- **A Git repository:**

  Tanzu Developer Portal stores the following in a Git repository:

    - The structure for your application catalog.
    - Your technical documentation about the catalog items, if you enable Tanzu Developer Portal TechDocs capabilities.

You can host the structure for your application catalog and your technical documentation in the same repository as your source code.

Overview of Tanzu Developer Portal

Tanzu Developer Portal is a tool for your developers to view your applications and services running for your organization. This portal provides a central location in which you can view dependencies, relationships, technical documentation, and the service status.

Tanzu Developer Portal is built from the Cloud Native Computing Foundation's project Backstage.

Tanzu Developer Portal consists of the following components:

- **Your organization catalog:**

  The catalog serves as the primary visual representation of your running services (components) and applications (systems).

- **Tanzu Developer Portal plug-ins:**

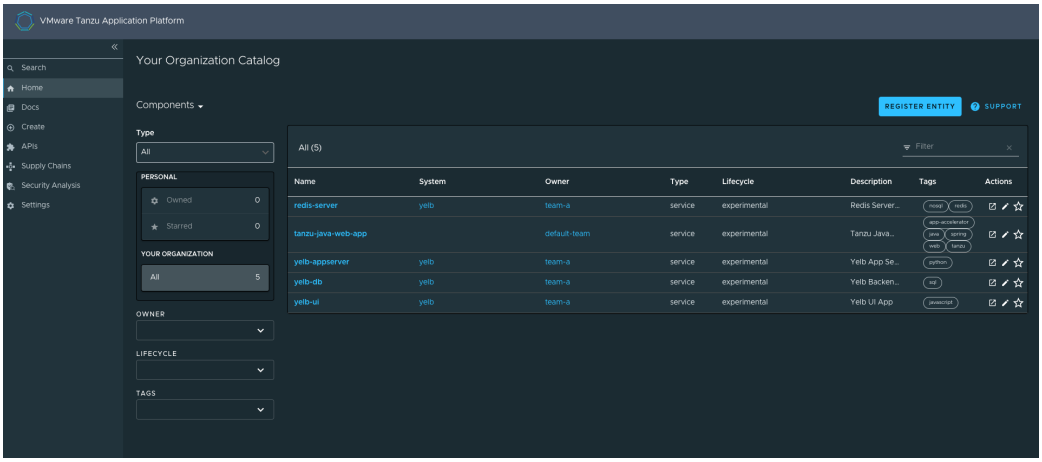
  These plug-ins expose capabilities regarding specific Tanzu Application Platform tools. Initially the included plug-ins are:

    - Runtime Resources Visibility
    - Application Live View
    - Application Accelerator
    - API Documentation

    - Supply Chain Choreographer

- **TechDocs:**

  This plug-in enables you to store your technical documentation in Markdown format in a source-code repository and display it alongside the relevant catalog entries.



- **Search:**

  This plug-in enables you to search your organization's catalog, including domains, systems, components, APIs, accelerators, and TechDocs.

- **A Git repository:**

  Tanzu Developer Portal stores the following in a Git repository:

    - The structure for your application catalog.
    - Your technical documentation about the catalog items, if you enable Tanzu Developer Portal TechDocs capabilities.

You can host the structure for your application catalog and your technical documentation in the same repository as your source code.

Install Tanzu Developer Portal

This topic tells you how to install Tanzu Developer Portal from the Tanzu Application Platform package repository.

> **Note**
>
> Follow the steps in this topic if you do not want to use a profile to install Tanzu Developer Portal. For more information about profiles, see Components and installation profiles.

Prerequisites

Before installing Tanzu Developer Portal:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see the Tanzu Application Platform Prerequisites.
- Create a Git repository for Tanzu Developer Portal software catalogs, with a token allowing read access. Supported Git infrastructure includes:
    - GitHub

    - GitLab
    - Azure DevOps
- Install Tanzu Developer Portal Blank Catalog
    1. Go to the Tanzu Application Platform section of VMware Tanzu Network.
    2. Under the list of available files to download, open the **tanzu-developer-portal-catalogs-latest** folder.
    3. Extract Tanzu Developer Portal Blank Catalog to your Git repository. This serves as the configuration location for your organization's Catalog inside Tanzu Developer Portal.

Procedure

To install Tanzu Developer Portal on a compliant Kubernetes cluster:

1. List version information for the package by running:

    ```
    tanzu package available list tap-gui.tanzu.vmware.com --namespace tap-install
    ```

    For example:

    ```
    $ tanzu package available list tap-gui.tanzu.vmware.com --namespace tap-install
    - Retrieving package versions for tap-gui.tanzu.vmware.com...
      NAME                      VERSION     RELEASED-AT
      tap-gui.tanzu.vmware.com  1.0.1       2022-01-10T13:14:23Z
    ```

2. (Optional) Make changes to the default installation settings by running:

    ```
    tanzu package available get tap-gui.tanzu.vmware.com/VERSION-NUMBER --values-sc
    hema --namespace \
    tap-install
    ```

    Where `VERSION-NUMBER` is the number you discovered previously. For example, `1.0.1`.

    For more information about values schema options, see the individual product documentation.

3. Create `tap-gui-values.yaml` and paste in the following YAML:

    ```
    ingressEnabled: true
    ingressDomain: "INGRESS-DOMAIN"
    app_config:
      catalog:
        locations:
          - type: url
            target: https://GIT-CATALOG-URL/catalog-info.yaml
    ```

    Where:

    - `INGRESS-DOMAIN` is the subdomain for the host name that you point at the `tanzu-shared-ingress` service's External IP address.
    - `GIT-CATALOG-URL` is the path to the `catalog-info.yaml` catalog definition file. It is from either the included Blank catalog (provided as an additional download named **Blank Tanzu Developer Portal Catalog**) or a Backstage-compliant catalog that you've already built and posted on the Git infrastructure specified in Adding Tanzu Developer Portal integrations.

4. Install the package by running:

```
tanzu package install tap-gui \
 --package tap-gui.tanzu.vmware.com \
 --version VERSION -n tap-install \
 --values-file tap-gui-values.yaml
```

Where `VERSION` is the version that you want. For example, `1.0.1`.

For example:

```
$ tanzu package install tap-gui --package tap-gui.tanzu.vmware.com --version 1.
0.1 -n \
tap-install --values-file tap-gui-values.yaml
- Installing package 'tap-gui.tanzu.vmware.com'
| Getting package metadata for 'tap-gui.tanzu.vmware.com'
| Creating service account 'tap-gui-default-sa'
| Creating cluster admin role 'tap-gui-default-cluster-role'
| Creating cluster role binding 'tap-gui-default-cluster-rolebinding'
| Creating secret 'tap-gui-default-values'
- Creating package resource
- Package install status: Reconciling

 Added installed package 'tap-gui' in namespace 'tap-install'
```

5. Verify that the package installed by running:

```
tanzu package installed get tap-gui -n tap-install
```

For example:

```
$ tanzu package installed get tap-gui -n tap-install
| Retrieving installation details for cc...
NAME:                    tap-gui
PACKAGE-NAME:            tap-gui.tanzu.vmware.com
PACKAGE-VERSION:         1.0.1
STATUS:                  Reconcile succeeded
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`.

6. To access Tanzu Developer Portal, use the service you exposed in the `service_type` field in the values file.

Runtime configuration options for Tanzu Developer Portal

You can provide a series of options to the Tanzu Developer Portal package to configure it and do some basic runtime customization.

Identify the Tanzu Developer Portal version you have available

From the Tanzu CLI, discover the Tanzu Developer Portal package version that is available to configure by running:

```
tanzu package available get tap-gui.tanzu.vmware.com -n INSTALL-NAMESPACE
```

Where `INSTALL-NAMESPACE` is the namespace in which you configured the Tanzu Application Platform installation. In most cases the namespace is `tap-install`.

For example:

```
$ tanzu package available get tap-gui.tanzu.vmware.com -n tap-install

NAME:                    tap-gui.tanzu.vmware.com
DISPLAY-NAME:            Tanzu Application Platform GUI
CATEGORIES:
SHORT-DESCRIPTION:       web app graphical user interface for Tanzu Application Platfor
m
LONG-DESCRIPTION:        web app graphical user interface for Tanzu Application Platfor
m
PROVIDER:                VMware
MAINTAINERS:             - name: VMware
SUPPORT-DESCRIPTION:     https://tanzu.vmware.com/support

  VERSION  RELEASED-AT
  1.7.6    2023-10-17 00:25:21 +0000 UTC
```

Display the possible values options for Tanzu Developer Portal

From the Tanzu CLI, identify possible values options for Tanzu Developer Portal by running:

```
tanzu package available get tap-gui.tanzu.vmware.com/VERSION --values-schema -n INSTAL
L-NAMESPACE
```

Where:

- `VERSION` is the Tanzu Developer Portal package version you learned earlier
- `INSTALL-NAMESPACE` is the namespace in which you configured the Tanzu Application Platform installation. In most cases the namespace is `tap-install`.

For example:

```
$ tanzu package available get tap-gui.tanzu.vmware.com/1.7.6 --values-schema -n tap-in
stall

  KEY                                                              DEFAULT   TYPE
DESCRIPTION
  #Details of all the possible configuration values
  ...
```

Runtime customization of Tanzu Developer Portal

This topic tells you how to use runtime configuration parameters to customize the existing functions of your currently running portal.

This configuration entails changing the values file that you built while installing the portal, and then re-applying those changes to the running configuration. The actual steps can vary depending on how you performed the installation. The basic steps of updating your values file are described in Customize your package installation.

To learn how to add additional plug-ins and features to your portal by using build-time customization, see Overview of Configurator.

Customize branding

To customize the branding in your portal, you can choose the name of the portal and the logo for it. To make these customizations:

1. Provide additional configuration parameters to the `app_config` section of `tap-values.yaml`:

```
tap_gui:
  app_config:
    customize:
      custom_logo: 'BASE-64-IMAGE'
      custom_name: 'PORTAL-NAME'
```

Where:

- `BASE-64-IMAGE` is the image encoded in base64. A 512-pixel by 512-pixel PNG image with a transparent background is optimal.
- `PORTAL-NAME` is the name of your portal, such as `Our Custom Developer Experience Portal`.

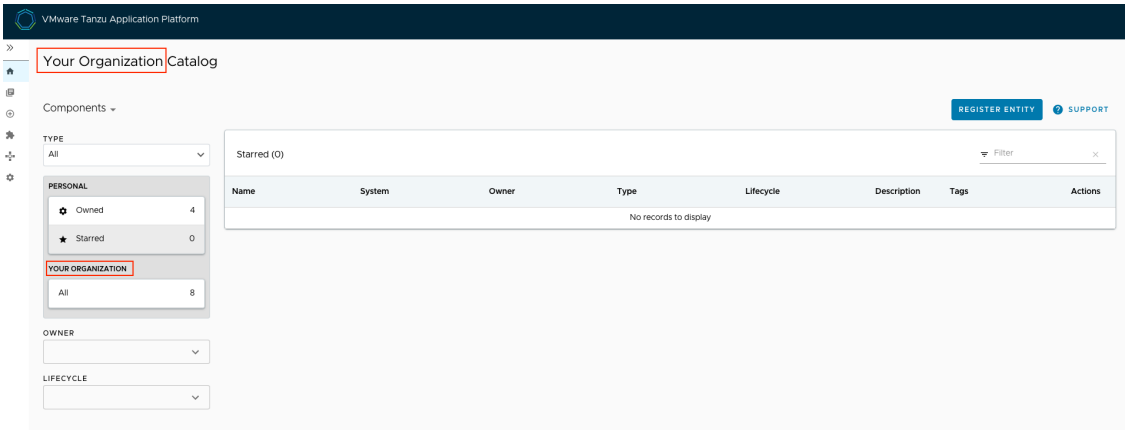
2. Reinstall your Tanzu Developer Portal package by following steps in Upgrading Tanzu Application Platform.

After the updated values configuration file is applied in Tanzu Developer Portal, you see the customized version of your portal.

If there is an error in any of the supplied images encoded in base64 or in your choice of portal name, Tanzu Developer Portal reverts to the original branding template.

Customize the Software Catalog page

You can customize the name of your organization on the Software Catalog page of Tanzu Developer Portal. By default, the portal displays **Your Organization** next to **Catalog** and in the selection box.



Customize the name of the organization

To customize the name of the organization for the software catalog in your portal:

1. Provide additional configuration parameters to the `app_config` section of your `tap-values.yaml` file:

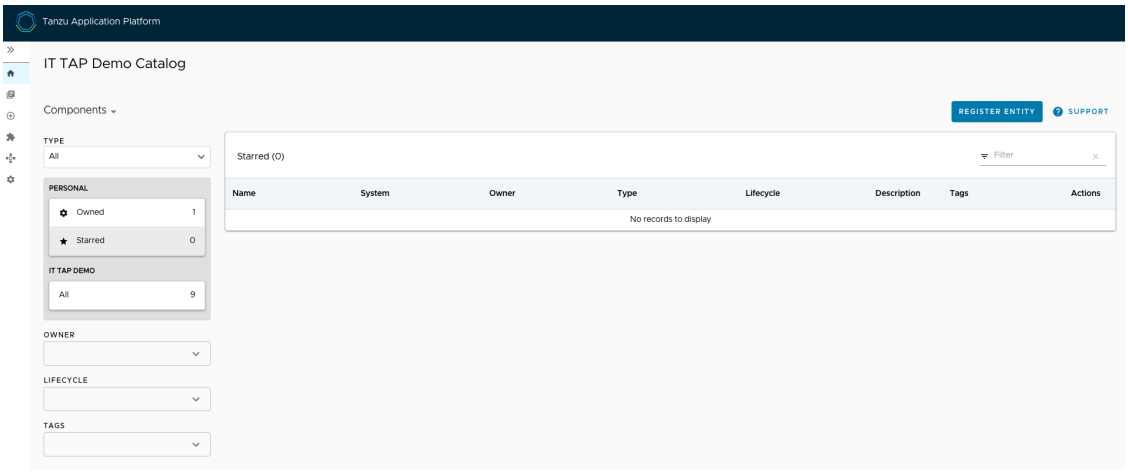
```
tap_gui:
  app_config:
    organization:
      name: 'ORG-NAME'
```

Where `ORG-NAME` is the name of your organization for the software catalog, such as `Our Organization Name`. You don't need to add `Catalog` to the `ORG-NAME`.

2. Reinstall your Tanzu Developer Portal package by following the steps in Upgrading Tanzu Application Platform.

After the updated values configuration file is applied in Tanzu Developer Portal, you see the customized version of your portal.

If there is an error in the provided configuration parameters, Tanzu Developer Portal reverts to the original organization name.



Prevent changes to the software catalog

You can deactivate the **Register Entity** button to prevent a user from making changes to the software catalog, including registering and deregistering locations. To do so, add `readonly: true` to the `catalog` section in `tap-values.yaml`, as in this example:

```
tap_gui:
  app_config:
    catalog:
      readonly: true
```

Customize the Authentication page

To customize the portal name on the **Authentication** page and the name of the browser tab for Tanzu Developer Portal:

1. Provide additional configuration parameters to the `app_config` section of your `tap-values.yaml` file:

   ```
   tap_gui:
     app_config:
       app:
         title: 'CUSTOM-TAB-NAME'
   ```

   Where `CUSTOM-TAB-NAME` is the name on the Authentication page and the browser tab of your portal, such as `Our Organization Full Name`.

2. Reinstall your Tanzu Developer Portal package by following the steps in Upgrading Tanzu Application Platform.

After the updated values configuration file is applied in Tanzu Developer Portal, you see the customized version of your portal.

Customize the default view

You can set your default route when the user is accessing your portal. Without this customization, when the user accesses the Tanzu Developer Portal URL, it displays the list of owned components of the software catalog.

To change the default view:

1. Provide additional configuration parameters to the `app_config` section of your `tap-values.yaml` file:

   ```
   tap_gui:
     app_config:
       customize:
         default_route: 'YOUR-PREFERRED-ROUTE'
   ```

   Where `YOUR-PREFERRED-ROUTE` is the path to the route that the portal uses by default. For example, you can type `/catalog?filters%5Bkind%5D=component&filters%5Buser%5D=all` to show all components of the software catalog instead of defaulting to owned components. As another example, you can type `/create` to show Application Accelerator when the portal starts.

   Caution

   Tanzu Developer Portal redirects you to `tap-gui.INGRESS-DOMAIN/YOUR-PREFERRED-ROUTE` even if there is an error in `YOUR-PREFERRED-ROUTE`.

2. Reinstall your Tanzu Developer Portal package by following the steps in Upgrading Tanzu Application Platform.

After the updated values configuration file is applied in Tanzu Developer Portal, you see the customized version of your portal.

Customize security banners

You can instruct Tanzu Developer Portal to create security banners on the top and bottom of the page. To add security banners to Tanzu Developer Portal:

1. Provide additional configuration parameters to the `app_config` section of your `tap-values.yaml` file, as in the following example:

   ```
   tap_gui:
     app_config:
       customize:
         banners:
           text: 'CUSTOM-TEXT'
           color: 'OPTIONAL-CUSTOM-TEXT-COLOR'
           bg: 'CUSTOM-BACKGROUND-COLOR'
           link: 'OPTIONAL-LINK'
   ```

   Where:

   - `CUSTOM-TEXT` is the text that is displayed in the banner. Keep this text short to accommodate various screen sizes.
   - `OPTIONAL-CUSTOM-TEXT-COLOR` is the color of the text displayed in the banner. Setting this is optional. It accepts CSS colors, such as `#ffffff`. The default color is `#FFFFFF`.
   - `CUSTOM-BACKGROUND-COLOR` is the color of the banner itself. Setting this is optional. It accepts CSS colors, such as `#ffffff`. The default color is `#C23B2E`
   - `OPTIONAL-LINK` is the link to which your text redirects. Setting this is optional.

2. Reinstall your Tanzu Developer Portal package by following the steps in Upgrading Tanzu Application Platform.

After the updated values configuration file is applied in Tanzu Developer Portal, the customized version of your portal is displayed.

Runtime customization of Tanzu Developer Portal

This topic tells you how to use runtime configuration parameters to customize the existing functions of your currently running portal.

This configuration entails changing the values file that you built while installing the portal, and then re-applying those changes to the running configuration. The actual steps can vary depending on how you performed the installation. The basic steps of updating your values file are described in Customize your package installation.

To learn how to add additional plug-ins and features to your portal by using build-time customization, see Overview of Configurator.

Customize branding

To customize the branding in your portal, you can choose the name of the portal and the logo for it. To make these customizations:

1. Provide additional configuration parameters to the `app_config` section of `tap-values.yaml`:

   ```
   tap_gui:
     app_config:
       customize:
         custom_logo: 'BASE-64-IMAGE'
         custom_name: 'PORTAL-NAME'
   ```

   Where:

   - `BASE-64-IMAGE` is the image encoded in base64. A 512-pixel by 512-pixel PNG image with a transparent background is optimal.
   - `PORTAL-NAME` is the name of your portal, such as `Our Custom Developer Experience Portal`.

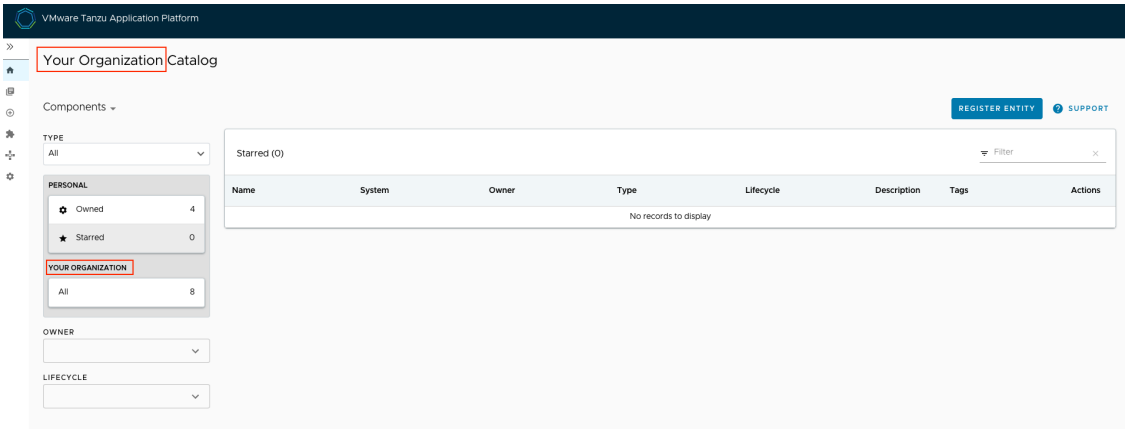
2. Reinstall your Tanzu Developer Portal package by following steps in Upgrading Tanzu Application Platform.

After the updated values configuration file is applied in Tanzu Developer Portal, you see the customized version of your portal.

If there is an error in any of the supplied images encoded in base64 or in your choice of portal name, Tanzu Developer Portal reverts to the original branding template.

Customize the Software Catalog page

You can customize the name of your organization on the Software Catalog page of Tanzu Developer Portal. By default, the portal displays **Your Organization** next to **Catalog** and in the selection box.



Customize the name of the organization

To customize the name of the organization for the software catalog in your portal:

1. Provide additional configuration parameters to the `app_config` section of your `tap-values.yaml` file:

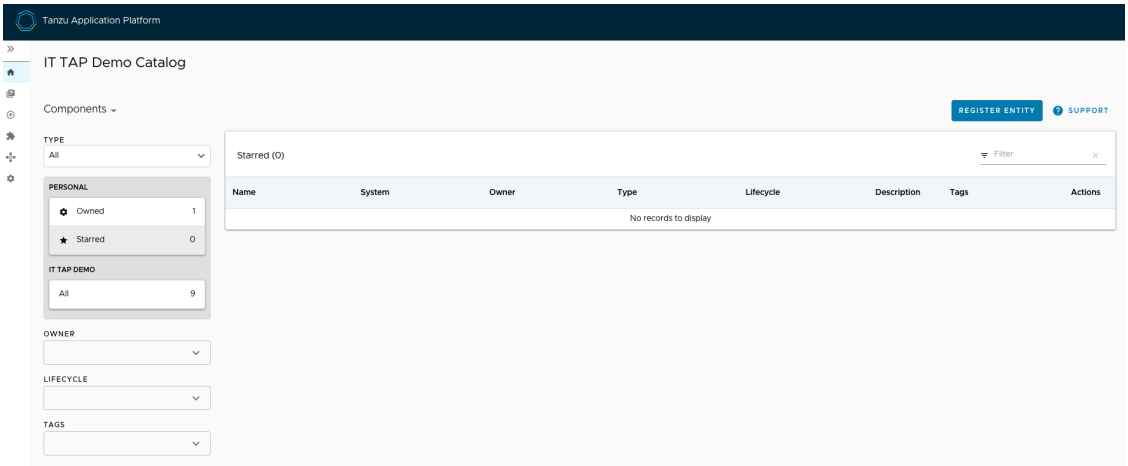
   ```
   tap_gui:
     app_config:
       organization:
         name: 'ORG-NAME'
   ```

   Where `ORG-NAME` is the name of your organization for the software catalog, such as `Our Organization Name`. You don't need to add `Catalog` to the `ORG-NAME`.

2. Reinstall your Tanzu Developer Portal package by following the steps in Upgrading Tanzu Application Platform.

After the updated values configuration file is applied in Tanzu Developer Portal, you see the customized version of your portal.

If there is an error in the provided configuration parameters, Tanzu Developer Portal reverts to the original organization name.



Prevent changes to the software catalog

You can deactivate the **Register Entity** button to prevent a user from making changes to the software catalog, including registering and deregistering locations. To do so, add `readonly: true` to the `catalog` section in `tap-values.yaml`, as in this example:

```
tap_gui:
  app_config:
    catalog:
      readonly: true
```

Customize the Authentication page

To customize the portal name on the **Authentication** page and the name of the browser tab for Tanzu Developer Portal:

1. Provide additional configuration parameters to the `app_config` section of your `tap-values.yaml` file:

   ```
   tap_gui:
     app_config:
       app:
         title: 'CUSTOM-TAB-NAME'
   ```

   Where `CUSTOM-TAB-NAME` is the name on the Authentication page and the browser tab of your portal, such as `Our Organization Full Name`.

2. Reinstall your Tanzu Developer Portal package by following the steps in Upgrading Tanzu Application Platform.

After the updated values configuration file is applied in Tanzu Developer Portal, you see the customized version of your portal.

Customize the default view

You can set your default route when the user is accessing your portal. Without this customization, when the user accesses the Tanzu Developer Portal URL, it displays the list of owned components of the software catalog.

To change the default view:

1. Provide additional configuration parameters to the `app_config` section of your `tap-values.yaml` file:

   ```
   tap_gui:
     app_config:
       customize:
         default_route: 'YOUR-PREFERRED-ROUTE'
   ```

   Where `YOUR-PREFERRED-ROUTE` is the path to the route that the portal uses by default. For example, you can type `/catalog?filters%5Bkind%5D=component&filters%5Buser%5D=all` to show all components of the software catalog instead of defaulting to owned components. As another example, you can type `/create` to show Application Accelerator when the portal starts.

   Caution

   Tanzu Developer Portal redirects you to `tap-gui.INGRESS-DOMAIN/YOUR-PREFERRED-ROUTE` even if there is an error in `YOUR-PREFERRED-ROUTE`.

2. Reinstall your Tanzu Developer Portal package by following the steps in Upgrading Tanzu Application Platform.

After the updated values configuration file is applied in Tanzu Developer Portal, you see the customized version of your portal.

Customize security banners

You can instruct Tanzu Developer Portal to create security banners on the top and bottom of the page. To add security banners to Tanzu Developer Portal:

1. Provide additional configuration parameters to the `app_config` section of your `tap-values.yaml` file, as in the following example:

   ```
   tap_gui:
     app_config:
       customize:
         banners:
           text: 'CUSTOM-TEXT'
           color: 'OPTIONAL-CUSTOM-TEXT-COLOR'
           bg: 'CUSTOM-BACKGROUND-COLOR'
           link: 'OPTIONAL-LINK'
   ```

   Where:

   - `CUSTOM-TEXT` is the text that is displayed in the banner. Keep this text short to accommodate various screen sizes.
   - `OPTIONAL-CUSTOM-TEXT-COLOR` is the color of the text displayed in the banner. Setting this is optional. It accepts CSS colors, such as `#ffffff`. The default color is `#FFFFFF`.
   - `CUSTOM-BACKGROUND-COLOR` is the color of the banner itself. Setting this is optional. It accepts CSS colors, such as `#ffffff`. The default color is `#C23B2E`
   - `OPTIONAL-LINK` is the link to which your text redirects. Setting this is optional.

2. Reinstall your Tanzu Developer Portal package by following the steps in Upgrading Tanzu Application Platform.

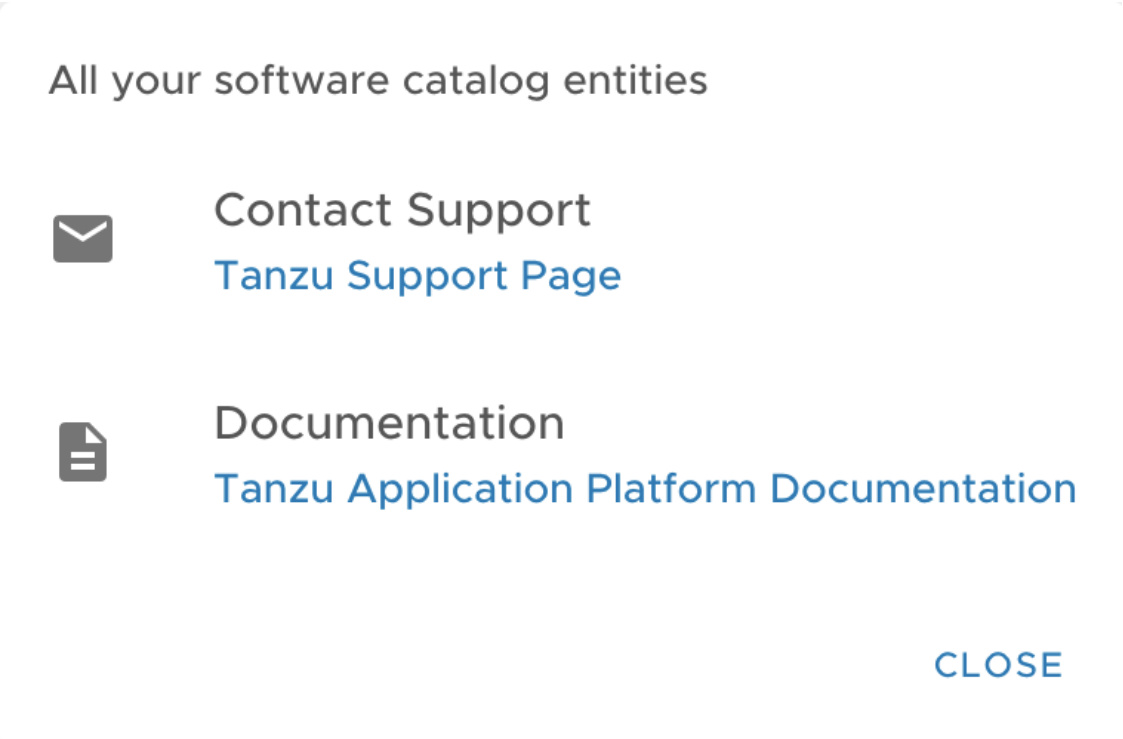
After the updated values configuration file is applied in Tanzu Developer Portal, the customized version of your portal is displayed.

Customize the Support menu

This topic describes how to customize the support menu.

Overview

Many important pages of Tanzu Developer Portal have a **Support** button that displays a pop-out menu. This menu contains a one-line description of the page the user is looking at, and a list of support item groupings.



As standard, there are two support item groupings:

- Contact Support, which is marked with an **email** icon and contains a link to VMware Tanzu's support portal.
- Documentation, which is marked with a **docs** icon and contains a link to the Tanzu Application Platform documentation that you are currently reading.

Customizing

The set of support item groupings is completely customizable. However, you might want to offer custom in-house links for your Tanzu Application Platform users rather than simply sending them to VMware support and documentation. You can provide this configuration by using your `tap-values.yaml`. Here is a configuration snippet, which produces the default support menu:

```
tap_gui:
  app_config:
    app:
      support:
        url: https://tanzu.vmware.com/support
        items:
          - title: Contact Support
            icon: email
            links:
              - url: https://tanzu.vmware.com/support
                title: Tanzu Support Page
          - title: Documentation
            icon: docs
            links:
              - url: https://docs.vmware.com/en/VMware-Tanzu-Application-Platform/inde
x.html
                title: Tanzu Application Platform Documentation
```

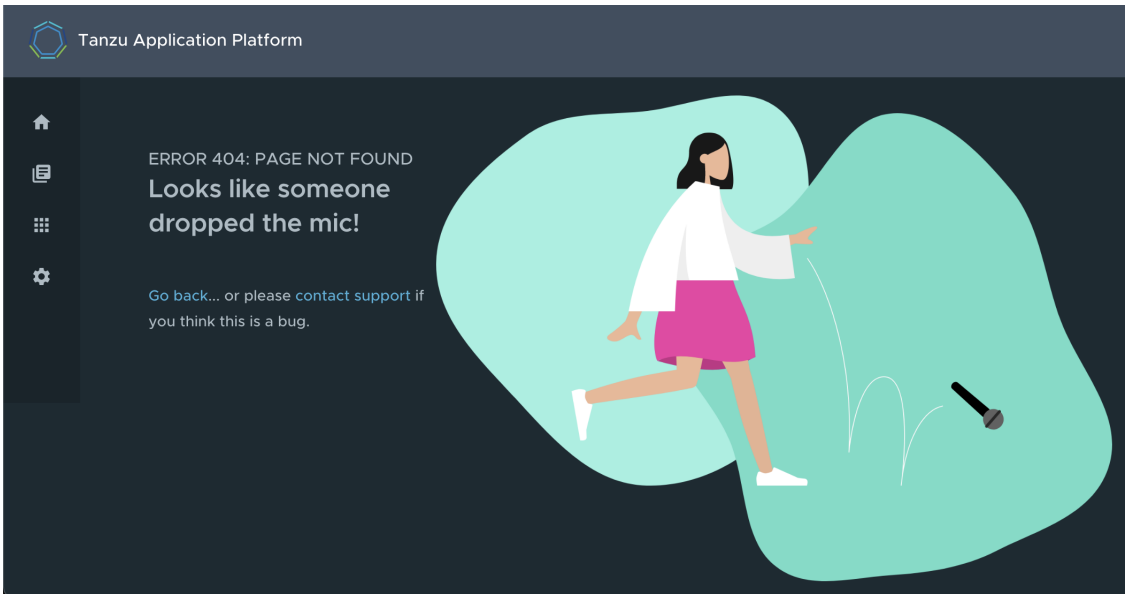
Structure of the support configuration

URL

The `url` field under the `support` section, for example,

```
    support:
      url: https://tanzu.vmware.com/support
```

provides the address of the **contact support** link that appears on error pages.



Items

The `items` field under the `support` section, for example, provides the set of support item groupings to display when the support menu is expanded.

Title

The `title` field on a support item grouping, for example,

```
      items:
        - title: Contact Support
```

provides the label for the grouping.

Icon

The `icon` field on a support item grouping, for example,

```
      items:
        - icon: email
```

provides the icon to use for that grouping. The valid choices are:

- `brokenImage`
- `catalog`
- `chat`
- `dashboard`
- `docs`
- `email`
- `github`

- `group`
- `help`
- `user`
- `warning`

Links

The `links` field on a support item grouping, for example,

```
        items:
          - links:
              - url: https://tanzu.vmware.com/support
                title: Tanzu Support Page
```

is a list of YAML objects that render as links. Each link has the text given by the `title` field and links to the value of the `url` field.

Customize the Tanzu Developer Portal telemetry collection

This section tells you how to customize telemetry collection for Tanzu Developer Portal.

Customize organization ID

By default, each instance of Tanzu Developer Portal is assigned to a random organization ID to ensure that your sensitive information is not revealed.

However, you can choose to customize your organization ID and self-identify. Doing so allows VMware to observe account-level telemetry, such as frequency of portal use, most popular function, and so on. All personally identifiable information remains anonymized in any case. The organization name is hashed to prevent VMware from identifying you by the value.

To customize:

1. Provide additional configuration parameters to the `app_config` section of `tap-values.yaml`:

   ```
   tap_gui:
    app_config:
      pendoAnalytics:
        organizationId: 'ORGANIZATION-NAME'
   ```

   Where `ORGANIZATION-NAME` is the name of your organization or the name of the Tanzu Developer Portal instance of your choice. `ORGANIZATION-NAME` must be unique and static across an instance of Tanzu Developer Portal so that your organization name remains the same across refreshes of the Tanzu Application Platform database.

2. Reinstall your Tanzu Developer Portal package by following the steps in Upgrade Tanzu Application Platform.

After the updated values configuration file is applied in Tanzu Developer Portal, your organization ID is updated.

Access Tanzu Developer Portal

This topic tells you how to access Tanzu Developer Portal by using one of the following methods:

- Access with the LoadBalancer method (default)
- Access with the shared Ingress method

Access with the LoadBalancer method (default)

1. Verify that you specified the `service_type` for Tanzu Developer Portal in `tap-values.yaml`, as in this example:

   ```
   tap_gui:
     service_type: LoadBalancer
   ```

2. Obtain the external IP address of your LoadBalancer by running:

   ```
   kubectl get svc -n tap-gui
   ```

3. Access Tanzu Developer Portal by using the external IP address with the default port of 7000. It has the following form:

   ```
   http://EXTERNAL-IP:7000
   ```

   Where `EXTERNAL-IP` is the external IP address of your LoadBalancer.

Access with the shared Ingress method

The Ingress method of access for Tanzu Developer Portal uses the shared `tanzu-system-ingress` instance of Contour that is installed as part of the Profile installation.

1. The Ingress method of access requires that you have a DNS host name that you can point at the External IP address of the `envoy` service that the shared `tanzu-system-ingress` uses. Retrieve this IP address by running:

   ```
   kubectl get service envoy -n tanzu-system-ingress
   ```

   This returns a value similar to this example:

   ```
   $ kubectl get service envoy -n tanzu-system-ingress
   NAME    TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)
   AGE
   envoy   LoadBalancer   10.0.242.171   40.118.168.232   80:31389/TCP,443:31780/T
   CP   27h
   ```

   The IP address in the `EXTERNAL-IP` field is the one that you point a DNS host record to. Tanzu Developer Portal prepends `tap-gui` to your provided subdomain. This makes the final host name `tap-gui.YOUR-SUBDOMAIN`. You use this host name in the appropriate fields in the `tap-values.yaml` file mentioned later.

2. Specify parameters in `tap-values.yaml` related to Ingress. For example:

   ```
   shared:
     ingress_domain: "example.com"
   ```

3. Update your other host names in the `tap_gui` section of your `tap-values.yaml` with the new host name. For example:

   ```
   shared:
     ingress_domain: "example.com"

   tap_gui:
   # Existing tap-values.yaml above
     app_config:
       app:
         baseUrl: http://tap-gui.example.com # No port needed with Ingress
   ```

```
    integrations:
      github: # Other are integrations available
        - host: github.com
          token: GITHUB-TOKEN
    catalog:
      locations:
        - type: url
          target: https://GIT-CATALOG-URL/catalog-info.yaml
    backend:
      baseUrl: http://tap-gui.example.com # No port needed with Ingress
      cors:
        origin: http://tap-gui.example.com # No port needed with Ingress
```

4. Update your package installation with your changed `tap-values.yaml` file by running:

   ```
   tanzu package installed update tap --package tap.tanzu.vmware.com --version VER
   SION-NUMBER \
   --values-file tap-values.yaml -n tap-install
   ```

   Where `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.9.1`.

5. Use a web browser to access Tanzu Developer Portal at the host name that you provided.

Catalog operations

The software catalog setup procedures in this topic make use of Backstage. For more information about Backstage, see the Backstage documentation.

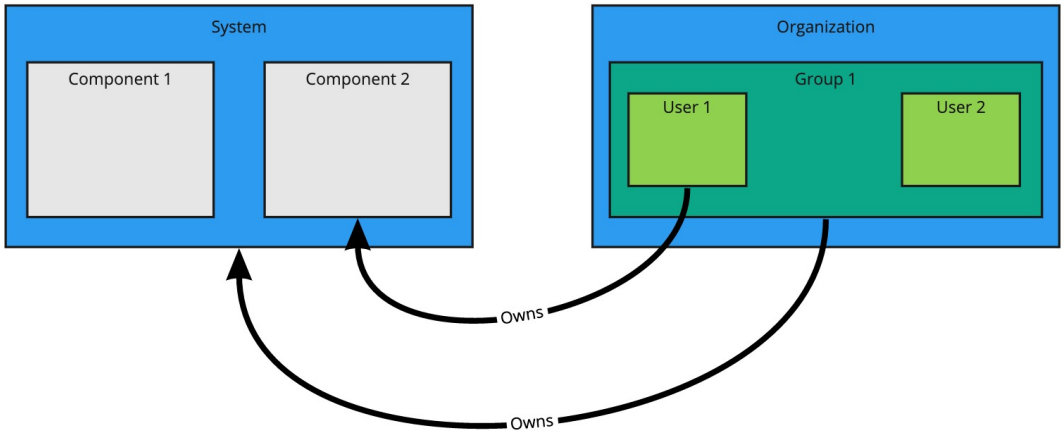
Adding catalog entities

This section describes how you can format your own catalog. Creating catalogs consists of building metadata YAML files stored together with the code. This information is read from a Git-compatible repository consisting of these YAML catalog definition files. Changes made to the catalog definitions on your Git infrastructure are automatically reflected every 200 seconds or when manually registered.

For each catalog entity kind you create, there is a file format you must follow. For information about all types of entities, see the Backstage documentation.

You can use the example blank catalog described in the Tanzu Developer Portal prerequisites as a foundation for creating user, group, system, and main component YAML files.

The organization contains Group 1, and Group 1 contains Users 1 and 2. System contains Components 1 and 2. User 1 owns Component 2. Group 1 owns System.



Users and groups

A user entity describes a specific person and is used for identity purposes. Users are members of one or more groups. A group entity describes an organizational team or unit.

Users and groups have different descriptor requirements in their descriptor files:

- User descriptor files require `apiVersion`, `kind`, `metadata.name`, and `spec.memberOf`.
- Group descriptor files require `apiVersion`, `kind`, and `metadata.name`. They also require `spec.type` and `spec.children` where `spec.children` is another group.

To link a logged-in user to a user entity, include the optional `spec.profile.email` field.

Sample user entity:

```
apiVersion: backstage.io/v1alpha1
kind: User
metadata:
  name: default-user
spec:
  profile:
    displayName: Default User
    email: guest@example.com
    picture: https://avatars.dicebear.com/api/avataaars/guest@example.com.svg?backgrou
nd=%23fff
  memberOf: [default-team]
```

Sample group entity:

```
apiVersion: backstage.io/v1alpha1
kind: Group
metadata:
  name: default-team
  description: Default Team
spec:
  type: team
  profile:
    displayName: Default Team
    email: team-a@example.com
    picture: https://avatars.dicebear.com/api/identicon/team-a@example.com.svg?backgro
und=%23fff
  parent: default-org
  children: []
```

For more information about user entities and group entities, see the Backstage documentation.

Systems

A system entity is a collection of resources and components.

System descriptor files require values for `apiVersion`, `kind`, `metadata.name`, and also `spec.owner` where `spec.owner` is a user or group.

A system has components when components specify the system name in the field `spec.system`.

Sample system entity:

```
apiVersion: backstage.io/v1alpha1
kind: System
metadata:
  name: backstage
  description: Tanzu Developer Portal System
spec:
  owner: default-team
```

For more information about system entities, see the Backstage documentation.

Components

A component describes a software component, or what might be described as a unit of software.

Component descriptor files require values for `apiVersion`, `kind`, `metadata.name`, `spec.type`, `spec.lifecycle`, and `spec.owner`.

Some useful optional fields are `spec.system` and `spec.subcomponentOf`, both of which link a component to an entity that it is part of.

```
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: backstage-component
  description: Tanzu Developer Portal Component
  annotations:
    'backstage.io/kubernetes-label-selector': 'app=backstage' #Identifies the Kubernet
es objects that make up this component
    'backstage.io/techdocs-ref': dir:. #TechDocs label
spec:
  type: service
  lifecycle: alpha
  owner: default-team
  system: backstage
```

For more information about component entities, see the Backstage documentation.

Update software catalogs

The following procedures describe how to update software catalogs.

Register components

To update your software catalog with new entities without re-deploying the entire `tap-gui` package:

1. Go to your **Software Catalog** page.
2. Click **Register Entity** at the top-right of the page.
3. Enter the full path to link to an existing entity file and start tracking your entity.
4. Import the entities and view them in your **Software Catalog** page.

Deregister components

To deregister an entity:

1. Go to your **Software Catalog** page.
2. Select the entity to deregister, such as component, group, or user.
3. Click the three dots at the top-right of the page and then click **Unregister...**.

Add or change organization catalog locations

To add or change organization catalog locations, you can use static configuration or you can use `GitLabDiscoveryProcessor` to discover and register catalog entities that match the configured path.

Use static configuration
To use static configuration to add or change catalog locations:

1. Update components by changing the catalog location in either the `app_config` section of `tap-gui-values.yaml` or the custom values file you used when installing. For example:

   ```
   tap_gui:
     app_config:
       catalog:
         locations:
           - type: url
             target: UPDATED-CATALOG-LOCATION
   ```

2. Register components by adding the new catalog location in either the `app_config` section of `tap-gui-values.yaml` or the custom values file you used when installing. For example:

   ```
   tap_gui:
     app_config:
       catalog:
         locations:
           - type: url
             target: EXISTING-CATALOG-LOCATION
           - type: url
             target: EXTRA-CATALOG-LOCATION
   ```

   When targeting GitHub, don't write the raw URL. Instead, use the URL that appears when you navigate to the file in the browser. The catalog processor cannot set up the files properly if you use the raw URL.

   - Example raw URL: `https://raw.githubusercontent.com/user/repo/catalog.yaml`
   - Example target URL: `https://github.com/user/repo/blob/main/catalog.yaml`

   When targeting GitLab, use a scoped route to the catalog file. This is a route with the `/-/` separator after the project name. If you don't use a scoped route, your entity fails to appear in the catalog.

   - Example unscoped URL: `https://gitlab.com/group/project/blob/main/catalog.yaml`
   - Example target URL: `https://gitlab.com/group/project/-/blob/main/catalog.yaml`

   For more information about static catalog configuration, see the Backstage documentation.

Use GitLabDiscoveryProcessor

To use `GitLabDiscoveryProcessor` to discover and register catalog entities:

1. Use `type: gitlab-discovery` to make `GitLabDiscoveryProcessor` crawl the GitLab instance to discover and register catalog entities that match the configured path. For more information, see the Backstage documentation.
2. Update the package to include the catalog:
   - If you installed Tanzu Developer Portal by using a profile, run:

     ```
     tanzu package installed update tap \
     --package tap.tanzu.vmware.com \
     --version PACKAGE-VERSION \
     --values-file tap-values.yaml \
     --namespace tap-install
     ```

   - If you installed Tanzu Developer Portal as an individual package, run:

```
tanzu package installed update tap-gui \
--package tap-gui.tanzu.vmware.com \
--version PACKAGE-VERSION \
--values-file tap-gui-values.yaml \
--namespace tap-install
```

3. Verify the status of this update by running:

```
tanzu package installed list -n tap-install
```

Install demo apps and their catalogs

To set up one of the demos, you can choose a blank catalog or a sample catalog.

Yelb system

The Yelb demo catalog in GitHub includes all the components that make up the Yelb system and the default Backstage components.

Install Yelb

To install Yelb:

1. Download the necessary file for running the Yelb application itself from GitHub.
2. Install the application on the Kubernetes cluster that you used for Tanzu Application Platform. Preserve the metadata labels on the Yelb application objects.

Install the Yelb catalog

To install the Yelb catalog:

1. In Tanzu Network, select your release from the drop-down menu.
2. Click **tanzu-developer-portal-catalogs-latest** > **Tanzu Application Platform Developer Portal Yelb Catalog** and download the catalog.
3. Unpack the downloaded TAR archive to a local drive.
4. Follow the earlier steps for Register components to register the `catalog-info.yaml` in the root of the unpacked archive and register all the catalog entities that constitute the Yelb system.

View resources on multiple clusters in Tanzu Developer Portal

You can configure Tanzu Developer Portal to retrieve Kubernetes object details from multiple clusters and then surface those details in the various Tanzu Developer Portal plug-ins.

Important

In this topic the terms `Build`, `Run`, and `View` describe the cluster's roles and distinguish which steps to apply to which cluster.

`Build` clusters are where the code is built and packaged, ready to be run.

`Run` clusters are where the Tanzu Application Platform workloads themselves run.

`View` clusters are where the Tanzu Developer Portal is run from.

In multicluster configurations, these can be separate clusters. However, in many configurations these can also be the same cluster.

Set up a Service Account to view resources on a cluster

To view resources on the `Build` or `Run` clusters, create a service account on the `View` cluster that can `get`, `watch`, and `list` resources on those clusters.

You first create a `ClusterRole` with these rules and a `ServiceAccount` in its own `Namespace`, and then bind the `ClusterRole` to the `ServiceAccount`. Depending on your topology, not every cluster has all of the following objects. For example, the `Build` cluster doesn't have any of the `serving.knative.dev` objects, by design, because it doesn't run the workloads themselves. You can edit the following object lists to reflect your topology.

To set up a Service Account to view resources on a cluster:

1. Copy this YAML content into a file called `tap-gui-viewer-service-account-rbac.yaml`.

```
apiVersion: v1
kind: Namespace
metadata:
  name: tap-gui
---
apiVersion: v1
kind: ServiceAccount
metadata:
  namespace: tap-gui
  name: tap-gui-viewer
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: tap-gui-read-k8s
subjects:
- kind: ServiceAccount
  namespace: tap-gui
  name: tap-gui-viewer
roleRef:
  kind: ClusterRole
  name: k8s-reader
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: k8s-reader
rules:
- apiGroups: ['']
  resources:
  - pods
  - pods/log
  - services
  - configmaps
  - limitranges
  - namespaces
  verbs: ['get', 'watch', 'list']
- apiGroups: ['metrics.k8s.io']
  resources: ['pods']
  verbs: ['get', 'watch', 'list']
- apiGroups: ['apps']
  resources: ['deployments', 'replicasets', 'statefulsets', 'daemonsets']
  verbs: ['get', 'watch', 'list']
- apiGroups: ['autoscaling']
```

```
  resources: ['horizontalpodautoscalers']
  verbs: ['get', 'watch', 'list']
- apiGroups: ['networking.k8s.io']
  resources: ['ingresses']
  verbs: ['get', 'watch', 'list']
- apiGroups: ['networking.internal.knative.dev']
  resources: ['serverlessservices']
  verbs: ['get', 'watch', 'list']
- apiGroups: [ 'autoscaling.internal.knative.dev' ]
  resources: [ 'podautoscalers' ]
  verbs: [ 'get', 'watch', 'list' ]
- apiGroups: ['serving.knative.dev']
  resources:
  - configurations
  - revisions
  - routes
  - services
  verbs: ['get', 'watch', 'list']
- apiGroups: ['carto.run']
  resources:
  - clusterconfigtemplates
  - clusterdeliveries
  - clusterdeploymenttemplates
  - clusterimagetemplates
  - clusterruntemplates
  - clustersourcetemplates
  - clustersupplychains
  - clustertemplates
  - deliverables
  - runnables
  - workloads
  verbs: ['get', 'watch', 'list']
- apiGroups: ['source.toolkit.fluxcd.io']
  resources:
  - gitrepositories
  verbs: ['get', 'watch', 'list']
- apiGroups: ['source.apps.tanzu.vmware.com']
  resources:
  - imagerepositories
  - mavenartifacts
  verbs: ['get', 'watch', 'list']
- apiGroups: ['conventions.apps.tanzu.vmware.com']
  resources:
  - podintents
  verbs: ['get', 'watch', 'list']
- apiGroups: ['kpack.io']
  resources:
  - images
  - builds
  verbs: ['get', 'watch', 'list']
- apiGroups: ['scanning.apps.tanzu.vmware.com']
  resources:
  - sourcescans
  - imagescans
  - scanpolicies
  - scantemplates
  verbs: ['get', 'watch', 'list']
- apiGroups: ['app-scanning.apps.tanzu.vmware.com']
  resources:
  - imagevulnerabilityscans
  verbs: ['get', 'watch', 'list']
- apiGroups: ['tekton.dev']
  resources:
  - taskruns
  - pipelineruns
  verbs: ['get', 'watch', 'list']
```

```
- apiGroups: ['kappctrl.k14s.io']
  resources:
  - apps
  verbs: ['get', 'watch', 'list']
- apiGroups: [ 'batch' ]
  resources: [ 'jobs', 'cronjobs' ]
  verbs: [ 'get', 'watch', 'list' ]
- apiGroups: ['conventions.carto.run']
  resources:
  - podintents
  verbs: ['get', 'watch', 'list']
- apiGroups: ['appliveview.apps.tanzu.vmware.com']
  resources:
  - resourceinspectiongrants
  verbs: ['get', 'watch', 'list', 'create']
- apiGroups: ['apiextensions.k8s.io']
  resources: ['customresourcedefinitions']
  verbs: ['get', 'watch', 'list']
- apiGroups: [data.packaging.carvel.dev]
  resources: [packages]
  verbs: ['get', 'watch', 'list']
```

This YAML content creates `Namespace`, `ServiceAccount`, `ClusterRole`, and `ClusterRoleBinding`.

2. On the `Build` and `Run` clusters, create `Namespace`, `ServiceAccount`, `ClusterRole`, and `ClusterRoleBinding` by running:

```
kubectl create -f tap-gui-viewer-service-account-rbac.yaml
```

3. Again, on the `Build` and `Run` clusters, discover the `CLUSTER_URL` and `CLUSTER_TOKEN` values.

v1.23 or earlier Kubernetes cluster

If you're watching a v1.23 or earlier Kubernetes cluster, run:

```
CLUSTER_URL=$(kubectl config view --minify -o jsonpath='{.clusters[0].cluste
r.server}')

CLUSTER_TOKEN=$(kubectl -n tap-gui get secret $(kubectl -n tap-gui get sa tap
-gui-viewer -o=json \
| jq -r '.secrets[0].name') -o=json \
| jq -r '.data["token"]' \
| base64 --decode)

echo CLUSTER_URL: $CLUSTER_URL
echo CLUSTER_TOKEN: $CLUSTER_TOKEN
```

v1.24 or later Kubernetes cluster

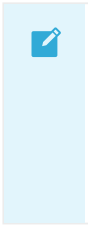
If you're watching a v1.24 or later Kubernetes cluster, run:

```
CLUSTER_URL=$(kubectl config view --minify -o jsonpath='{.clusters[0].cluste
r.server}')

kubectl apply -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
  name: tap-gui-viewer
  namespace: tap-gui
  annotations:
    kubernetes.io/service-account.name: tap-gui-viewer
type: kubernetes.io/service-account-token
EOF
```

```
CLUSTER_TOKEN=$(kubectl -n tap-gui get secret tap-gui-viewer -o=json \
| jq -r '.data["token"]' \
| base64 --decode)

echo CLUSTER_URL: $CLUSTER_URL
echo CLUSTER_TOKEN: $CLUSTER_TOKEN
```



Note

You can create a short-lived token with the `kubectl create token` command if that is the preferred method. This method requires frequent token rotation.

4. (Optional) Configure the Kubernetes client to verify the TLS certificates presented by a cluster's API server. To do this, discover `CLUSTER_CA_CERTIFICATES` by running:

```
CLUSTER_CA_CERTIFICATES=$(kubectl config view --raw -o jsonpath='{.clusters[?
(@.name=="CLUSTER-NAME")].cluster.certificate-authority-data}')

echo CLUSTER_CA_CERTIFICATES: $CLUSTER_CA_CERTIFICATES
```

   Where `CLUSTER-NAME` is your cluster name.

5. Record the `Build` and `Run` clusters' `CLUSTER_URL` and `CLUSTER_TOKEN` values for when you Update Tanzu Developer Portal to view resources on multiple clusters later.

Update Tanzu Developer Portal to view resources on multiple clusters

The clusters must be identified to Tanzu Developer Portal with the `ServiceAccount` token and the cluster Kubernetes control plane URL.

You must add a `kubernetes` section to the `app_config` section in the `tap-values.yaml` file that Tanzu Application Platform used when you installed it. This section must have an entry for each `Build` and `Run` cluster that has resources to view.

To do so:

1. Copy this YAML content into `tap-values.yaml`:

```
tap_gui:
## Previous configuration above
  app_config:
    kubernetes:
      serviceLocatorMethod:
        type: 'multiTenant'
      clusterLocatorMethods:
        - type: 'config'
          clusters:
          ## Cluster 1
            - url: CLUSTER-URL
              name: CLUSTER-NAME
              authProvider: serviceAccount
              serviceAccountToken: "CLUSTER-TOKEN"
              skipTLSVerify: true
              skipMetricsLookup: true
          ## Cluster 2+
            - url: CLUSTER-URL
              name: CLUSTER-NAME
```

```
      authProvider: serviceAccount
      serviceAccountToken: "CLUSTER-TOKEN"
      skipTLSVerify: true
      skipMetricsLookup: true
```

Where:

- `CLUSTER-URL` is the value you discovered earlier.
- `CLUSTER-TOKEN` is the value you discovered earlier.
- `CLUSTER-NAME` is a unique name of your choice.

If there are resources to view on the `View` cluster that hosts Tanzu Developer Portal, add an entry to `clusters` for it as well.

If you would like the Kubernetes client to verify the TLS certificates presented by a cluster's API server, set the following properties for the cluster:

```
skipTLSVerify: false
caData: CLUSTER-CA-CERTIFICATES
```

Where `CLUSTER-CA-CERTIFICATES` is the value you discovered earlier.

2. Update the `tap` package by running this command:

```
tanzu package installed update tap -n tap-install --values-file tap-values.yaml
```

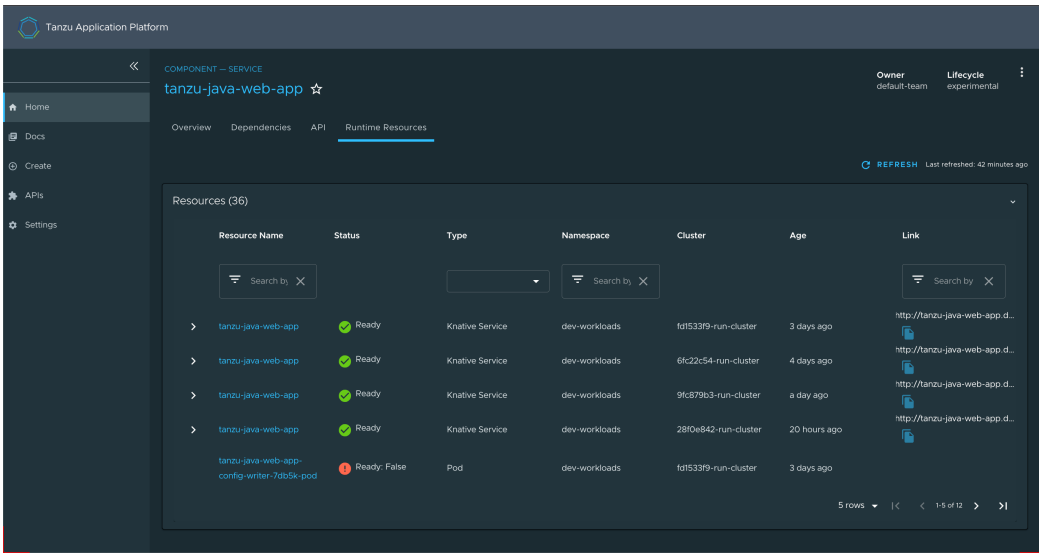
3. Wait a moment for the `tap` and `tap-gui` packages to update and then verify that `STATUS` is `Reconcile succeeded` by running:

```
tanzu package installed get all -n tap-install
```

View resources on multiple clusters in the Runtime Resources Visibility plug-in

To view resources on multiple clusters in the Runtime Resources Visibility plug-in:

1. Go to the Runtime Resources Visibility plug-in for a component that is running on multiple clusters.
2. View the multiple resources and their statuses across the clusters.



Set up authentication for Tanzu Developer Portal

Tanzu Developer Portal extends the current Backstage authentication plug-in so that you can see a login page based on the authentication providers configured at installation. This feature is a work in progress.

Tanzu Developer Portal currently supports the following authentication providers:

- Auth0
- Azure
- Bitbucket
- GitHub
- GitLab
- Google
- Okta
- OneLogin

You can also configure a custom OpenID Connect (OIDC) provider.

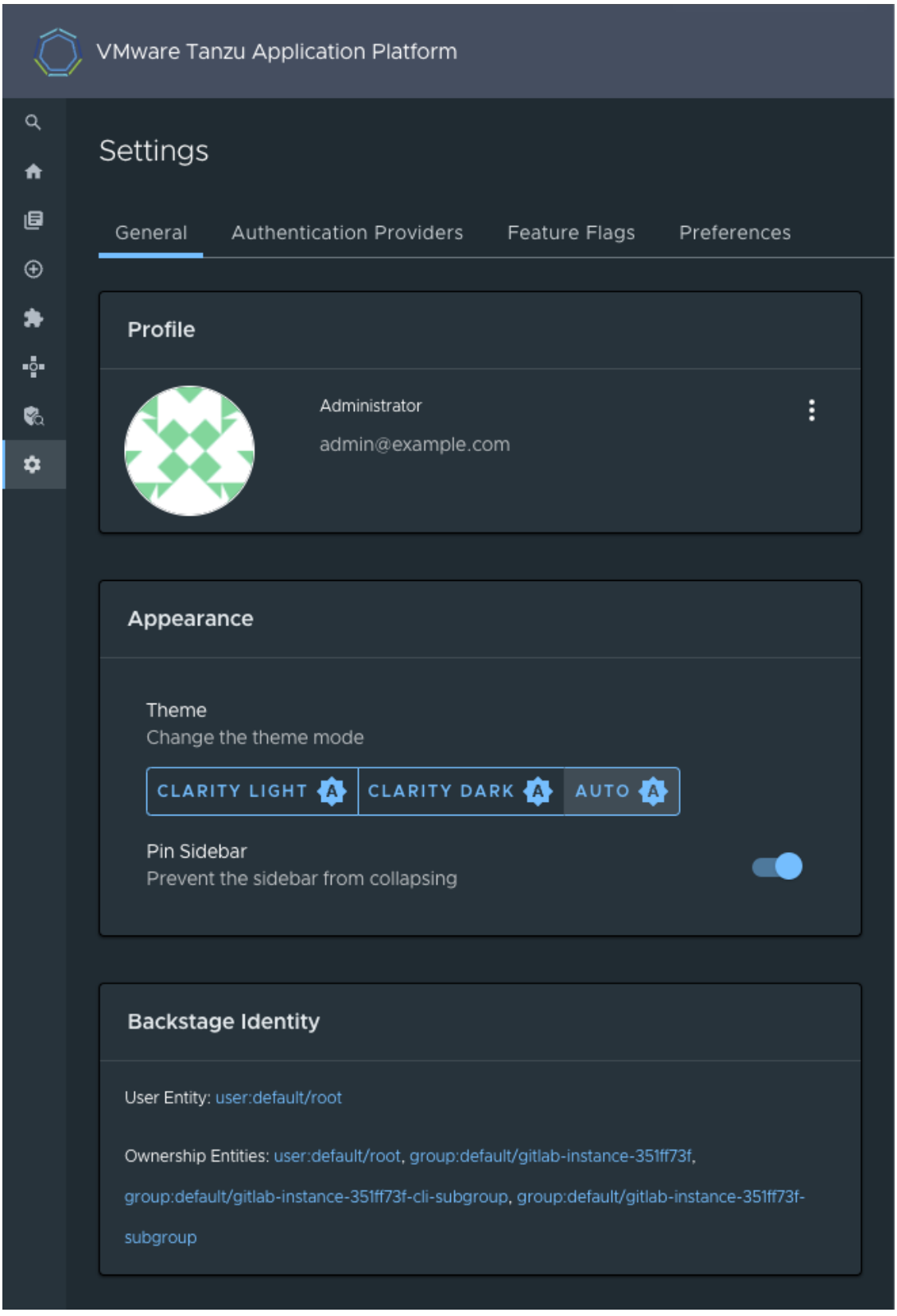
View your Backstage Identity

A Backstage identity is defined as a combination of:

- The user reference: each entity in the catalog is uniquely identified by the triplet of its kind
- A namespace
- A name

For example, the user Jane can be assigned to the user entity `user:default/jane` and an ownership reference, which is used to determine what that user owns. Jane (`user:default/jane`) might have the ownership references `user:default/jane`, `group:default/team-a`, and `group:default/admins`. This would mean that Jane belongs to those groups and, therefore, owns those references.

To view your current Backstage identity, in the **Settings** section of the left side navigation pane click the **General** tab.



Configure an authentication provider

Configure a supported authentication provider or a custom OIDC provider:

- To configure a supported authentication provider, see the Backstage authentication documentation.

- To configure a custom OIDC provider, edit your `tap-values.yaml` file or your custom configuration file to include an OIDC authentication provider. Configure the OIDC provider with your OAuth App values. For example:

```
shared:
  ingress_domain: "INGRESS-DOMAIN"

# ... any existing values

tap_gui:
  # ... any other Tanzu Developer Portal values
  app_config:
    auth:
      environment: development
      session:
        secret: custom session secret
      providers:
        oidc:
          development:
            metadataUrl: AUTH-OIDC-METADATA-URL
            clientId: AUTH-OIDC-CLIENT-ID
            clientSecret: AUTH-OIDC-CLIENT-SECRET
            tokenSignedResponseAlg: AUTH-OIDC-TOKEN-SIGNED-RESPONSE-ALG # defau
lt='RS256'
            scope: AUTH-OIDC-SCOPE # default='openid profile email'
            prompt: auto # default=none (allowed values: auto, none, consent, l
ogin)
```

  Where `AUTH-OIDC-METADATA-URL` is a JSON file with generic OIDC provider configuration. It contains `authorizationUrl` and `tokenUrl`. Tanzu Developer Portal reads these values from `metadataUrl`, so you must not specify these values explicitly in the earlier authentication configuration.

  You must also the provide the redirect URI of the Tanzu Developer Portal instance to your identity provider. The redirect URI is sometimes called the redirect URL, the callback URL, or the callback URI. The redirect URI takes the following form:

```
SCHEME://tap-gui.INGRESS-DOMAIN/api/auth/oidc/handler/frame
```

  Where:

  - `SCHEME` is the URI scheme, most commonly `http` or `https`
  - `INGRESS-DOMAIN` is the host name you selected for your Tanzu Developer Portal instance

  When using `https` and `example.com` as examples for the two placeholders respectively, the redirect URI reads as follows:

```
https://tap-gui.example.com/api/auth/oidc/handler/frame
```

  For more information, see this example in GitHub.

- (Optional) Configure offline access scope for the OIDC provider by adding the `scope` parameter `offline_access` to either `tap-values.yaml` or your custom configuration file. For example:

```
auth:
  providers:
    oidc:
      development:
```

```
        ... # auth configs
        scope: 'openid profile email offline_access'
```

By default, `scope` is not configured to provide persistence to user login sessions, such as in the case of a page refresh. Not all identity providers support the `offline_access` scope. For more information, see your identity provider documentation.

(Optional) Allow guest access

Enable guest access with other providers by adding the following flag under your authentication configuration:

```
auth:
  allowGuestAccess: true
```

(Optional) Customize the login page

Change the card's title or description for a specific provider with the following configuration:

```
auth:
  environment: development
  providers:
    ... # auth providers config
  loginPage:
    github:
      title: Github Login
      message: Enter with your GitHub account
```

For a provider to appear on the login page, ensure that it is properly configured under the `auth.providers` section of your values file.

View resources on remote clusters

You can control the access to Kubernetes runtime resources on Tanzu Developer Portal (commonly called TDP) based on user roles and permissions for each of the visible remote clusters.

Caution

In Tanzu Application Platform v1.6 and earlier, setting up role-based access control (RBAC) might impact your ability to view workloads in the Security Analysis GUI and the Workloads table of the Supply Chain Choreographer plug-in GUI.

RBAC is currently supported for the following Kubernetes cluster providers:

- EKS (Elastic Kubernetes Service) on AWS
- GKE (Google Kubernetes Engine) on Google Cloud Platform (GCP)

Support for other Kubernetes providers is planned for future releases of Tanzu Application Platform.

Tanzu Developer Portal is designed under the assumption that the roles and permissions for the Kubernetes clusters are already defined and that the users are already assigned to their roles. For information about assigning roles and permissions to users, see Assigning roles and permissions on Kubernetes clusters.

Adding access-controlled visibility for a remote cluster is similar to Setting up unrestricted remote cluster visibility.

The steps are:

1. Set up the OIDC provider
2. Configure the Kubernetes cluster with the OIDC provider
3. Configure the Tanzu Developer Portal to view the remote cluster
4. Upgrade the Tanzu Developer Portal package

After following these steps, you can view your runtime resources on a remote cluster in Tanzu Developer Portal. For more information, see View runtime resources on remote clusters.

View resources on remote clusters

You can control the access to Kubernetes runtime resources on Tanzu Developer Portal (commonly called TDP) based on user roles and permissions for each of the visible remote clusters.

Caution

In Tanzu Application Platform v1.6 and earlier, setting up role-based access control (RBAC) might impact your ability to view workloads in the Security Analysis GUI and the Workloads table of the Supply Chain Choreographer plug-in GUI.

RBAC is currently supported for the following Kubernetes cluster providers:

- EKS (Elastic Kubernetes Service) on AWS
- GKE (Google Kubernetes Engine) on Google Cloud Platform (GCP)

Support for other Kubernetes providers is planned for future releases of Tanzu Application Platform.

Tanzu Developer Portal is designed under the assumption that the roles and permissions for the Kubernetes clusters are already defined and that the users are already assigned to their roles. For information about assigning roles and permissions to users, see Assigning roles and permissions on Kubernetes clusters.

Adding access-controlled visibility for a remote cluster is similar to Setting up unrestricted remote cluster visibility.

The steps are:

1. Set up the OIDC provider
2. Configure the Kubernetes cluster with the OIDC provider
3. Configure the Tanzu Developer Portal to view the remote cluster
4. Upgrade the Tanzu Developer Portal package

After following these steps, you can view your runtime resources on a remote cluster in Tanzu Developer Portal. For more information, see View runtime resources on remote clusters.

View resources on remote EKS clusters

This topic tells you how to view your runtime resources on a remote EKS cluster in Tanzu Developer Portal. For more information, see View runtime resources on remote clusters.

Set up the OIDC provider

You must set up the OIDC provider to enable RBAC visibility of remote EKS clusters. You can see the list of supported OIDC providers in Setting up a Tanzu Developer Portal authentication

provider.

Tanzu Developer Portal supports multiple OIDC providers. Auth0 is used here as an example.

1. Log in to the Auth0 dashboard.
2. Go to **Applications**.
3. Create an application of the type `Single Page Web Application` named `TAP-GUI` or a name of your choice.
4. Click the **Settings** tab.
5. Under **Application URIs > Allowed Callback URLs**, add

   ```
   https://tap-gui.INGRESS-DOMAIN/api/auth/auth0/handler/frame
   ```

   Where `INGRESS-DOMAIN` is the domain you chose for your Tanzu Developer Portal in Installing the Tanzu Application Platform package and profiles.
6. Click **Save Changes**.

After creating an application with your OIDC provider, you receive the following credentials for setting up RBAC for your remote cluster:

- **Domain**, which is used as `ISSUER-URL` in the following sections (`AUTH0_DOMAIN` for Auth0)
- **Client ID**, which is used as `CLIENT-ID` in the following sections
- **Client Secret**, which is used as `CLIENT-SECRET` in the following sections

For more information, see Auth0 Setup Walkthrough in the Backstage documentation. To configure other OIDC providers, see Authentication in Backstage in the Backstage documentation.

Configure the Kubernetes cluster with the OIDC provider

To configure the cluster with the OIDC provider's credentials:

1. Create a file with the following content and name it `rbac-setup.yaml`. This content applies to EKS clusters.

   ```
   apiVersion: eksctl.io/v1alpha5
   kind: ClusterConfig
   metadata:
     name: "CLUSTER-NAME"
     region: "AWS-REGION"
   identityProviders:
     - name: auth0
       type: oidc
       issuerUrl: "ISSUER-URL"
       clientId: "CLIENT-ID"
       usernameClaim: email
   ```

   Where:

   - `CLUSTER-NAME` is the cluster name for your EKS cluster as an AWS identifier
   - `AWS-REGION` is the AWS region of the EKS cluster
   - `CLIENT-ID` is the Client ID you obtained while setting up the OIDC provider
   - `ISSUER-URL` is the Issuer URL you obtained while setting up the OIDC provider. For Auth0, this is `https://${AUTH0_DOMAIN}/`.
2. Using `eksctl`, run:

```
eksctl associate identityprovider -f rbac-setup.yaml
```

3. Verify that the association of the OIDC provider with the EKS cluster was successful by running:

   ```
   eksctl get identityprovider --cluster CLUSTER-NAME
   ```

   Where `CLUSTER-NAME` is the cluster name for your EKS cluster as an AWS identifier

   Verify that the output shows `ACTIVE` in the `STATUS` column.

Configure the Tanzu Developer Portal

Configure visibility of the remote cluster in Tanzu Developer Portal:

1. Obtain your cluster's URL by running:

   ```
   CLUSTER_URL=$(kubectl config view --minify -o jsonpath='{.clusters[0].cluster.s
   erver}')

   echo CLUSTER-URL: $CLUSTER_URL
   ```

   This command returns the URL of the first configured cluster in your `kubeconfig` file. To view other clusters one by one, edit the number in `.clusters[0].cluster.server` or edit the command to view all the configured clusters.

2. Ensure you have an `auth` section in the `app_config` section that Tanzu Developer Portal uses. In the example for Auth0, copy this YAML content into `tap-values.yaml`:

   ```
   auth:
     environment: development
     providers:
       auth0:
         development:
           clientId: "CLIENT-ID"
           clientSecret: "CLIENT-SECRET"
           domain: "ISSUER-URL"
   ```

   Where:

   - `CLIENT-ID` is the Client ID you obtained while setting up the OIDC provider.
   - `CLIENT-SECRET` is the Client Secret you obtained while setting up the OIDC provider.
   - `ISSUER-URL` is the Issuer URL you obtained while setting up the OIDC provider. For Auth0, it is only `AUTH0_DOMAIN`.

3. Add a `kubernetes` section to the `app_config` section that Tanzu Developer Portal uses. This section must have an entry for each cluster that has resources to view. To do so, copy this YAML content into `tap-values.yaml`:

   ```
   kubernetes:
     serviceLocatorMethod:
       type: 'multiTenant'
     clusterLocatorMethods:
       - type: 'config'
         clusters:
           - name: "CLUSTER-NAME-UNCONSTRAINED"
             url: "CLUSTER-URL"
             authProvider: oidc
             oidcTokenProvider: auth0
   ```

```
          skipTLSVerify: true
          skipMetricsLookup: true
```

Where:

- `CLUSTER-NAME-UNCONSTRAINED` is the cluster name of your choice for your EKS cluster
- `CLUSTER-URL` is the URL for the remote cluster you are connecting to Tanzu Developer Portal. You obtained this earlier in the procedure.

If there are any other clusters that you want to make visible in Tanzu Developer Portal, add their entries to `clusters` as well.

Upgrade the Tanzu Developer Portal package

After the new configuration file is ready, update the `tap` package:

1. Run:

   ```
   tanzu package installed update tap --values-file tap-values.yaml
   ```

2. Wait a moment for the `tap-gui` package to update and then verify that `STATUS` is `Reconcile succeeded` by running:

   ```
   tanzu package installed get tap-gui -n tap-install
   ```

View resources on remote GKE clusters

This topic tells you about two supported options to add access-controlled visibility for a remote GKE cluster:

- Leverage an external OIDC provider
- Leveraging Google's OIDC provider

After the authorization is enabled, you can view your runtime resources on a remote cluster in Tanzu Developer Portal. For more information, see View runtime resources on remote clusters.

Leverage an external OIDC provider

To leverage an external OIDC provider, such as Auth0:

1. Set up the OIDC provider
2. Configure the GKE cluster with the OIDC provider
3. Configure the Tanzu Developer Portal to view the remote GKE cluster
4. Update the `tap-gui` package

Set up the OIDC provider

You must set up the OIDC provider to enable RBAC visibility of remote clusters. You can see the list of supported OIDC providers in Setting up a Tanzu Developer Portal authentication provider.

Tanzu Developer Portal supports multiple OIDC providers. Auth0 is used here as an example.

1. Log in to the Auth0 dashboard.
2. Go to **Applications**.
3. Create an application of the type `Single Page Web Application` named `TAP-GUI` or a name of your choice.

4. Click the **Settings** tab.
5. Under **Application URIs** > **Allowed Callback URLs**, add

   ```
   https://tap-gui.INGRESS-DOMAIN/api/auth/auth0/handler/frame
   ```

   Where `INGRESS-DOMAIN` is the domain you chose for your Tanzu Developer Portal in Installing the Tanzu Application Platform package and profiles.
6. Click **Save Changes**.

After creating an application with your OIDC provider, you receive the following credentials for setting up RBAC for your remote cluster:

- **Domain**, which is used as `issuerURL` in the following sections
- **Client ID**, which is used as `CLIENT-ID` in the following sections
- **Client Secret**, which is used as `CLIENT-SECRET` in the following sections

For more information, see Auth0 Setup Walkthrough in the Backstage documentation. To configure other OIDC providers, see Authentication in Backstage in the Backstage documentation.

Configure the GKE cluster with the OIDC provider

Add redirect configuration on the OIDC side by following the Google Cloud documentation.

Configure visibility of the remote cluster

Configure visibility of the remote cluster in Tanzu Developer Portal:

1. Obtain your cluster's URL by running:

   ```
   CLUSTER_URL=$(kubectl config view --minify -o jsonpath='{.clusters[0].cluster.s
   erver}')

   echo CLUSTER-URL: $CLUSTER_URL
   ```

   This command returns the URL of the first configured cluster in your `kubeconfig` file. To view other clusters one by one, edit the number in `.clusters[0].cluster.server` or edit the command to view all the configured clusters.
2. Ensure you have an `auth` section in the `app_config` section that Tanzu Developer Portal uses. In the example for Auth0, copy this YAML content into `tap-values.yaml`:

   ```
   auth:
     environment: development
     providers:
       auth0:
         development:
           clientId: "CLIENT-ID"
           clientSecret: "CLIENT-SECRET"
           domain: "ISSUER-URL"
   ```

   Where:

   - `CLIENT-ID` is the Client ID you obtained while setting up the OIDC provider
   - `CLIENT-SECRET` is the Client Secret you obtained while setting up the OIDC provider
   - `ISSUER-URL` is the Issuer URL you obtained while setting up the OIDC provider
3. Add a `kubernetes` section to the `app_config` section that Tanzu Developer Portal uses. This section must have an entry for each cluster that has resources to view. To do so, copy this YAML content into `tap-values.yaml`:

```
kubernetes:
  serviceLocatorMethod:
    type: 'multiTenant'
  clusterLocatorMethods:
    - type: 'config'
      clusters:
        - name: "CLUSTER-NAME-UNCONSTRAINED"
          url: "CLUSTER-URL"
          authProvider: oidc
          oidcTokenProvider: auth0
          skipTLSVerify: true
          skipMetricsLookup: true
```

Where:

- `CLUSTER-NAME-UNCONSTRAINED` is the cluster name of your choice for your GKE cluster
- `CLUSTER-URL` is the URL for the remote cluster you are connecting to Tanzu Developer Portal. You obtained this earlier in the procedure.

If there are any other clusters that you want to make visible in Tanzu Developer Portal, add their entries to `clusters` as well.

Update the `tap-gui` package to finish leveraging the external OIDC provider

After the new configuration file is ready, update the `tap-gui` package:

1. Run:

   ```
   tanzu package installed update tap --values-file tap-values.yaml
   ```

2. Wait a moment for the `tap-gui` package to update and then verify that `STATUS` is `Reconcile succeeded` by running:

   ```
   tanzu package installed get tap-gui -n tap-install
   ```

Leverage Google's OIDC provider

When leveraging Google's OIDC provider, fewer steps are needed to enable authorization:

1. Add redirect configuration on the OIDC side.
2. Configure the Tanzu Developer Portal to view the remote GKE cluster
3. Upgrade the Tanzu Developer Portal package

Add redirect configuration on the OIDC side

Add redirect configuration on the OIDC side by following the Google Cloud documentation.

Configure visibility of the remote GKE cluster

Configure visibility of the remote GKE cluster in Tanzu Developer Portal:

1. Obtain your cluster's URL by running:

   ```
   CLUSTER_URL=$(kubectl config view --minify -o jsonpath='{.clusters[0].cluster.s
   erver}')

   echo CLUSTER-URL: $CLUSTER_URL
   ```

This command returns the URL of the first configured cluster in your `kubeconfig` file. To view other clusters one by one, edit the number in `.clusters[0].cluster.server` or edit the command to view all the configured clusters.

2. Ensure you have an `auth` section in the `app_config` section that Tanzu Developer Portal uses. In the example for Auth0, copy this YAML content into `tap-values.yaml`:

```
auth:
  environment: development
  providers:
    google:
      development:
        clientId: "CLIENT-ID"
        clientSecret: "CLIENT-SECRET"
```

Where:

- `CLIENT-ID` is the Client ID you obtained while setting up the OIDC provider
- `CLIENT-SECRET` is the Client Secret you obtained while setting up the OIDC provider

3. Add a `kubernetes` section to the `app_config` section that Tanzu Developer Portal uses. This section must have an entry for each cluster that has resources to view. To do so, copy this YAML content into `tap-values.yaml`:

```
kubernetes:
  clusterLocatorMethods:
    - type: 'config'
      clusters:
        - name: "CLUSTER-NAME-UNCONSTRAINED"
          url: "CLUSTER-URL"
          authProvider: google
          caData: "CA-DATA"
```

Where:

- `CLUSTER-NAME-UNCONSTRAINED` is the cluster name of your choice for your GKE cluster.
- `CLUSTER-URL` is the URL for the remote cluster you are connecting to Tanzu Developer Portal. You obtained this earlier in the procedure.
- `CA-DATA` is the CA certificate data.

If there are any other clusters that you want to make visible in Tanzu Developer Portal, add their entries to `clusters` as well.

Update the `tap-gui` package to finish leveraging the Google OIDC provider

After the new configuration file is ready, update the `tap-gui` package:

1. Run:

```
tanzu package installed update tap --values-file tap-values.yaml
```

2. Wait a moment for the `tap-gui` package to update and then verify that `STATUS` is `Reconcile succeeded` by running:

```
tanzu package installed get tap-gui -n tap-install
```

Enable role-based access control for the Secure Supply Chains UI and Security Analysis UI plug-ins

This topic tells you how to make workloads visible on the Secure Supply Chains UI and the Security Analysis UI when using a cluster with role-based access control (RBAC) and namespace-scoped access.

Add permissions to list namespaces

To be able to get the information from the scoped namespaces, users must have permission to list namespaces. To grant this permission, create a new `ClusterRole` and `ClusterRoleBinding`:

1. Add the following to `namespace-cluster-role.yaml`:

```
# namespace-cluster-role.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: namespaces-role
rules:
- apiGroups: ['']
  resources: ['namespaces']
  verbs: ['get', 'list']
```

2. Add the following to `namespace-cluster-role-binding.yaml`:

```
# namespace-cluster-role-binding.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: namespaces-rolebinding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: namespaces-role
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: YOUR-USER-ID
```

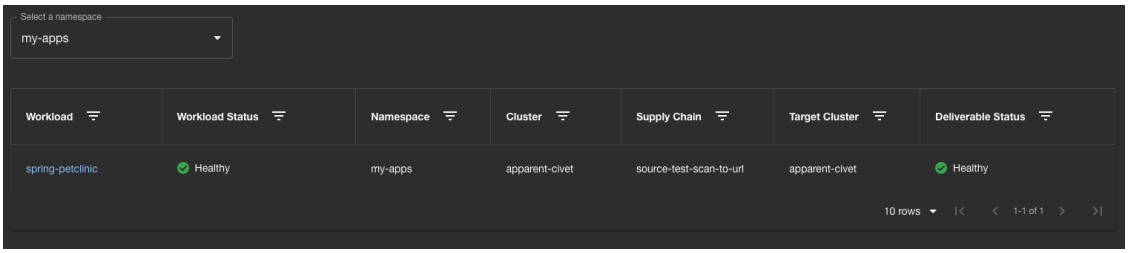
Add a label to the scoped namespaces

The current implementation requires the scoped namespaces to have a specific label or annotation. If you are using Namespace Provisioner, one of the required labels is present. However, to be safe, add the supported annotation to the scoped namespaces by running:

```
kubectl annotate namespaces NAMESPACE apps.tanzu.vmware.com/tap-managed-ns=""
```

Where `NAMESPACE` is your namespace

With this annotation the UI can target the scoped namespaces and show you workloads on such namespaces.



Set up the Backstage RBAC plug-in for Tanzu Developer Portal

This topic gives you an overview of the Backstage role-based access control (RBAC) plug-in and demonstrates how to enable it in Tanzu Developer Portal. For more information about the plug-in, see the:

- Backstage official website
- Front-end plug-in documentation
- Back-end plug-in documentation

Overview of the RBAC plug-in

The Backstage RBAC plug-in works with the permission framework to enable support for RBAC for Tanzu Application Platform operators. For more information, see Set up the permission framework for your Tanzu Developer Portal.

Install the RBAC plug-in

Creating a customized Tanzu Developer Portal with the RBAC plug-in is similar to building a customized Tanzu Developer Portal with Configurator.

To create a customized Tanzu Developer Portal with the RBAC plug-in:

1. Create a `tdp-config.yaml` file with the following structure:

```
app:
  plugins:
    - name: "@vmware-tanzu/tdp-plugin-rbac"
      version: "2.0.0"

backend:
  plugins:
    - name: "@vmware-tanzu/tdp-plugin-rbac-backend"
      version: "2.0.0"
    - name: "@vmware-tanzu/tdp-plugin-permission-backend"
      version: "2.0.0"
```

> **Important**
>
> If you want to install additional plug-ins, `@vmware-tanzu/tdp-plugin-rbac-backend` must still come before `@vmware-tanzu/tdp-plugin-permission-backend` in the `backend.plugins` section in `tdp-config.yaml`.

2. Encode the `tdp-config.yaml` file in Base64 by running:

```
base64 -i tdp-config.yaml
```

3. In your `tdp-workload.yaml` file, replace `ENCODED-TDP-CONFIG-VALUE` with the result of the Base64-encoded value from the prior step. Here is an example of what the `tdp-workload.yaml` file can look like:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: tdp-configurator
  labels:
    apps.tanzu.vmware.com/workload-type: web
    app.kubernetes.io/part-of: tdp-configurator
spec:
  build:
    env:
      - name: BP_NODE_RUN_SCRIPTS
        value: "set-tdp-config,portal:pack"
      - name: TPB_CONFIG
        value: /tmp/tdp-config.yaml
      - name: TPB_CONFIG_STRING
        value: YXBwOgogIHBsdWdpbnM6CiAgICAtIG5hbWU6ICJAdm13YXJlLXRhbnp1L3RkcC1w
bHVnaW4tcmJhYyIKICAgICAgdmVyc2lvbjogIjIuMC4wIgoKYmFja2VuZDoKICBwbHVnaW5zOgogICA
gLSBuYW1lOiAiQHZtd2FyZS10YW56dS90ZHAtcGx1Z2luLXJiYWMtYmFja2VuZCIKICAgICAgdmVyc2
lvbjogIjIuMC4wIgogICAgLSBuYW1lOiAiQHZtd2FyZS10YW56dS90ZHAtcGx1Z2luLXBlcm1pc3Npb
24tYmFja2VuZCIKICAgICAgdmVyc2lvbjogIjIuMC4wIgo=
  source:
    image: TDP-IMAGE-LOCATION
    subPath: builder
```

   Where `TDP-IMAGE-LOCATION` is the location of the Configurator image.

4. Apply the `tdp-workload.yaml` file as a workload by running:

```
tanzu apps workload apply -f tdp-workload.yaml -n DEVELOPER-NAMESPACE
```

   Where `DEVELOPER-NAMESPACE` is the Kubernetes namespace you created to run your workloads.

   An example command to create such a namespace is:

```
$ kubectl create ns my-apps
```

5. Wait for the workload to go through the `image-provider` stage in the supply chain.
6. Follow the steps in running a customized Tanzu Developer Portal to retrieve the location of the customized Tanzu Developer Portal image that the workload produces and overlay the customized image on to the instance of Tanzu Developer Portal currently running.

Enable the RBAC plug-in

To enable the RBAC plug-in:

1. Ensure that the permission framework is enabled in the `tap_gui.app_config` section of `tap-values.yaml`:

```
permission:
  enabled: true
```

2. Add the plug-ins that use permissions and entity references for users and groups that you want to permit to configure RBAC. Also add the license key to use the RBAC plug-in:

```
permission:
  enabled: true
```

```
  permissionedPlugins:
    - PLUG-IN-NAME
  rbac:
    authorizedUsers:
      - group:NAMESPACE/NAME
      - user:NAMESPACE/NAME
spotify:
  licenseKey: SPOTIFY-LICENSE-KEY
```

Where:

- `PLUG-IN-NAME` is a plug-in that includes permissions. Currently, the Catalog plug-in is the only first-party Backstage plug-in that includes permissions.
- `NAMESPACE` is usually `default` unless defined otherwise in the definition file.
- `NAME` is the name of the group or user.
- `SPOTIFY-LICENSE-KEY` is the license key that you received from Spotify.

Example:

```
tap_gui:
  app_config:
    ... # other Tanzu Developer Portal app configuration
    permission:
      enabled: true
      permissionedPlugins:
        - PLUG-IN-NAME
      rbac:
        authorizedUsers:
          - group:NAMESPACE/NAME
          - user:NAMESPACE/NAME
    spotify:
      licenseKey: SPOTIFY-LICENSE-KEY
```

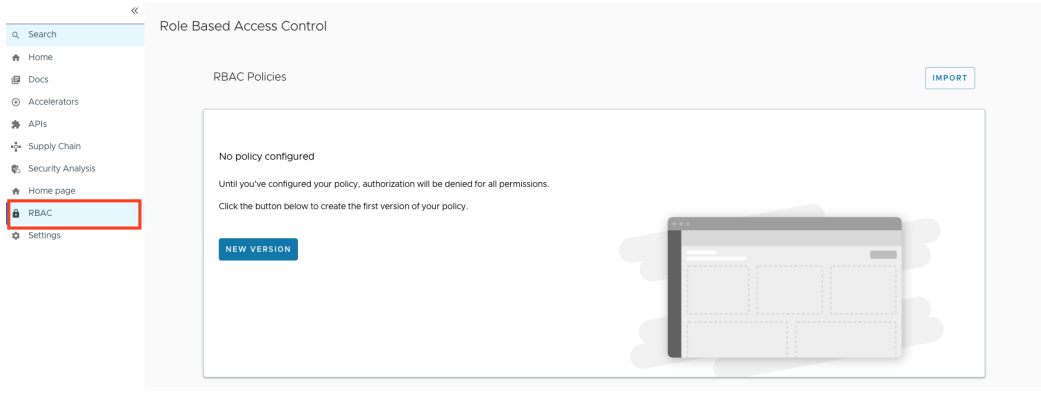
3. With the overlay from running a customized Tanzu Developer Portal, ensure that the `tap-values.yaml` file has the `tap_gui` section and `package_overlays` section.

   Example:

```
tap_gui:
  app_config:
    ... # other Tanzu Developer Portal app configuration
    permission:
      enabled: true
      permissionedPlugins:
        - PLUG-IN-NAME
      rbac:
        authorizedUsers:
          - group:NAMESPACE/NAME
          - user:NAMESPACE/NAME
    spotify:
      licenseKey: SPOTIFY-LICENSE-KEY

package_overlays:
- name: tap-gui
  secrets:
  - name: tdp-app-image-overlay-secret
```

4. Reinstall the Tanzu Developer Portal package by following the steps in Upgrade Tanzu Application Platform.
5. Open Tanzu Developer Portal in your browser and verify that the **RBAC** tab is in the sidebar.



(Optional) Identify a user authorized to author RBAC policies

You can identify someone who used an authentication provider to log in as a user authorized to author RBAC policies.

To do so, edit your `tap_gui` section in `tdp-values.yaml` to look similar to this example that uses Google authentication:

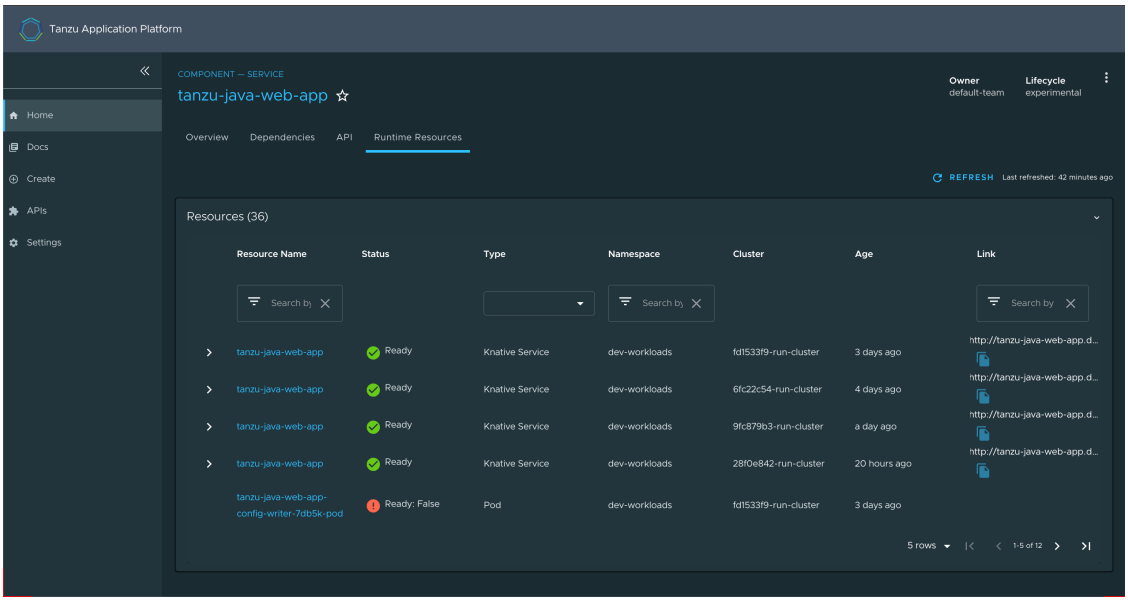
```
tap_gui:
  app_config:
    auth:
      environment: development
      providers:
        google: # https://backstage.io/docs/auth/google/provider/
          development:
            clientId: GOOGLE-CLIENT-ID
            clientSecret: GOOGLE-CLIENT-SECRET
    permission:
      enabled: true
      permissionedPlugins:
      - catalog
      rbac:
        authorizedUsers:
        - group:default/admins
        - user:default/USER-GOOGLE-EMAIL
    spotify:
      licenseKey: SPOTIFY-LICENSE-KEY
```

Where:

- `GOOGLE-CLIENT-ID` and `GOOGLE-CLIENT-SECRET` are credentials provided after setting up Google authentication by following steps in the Backstage documentation
- `USER-GOOGLE-EMAIL` is the Google account email address used to log in to Tanzu Developer Portal
- `SPOTIFY-LICENSE-KEY` is the license key provided by Spotify

View runtime resources on authorization-enabled clusters

To visualize runtime resources on authorization-enabled clusters in Tanzu Developer Portal, proceed to the software catalog component of choice and click the **Runtime Resources** tab on top of the ribbon.



After you click **Runtime Resources**, Tanzu Developer Portal uses your credentials to query the clusters for the respective runtime resources. The system verifies that you are authenticated with the OIDC providers configured for the remote clusters. If you are not authenticated, the system prompts you for your OIDC credentials.

Remote clusters that are not restricted by authorization are visible by using the general Service Account of Tanzu Developer Portal. It is not restricted for users. For more information about how to set up unrestricted remote cluster visibility, see Viewing resources on multiple clusters in Tanzu Developer Portal.

The type of query to the remote cluster depends on the definition of the software catalog component. In Tanzu Developer Portal, there are globally-scoped components and namespace-scoped components.

This property of the component affects runtime resource visibility, depending on your permissions on a specific cluster.

If your permissions on the authorization-enabled cluster are limited to specific namespaces, you do not have visibility into runtime resources of globally-scoped components.

You need cluster-scoped access to have visibility into runtime resources of globally-scoped components.

Globally-scoped components

For globally-scoped components, when you access **Runtime Resources** Tanzu Developer Portal queries all Kubernetes namespaces for runtime resources that have a matching `kubernetes-label-selector`, usually with a `part-Of` prefix.

For example, `demo-component-a` does not have a `backstage.io/kubernetes-namespace` in the `metadata.annotations` section. This makes it a globally-scoped component. See the following example YAML.

```
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: demo-component-a
  description: Demo Component A
  tags:
    - java
  annotations:
```

```
    'backstage.io/kubernetes-label-selector': 'app.kubernetes.io/part-of=component-a'
spec:
  type: service
  lifecycle: experimental
  owner: team-a
```

Namespace-scoped components

If a component is namespace-scoped, when you access **Runtime Resources** Tanzu Developer Portal queries only the associated Kubernetes namespace for each remote cluster that is visible to Tanzu Developer Portal.

To make a component namespace-scoped, pass the following annotation to the definition YAML file of the component:

```
annotations:
  'backstage.io/kubernetes-namespace': NAMESPACE-NAME
```

Where `NAMESPACE-NAME` is the Kubernetes namespace you want to associate your component with.

For example, `demo-component-b` has a `kubernetes-namespace` in the `metadata.annotations` section, which associates it with the `component-b` namespaces on each of the visible clusters. This makes it a namespace-scoped component. See the following example YAML.

```
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: demo-component-b
  description: Demo Component B
  tags:
    - java
  annotations:
    'backstage.io/kubernetes-label-selector': 'app.kubernetes.io/part-of=component-b'
    'backstage.io/kubernetes-namespace': component-b
spec:
  type: service
  lifecycle: experimental
  owner: team-b
```

When the `kubernetes-namespace` annotation is absent, the component is considered globally-scoped by default. For more information, see Adding Namespace Annotation in the Backstage documentation.

Assign roles and permissions on Kubernetes clusters

This topic gives you an overview of creating roles and permissions on Kubernetes clusters and assigning these roles to users. For more information, see Using RBAC Authorization in the Kubernetes documentation.

The steps to define and assign roles are:

1. Create roles
2. Create users
3. Assign users to their roles

Create roles

To control the access to Kubernetes runtime resources on Tanzu Developer Portal based on users' roles and permissions for each of visible remote clusters, VMware recommends two role types:

- Cluster-scoped roles
- Namespace-scoped roles

Cluster-scoped roles

Cluster-scoped roles provide cluster-wide privileges. They enable visibility into runtime resources across all of a cluster's namespaces.

In this example YAML snippet, the `pod-viewer` role enables pod visibility on the cluster:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: pod-viewer
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

Namespace-scoped roles

Namespace-scoped roles provide privileges that are limited to a certain namespace. They enable visibility into runtime resources inside namespaces.

In this example YAML snippet, the `pod-viewer-app1` role enables pod visibility in the `app1` namespace:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: app1
  name: pod-viewer-app1
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list"]
```

Create users

You can create users by running the `kubectl create` command. In this example YAML snippet, the user `john` is defined:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: User
metadata:
  namespace: default
  name: john
```

Assign users to their roles

After the users and role are created, the next step is to bind them together.

To bind a Tanzu Application Platform default role, see Bind a user or group to a default role.

In this example YAML snippet, the user `john` is bound with the `pod-viewer` cluster role:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: john-pod-viewer
  namespace: default
subjects:
- kind: User
  name: john
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: pod-viewer
  apiGroup: rbac.authorization.k8s.io
```

In this example YAML snippet, the user `john` is bound with the `pod-viewer-app1` namespace-specific role:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: john-pod-viewer-app1
  namespace: app1
subjects:
- kind: User
  name: john
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-viewer-app1
  apiGroup: rbac.authorization.k8s.io
```

To verify the user's permissions, run the `can-i` commands to get a `yes` or `no` answer. To verify that you can list pods in your cluster-wide role, run:

```
kubectl auth can-i get pods --all-namespaces
```

To verify that you can list pods in namespace `app1` in your namespace-specific role, run:

```
kubectl auth can-i get pods --namespace app1
```

Set up the permission framework for your Tanzu Developer Portal

This topic gives you an overview of the Backstage permission framework and tells you how to enable it for Tanzu Developer Portal. For more information, see the Backstage documentation.

Caution

The permission framework functions are in alpha. It is not recommended for use in production environment.

Overview of the permission framework

The permission framework enables Tanzu Application Platform operators to enforce policies that limit visibility of certain parts of Tanzu Developer Portal.

The current release features the alpha version of this function and only applies to the Software Catalog entities. `owner-of` is the only policy available and it is embedded in the GUI.

`owner-of` enables the operator to define which admin groups and users have unrestricted visibility of the Software Catalog entities. If a user is not listed as an admin or part of the admin group, that user can see only the Software Catalog entities that are owned by their teams.

Important

The permission framework is deactivated by default. The built-in `owner-of` policy is primarily applied to the component entities of the software catalog. Accelerators (templates) in Application Accelerator become invisible if the permission framework is enabled.

Enable the permission framework

To enable the permission framework and limit Software Catalog entities' visibility to owners, allowlist users, allowlist groups, and catalog admins, add the following parameters to the `tap_gui.app_config` section:

```
permission:
  enabled: true
  adminRefs:
    - user:NAMESPACE/NAME
    - group:NAMESPACE/NAME
```

Where:

- `NAMESPACE` is usually `default` unless defined otherwise in the definition file
- `NAME` is the name of the group or user

The `adminRefs` section lists all users and groups to grant access to. Each user and group is defined by the `NAMESPACE` and the `NAME`.

You can view your user entity in the `Settings`/`General` section in your Tanzu Application Platform GUI, or in the YAML definition file of the user or group.

For example:

```
- user:default/admin
- group:test-namespace/operators
```

After you have updated your configuration file, reinstall your Tanzu Developer Portal package by following the steps in Upgrade Tanzu Application Platform.

After the updated values configuration file is applied in Tanzu Developer Portal, the permission framework is enabled.

Important

After the permission framework is enabled, all Software Catalog entities without proper annotations are invisible to all users except admins.

Enable catalog entity visibility

With the permission framework enabled, annotate entities in your Software Catalog to make the relevant users or groups, who are not the owner of the entity, visible. To do this, add annotations to the `metadata.annotations` section of the entity's definition file:

```
metadata:
  annotations:
    backstage.tanzu.vmware.com/GROUP-OR-USER.NAMESPACE.NAME: 'COMMA-SEPARATED-PERMISSI
ONS'
```

Where:

- `GROUP-OR-USER` is either a `group` to describe a group or a `user` to describe an individual user
- `NAMESPACE` is the namespace that the group or user is attributed to. If this value is not specified, it is `default`.
- `NAME` is the name of the user or group
- `COMMA-SEPARATED-PERMISSIONS` is the list of comma-delimited permissions for the specified user or group. Valid catalog entity permissions include:
  - `catalog.entity.read`
  - `catalog.entity.update`
  - `catalog.entity.delete`

Note

`catalog.entity.create` is another permission that is valid for the permission framework, but it is not meaningful for the Software Catalog entities that have already been created.

You can add several groups or users by adding new annotations with their respective comma-delimited permissions, for example:

```
metadata:
  annotations:
    backstage.tanzu.vmware.com/user.default.user-a: 'catalog.entities.read'
    backstage.tanzu.vmware.com/user.default.user-b: 'catalog.entities.read,catalog.ent
ities.update'
```

For example, this annotation enables users that are part of `group-a` to read and delete `component-a`:

```
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: component-a
  annotations:
    backstage.tanzu.vmware.com/group.test-namespace.group-a: 'catalog.entity.read, cat
alog.entity.delete'
```

If a group is granted ownership or access to an entity, all users within that group automatically inherit the same access. But it does not work for nested group ownerships, when a group is listed as a member of another group.

Caution

When the permission plug-in is turned on with the default permission policy, API Auto Registration no longer works due to the lack of user identities. Requests against the Catalog API will no longer work as before. For more information, see the known issues in the release notes.

Add Tanzu Developer Portal integrations

You can integrate Tanzu Developer Portal with several Git providers. To use an integration, you must enable it and provide the necessary token or credentials in `tap-values.yaml`.

Add a GitHub provider integration

To add a GitHub provider integration, edit `tap-values.yaml` as in this example:

```
app_config:
  app:
    baseUrl: http://EXTERNAL-IP:7000
  # Existing tap-values.yaml above
  integrations:
    github: # Other integrations available see NOTE below
      - host: github.com
        token: GITHUB-TOKEN
```

Where:

- `EXTERNAL-IP` is the external IP address.
- `GITHUB-TOKEN` is a valid token generated from your Git infrastructure of choice. Ensure that `GITHUB-TOKEN` has the necessary read permissions for the catalog definition files you extracted from the blank software catalog introduced in the Tanzu Developer Portal prerequisites.

Add a Git-based provider integration that isn't GitHub

To enable Tanzu Developer Portal to read Git-based non-GitHub repositories containing component information:

1. Add the following YAML to `tap-values.yaml`:

   ```
   app_config:
     # Existing tap-values.yaml above
     backend:
       reading:
         allow:
           - host: "GIT-CATALOG-URL-1"
           - host: "GIT-CATALOG-URL-2" # Including more than one URL is optional
   ```

   Where `GIT-CATALOG-URL-1` and `GIT-CATALOG-URL-2` are URLs in a list of URLs that Tanzu Developer Portal can read when registering new components. For example, `git.example.com`. For more information about registering new components, see Adding catalog entities.

2. Adding the YAML from the previous step currently causes the **Accelerators** page to break and not show any accelerators. Provide a value for Application Accelerator as a workaround, as in this example:

   ```
   app_config:
     # Existing tap-values.yaml above
     backend:
       reading:
         allow:
           - host: acc-server.accelerator-system.svc.cluster.local
   ```

Add a non-Git provider integration

To add an integration for a provider that isn't associated with GitHub, see the Backstage documentation.

Update the package profile

After changing `tap-values.yaml`, update the package profile by running:

```
tanzu package installed update  tap --package tap.tanzu.vmware.com --version VERSION-N
UMBER \
--values-file tap-values.yaml -n tap-install
```

Where `VERSION-NUMBER` is the Tanzu Application Platform version. For example, `1.9.1`.

For example:

```
$ tanzu package installed update  tap --package tap.tanzu.vmware.com --version \
1.9.1 --values-file tap-values.yaml -n tap-install
| Updating package 'tap'
| Getting package install for 'tap'
| Getting package metadata for 'tap.tanzu.vmware.com'
| Updating secret 'tap-tap-install-values'
| Updating package install for 'tap'
/ Waiting for 'PackageInstall' reconciliation for 'tap'


Updated package install 'tap' in namespace 'tap-install'
```

Configure the Tanzu Developer Portal database

The Tanzu Developer Portal catalog gives you two approaches for storing catalog information:

- **In-memory database:**

  The default option uses an in-memory database and is suitable for test and development scenarios only. The in-memory database reads the catalog data from Git URLs that you write in `tap-values.yaml`.

  This data is temporary. Any operations that cause the `server` pod in the `tap-gui` namespace to be re-created also cause this data to be rebuilt from the Git location.

  This can cause issues when you manually register entities by using the UI because they only exist in the database and are lost when that in-memory database is rebuilt. If you choose this method, you lose all user preferences and any manually registered entities when the Tanzu Developer Portal server pod is re-created.

- **PostgreSQL database:**

  For production use-cases, use a PostgreSQL database that exists outside the Tanzu Application Platform packaging. The PostgreSQL database stores all the catalog data persistently both from the Git locations and the UI manual entity registrations.

For production or general-purpose use-cases, a PostgreSQL database is recommended.

Configure a PostgreSQL database

See the following sections for configuring Tanzu Developer Portal to use a PostgreSQL database.

Edit `tap-values.yaml`

Apply the following values in `tap-values.yaml`:

```
# ... existing tap-values.yaml above
tap_gui:
  # ... existing tap_gui values
  app_config:
    backend:
      database:
        client: pg
        connection:
          host: PG-SQL-HOSTNAME
          port: 5432
          user: PG-SQL-USERNAME
          password: PG-SQL-PASSWORD
          ssl: {rejectUnauthorized: false} # Set to true if using SSL
```

Where:

- `PG-SQL-HOSTNAME` is the host name of your PostgreSQL database
- `PG-SQL-USERNAME` is the user name of your PostgreSQL database
- `PG-SQL-PASSWORD` is the password of your PostgreSQL database

(Optional) Configure extra parameters

Beyond the minimum configuration options needed to make Tanzu Developer Portal work with the `pg` driver, there are many more configuration options for other purposes. For example, you can restrict Tanzu Developer Portal to a single database. For more information about this restriction, see the Backstage documentation.

By default, Tanzu Developer Portal creates a database for each plug-in, but you can configure it to divide plug-ins based on different PostgreSQL schemas and use a single specified database.

See the following example of extra configuration parameters:

```
# ... existing tap-values.yaml above
tap_gui:
  # ... existing tap_gui values
  app_config:
    backend:
      # ... other backend details
      database:
        client: pg

        # This parameter tells Tanzu Developer Portal to put plug-ins in their own sch
ema instead
        # of their own database.
        # default: database
        pluginDivisionMode: schema

        connection:
          # ... other connection details
          database: PG-SQL-DATABASE
```

Where `PG-SQL-DATABASE` is the database name for Tanzu Developer Portal to use

For the complete list of these configuration options, see the node-postgres documentation.

Update the package profile

You can apply your new configuration by updating Tanzu Application Platform with your modified values. Doing so updates Tanzu Developer Portal because it belongs to Tanzu Application Platform.

To apply your new configuration, run:

```
tanzu package installed update  tap --package tap.tanzu.vmware.com --version VERSION-N
UMBER --values-file tap-values.yaml -n tap-install
```

Where `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.9.1`.

For example:

```
$ tanzu package installed update  tap --package tap.tanzu.vmware.com --version {{ var
s.tap_version }} --values-file tap-values.yaml -n tap-install
| Updating package 'tap'
| Getting package install for 'tap'
| Getting package metadata for 'tap.tanzu.vmware.com'
| Updating secret 'tap-tap-install-values'
| Updating package install for 'tap'
/ Waiting for 'PackageInstall' reconciliation for 'tap'


Updated package install 'tap' in namespace 'tap-install'
```

Generate and publish TechDocs

This topic tells you how to generate and publish TechDocs for catalogs as part of Tanzu Developer Portal. For more information about TechDocs, see the Backstage.io documentation.

Create an Amazon S3 bucket

To create an Amazon S3 bucket:

1. Go to Amazon S3.
2. Click **Create bucket**.
3. Give the bucket a name.
4. Select the AWS region.
5. Keep **Block all public access** checked.
6. Click **Create bucket**.

Configure Amazon S3 access

The TechDocs are published to the S3 bucket that was recently created. You need an AWS user's access key to read from the bucket when viewing TechDocs.

Create an AWS IAM user group

To create an AWS IAM User Group:

1. Click **Create Group**.
2. Give the group a name.
3. Click **Create Group**.
4. Click the new group and navigate to **Permissions**.
5. Click **Add permissions** and click **Create Inline Policy**.
6. Click the **JSON** tab and replace contents with this JSON replacing `BUCKET-NAME` with the bucket name.

```
{
  "Version": "2012-10-17",
  "Statement": [
      {
          "Sid": "ReadTechDocs",
          "Effect": "Allow",
          "Action": [
              "s3:ListBucket",
              "s3:GetObject"
          ],
          "Resource": [
              "arn:aws:s3:::BUCKET-NAME",
              "arn:aws:s3:::BUCKET-NAME/*"
          ]
      }
  ]
}
```

7. Click **Review policy**.
8. Give the policy a name and click **Create policy**.

Create an AWS IAM user

To create an AWS IAM User to add to this group:

1. Click **Add users**.
2. Give the user a name.
3. Verify **Access key - Programmatic access** and click **Next: Permissions**.
4. Verify the IAM Group to add the user to and click **Next: Tags**.
5. Click **Next: Review** then click **Create user**.
6. Record the **Access key ID** (`AWS_READONLY_ACCESS_KEY_ID`) and the **Secret access key** (`AWS_READONLY_SECRET_ACCESS_KEY`) and click **Close**.

Find the catalog locations and their entities' namespace, kind, and name

TechDocs are generated for catalogs that have Markdown source files for TechDocs. To find the catalog locations and their entities' namespace, kind, and name:

1. The catalogs appearing in Tanzu Developer Portal are listed in the config values under `app_config.catalog.locations`.
2. For a catalog, clone the catalog's repository to the local file system.
3. Find the `mkdocs.yml` that is at the root of the catalog. There is a YAML file describing the catalog at the same level called `catalog-info.yaml`.
4. Record the values for `namespace`, `kind`, and `metadata.name`, and the directory path containing the YAML file.
5. Record the `spec.targets` in that file.
6. Find the namespace, kind, or name for each of the targets:
    1. Go to the target's YAML file.
    2. The `namespace` value is the value of `namespace`. If it is not specified, it has the value `default`.
    3. The `kind` value is the value of `kind`.

4. The `name` value is the value of `metadata.name`.
5. Record the directory path containing the YAML file.

Use the TechDocs CLI to generate and publish TechDocs

VMware uses `npx` to run the TechDocs CLI, which requires `Node.js` and `npm`. To generate and publish TechDocs by using the TechDocs CLI:

1. Download and install Node.js and npm.
2. Install `npx` by running:

   ```
   npm install -g npx
   ```

3. Generate the TechDocs for the root of the catalog by running:

   ```
   npx @techdocs/cli generate --source-dir DIRECTORY-CONTAINING-THE-ROOT-YAML-FILE
   --output-dir ./site
   ```

   Note

   This creates a temporary `site` directory in your current working directory that contains the generated TechDocs files.

4. Review the contents of the `site` directory to verify the TechDocs were generated.
5. Set environment variables for authenticating with Amazon S3 with an account that has read/write access:

   ```
   export AWS_ACCESS_KEY_ID=AWS-ACCESS-KEY-ID
   export AWS_SECRET_ACCESS_KEY=AWS-SECRET-ACCESS-KEY
   export AWS_REGION=AWS-REGION
   ```

6. Publish the TechDocs for the root of the catalog to the Amazon S3 bucket you created earlier by running:

   ```
   npx @techdocs/cli publish --publisher-type awsS3 --storage-name BUCKET-NAME --e
   ntity \
   NAMESPACE/KIND/NAME --directory ./site
   ```

   Where `NAMESPACE/KIND/NAME` are the values for `namespace`, `kind`, and `metadata.name` you recorded earlier. For example, `default/location/yelb-catalog-info`.

7. For each of the `spec.targets` found earlier, repeat the `generate` and `publish` commands.

   Note

   The `generate` command erases the contents of the `site` directory before creating new TechDocs files. Therefore, the `publish` command must follow the `generate` command for each target.

Update the `techdocs` section in `app-config.yaml` to point to the Amazon S3 bucket

Update the config values you used during installation to point to the Amazon S3 bucket that has the published TechDocs files:

1. Add or edit the `techdocs` section under `app_config` in the config values with the following YAML, replacing placeholders with the appropriate values.

   ```
   techdocs:
     builder: 'external'
     publisher:
       type: 'awsS3'
       awsS3:
         bucketName: BUCKET-NAME
         accountId: ACCOUNT-ID
         region: AWS-REGION
   aws:
     accounts:
       - accountId: ACCOUNT-ID
         accessKeyId: AWS-READONLY-ACCESS-KEY-ID
         secretAccessKey: AWS-READONLY-SECRET-ACCESS-KEY
   ```

   For more information about authentication to an Amazon S3 bucket through an assumed role, see the Backstage documentation.

2. Update your installation from the Tanzu CLI:

   Tanzu Application Platform package installation
   If you installed Tanzu Developer Portal as part of the Tanzu Application Platform package (in other words, if you installed it by running `tanzu package install tap ...`) then run:

   ```
   tanzu package installed update tap \
     --version PACKAGE-VERSION \
     --values-file VALUES-FILE
   ```

   Where `PACKAGE-VERSION` is your package version and `VALUES-FILE` is your values file

   Separate package installation
   If you installed Tanzu Developer Portal as its own package (in other words, if you installed it by running `tanzu package install tap-gui ...`) then run:

   ```
   tanzu package installed update tap-gui \
     --version PACKAGE-VERSION \
     --values-file VALUES-FILE
   ```

   Where `PACKAGE-VERSION` is your package version and `VALUES-FILE` is your values file

3. Verify the status of the update by running:

   ```
   tanzu package installed list
   ```

4. Go to the **Docs** section of your catalog and view the TechDocs pages to verify the content is loaded from the S3 bucket.

Overview of Configurator

Use the Tanzu Developer Portal Configurator tool to add custom functions to Tanzu Developer Portal.

Differences between the pre-built Tanzu Developer Portal and a customized portal

Tanzu Application Platform (commonly known as TAP) has a pre-built version of Tanzu Developer Portal. The pre-built Tanzu Developer Portal includes all the core functions of Backstage (software catalog, TechDocs, API Docs, templates, accelerators, and Kubernetes).

You can choose to use just the pre-built Tanzu Developer Portal and the integrations with Tanzu Application Platform. Alternatively, you can customize Tanzu Developer Portal for your needs by using the Configurator tool to add additional Tanzu Developer Portal plug-ins.

Tanzu Developer Portal plug-ins

Tanzu Developer Portal is built using the Cloud Native Computing Foundation's project Backstage.

Backstage offers Backstage plug-ins for adding custom functions, and Tanzu Developer Portal does the same. A Tanzu Developer Portal plug-in is simply a Backstage plug-in that is wrapped in a small amount of code to make integrating with Configurator and Tanzu Developer Portal easier.

How Configurator works

Configurator takes a list of Tanzu Developer Portal plug-ins that you want to integrate with your Tanzu Developer Portal. With that plug-in list, Configurator generates a portal customized to your specifications.

With these new plug-ins, maintaining a customized portal remains similar to maintaining a pre-built Tanzu Developer Portal. Tanzu Application Platform automation handles the maintenance.

Next steps

- Learn the Tanzu Developer Portal Configurator concepts.
- See the list of Tanzu Developer Portal plug-ins currently available for use.
- When you are ready, build your customized Tanzu Developer Portal with Configurator.
- Learn how to create your own Tanzu Developer Portal plug-in of an existing Backstage plug-in.
- For more information about how to create Backstage plug-ins, see the Backstage plug-in documentation.

Overview of Configurator

Use the Tanzu Developer Portal Configurator tool to add custom functions to Tanzu Developer Portal.

Differences between the pre-built Tanzu Developer Portal and a customized portal

Tanzu Application Platform (commonly known as TAP) has a pre-built version of Tanzu Developer Portal. The pre-built Tanzu Developer Portal includes all the core functions of Backstage (software catalog, TechDocs, API Docs, templates, accelerators, and Kubernetes).

You can choose to use just the pre-built Tanzu Developer Portal and the integrations with Tanzu Application Platform. Alternatively, you can customize Tanzu Developer Portal for your needs by using the Configurator tool to add additional Tanzu Developer Portal plug-ins.

Tanzu Developer Portal plug-ins

Tanzu Developer Portal is built using the Cloud Native Computing Foundation's project Backstage.

Backstage offers Backstage plug-ins for adding custom functions, and Tanzu Developer Portal does the same. A Tanzu Developer Portal plug-in is simply a Backstage plug-in that is wrapped in a small amount of code to make integrating with Configurator and Tanzu Developer Portal easier.

How Configurator works

Configurator takes a list of Tanzu Developer Portal plug-ins that you want to integrate with your Tanzu Developer Portal. With that plug-in list, Configurator generates a portal customized to your specifications.

With these new plug-ins, maintaining a customized portal remains similar to maintaining a pre-built Tanzu Developer Portal. Tanzu Application Platform automation handles the maintenance.

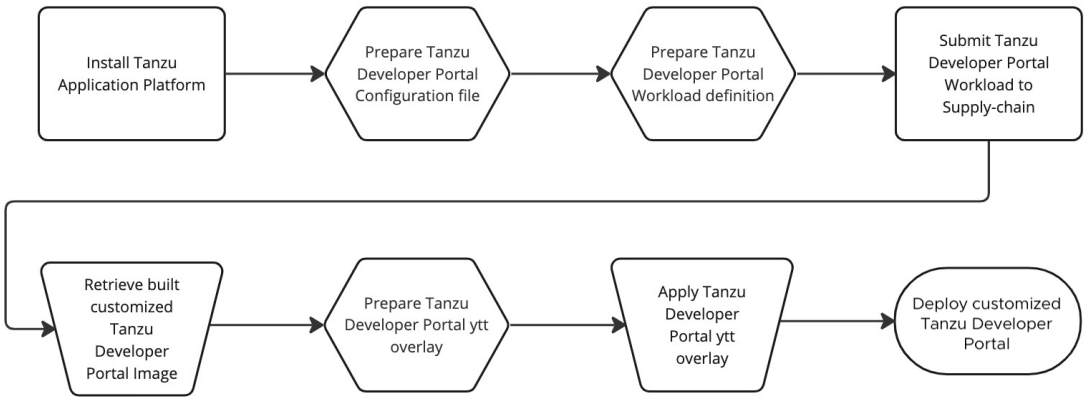
Next steps

- Learn the Tanzu Developer Portal Configurator concepts.
- See the list of Tanzu Developer Portal plug-ins currently available for use.
- When you are ready, build your customized Tanzu Developer Portal with Configurator.
- Learn how to create your own Tanzu Developer Portal plug-in of an existing Backstage plug-in.
- For more information about how to create Backstage plug-ins, see the Backstage plug-in documentation.

Tanzu Developer Portal Configurator Concepts

This topic gives you conceptual overviews of how Tanzu Developer Portal Configurator works.

Overview of how to customize your portal



To use your customized portal with all the runtime configuration values used in your pre-built version:

1. Install a working installation of Tanzu Application Platform with a working instance of the pre-built Tanzu Developer Portal.
2. Prepare your Configurator buildtime configuration file.

3. Prepare your Configurator workload definition YAML.
4. Submit your Configurator workload definition YAML to your supplychain.
5. After the image is built, retrieve your customized Tanzu Developer Portal image from your supplychain deliverables.
6. To run your customized portal, prepare a ytt overlay to replace the pre-built Tanzu Developer Portal in Tanzu Application Platform with your customized version. For more information about ytt, see the Carvel documentation.
7. Apply the ytt overlay to your cluster.

Overviews of buildtime configuration and runtime configuration

The following sections describe the differences between buildtime configuration and runtime configuration.

Runtime configuration

Runtime configuration refers to the values that you use to configure your existing portal image. You provide these values in `tap-values.yaml` when you install and run your portal. Runtime configuration values can include:

- The name of your portal
- Integrations (GitHub or GitLab keys, identity provider configuration, and so on)
- The locations of any catalogs on GitHub or GitLab
- Security credentials
- Configuration made available by any Tanzu Developer Portal plug-ins

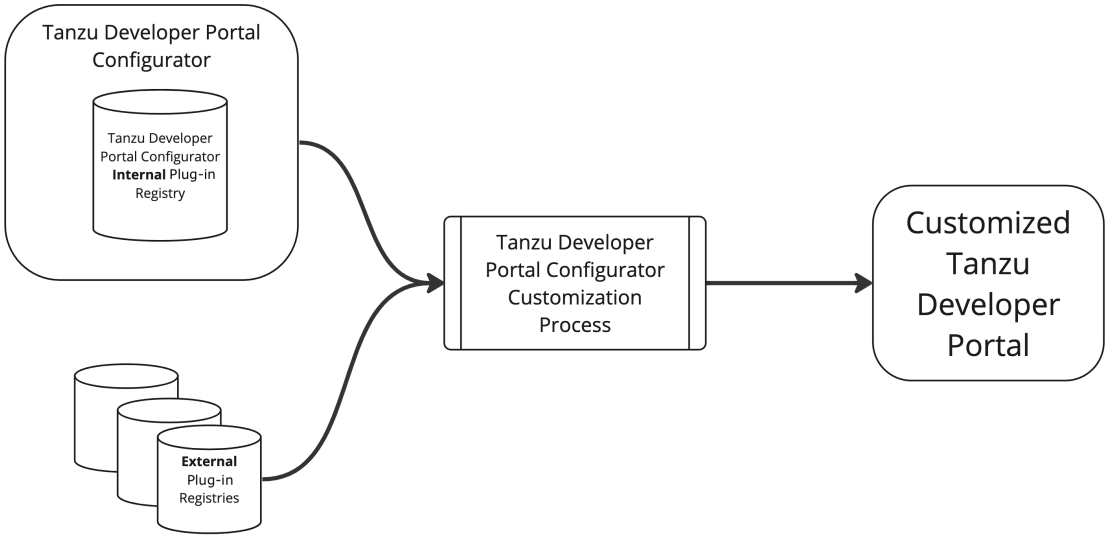
Buildtime configuration

Buildtime configuration refers to the values that you pass to Configurator to generate a new portal image. This configuration consists of a list of Tanzu Developer Portal plug-ins and any default runtime configuration for the portal image.

Tanzu Developer Portal Configurator

Tanzu Developer Portal Configurator is the image that contains everything necessary to build a customized version of Tanzu Developer Portal.

Configurator includes a templated version of Tanzu Developer Portal, an internal registry of Tanzu Developer Portal plug-ins, and tools to enable the build process to incorporate external plug-ins.



Tanzu Developer Portal plug-ins

While Backstage uses Backstage plug-ins to enable the user to customize functions, Tanzu Developer Portal uses Tanzu Developer Portal plug-ins to do the same thing.

A Tanzu Developer Portal plug-in is simply a Backstage plug-in wrapped in a small amount of code to facilitate easy integration into Configurator and Tanzu Developer Portal.

Internal plug-ins and external plug-ins

Internal Tanzu Developer Portal plug-ins are included inside the Tanzu Developer Portal Configurator image. These include:

- Tanzu Developer Portal plug-ins specifically built for Tanzu Application Platform
- Tanzu Developer Portal plug-ins for core Backstage plug-ins

External Tanzu Developer Portal plug-ins are not in the Tanzu Developer Portal Configurator image. They are hosted in, and installed, from an external registry, such as npmjs.com. External Tanzu Developer Portal plug-ins can include custom functions or wrap existing third-party Backstage plug-ins.

Tanzu Developer Portal plug-in surfaces

Tanzu Developer Portal plug-in surfaces are the mechanism that allows Tanzu Developer Portal plug-ins to change the behavior of the portal.

When adding a Backstage plug-in to an instance of Backstage, code modifications are required. Rather than editing code, Tanzu Developer Portal exposes extension points where Tanzu Developer Portal plug-ins can inject code. These extension points are known as surfaces.

The pre-built version of Tanzu Developer Portal has several surfaces to edit existing functions, including:

- Allowing new items to be added to the sidebar
- Adding new routes, such as `https://YOUR_PORTAL_URL/plugin`
- The ability to appear as a Catalog Overview tab

This pattern of exposing surfaces where Tanzu Developer Portal plug-ins can inject new behavior is not limited to what is offered in the pre-built version of Tanzu Developer Portal.

This pattern can also be applied on top of Tanzu Developer Portal plug-ins themselves. This allows for one Tanzu Developer Portal plug-in to build on top of another Tanzu Developer Portal plug-in.

Build your customized Tanzu Developer Portal with Configurator

This topic tells you how to build your customized Tanzu Developer Portal with Configurator.

Prerequisites

Ensure that the following is true:

- Configurator has a running and operating Tanzu Application Platform instance to build the customized portal and run the resulting customized image. You can use a `full` profile for everything or you can use a `build` profile for customizing the portal and a `view` profile for running the customized portal. For more information, see Components and installation profiles for Tanzu Application Platform.
- Your instance of Tanzu Application Platform has a working supply chain that can build the Tanzu Developer Portal bundle. It doesn't need to be able to deliver it because currently an overlay is used to place the built image on the cluster where the pre-built Tanzu Developer Portal resides.
- Your developer namespace has access to both:
  - Your installation registry where the source Tanzu Application bundles are located
  - Your build registry where your built images are staged
- You have a working Tanzu Application Platform installation and you can build a sample application, such as `Tanzu-Java-Web-App` in Generate an application with Application Accelerator.
- Your extra plug-ins are in the npmjs.com registry or a private registry.
- Carvel tools is installed on your workstation. `imgpkg`, in particular, must be installed to perform some of the build steps.
- The yq tool is installed.
- The Docker CLI is installed and you are logged into your registry.

Important

Tanzu Application Platform plug-ins cannot be removed from customized portals. However, if you decide you want to hide them, you can use the runtime configuration options in your `tap-values.yaml` file.

Prepare your Configurator configuration file

To prepare your Configurator configuration file:

1. Create a new file called `tdp-config.yaml` by using the following template:

```
app:
  plugins:
    - name: "NPM-PLUGIN-FRONTEND"
      version: "NPM-PLUGIN-FRONTEND-VERSION"
backend:
  plugins:
```

```
    - name: "NPM-PLUGIN-BACKEND"
      version: "NPM-PLUGIN-BACKEND-VERSION"
```

Where:

- `NPM-PLUGIN-FRONTEND` is the npm registry and module name of the front-end plug-in
- `NPM-PLUGIN-FRONTEND-VERSION` is the version of your desired front-end plug-in that exists in the npm registry
- `NPM-PLUGIN-BACKEND` is the npm registry and module name of the back-end plug-in that you want
- `NPM-PLUGIN-BACKEND-VERSION` is the version of your desired back-end plug-in that exists in the npm registry

The following example adds the sample `techinsights` plug-in. The plug-in wrapper is available in the VMware NPM repository:

```
app:
  plugins:
    - name: "@vmware-tanzu/tdp-plugin-techinsights"
      version: "0.0.2"

backend:
  plugins:
    - name: "@vmware-tanzu/tdp-plugin-techinsights-backend"
      version: "0.0.2"
```

2. If you plan to add plug-ins that exist in a private registry, configure the Configurator with a private registry.
3. Encode the file in Base64, to later embed `tdp-config.yaml` in the workload definition file, by running:

   ```
   base64 -i tdp-config.yaml
   ```

   If no plug-ins are specified in your `tdp-config.yaml` file, the following plug-ins are present by default:

   - Runtime Resources Visibility
   - Application Live View
   - Application Accelerator
   - API Documentation
   - Supply Chain Choreographer
   - Security Analysis
   - DORA Metrics

Identify your Configurator image

To build a customized Tanzu Developer Portal, you must identify the Configurator image to pass through the supply chain. Depending on your choices during installation, this is on either `registry.tanzu.vmware.com` or the local image registry (`imgpkg`) that you moved the installation packages to.

Using imgpkg v0.39.0 or earlier
If using imgpkg v0.39.0 or earlier:

1. Using the `imgpkg` tool, retrieve the image location by running:

   ```
   imgpkg describe -b $(kubectl get -n tap-install $(kubectl get package -n tap-
   install \
   --field-selector spec.refName=tpb.tanzu.vmware.com -o name) -o \
   jsonpath="{.spec.template.spec.fetch[0].imgpkgBundle.image}") -o yaml --tty=t
   rue | grep -A 1 \
   "kbld.carvel.dev/id:[[:blank:]]*[^[:blank:]]*configurator" | grep "image:" |
   sed 's/[[:blank:]]*image:[[:blank:]]*//g'
   ```

   Output similar to the following appears:

   ```
   IMAGE-REGISTRY/tap-packages@sha256:bea2f5bec5c5102e2a69a4c5047fae3d51f2974191
   1cf5bb588893aa4e03ca27
   ```

2. Record this value to later use it in place of the `TDP-IMAGE-LOCATION` placeholder in the workload definition.

Using imgpkg v0.40.0 or later

If using imgpkg v0.40.0 or later:

1. Using the `imgpkg` tool, download the bundle by running:

   ```
   imgpkg pull -b $(kubectl get -n tap-install $(kubectl get package -n tap-inst
   all \
   --field-selector spec.refName=tpb.tanzu.vmware.com -o name) -o \
   jsonpath="{.spec.template.spec.fetch[0].imgpkgBundle.image}") -o bundle
   ```

2. Retrieve the image location by running:

   ```
   cat bundle/.imgpkg/images.yml | grep -A 1 "kbld.carvel.dev/id:[[:blank:]]*[^
   [:blank:]]*configurator" | \
   grep "image:" | sed 's/[[:blank:]]*image:[[:blank:]]*//g'
   ```

   Output similar to the following appears:

   ```
   IMAGE-REGISTRY/tap-packages@sha256:bea2f5bec5c5102e2a69a4c5047fae3d51f2974191
   1cf5bb588893aa4e03ca27
   ```

3. Record this value to later use it in place of the `TDP-IMAGE-LOCATION` placeholder in the workload definition.

Build your customized portal

There are two options for passing your workload through a supply chain and building your customized portal:

Use an existing supply chain

To use an existing supply chain to build your custom portal:

1. Create a file called `tdp-workload.yaml` with the following content:

   ```
   apiVersion: carto.run/v1alpha1
   kind: Workload
   metadata:
     name: tdp-configurator
     namespace: DEVELOPER-NAMESPACE
     labels:
       apps.tanzu.vmware.com/workload-type: web
       app.kubernetes.io/part-of: tdp-configurator
   ```

```
spec:
  build:
    env:
      - name: BP_NODE_RUN_SCRIPTS
        value: 'set-tpb-config,portal:pack'
      - name: TPB_CONFIG
        value: /tmp/tpb-config.yaml
      - name: TPB_CONFIG_STRING
        value: ENCODED-TDP-CONFIG-VALUE

  source:
    image: TDP-IMAGE-LOCATION
    subPath: builder
```

Where:

- `DEVELOPER-NAMESPACE` is an appropriately configured developer namespace on the cluster.
- `ENCODED-TDP-CONFIG-VALUE` is the Base64-encoded value that you encoded earlier.
- `TDP-IMAGE-LOCATION` is the location of the Configurator image in the image registry from which you installed Tanzu Application Platform. You discovered this location earlier when you identified your Configurator image.

Depending on which supply chain you're using or how you configured it, you might need to add extra sections to your workload definition file to accommodate activities such as testing.

For example:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: tdp-configurator
  namespace: default
  labels:
    apps.tanzu.vmware.com/workload-type: web
    app.kubernetes.io/part-of: tdp-configurator
spec:
  build:
    env:
      - name: BP_NODE_RUN_SCRIPTS
        value: "set-tpb-config,portal:pack"
      - name: TPB_CONFIG
        value: /tmp/tpb-config.yaml
      - name: TPB_CONFIG_STRING
        value: YXBwOgogIHBsdWdpbnM6CiAgICAtIG5hbWU6ICdAdm13YXJlLXRhbnp1L3RkcC
1wbHVnaW4tdGVjaGluc2lnaHRzJwogICAgICB2ZXJzaW9uOiAnMC4wLjInCgpiYWNrZW5kOgogIHB
sdWdpbnM6IAogICAgLSBuYW1lOiAnQHZtd2FyZS10YW56dS90dS1wbHVnaW4tdGVjaHBsNpZ2h0
cy1iYWNrZW5kJwogICAgICB2ZXJzaW9uOiAnMC4wLjIn
  source:
    image: TDP-IMAGE-LOCATION
    subPath: builder
```

The `TPB_CONFIG_STRING` value can be decoded as the earlier example that includes the TechInsights front-end and back-end plug-ins.

`TDP-IMAGE-LOCATION` is the location of your Configurator image identified in earlier steps.

2. Submit the workload definition file by running:

```
kubectl apply -f tdp-workload.yaml
```

The supply chain does not need to go beyond the image-provider stage. After an image is built, you can proceed to run your customized Tanzu Developer Portal.

Use a custom supply chain

To use a custom supply chain to build your custom portal, follow these steps. This method creates a custom supply chain for `workload-type: tdp` that encompasses only the steps necessary to build the customized image:

1. Create a file called `tdp-sc.yaml` with the following content:

```
apiVersion: carto.run/v1alpha1
kind: ClusterSupplyChain
metadata:
  name: tdp-configurator
spec:
  resources:
  - name: source-provider
    params:
    - default: default
      name: serviceAccount
    - default: TDP-IMAGE-LOCATION
      name: tdp_configurator_bundle
    templateRef:
      kind: ClusterSourceTemplate
      name: tdp-source-template
  - name: image-provider
    params:
    - default: default
      name: serviceAccount
    - name: registry
      default:
        ca_cert_data: ""
        repository: IMAGE-REPOSITORY
        server: REGISTRY-HOSTNAME
    - default: default
      name: clusterBuilder
    sources:
    - name: source
      resource: source-provider
    templateRef:
      kind: ClusterImageTemplate
      name: tdp-kpack-template

  selectorMatchExpressions:
  - key: apps.tanzu.vmware.com/workload-type
    operator: In
    values:
    - tdp
---
apiVersion: carto.run/v1alpha1
kind: ClusterImageTemplate
metadata:
  name: tdp-kpack-template
spec:
  healthRule:
    multiMatch:
      healthy:
        matchConditions:
        - status: "True"
          type: BuilderReady
        - status: "True"
          type: Ready
      unhealthy:
        matchConditions:
        - status: "False"
```

```
        type: BuilderReady
      - status: "False"
        type: Ready
imagePath: .status.latestImage
lifecycle: mutable
params:
- default: default
  name: serviceAccount
- default: default
  name: clusterBuilder
- name: registry
  default: {}
ytt: |
  #@ load("@ytt:data", "data")
  #@ load("@ytt:regexp", "regexp")

  #@ def merge_labels(fixed_values):
  #@   labels = {}
  #@   if hasattr(data.values.workload.metadata, "labels"):
  #@     exclusions = ["kapp.k14s.io/app", "kapp.k14s.io/association"]
  #@     for k,v in dict(data.values.workload.metadata.labels).items():
  #@       if k not in exclusions:
  #@         labels[k] = v
  #@       end
  #@     end
  #@   end
  #@   labels.update(fixed_values)
  #@   return labels
  #@ end

  #@ def image():
  #@   return "/".join([
  #@    data.values.params.registry.server,
  #@    data.values.params.registry.repository,
  #@    "-".join([
  #@      data.values.workload.metadata.name,
  #@      data.values.workload.metadata.namespace,
  #@    ])
  #@   ])
  #@ end

  #@ bp_node_run_scripts = "set-tpb-config,portal:pack"
  #@ tpb_config = "/tmp/tpb-config.yaml"

  #@ for env in data.values.workload.spec.build.env:
  #@   if env.name == "TPB_CONFIG_STRING":
  #@     tpb_config_string = env.value
  #@   end
  #@   if env.name == "BP_NODE_RUN_SCRIPTS":
  #@     bp_node_run_scripts = env.value
  #@   end
  #@   if env.name == "TPB_CONFIG":
  #@     tpb_config = env.value
  #@   end
  #@ end

  apiVersion: kpack.io/v1alpha2
  kind: Image
  metadata:
    name: #@ data.values.workload.metadata.name
    labels: #@ merge_labels({ "app.kubernetes.io/component": "build" })
  spec:
    tag: #@ image()
    serviceAccountName: #@ data.values.params.serviceAccount
    builder:
      kind: ClusterBuilder
```

```
        name: #@ data.values.params.clusterBuilder
      source:
        blob:
          url: #@ data.values.source.url
        subPath: builder
      build:
        env:
        - name: BP_OCI_SOURCE
          value: #@ data.values.source.revision
        #@  if regexp.match("^([a-zA-Z0-9\/_-]+)(\@sha1:)?[0-9a-f]{40}$", dat
a.values.source.revision):
        - name: BP_OCI_REVISION
          value: #@ data.values.source.revision
        #@ end
        - name: BP_NODE_RUN_SCRIPTS
          value: #@ bp_node_run_scripts
        - name: TPB_CONFIG
          value: #@ tpb_config
        - name: TPB_CONFIG_STRING
          value: #@ tpb_config_string

---
apiVersion: carto.run/v1alpha1
kind: ClusterSourceTemplate
metadata:
  name: tdp-source-template
spec:
  healthRule:
    singleConditionType: Ready
  lifecycle: mutable
  params:
  - default: default
    name: serviceAccount
  revisionPath: .status.artifact.revision
  urlPath: .status.artifact.url
  ytt: |
    #@ load("@ytt:data", "data")

    #@ def merge_labels(fixed_values):
    #@   labels = {}
    #@   if hasattr(data.values.workload.metadata, "labels"):
    #@     exclusions = ["kapp.k14s.io/app", "kapp.k14s.io/association"]
    #@     for k,v in dict(data.values.workload.metadata.labels).items():
    #@       if k not in exclusions:
    #@         labels[k] = v
    #@       end
    #@     end
    #@   end
    #@   labels.update(fixed_values)
    #@   return labels
    #@ end

    ---
    apiVersion: source.apps.tanzu.vmware.com/v1alpha1
    kind: ImageRepository
    metadata:
      name: #@ data.values.workload.metadata.name
      labels: #@ merge_labels({ "app.kubernetes.io/component": "source" })
    spec:
      serviceAccountName: #@ data.values.params.serviceAccount
      interval: 10m0s
      #@ if hasattr(data.values.workload.spec, "source") and hasattr(data.val
ues.workload.spec.source, "image"):
      image: #@ data.values.workload.spec.source.image
      #@ else:
```

```
      image: #@ data.values.params.tdp_configurator_bundle
      #@ end
```

Where:

- `TDP-IMAGE-LOCATION` is the location of the Configurator image in the image registry from which you installed Tanzu Application Platform. You discovered this location earlier when you identified your Configurator image.
- `REGISTRY-HOSTNAME` is the name of the container registry that your developer namespace was configured to push artifacts to.
- `IMAGE-REPOSITORY` is the name of the repository (folder) on the `REGISTRY-HOSTNAME` that you want to push built artifacts to.

2. Submit the custom supply chain file by running:

```
kubectl apply -f tdp-sc.yaml
```

3. Create a file called `tdp-workload.yaml` with the following content:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: tdp-configurator-1-sc
  namespace: DEVELOPER-NAMESPACE
  labels:
    apps.tanzu.vmware.com/workload-type: tdp
    app.kubernetes.io/part-of: tdp-configurator-1-custom
spec:
  build:
    env:
      - name: TPB_CONFIG_STRING
        value: ENCODED-TDP-CONFIG-VALUE
```

Where:

- `DEVELOPER-NAMESPACE` is an appropriately configured developer namespace on the cluster
- `ENCODED-TDP-CONFIG-VALUE` is the Base64-encoded value that you encoded earlier

4. Submit the workload definition file by running:

```
tanzu apps workload create -f tdp-workload.yaml
```

After the job finishes the Image Provider stage of your supply chain, run your customized Tanzu Developer Portal instance.

Create a Tanzu Developer Portal plug-in

This topic teaches you how to create a Tanzu Developer Portal plug-in by wrapping an existing Backstage plug-in. After you create a Tanzu Developer Portal plug-in, you can build a customized Tanzu Developer Portal with Configurator.

Prerequisites

Meet the following prerequisites before creating a Tanzu Developer Portal plug-in.

Software

Ensure that you have the following software installed locally to develop a Tanzu Developer Portal plug-in:

- Node 18: `nvm` is recommended. For how to install `nvm`, see the `nvm` GitHub repository. For how to install a specific version of `nvm`, see the NodeJS documentation.
- yarn v1
- (Optional) A UNIX-based OS: If you use Windows, you must find alternatives to some commands in this topic.

A Backstage plug-in in an accessible npm registry

Ensure that the Backstage plug-in you want to wrap is in an npm registry. You can use your own private registry or a public registry, such as npm JS. Both your development machine and your Tanzu Application Platform cluster must have access to the registry.

This topic tells you, by way of example, how to wrap the Backstage TechInsights plug-in. This plug-in consists of back-end and front-end components, both of which are available on npm JS:

- @backstage/plugin-tech-insights v0.3.19
- @backstage/plugin-tech-insights-backend v0.5.21

Set up a development environment

This topic tells you how to create two Tanzu Developer Portal plug-ins by wrapping the `@backstage/plugin-tech-insights` and `@backstage/plugin-tech-insights-backend` Backstage plug-ins. You can create a separate repository for each of these plug-ins, but it's easier and simpler to do the work for both in a single monorepo.

Generate a Backstage app for the monorepo

This topic describes how to use the Backstage tools `@backstage/create-app` and `backstage-cli` to manage your monorepo. The Backstage tools make managing packages easier. However, you will not develop a traditional Backstage app, and you will remove some portions of generated code later.

1. This tutorial uses `plugin-wrappers` as the app name. Run the `create-app` script and, when prompted, enter a name for your app:

   ```
   npx @backstage/create-app@0.5.7 --skip-install
   ```

   `@backstage/create-app` v0.5.7 is used because the Tanzu Developer Portal version that ships with Tanzu Application Platform v1.8 uses Backstage v1.20.3. Backstage v1.20.3 uses `@backstage/create-app` v0.5.7. For more information, see the Backstage version manifest.

   Important

   Ensure that you use the correct versions of dependencies for your Tanzu Application Platform version. Use the Backstage version compatibility reference table to find which versions of Backstage dependencies work with your version of Tanzu Application Platform.

   The `--skip-install` flag tells the script to not run `yarn install`. This is because you will remove the unnecessary dependencies that would have been needed if you were building a traditional Backstage app before installing your dependencies.

The `create-app` command scaffolds a Backstage project structure under a directory matching your project name.

2. Run:

```
cd APP-NAME
```

Where `APP-NAME` is your application name. For example, `plugin-wrappers`.

Remove some dependencies

To remove unnecessary dependencies you must:

1. Remove the packages directory by running:

```
rm -rf packages/
```

The packages directory contains a scaffolded Backstage `app` and `backend`, which are only necessary for a traditional Backstage app.

2. Remove the packages directory from the `yarn` workspaces by deleting the `"packages/*"` line within the `workspaces` attribute in `package.json`. For example:

```
diff --git a/package.json b/package.json
index 00d64c9..77f38f3 100644
--- a/package.json
+++ b/package.json
@@ -24,7 +24,6 @@
  },
  "workspaces": {
    "packages": [
-      "packages/*",
      "plugins/*"
    ]
  },
```

3. Install the dependencies by running:

```
yarn install
```

This command installs `backstage-cli` and a few other dependencies.

Create the Tech Insights front-end Tanzu Developer Portal plug-in

Now that you have an environment to develop your Tanzu Developer Portal plug-ins, you can begin wrapping Backstage plug-ins. You will start with the Tech Insight front-end plug-in.

Generate a front-end plug-in

This section describes how to generate a front-end plug-in:

1. Generate a front-end plug-in by running the following command, replacing `PACKAGE-NAMESPACE` with your namespace (for example, `@mycompany`):

```
yarn backstage-cli new --select plugin --option id=tech-insights-wrapper --scop
e PACKAGE-NAMESPACE --no-private
```

Important

> The `yarn install` step of the previous command fails because of a Node version issue. This is handled in a later step.

Here is a summary of what the `backstage-cli new` script does:

- `--select plugin` creates a front-end plug-in
- `--option id=tech-insights-wrapper` names the plug-in `tech-insights-wrapper`
- `--scope PACKAGE-NAMESPACE` scopes the package under the `PACKAGE-NAMESPACE` namespace
- `--no-private` sets the package to public

2. Open the `plugins/tech-insights-wrapper/package.json` to see how these options were mapped to the generated `package.json`.

Add dependencies for the front-end plug-in

Update your dependencies for the specific Backstage plug-in you want to wrap:

1. Replace the `dependencies` in the `package.json` with:

```
...
"dependencies": {
 "@backstage/plugin-tech-insights": "0.3.19",
 "@backstage/plugin-catalog": "1.15.1",
 "@vmware-tanzu/core-common": "2.0.0",
 "@vmware-tanzu/core-frontend": "2.0.0"
},
...
```

2. The dependency on `@backstage/plugin-tech-insights` is obvious, but verify the version is compatible with your Tanzu Application Platform version by reading the Backstage version compatibility table and checking the version listed in the Backstage Manifest file for your specific Tanzu Application Portal version.
3. `@backstage/plugin-catalog` is needed for a UI component that you use later. Verify its version by using the Backstage version compatibility table.
4. Verify that you are using the correct versions of `@vmware-tanzu/core-common` and `@vmware-tanzu/core-frontend` by cross-referencing the dependency name with your Tanzu Application Platform version in the Tanzu Developer Portal plug-in libraries compatibility tables. You use `@vmware-tanzu/core-common` and `@vmware-tanzu/core-frontend` later for integrating the Backstage plug-in with Tanzu Developer Portal.
5. Install the dependencies you added by running:

```
cd plugins/tech-insights-wrapper
yarn install --ignore-engines
```

Remove unnecessary code for the front-end plug-in

The `backstage-cli new` command created example code that you don't need. Remove this code and start with an empty `src` directory by running:

```
rm -rf dev/ src/ && mkdir src
```

Wrap the Backstage front-end plug-in

1. Read the documentation for @backstage/plugin-tech-insights. You will see that to use this Backstage plug-in, you must edit the content of the serviceEntityPage constant. Because you do not have access to the Tanzu Developer Portal source code, you cannot change that constant directly. Instead, you must use a surface to make the equivalent change.
2. Create the file where you will use a surface to edit the `serviceEntityPage` constant by running:

   ```
   touch src/TechInsightsFrontendPlugin.tsx
   ```

3. In the `TechInsightsFrontendPlugin.tsx` file, add the following code:

   ```
   import { EntityLayout } from '@backstage/plugin-catalog';
   import { EntityTechInsightsScorecardContent } from '@backstage/plugin-tech-insi
   ghts';
   import {
    AppPluginInterface,
    SurfaceStoreInterface,
    EntityPageSurface,
   } from '@vmware-tanzu/core-frontend';
   import React from 'react';

   export const TechInsightsFrontendPlugin: AppPluginInterface =
    () => (context: SurfaceStoreInterface) => {
      context.applyTo(
        EntityPageSurface,
        (entityPageSurface) => {
          entityPageSurface.servicePage.addTab(
            <EntityLayout.Route path="/techinsights" title="TechInsights">
              <EntityTechInsightsScorecardContent
                title="TechInsights Scorecard."
                description="TechInsight's default fact-checkers"
              />
            </EntityLayout.Route>,
          );
        },
      );
    };
   ```

   Where:

   - `context.applyTo` is a function that takes the class of the surface you want to interact with, and a function that is passed the instance of that class.
   - The `EntityPageSurface` keeps track of tabs that appear on the service page. You add a new tab by calling `entityPageSurface.servicePage.addTab` and passing it the UI component you want it to render.
   - `TechInsightsFrontendPlugin: AppPluginInterface = () => (context: SurfaceStoreInterface) => {}` is boilerplate code that enables you to interact with the various front-end surfaces in Tanzu Developer Portal.
   - `EntityPageSurface` is one example of the many surfaces available in Tanzu Developer Portal. To discover all the surfaces currently available, see How to use surfaces. For surface API reference information, see API documentation for surfaces.

   This code accomplishes the same thing as the @backstage/plugin-tech-insights, but for an integration with Tanzu Developer Portal instead of a traditional Backstage app.

Expose and build the Tanzu Developer Portal front-end plug-in

To expose and then build the front-end plug-in:

1. Create an `index.ts` file under the `plugins/tech-insights-wrapper/src` directory:

   ```
   touch src/index.ts
   ```

2. In the `index.ts` file write:

   ```
   export { TechInsightsFrontendPlugin as plugin } from './TechInsightsFrontendPlu
   gin';
   ```

   This exports `TechInsightsFrontendPlugin` in a way that enables Configurator to use your plug-in. You need to alias `TechInsightsFrontendPlugin` to `plugin` because the Tanzu Developer Portal Configurator expects compatible plug-ins to export a symbol with the name `plugin`.

3. Build your Tanzu Developer Portal plug-in by running:

   ```
   yarn tsc && yarn build
   ```

You can now publish this plug-in to your npm registry. However, you cannot use the plug-in functions without the back-end portion.

Create the Tech Insights back-end Tanzu Developer Portal plug-in

Creating the back-end plug-in is very similar to the work you did for the front-end plug-in. This section does not describe in detail what is happening at each step except for where it differs from the previous work.

Generate a back-end plug-in

This describes how to generate a back-end plug-in.

From the root of your project, generate a back-end plug-in by running:

```
yarn backstage-cli new --select backend-plugin --option id=tech-insights-wrapper --sco
pe PACKAGE-NAMESPACE --no-private
```

Where:

- `PACKAGE-NAMESPACE` is the namespace for your package. For example, `@mycompany`.
- `--select backend-plugin` tells the `backstage-cli` to generate a back-end plug-in.
- `--option id=tech-insights-wrapper` provides the name for the plug-in. The ID you provide is the same as the front-end plug-in. `backstage-cli` automatically appends `-backend` to the directory and package-name of back-end plug-ins to prevent conflict with the front-end plug-in.

Add dependencies for the back-end plug-in

To add your dependencies for the specific Backstage plug-in you want to wrap:

1. Update the dependencies in `package.json` as follows:

   ```
   "dependencies": {
    "@backstage/plugin-tech-insights-backend": "0.5.21",
    "@backstage/plugin-tech-insights-backend-module-jsonfc": "0.1.39",
    "@vmware-tanzu/core-backend": "2.0.1",
    "express": "4.18.2"
   },
   ```

2. Install your dependencies by running:

```
cd plugins/tech-insights-wrapper-backend/
yarn install --ignore-engines
```

Remove unnecessary code for the back-end plug-in

Remove the Backstage scaffolded example code by running:

```
rm -rf src/ && mkdir src
```

Wrap the Backstage back-end plug-in

1. Within the `src/` directory, create a file called `TechInsightsBackendPlugin.ts` by running:

```
touch src/TechInsightsBackendPlugin.ts
```

2. In `TechInsightsBackendPlugin.ts`, add the following code:

```
import {
 createRouter,
 buildTechInsightsContext,
 createFactRetrieverRegistration,
 entityOwnershipFactRetriever,
 entityMetadataFactRetriever,
 techdocsFactRetriever,
} from '@backstage/plugin-tech-insights-backend';
import { Router } from 'express';
import {
 JsonRulesEngineFactCheckerFactory,
 JSON_RULE_ENGINE_CHECK_TYPE,
} from '@backstage/plugin-tech-insights-backend-module-jsonfc';
import {
 BackendPluginInterface,
 BackendPluginSurface,
 PluginEnvironment,
} from '@vmware-tanzu/core-backend';

const ttlTwoWeeks = { timeToLive: { weeks: 2 } };

export default async function createPlugin(
       env: PluginEnvironment,
): Promise<Router> {
 const techInsightsContext = await buildTechInsightsContext({
   logger: env.logger,
   config: env.config,
   database: env.database,
   discovery: env.discovery,
   tokenManager: env.tokenManager,
   scheduler: env.scheduler,
   factRetrievers: [
     createFactRetrieverRegistration({
       cadence: '0 */6 * * *', // Run every 6 hours - https://crontab.guru/#0_
*/6_*_*_*
       factRetriever: entityOwnershipFactRetriever,
       lifecycle: ttlTwoWeeks,
     }),
     createFactRetrieverRegistration({
       cadence: '0 */6 * * *',
       factRetriever: entityMetadataFactRetriever,
       lifecycle: ttlTwoWeeks,
     }),
     createFactRetrieverRegistration({
```

```
      cadence: '0 */6 * * *',
      factRetriever: techdocsFactRetriever,
      lifecycle: ttlTwoWeeks,
    }),
  ],
  factCheckerFactory: new JsonRulesEngineFactCheckerFactory({
    logger: env.logger,
    checks: [
      {
        id: 'groupOwnerCheck',
        type: JSON_RULE_ENGINE_CHECK_TYPE,
        name: 'Group Owner Check',
        description:
                'Verifies that a Group has been set as the owner for this enti
ty',
        factIds: ['entityOwnershipFactRetriever'],
        rule: {
          conditions: {
            all: [
              {
                fact: 'hasGroupOwner',
                operator: 'equal',
                value: true,
              },
            ],
          },
        },
      },
      {
        id: 'titleCheck',
        type: JSON_RULE_ENGINE_CHECK_TYPE,
        name: 'Title Check',
        description:
                'Verifies that a Title, used to improve readability, has been
set for this entity',
        factIds: ['entityMetadataFactRetriever'],
        rule: {
          conditions: {
            all: [
              {
                fact: 'hasTitle',
                operator: 'equal',
                value: true,
              },
            ],
          },
        },
      },
      {
        id: 'techDocsCheck',
        type: JSON_RULE_ENGINE_CHECK_TYPE,
        name: 'TechDocs Check',
        description:
                'Verifies that TechDocs has been enabled for this entity',
        factIds: ['techdocsFactRetriever'],
        rule: {
          conditions: {
            all: [
              {
                fact: 'hasAnnotationBackstageIoTechdocsRef',
                operator: 'equal',
                value: true,
              },
            ],
          },
        },
```

```
        },
      ],
    }),
  });

  return await createRouter({
    ...techInsightsContext,
    logger: env.logger,
    config: env.config,
  });
}

export const TechInsightsBackendPlugin: BackendPluginInterface =
  () => surfaces =>
    surfaces.applyTo(BackendPluginSurface, backendPluginSurface => {
      backendPluginSurface.addPlugin({
        name: 'tech-insights',
        pluginFn: createPlugin,
      });
    });
```

The majority of this code comes from the npm JS documentation. The Backstage plug-in documentation instructs you to create a constant for `techInsightsEnv` and then configure the router by using `apiRouter.use('/tech-insights', await techInsights(techInsightsEnv))` all in the Backstage source code. Because you are unable to edit the source code of Tanzu Developer Portal, this code accomplishes the same thing by:

- Getting an instance of `BackendPluginSurface`. This surface keeps track of all the back-end plug-ins.
- Adding your plug-in by using the `addPlugin` function. The `name` argument is used to configure the path in the router.

Expose and build the Tanzu Developer Portal back-end plug-in

To expose and then build the back-end plug-in:

1. Create an `index.ts` file by running:

   ```
   touch src/index.ts
   ```

2. In the `index.ts` file, write:

   ```
   export { TechInsightsBackendPlugin as plugin } from './TechInsightsBackendPlugi
   n';
   ```

   This exposes the Tanzu Developer Portal plug-in.

3. Build your plug-in by running:

   ```
   yarn tsc && yarn build
   ```

Next steps

You can now publish your Tanzu Developer Portal plug-ins to your registry of choice. If you want to publish your plug-ins to a private registry, see Configure the Configurator with a private registry.

After publishing your plug-ins, you can build a customized Tanzu Developer Portal with Configurator.

Configure the Configurator with a private registry

If you have plug-ins stored in a private registry that you want to include in your customized Tanzu Developer Portal, you can configure Configurator in the `tdp-config.yaml` file.

Caution

This feature is in the alpha stage. Do not use this feature in a production environment.

Methods

Use one of two methods to configure Configurator:

- Provide the private registry token in the `tpb` configuration
- Store the private registry token in a secret

tpb config method

This method makes your credentials Base64-encoded in the workload definition. Therefore, use this method with caution.

1. In your `tdp-config.yaml` file, add a `registry.uplinks` section to include your private registry. Add the `auth` property to the `registry.uplinks` section, as seen in the Verdaccio documentation.
2. Use an authentication token with the uplink by using one of three methods:
    - Use the default environment variable
    - Use a specified custom environment variable
    - Directly specify a token in the configuration file

    You cannot create an uplink with the name `tpb` or `npmjs` because these packages are reserved and cannot be overridden.
3. In your `tdp-config.yaml` file, add a `registry.packages` section. This section lists access rules for different packages. For more information about this section, see the Verdaccio documentation. You cannot create a package with the name `@tpb/*` because these packages are reserved and cannot be overridden.

    Here is a full example that directly specifies the authentication token in the configuration file:

```
registry:
 uplinks:
   private-registry:
     url: PRIVATE-REGISTRY-ENDPOINT
     auth:
       type: bearer
       token: VALID-TOKEN-FOR-PRIVATE-REGISTRY
 packages:
   '@company/*':
     access: $all
     proxy: private-registry
app:
 plugins:
   - name: '@tpb/plugin-hello-world'
backend:
```

```
 plugins:
   - name: '@tpb/plugin-hello-world-backend'
```

This configuration instructs to download any package with the scope `@company` from the `private-registry` registry with the specified credentials.

secret method

This method is more appropriate if you don't want the registry token to appear in the workload. It is based on the secrets mechanism available in Kubernetes.

1. Create a secret resource that will contain the private registry token. For example:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
    namespace: DEVELOPER-NAMESPACE
    name: private-registry
   data:
    registry_token: A-TOKEN
   ```

   Where `A-TOKEN` is the token value encoded in Base64.

2. In your `tdp-config.yaml` file, add a `registry.uplinks` section to include your private registry. Add the `auth` property to the `registry.uplinks` section, as seen in the Verdaccio documentation, to use an authentication token with an uplink.

   You cannot create an uplink with the name `tpb` or `npmjs` because these uplinks are reserved and cannot be overridden.

3. In your `tdp-config.yaml` file, add a `registry.packages` section. This section lists access rules for different packages. For more information about this section, see the Verdaccio documentation.

   You cannot create a package with the name `@tpb/*` because these packages are reserved and cannot be overridden.

   Here is the full example that uses an `NPM_TOKEN` environment variable:

   ```
   registry:
    uplinks:
      private:
        url: PRIVATE-REGISTRY-ENDPOINT
        auth:
          type: bearer
          token_env: NPM_TOKEN
    packages:
      '@company/*':
        access: $all
        proxy: private
   app:
    plugins:
      - name: '@company/company-plugin'
   backend:
    plugins:
      - name: '@company/company-plugin-backend'
   ```

   This configuration instructs to download any package with the scope `@company` from the `private-registry` registry with the specified credentials.

4. Populate the environment variable value with the secret value in the workload file:

   ```
   apiVersion: carto.run/v1alpha1
   kind: Workload
   # ... remaining of the workload file
   ```

```
spec:
 build:
   env:
     - name: NPM_TOKEN
       valueFrom:
         secretKeyRef:
           key: registry_token
           name: private-registry
# ... remaining of the workload file
```

(Optional) Redirect all the npm requests to a private registry

By default, Configurator redirects all the external requests to `npmjs`. If, for some reason, you need to redirect those requests to your private registry, edit the `registry.packages` section as follows:

```
registry:
  uplinks:
    private:
      url: PRIVATE-REGISTRY-ENDPOINT
      auth:
        type: bearer
        token_env: NPM_TOKEN
  packages:
    '**':
      access: $all
      proxy: private
```

Run your Customized Tanzu Developer Portal

At this stage, you have built your customized Tanzu Developer Portal with Configurator.

After the build has completed, you retrieve the image reference of the built portal. You then use the image reference to perform a ytt overlay to substitute that image name for the one running the pre-built Tanzu Developer Portal on your cluster.

Identify the customized image reference

Identify the image that the supply chain built so that you can use the image reference in your ytt overlay in the next section. There are several ways to retrieve this image name, including using the Kubernetes command-line tool (kubectl) or using the Tanzu Developer Portal GUI.

kubectl
Run:

```
kubectl -n DEVELOPER-NAMESPACE get images.kpack.io WORKLOAD-NAME -o jsonpath={.statu
s.latestImage}
```

Where:

- `DEVELOPER-NAMESPACE` is the configured developer namespace on the cluster where you ran the workload.
- `WORKLOAD-NAME` is the name of the workload you used.

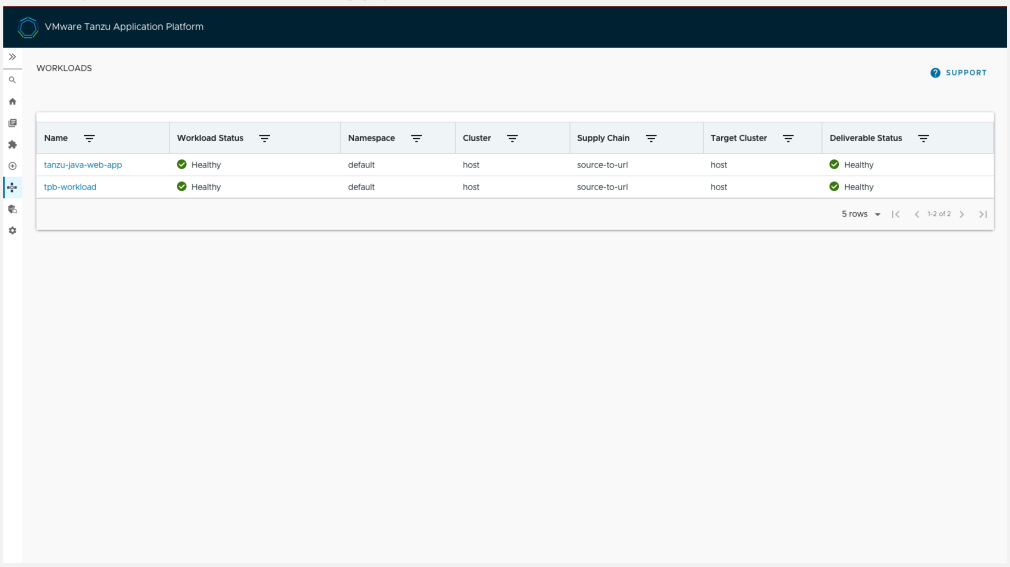
For example:

```
> kubectl -n default get images.kpack.io tdp-workload -o jsonpath={.status.latestIma
ge}
> kapplegate.azurecr.io/demo/workloads/tdp-workload-default@sha256:bae710386f7d81a72
5ce5ab15d76a3dd4f6ea79804ae0a475cf98f5e3dd6cf82
```

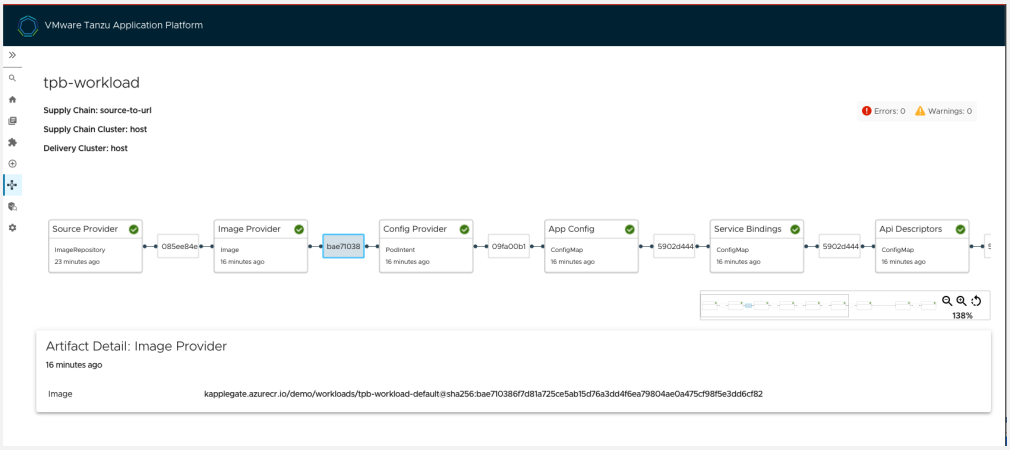
Tanzu Developer Portal GUI

Within the Tanzu Developer Portal GUI:

1. Select your workload in the Supply Chain view.



2. Record the Artifact Detail image value you see in the detailed view of the workload.



Prepare to overlay your customized image onto the currently running instance

To prepare to overlay your customized image onto the currently running instance:

1. Create the ytt overlay secret.
2. Create a file called `tdp-overlay-secret.yaml` with the following content to replace `IMAGE-REFERENCE` with the customized image you retrieved earlier. There is one of two possible values. The value depends on whether you installed Tanzu Application Platform with the `lite` dependencies (default) or the `full` dependencies.

Content for lite

This is the content for `tdp-overlay-secret.yaml` if you installed the `lite` dependencies (default):

```
apiVersion: v1
kind: Secret
metadata:
  name: tdp-app-image-overlay-secret
  namespace: tap-install
stringData:
  tdp-app-image-overlay.yaml: |
    #@ load("@ytt:overlay", "overlay")

    #! makes an assumption that tap-gui is deployed in the namespace: "tap-gu
i"
    #@overlay/match by=overlay.subset({"kind": "Deployment", "metadata": {"na
me": "server", "namespace": "tap-gui"}}), expects="1+"
    ---
    spec:
      template:
        spec:
          containers:
            #@overlay/match by=overlay.subset({"name": "backstage"}),expects
="1+"
            #@overlay/match-child-defaults missing_ok=True
            - image: IMAGE-REFERENCE
            #@overlay/replace
              args:
              - -c
              - |
                export KUBERNETES_SERVICE_ACCOUNT_TOKEN="$(cat /var/run/secre
ts/kubernetes.io/serviceaccount/token)"
                exec /layers/tanzu-buildpacks_node-engine-lite/node/bin/node
portal/dist/packages/backend  \
                --config=portal/app-config.yaml \
                --config=portal/runtime-config.yaml \
                --config=/etc/app-config/app-config.yaml
```

Content for full

This is the content for `tdp-overlay-secret.yaml` if you installed the `full` dependencies:

```
apiVersion: v1
kind: Secret
metadata:
  name: tdp-app-image-overlay-secret
  namespace: tap-install
stringData:
  tdp-app-image-overlay.yaml: |
    #@ load("@ytt:overlay", "overlay")

    #! makes an assumption that tap-gui is deployed in the namespace: "tap-gu
i"
    #@overlay/match by=overlay.subset({"kind": "Deployment", "metadata": {"na
me": "server", "namespace": "tap-gui"}}), expects="1+"
    ---
    spec:
      template:
        spec:
          containers:
            #@overlay/match by=overlay.subset({"name": "backstage"}),expects
="1+"
            #@overlay/match-child-defaults missing_ok=True
            - image: IMAGE-REFERENCE
            #@overlay/replace
              args:
              - -c
```

```
              - |
                export KUBERNETES_SERVICE_ACCOUNT_TOKEN="$(cat /var/run/secre
ts/kubernetes.io/serviceaccount/token)"
                exec /layers/tanzu-buildpacks_node-engine/node/bin/node porta
l/dist/packages/backend  \
                --config=portal/app-config.yaml \
                --config=portal/runtime-config.yaml \
                --config=/etc/app-config/app-config.yaml
```

3. Apply the secret by running:

```
kubectl apply -f tdp-overlay-secret.yaml
```

4. Add the secret to `tap-values.yaml`, the file used to install Tanzu Application Platform. For example:

```
profile: full
tap_gui:
  ...
package_overlays:
- name: tap-gui
  secrets:
  - name: tdp-app-image-overlay-secret
```

5. Update your installation to use the modified `tap-values.yaml` file. The exact steps vary depending on your installation method (GitOps, online install, or offline install).

   For how to do so for an online installation, see Install your Tanzu Application Platform package.

Important

The cluster that is running Tanzu Developer Portal (formerly known as `TAP-GUI`) needs the relevant credentials to pull images from the container registry that contains your customized portal.

If the cluster is in a `full` profile, it likely has the necessary credentials. In a multicluster installation (`view` profile) you likely need to use the Tanzu CLI to add the relevant credentials.

Troubleshoot Tanzu Developer Portal Configurator

This topic helps you troubleshoot Tanzu Developer Portal Configurator.

No supply chain found in `tdp-workload.yaml`

Symptom

No supply chain is found in `tdp-workload.yaml`

Cause

You might not have specified all of the correct information regarding your workload if you're using a supply chain other than `basic`. This documentation has the assumption that you're running the `basic` supply chain, so it doesn't tell you to add any tests. Because supply chains are configurable, there might be stages unique to your configuration.

Solution

If you have a supply chain that requires testing, use a no-op set of tests to make the workload pass through. For more information about no-op, see Wikipedia.

Add the following test YAML to your developer namespace:

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  labels:
    apps.tanzu.vmware.com/pipeline: noop-pipeline
  name: developer-defined-noop-tekton-pipeline
  namespace: DEVELOPER-NAMESPACE
spec:
  params:
  - name: source-url
    type: string
  - name: source-revision
    type: string
  tasks:
  - name: noop-task
    params:
    - name: source-url
      value: $(params.source-url)
    - name: source-revision
      value: $(params.source-revision)
    taskSpec:
      metadata: {}
      params:
      - name: source-url
        type: string
      - name: source-revision
        type: string
      steps:
      - image: bash
        name: noop
        resources: {}
        script: |
          echo "Nothing to do for $(params.source-url)/$(params.source-revision)"%
```

Where `DEVELOPER-NAMESPACE` is an appropriately configured developer namespace on the cluster

Add the following YAML to your workload definition:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  labels:
    # Existing labels
    apps.tanzu.vmware.com/has-tests: 'true'
spec:
  # Existing parameters
  params:
    - name: testing_pipeline_matching_labels
      value:
        apps.tanzu.vmware.com/pipeline: noop-pipeline
    - name: testing_pipeline_params
      value: {}
```

Now when you submit your workload, the testing stage finishes.

Convention controller fails frequently

Symptom

Convention controller fails frequently and prevents other workloads from completing.

Cause

The frequent failures are caused by the large memory limit of the Software Bill of Materials (SBOM). The Tanzu Developer Portal Configurator build process produces this SBOM, and `PodIntent` reports an out-of-memory error.

Solution

Create an overlay to increase the memory limit of `PodIntent` to 512 MB. To learn how to do so, see Troubleshoot Cartographer Conventions.

Using surfaces

This topic gives you small basic examples of how to use the various surfaces when creating a Tanzu Developer Portal plug-in.

BackendPluginSurface

Use `BackendPluginSurface` if you must add routes to the `apiRouter` commonly found in `packages/backend/src/index.ts`.

Example:

```
import {
  createRouter,
  buildTechInsightsContext,
  createFactRetrieverRegistration,
  entityOwnershipFactRetriever,
  entityMetadataFactRetriever,
  techdocsFactRetriever,
} from '@backstage/plugin-tech-insights-backend';
import { Router } from 'express';
import {
  JsonRulesEngineFactCheckerFactory,
  JSON_RULE_ENGINE_CHECK_TYPE,
} from '@backstage/plugin-tech-insights-backend-module-jsonfc';
import {
  BackendPluginInterface,
  BackendPluginSurface,
  PluginEnvironment,
} from '@vmware-tanzu/core-backend';

const ttlTwoWeeks = { timeToLive: { weeks: 2 } };

export default async function createPlugin(
        env: PluginEnvironment,
): Promise<Router> {
  const techInsightsContext = await buildTechInsightsContext({
    logger: env.logger,
    config: env.config,
    database: env.database,
    discovery: env.discovery,
    tokenManager: env.tokenManager,
    scheduler: env.scheduler,
    factRetrievers: [
      createFactRetrieverRegistration({
        cadence: '0 */6 * * *', // Run every 6 hours - https://crontab.guru/#0_*/6_*_*
_*
```

```
        factRetriever: entityOwnershipFactRetriever,
        lifecycle: ttlTwoWeeks,
      }),
      createFactRetrieverRegistration({
        cadence: '0 */6 * * *',
        factRetriever: entityMetadataFactRetriever,
        lifecycle: ttlTwoWeeks,
      }),
      createFactRetrieverRegistration({
        cadence: '0 */6 * * *',
        factRetriever: techdocsFactRetriever,
        lifecycle: ttlTwoWeeks,
      }),
    ],
    factCheckerFactory: new JsonRulesEngineFactCheckerFactory({
      logger: env.logger,
      checks: [
        {
          id: 'groupOwnerCheck',
          type: JSON_RULE_ENGINE_CHECK_TYPE,
          name: 'Group Owner Check',
          description:
                  'Verifies that a Group has been set as the owner for this entity',
          factIds: ['entityOwnershipFactRetriever'],
          rule: {
            conditions: {
              all: [
                {
                  fact: 'hasGroupOwner',
                  operator: 'equal',
                  value: true,
                },
              ],
            },
          },
        },
        {
          id: 'titleCheck',
          type: JSON_RULE_ENGINE_CHECK_TYPE,
          name: 'Title Check',
          description:
                  'Verifies that a Title, used to improve readability, has been set fo
r this entity',
          factIds: ['entityMetadataFactRetriever'],
          rule: {
            conditions: {
              all: [
                {
                  fact: 'hasTitle',
                  operator: 'equal',
                  value: true,
                },
              ],
            },
          },
        },
        {
          id: 'techDocsCheck',
          type: JSON_RULE_ENGINE_CHECK_TYPE,
          name: 'TechDocs Check',
          description:
                  'Verifies that TechDocs has been enabled for this entity',
          factIds: ['techdocsFactRetriever'],
          rule: {
            conditions: {
              all: [
```

```
                {
                  fact: 'hasAnnotationBackstageIoTechdocsRef',
                  operator: 'equal',
                  value: true,
                },
              ],
            },
          },
        },
      ],
    }),
  });

  return await createRouter({
    ...techInsightsContext,
    logger: env.logger,
    config: env.config,
  });
}

export const TechInsightsBackendPlugin: BackendPluginInterface =
  () => surfaces =>
    surfaces.applyTo(BackendPluginSurface, backendPluginSurface => {
      backendPluginSurface.addPlugin({
        name: 'tech-insights',
        pluginFn: createPlugin,
      });
    });
```

BannerSurface

`BannerSurface` enables you to add to the DOM at the top of the page before the content, which is commonly used for banners. The only elements that can appear before the banners in the DOM are `AlertDisplay` and `OAuthRequestDialog`. You can add multiple banners to the surface. The multiple banners are rendered in order of registration.

Example:

```
import React from 'react';
import {
  AppPluginInterface,
  BannerSurface,
} from '@vmware-tanzu/core-frontend';

export const HelloWorldPlugin: AppPluginInterface = () => context => {
  context.applyTo(BannerSurface, banners => {
    banners.add(<div>Hello World Banner</div>);
  });
};
```

EntityPageSurface

Use `EntityPageSurface` if you must add to the `serviceEntityPage` constant found in `packages/app/src/components/catalog/EntityPage.tsx`.

Example:

```
import { EntityLayout } from '@backstage/plugin-catalog';
import { EntityTechInsightsScorecardContent } from '@backstage/plugin-tech-insights';
import {
  AppPluginInterface,
  SurfaceStoreInterface,
```

```
  EntityPageSurface,
} from '@vmware-tanzu/core-frontend';
import React from 'react';

export const TechInsightsFrontendPlugin: AppPluginInterface =
  () => (context: SurfaceStoreInterface) => {
    context.applyTo(
      EntityPageSurface,
      (entityPageSurface) => {
        entityPageSurface.servicePage.addTab(
          <EntityLayout.Route path="/techinsights" title="TechInsights">
            <EntityTechInsightsScorecardContent
              title="TechInsights Scorecard."
              description="TechInsight's default fact-checkers"
            />
          </EntityLayout.Route>,
        );
      },
    );
  };
```

SettingsTabsSurface

Use `SettingsTabsSurface` if you want to add child routes to the `/settings` route defined in the `route` constant of `packages/app/src/App.tsx`. This action also adds a tab to the settings page so that you can use the UI to access your new route.

Example:

```
import React from 'react';
import {
  AppPluginInterface,
  SettingsTabsSurface,
} from '@vmware-tanzu/core-frontend';
import { SettingsLayout } from '@backstage/plugin-user-settings';

export const HelloWorldPlugin: AppPluginInterface = () => context => {
  context.applyTo(SettingsTabsSurface, tabs =>
    tabs.add(
      <SettingsLayout.Route path="/hello-world" title="Hello World Tab">
        <div>Hello World Settings Tab Content</div>
      </SettingsLayout.Route>,
    ),
  );
};
```

SidebarItemSurface

`SidebarItemSurface` enables you to render links in the sidebar to quickly access your plug-in. `SidebarItemsSurface` enables you to change the content of the `root` constant in `packages/app/src/components/Root/Root.tsx`. All items you add to the sidebar appear at the bottom of existing sidebar items, in the order that they were registered.

Example:

```
import React from 'react';
import { SidebarItem } from '@backstage/core-components';
import {
  AppPluginInterface,
  SidebarItemSurface,
} from '@vmware-tanzu/core-frontend';
```

```
import AlarmIcon from '@material-ui/icons/Alarm';

export const HelloWorldPlugin: AppPluginInterface = () => context => {
  context.applyTo(SidebarItemSurface, sidebar =>
    sidebar.addMainItem(
      <SidebarItem icon={AlarmIcon} to="hello-world" text="Hello" />,
    ),
  );
};
```

API documentation for surfaces

This topic tells you about API references for surfaces in Tanzu Developer Portal.

The following packages are described:

- @tpb/plugin-catalog
- @vmware-tanzu/core-backend
- @vmware-tanzu/core-frontend
- @vmware-tanzu/tdp-plugin-auth-backend
- @vmware-tanzu/tdp-plugin-custom-logger
- @vmware-tanzu/tdp-plugin-home
- @vmware-tanzu/tdp-plugin-ldap-backend
- @vmware-tanzu/tdp-plugin-login
- @vmware-tanzu/tdp-plugin-microsoft-graph-org-reader-processor
- @vmware-tanzu/tdp-plugin-permission-backend

Package `@tpb/plugin-catalog`

The following sections describe the package.

Class `EntityPageSurface`

This class can manipulate the contents of pages for Backstage catalog entities.

Each catalog entity subtype can specify a dedicated template. This surface standardizes the contents of those templates and allows for some amount of external configuration.

This example adds a tab to the template for Backstage Service entities:

```
context.applyWithDependency(
  AppRouteSurface,
  EntityPageSurface,
  (_, surface) => {
       surface.servicePage.addTab(
         <EntityLayout.Route path="/some-subpath" title="Additional content for Servic
es">
           <MyAdditionalContent />
         </EntityLayout.Route>,
       );
     },
  );
```

Properties

This table describes the properties.

Property	Type	Description
apiPage	ApiPage	
componentPageCases	ReactElement[]	
defaultPage	BasicPage	
domainPage	BasicPage	
groupPage	BasicPage	
packagePage	BasicPage	
servicePage	BasicPage	
systemPage	BasicPage	
userPage	BasicPage	
websitePage	BasicPage	

Constructors

The package has the following constructor.

constructor()

This constructor constructs a new instance of the `EntityPageSurface` class.

Methods

The package has the following methods.

`addComponentPageCase(pageCase: ReactElement)`: void

This table describes the parameters.

Parameter	Type	Description
pageCase	ReactElement	

`addOverviewContent(content: ReactElement)`: void

This table describes the parameters.

Parameter	Type	Description
content	ReactElement	

Package `@vmware-tanzu/core-backend`

The following sections describe the package.

Types

The package has the following types.

```
BackendPluginInterface<T = {}> = (config?: T & BackendPluginConfig) =>
TpbPluginInterface;
```

```
CatalogProcessorBuilder = (env: PluginEnvironment) => CatalogProcessor[] |
CatalogProcessor;

EntityProviderBuilder = (env: PluginEnvironment) => EntityProvider[] | EntityProvider;

PermissionRuleBuilder = (env: PluginEnvironment) => CatalogPermissionRule[] |
CatalogPermissionRule;

PluginFn = (env: PluginEnvironment) => Promise<Router>;

RouterBuilder = (env: PluginEnvironment) => Promise<Router>;
```

Class `BackendCatalogSurface`

This class can manipulate the catalog backend.

This class provides extension points for registering catalog processors, entity providers, routes, and permissions.

Constructors

The package has the following constructor.

constructor()

This constructor constructs a new instance of the `BackendCatalogSurface` class.

Methods

The package has the following methods.

`addCatalogProcessorBuilder(builder: CatalogProcessorBuilder)`: void

This method registers a Backstage `CatalogProcessor` that applies transformations to unprocessed entities.

For more information about processors, see the Backstage documentation.

For more information about processing, see the Backstage documentation.

Parameter	Type	Description
`builder`	CatalogProcessorBuilder	Builds a `CatalogProcessor` for a given `PluginEnvironment`.

`addEntityProviderBuilder(builder: EntityProviderBuilder)`: void

This method registers a Backstage `EntityProvider` to ingest generate catalog entries for further processing.

For more information about ingestion, see the Backstage documentation.

Parameter	Type	Description
`builder`	EntityProviderBuilder	Builds an `EntityProvider` for a given `PluginEnvironment`.

`addPermissionRuleBuilder(builder: PermissionRuleBuilder)`: void

This method registers Backstage `CatalogPermissionRules` to limit access to catalog entries.

For more information about custom permission rules, see the Backstage documentation.

Parameter	Type	Description
`builder`	PermissionRuleBuilder	Builds `CatalogPermissionRules` for a given `PluginEnvironment`.

`addRouterBuilder(routerBuilder: RouterBuilder)`: void

This method registers an Express Router to handle arbitrary requests made to the Backstage backend.

Parameter	Type	Description
`routerBuilder`	RouterBuilder	Builds an Express Router for a given `PluginEnvironment`.

Class `BackendPluginSurface`

You can use this class to add plug-ins to the Backstage backend.

Constructors

The package has the following constructor.

constructor()

This constructor constructs a new instance of the `BackendPluginSurface` class.

Methods

The package has the following methods.

`addPlugin(plugin: Plugin)`: void

This method registers a named plug-in to handle backend requests at a subpath. Plug-ins must have unique names. The first plug-in registered with a given name wins.

Parameter	Type	Description
`plugin`	Plugin	Plug-in to add

Package `@vmware-tanzu/core-frontend`

The following sections describe the package.

Types

The package has the following type.

```
AppPluginInterface<T = {}> = (config?: T) => TpbPluginInterface;
```

Class `ApiSurface`

You can use this class to add Backstage `ApiFactories`.

Methods

The package has the following methods.

`add(factory: AnyApiFactory)`: void

This method registers a new Backstage `ApiFactory`. Factories must specify an API with a unique ID. The last factory registered with a given ID wins. Factories that specify APIs with IDs on the ignored list will not be registered.

Parameter	Type	Description
`factory`	`AnyApiFactory`	

Class `AppComponentSurface`

You can use this class to replace core components of the Backstage app.

Constructors

The package has the following constructor.

constructor()

This constructor constructs a new instance of the `AppComponentSurface` class.

Methods

The package has the following methods.

`add&lt;K extends keyof AppComponents&gt;(key: K, component: AppComponents\[K\])`: void

This method replaces a given Backstage `AppComponent` with another. `AppComponents` must be unique by key. The last `AppComponent` registered with a given key wins.

Parameter	Type	Description
`key`		Unique key for the `AppComponent`.
`component`	`AppComponents`	Custom `AppComponent` to substitute for Backstage's default.

Class `AppPluginSurface`

You can use this class to register additional frontend plug-ins with the Backstage app.

https://backstage.io/docs/plugins/

Constructors

The package has the following constructor.

constructor()

This constructor constructs a new instance of the `AppPluginSurface` class.

Methods

The package has the following methods.

`add(plugin: BackstagePlugin)`: void

This method registers a frontend `BackstagePlugin`.

Parameter	Type	Description
`plugin`	`BackstagePlugin`	

Class `AppRouteSurface`

Manipulate routes in the frontend Backstage app.

https://backstage.io/docs/plugins/composability/#routing-system

Constructors

The package has the following constructor.

constructor()

Constructs a new instance of the `AppRouteSurface` class

Methods

The package has the following methods.

`add(route: ReactElement)`: void

This method registers a react-router route.

Parameter	Type	Description
`route`	`ReactElement`	Route to register. Intended to be a react-router#Route or a subclass of it.

`addRouteBinder(routeBinder: RouteBinder)`: void

This method binds external routes. For more information about binding external routes to apps, see the Backstage documentation.

Parameter	Type	Description
`routeBinder`	`RouteBinder`	

Class `BannerSurface`

You can use this class to add page-level banners to the Backstage app.

Constructors

The package has the following constructor.

constructor()

This constructor constructs a new instance of the `BannerSurface` class.

Methods

The package has the following methods.

`add(banner: ReactElement)`: void

This method adds a new page-level banner.

Parameter	Type	Description
`banner`	`ReactElement`	

Class `SettingsTabsSurface`

You can use this class to add tabs to the Settings page.

Constructors

The package has the following constructor.

constructor()

This constructor constructs a new instance of the `SettingsTabsSurface` class.

Methods

The package has the following methods.

`add(tab: ReactElement)`: void

This method adds a new tab to the **Settings** page.

Parameter	Type	Description
`tab`	`ReactElement`	Typically an instance of @backstage/@plugin-user-settings#SettingsLayout.Route

Class `SidebarItemSurface`

You can use this class to manipulate entries in sidebar.

Entries are split into two sections: top items appear above a divider and main items appear below it. Tabs within each section appear in order of registration.

Constructors

The package has the following constructor.

constructor()

This constructor constructs a new instance of the `SidebarItemSurface` class.

Methods

The package has the following methods.

`addMainItem(item: ReactElement)`: void

This method adds an item to the main items section.

Parameter	Type	Description
`item`	`ReactElement`	

`addTopItem(item: ReactElement)`: void

This method adds an item to the top items section.

Parameter	Type	Description
item	ReactElement	

Class ThemeSurface

You can use this class to manipulate themes available to the Backstage application.

Constructors

The package has the following constructor.

constructor()

This constructor constructs a new instance of the ThemeSurface class.

Methods

The package has the following methods.

addTheme(theme: AppTheme): void

This method registers an additional theme.

Parameter	Type	Description
theme	AppTheme	

setRootBuilder(builder: (children: JSX.Element) => ReactElement): void

This method registers the factory that renders the application's root element.

Parameter	Type	Description
builder	JSX.ElementReactElement	

Package @vmware-tanzu/tdp-plugin-auth-backend

The following sections describe the package.

Class SignInProviderSurface

You can use this class to register additional Auth Providers.

https://backstage.io/docs/auth/add-auth-provider

Methods

The package has the following methods.

add(signInProvider: SignInProvider): void

This method registers an additional Auth Provider. The Auth Provider named last wins.

Parameter	Type	Description
signInProvider	SignInProvider	Map of @backstage/plugin-auth-backend#AuthProviderFactory instances.

Class SignInResolverSurface

You can use this class to register SignInResolvers and provide utility methods to log in with them.

Methods

The package has the following methods.

add(authProviderKey: string, resolver: SignInResolver<any>): void

This method registers a new SignInResolver.

Parameter	Type	Description
authProviderKey		Name resolver to add. Uniqueness of name is not enforced, but the behavior of duplicate named entries is undefined.
resolver	SignInResolver	Resolver to add.

getResolver<TAuthResult>(authProviderKey: string): SignInResolver<TAuthResult>

This method gets the first named provider, or falls back to the guest resolver.

Parameter	Type	Description
authProviderKey		

signInAsGuestResolver<TAuthResult>(): SignInResolver<TAuthResult>

This method gets a resolver that creates an ad-hoc guest user.

signInWithEmail(email: string, context: AuthResolverContext): Promise<BackstageSignInResult>

This method provides a utility function to sign in as a user by email address.

The method attempts to match the given email address with a catalog user's email address for sign-in. If that fails, the method signs the user in without associating with a catalog user.

Parameter	Type	Description
email		Email address to attempt sign-in with.
context	AuthResolverContext	

signInWithName(name: string, context: AuthResolverContext): Promise<BackstageSignInResult>

This method provides a utility function to sign in as a user by name.

The method attempts to match the name with a catalog user for signing in. If that fails, the method signs the user in with that name without associating with a catalog user.

Parameter	Type	Description
name		Username to attempt sign-in with.

Parameter	Type	Description
context	AuthResolverContext	

Package @vmware-tanzu/tdp-plugin-custom-logger

The following sections describe the package.

Class LoggerOptionsSurface

You can use this class to allow for configuration of the Backstage logger.

Constructors

The package has the following constructor.

constructor()

This constructor constructs a new instance of the LoggerOptionsSurface class.

Methods

The package has the following methods.

setLoggerOptions(loggerOptions: LoggerOptions): void

This method sets options for the Backstage logger. This method can be invoked multiple times. The last invocation wins.

Parameter	Type	Description
loggerOptions	LoggerOptions	

Package @vmware-tanzu/tdp-plugin-home

The following sections describe the package.

Class HomeSurface

You can use this class to add content to the Backstage Home screen.

Constructors

The package has the following constructor.

constructor()

This constructor constructs a new instance of the HomeSurface class.

Methods

The package has the following methods.

addContent(item: ReactElement): void

This method adds content to the main section below the widget grid.

Parameter	Type	Description
`item`	`ReactElement`	Item to add.

`addWidget(item: ReactElement, config?: LayoutConfiguration)`: void

This method adds a widget to the Home screen's widget grid.

Parameter	Type	Description
`item`	`ReactElement`	Widget to add.
`config`	`LayoutConfiguration`	(Optional) layout configuration for the widget. The `config.component` field must match the added widget either by name (if provided as string) or by direct reference (if provided as a React element).

`addWidgetConfig(config: LayoutConfiguration)`: void

This method specifies the layout configuration for a widget.

Parameter	Type	Description
`config`	`LayoutConfiguration`	Optional layout configuration for a widget. The `config.component` field must match some registered widget either by name (if provided as string) or by direct reference (if provided as a React element).

Package `@vmware-tanzu/tdp-plugin-ldap-backend`

The following sections describe the package.

Class `LdapSurface`

You can use this class to allow for manipulation of LDAP groups and users during ingestion.

Constructors

The package has the following constructor.

constructor()

This constructor constructs a new instance of the `LdapSurface` class.

Methods

The package has the following methods.

`setGroupTransformerBuilder(builder: GroupTransformerBuilder)`: void

This method specifies the transformation of LDAP groups during ingestion.

Parameter	Type	Description
`builder`	`GroupTransformerBuilder`	

`setUserTransformerBuilder(builder: UserTransformerBuilder)`: void

This method specifies the transformation of LDAP users during ingestion.

Parameter	Type	Description
builder	UserTransformerBuilder	

Package @vmware-tanzu/tdp-plugin-login

The following sections describe the package.

Class LoginSurface

You can use this class to register new login providers with Backstage's SignIn page.

For more information about sign-in configuration, see the Backstage documentation.

Methods

The package has the following methods.

add(provider: Provider): void

This method registers a login provider.

Parameter	Type	Description
provider	Provider	

Package @vmware-tanzu/tdp-plugin-microsoft-graph-org-reader-processor

The following sections describe the package.

Class MicrosoftGraphOrgReaderProcessorTransformersSurface

You can use this class to allow for manipulation of Microsoft Graph groups and users during ingestion.

Methods

The package has the following methods.

setGroupTransformer(groupTransformer: GroupTransformer): void

This method specifies the transformation of Microsoft Graph groups during ingestion.

Parameter	Type	Description
groupTransformer	GroupTransformer	

setOrganizationTransformer(organizationTransformer: OrganizationTransformer): void

This method specifies the transformation of Microsoft Graph organizations during ingestion.

Parameter	Type	Description
organizationTransformer	OrganizationTransformer	

setUserTransformer(userTransformer: UserTransformer): void

This method specifies the transformation of Microsoft Graph users during ingestion.

Parameter	Type	Description
`userTransformer`	`UserTransformer`	

Package `@vmware-tanzu/tdp-plugin-permission-backend`

The following sections describe the package.

Class `CustomPermissionPolicy`

Methods

The package has the following methods.

`handle(request: PolicyQuery): Promise<PolicyDecision>`

Parameter	Type	Description
`request`	`PolicyQuery`	

Class `PermissionPolicySurface`

You can use this class to define a `PermissionPolicy` for the Backstage application.

https://backstage.io/docs/permissions/writing-a-policy

Methods

The package has the following methods.

`set(permissionPolicy: PermissionPolicy)`: void

This method defines the `PermissionPolicy` for the Backstage application. This method can be invoked only once.

Parameter	Type	Description
`permissionPolicy`	`PermissionPolicy`	

Dependency version reference

When using any external dependencies with Tanzu Developer Portal, you must verify that the versions are compatible. The following are references for some external dependencies that you might want to use.

External Tanzu Developer Portal plug-ins compatibility

The following are Tanzu Developer Portal plug-ins that are not included by default. To use these with Configurator, you must choose a compatible version based on your version of Tanzu Application Platform.

Tanzu Application Platform v1.9.1 offers the following support:

Plug-in package name	Compatible npm version
`@vmware-tanzu/tdp-plugin-auth-backend`	2.0.0
`@vmware-tanzu/tdp-plugin-backstage-grafana`	2.0.0
`@vmware-tanzu/tdp-plugin-backstage-jira`	2.0.0
`@vmware-tanzu/tdp-plugin-backstage-sonarqube`	2.0.0
`@vmware-tanzu/tdp-plugin-backstage-sonarqube-backend`	2.0.0
`@vmware-tanzu/tdp-plugin-custom-logger`	2.0.0
`@vmware-tanzu/tdp-plugin-github-actions`	2.0.0
`@vmware-tanzu/tdp-plugin-home`	2.0.0
`@vmware-tanzu/tdp-plugin-ldap-backend`	2.0.0
`@vmware-tanzu/tdp-plugin-login`	2.0.0
`@vmware-tanzu/tdp-plugin-microsoft-graph-org-reader-processor`	2.0.0
`@vmware-tanzu/tdp-plugin-permission-backend`	2.0.0
`@vmware-tanzu/tdp-plugin-prometheus`	2.0.0
`@vmware-tanzu/tdp-plugin-rbac`	2.0.0
`@vmware-tanzu/tdp-plugin-rbac-backend`	2.0.0
`@vmware-tanzu/tdp-plugin-snyk`	2.0.0
`@vmware-tanzu/tdp-plugin-stack-overflow`	2.0.0
`@vmware-tanzu/tdp-plugin-techinsights`	2.0.0
`@vmware-tanzu/tdp-plugin-techinsights-backend`	2.0.1

Tanzu Developer Portal plug-in libraries compatibility

The following are libraries used to create Tanzu Developer Portal plug-ins.

When using these libraries, you must choose a compatible version based on your version of Tanzu Application Platform.

Tanzu Application Platform v1.9.1 offers the following support:

Library package name	Compatible npm version
`@vmware-tanzu/core-backend`	2.0.1
`@vmware-tanzu/core-common`	2.0.0
`@vmware-tanzu/core-frontend`	2.0.0

Backstage version compatibility

Tanzu Application Platform v1.9.1 is compatible with the following Backstage version. Developers must verify that a dependency used by Backstage is compatible with your current Tanzu Application Platform installation. When considering upgrading Tanzu Application Platform, read the relevant documentation to see if compatibility would change after upgrading.

Tanzu Application Platform version	Backstage version	Dependencies manifest
1.8.x	v1.20.3	Manifest file

Overview of Tanzu Developer Portal plug-ins

This topic gives you an overview of the different plug-in types that Tanzu Developer Portal supports. Some plug-ins are already integrated with Tanzu Developer Portal. Other plug-ins require you to use Configurator to integrate them.

Tanzu Developer Portal plug-ins

A Tanzu Developer Portal plug-in is a Backstage plug-in inside a wrapper. The wrapper makes it possible to dynamically integrate a Backstage plug-in into Tanzu Developer Portal. The wrapper defines integration points with the underlying Backstage instance to remove the need to manually update source code when adding a Backstage plug-in.

Tanzu Application Platform plug-ins

Tanzu Application Platform includes some pre-built Tanzu Developer plug-ins. These Tanzu Application Platform plug-ins are already integrated with Tanzu Developer Portal. You don't need to configure these plug-ins. To use a Tanzu Application Platform plug-in, you must install the relevant Tanzu Application Platform component.

Tanzu Application Platform has the following Tanzu Developer Portal plug-ins:

- Runtime Resources Visibility
- Application Live View
- Application Accelerator
- API Documentation
- Supply Chain Choreographer
- Security Analysis
- DORA Metrics

Backstage plug-ins

Backstage plug-ins are developed by Backstage and are in the `@backstage` namespace.

Community plug-ins

Community plug-ins are not developed by Backstage or VMware. These plug-ins are not in the `@backstage` namespace.

Validated plug-ins

Validated plug-ins are Backstage plug-ins or community plug-ins that VMware has validated for use with Tanzu Developer Portal. You don't need to create custom wrappers to integrate these plug-ins with Tanzu Developer Portal.

For more information, see the validated community plug-ins section.

External plug-ins

External plug-ins (also referred to as custom plug-ins) are plug-ins that are not in Tanzu Developer Portal Configurator.

These plug-ins are typically published to npmjs.org or a similar npm registry alternative. If you want to integrate an external plug-in that is not validated for use with Tanzu Developer Portal, you must write a custom wrapper for it.

Overview of Tanzu Developer Portal plug-ins

This topic gives you an overview of the different plug-in types that Tanzu Developer Portal supports. Some plug-ins are already integrated with Tanzu Developer Portal. Other plug-ins require you to use Configurator to integrate them.

Tanzu Developer Portal plug-ins

A Tanzu Developer Portal plug-in is a Backstage plug-in inside a wrapper. The wrapper makes it possible to dynamically integrate a Backstage plug-in into Tanzu Developer Portal. The wrapper defines integration points with the underlying Backstage instance to remove the need to manually update source code when adding a Backstage plug-in.

Tanzu Application Platform plug-ins

Tanzu Application Platform includes some pre-built Tanzu Developer plug-ins. These Tanzu Application Platform plug-ins are already integrated with Tanzu Developer Portal. You don't need to configure these plug-ins. To use a Tanzu Application Platform plug-in, you must install the relevant Tanzu Application Platform component.

Tanzu Application Platform has the following Tanzu Developer Portal plug-ins:

- Runtime Resources Visibility
- Application Live View
- Application Accelerator
- API Documentation
- Supply Chain Choreographer
- Security Analysis
- DORA Metrics

Backstage plug-ins

Backstage plug-ins are developed by Backstage and are in the `@backstage` namespace.

Community plug-ins

Community plug-ins are not developed by Backstage or VMware. These plug-ins are not in the `@backstage` namespace.

Validated plug-ins

Validated plug-ins are Backstage plug-ins or community plug-ins that VMware has validated for use with Tanzu Developer Portal. You don't need to create custom wrappers to integrate these plug-ins with Tanzu Developer Portal.

For more information, see the validated community plug-ins section.

External plug-ins

External plug-ins (also referred to as custom plug-ins) are plug-ins that are not in Tanzu Developer Portal Configurator.

These plug-ins are typically published to npmjs.org or a similar npm registry alternative. If you want to integrate an external plug-in that is not validated for use with Tanzu Developer Portal, you must write a custom wrapper for it.

Runtime resources visibility in Tanzu Developer Portal

This topic tells you about runtime resources visibility.

The Runtime Resources Visibility plug-in enables users to visualize their Kubernetes resources associated with their workloads.

Prerequisite

Do one of the following actions to access the Runtime Resources Visibility plug-in:

- Install the Tanzu Application Platform Full or View profile
- Install Tanzu Application Platform without using a profile and then install Tanzu Developer Portal separately
- Review the section If you have a metrics server

If you have a metrics server

By default, the Kubernetes API does not attempt to use any metrics servers on your clusters. To access metrics information for a cluster, set `skipMetricsLookup` to `false` for that cluster in the `kubernetes` section of `app-config.yaml`. Example:

```
tap_gui:
  # ... existing configuration
  app_config:
    # ... existing configuration
    kubernetes:
      clusterLocatorMethods:
        - type: 'config'
          clusters:
            - url: https://KUBERNETES-SERVICE-HOST:KUBERNETES-SERVICE-PORT
              name: host
              authProvider: serviceAccount
              serviceAccountToken: KUBERNETES-SERVICE-ACCOUNT-TOKEN
              skipTLSVerify: true
              skipMetricsLookup: false
```

Where:

- `KUBERNETES-SERVICE-HOST` and `KUBERNETES-SERVICE-PORT` are the URL and ports of your Kubernetes cluster. You can gather these through `kubectl cluster-info`.
- `KUBERNETES-SERVICE-ACCOUNT-TOKEN` is the token from your `tap-gui-token-id`.

You can retrieve this secret's ID by running:

```
kubectl get secrets -n tap-gui
```

and then running

```
kubectl describe secret tap-gui-token-ID
```

Where `ID` is the secret name from the first step.

> **Caution**
>
> If you enable metrics for a cluster but do not have a metrics server running on it, Tanzu Application Platform web interface users see an error notifying them that there is a problem connecting to the back end.

Visualize Workloads on Tanzu Developer Portal

To view your applications on Tanzu Developer Portal, use the following steps:

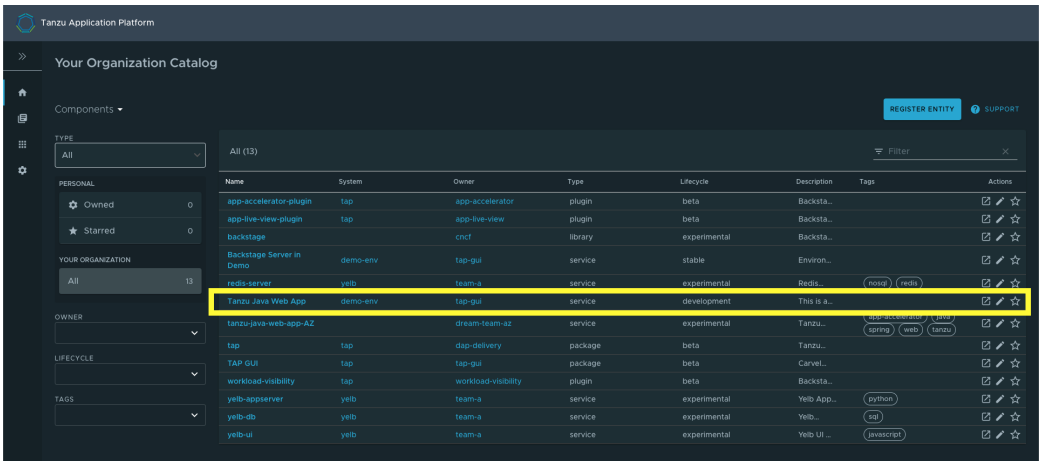
1. Deploy your first application on the Tanzu Application Platform
2. Add your application to Tanzu Developer Portal Software Catalog

Navigate to the **Runtime Resources Visibility** screen

You can view the list of running resources and the details of their status, type, namespace, cluster, and public URL if applicable for the resource type.

To view the list of your running resources:

1. Select your component from the Catalog index page.



2. Select the **Runtime Resources** tab.

Resources

Built-in Kubernetes resources in this view are:

- Services
- Deployments
- ReplicaSets
- Pods
- Jobs
- Cronjobs
- DaemonSets

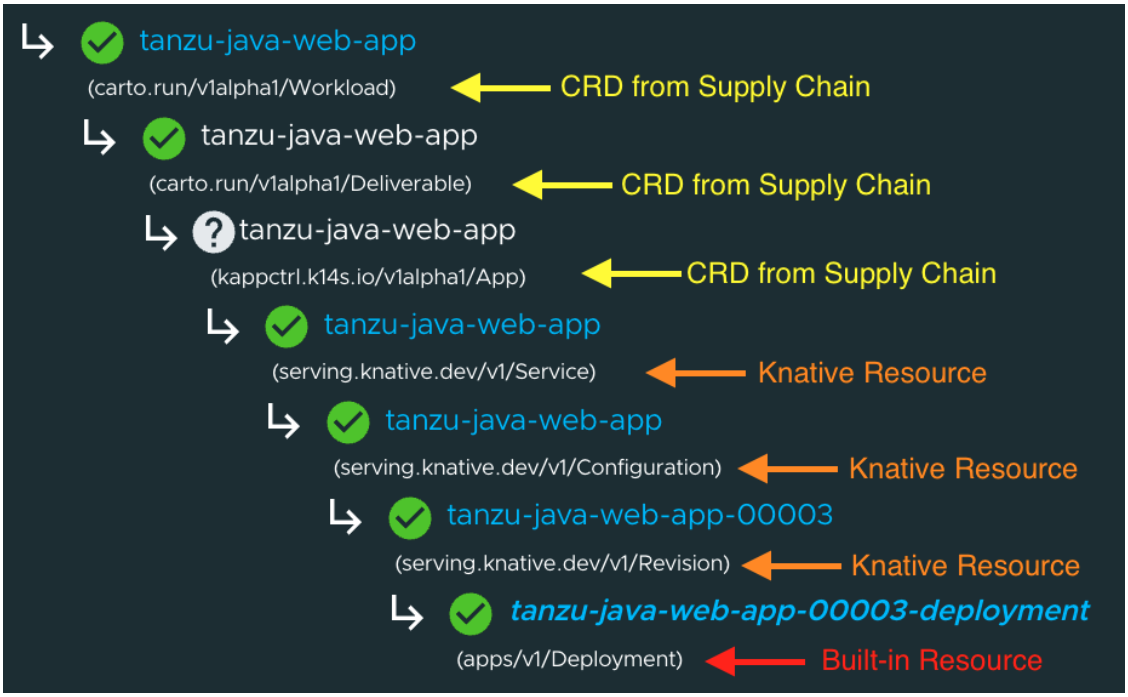
- ReplicaSets

The Runtime Resource Visibility plug-in also displays CRDs created with the Supply Chain, including:

- Cartographer Workloads
- Knative Services, Configurations, Revisions, and Routes

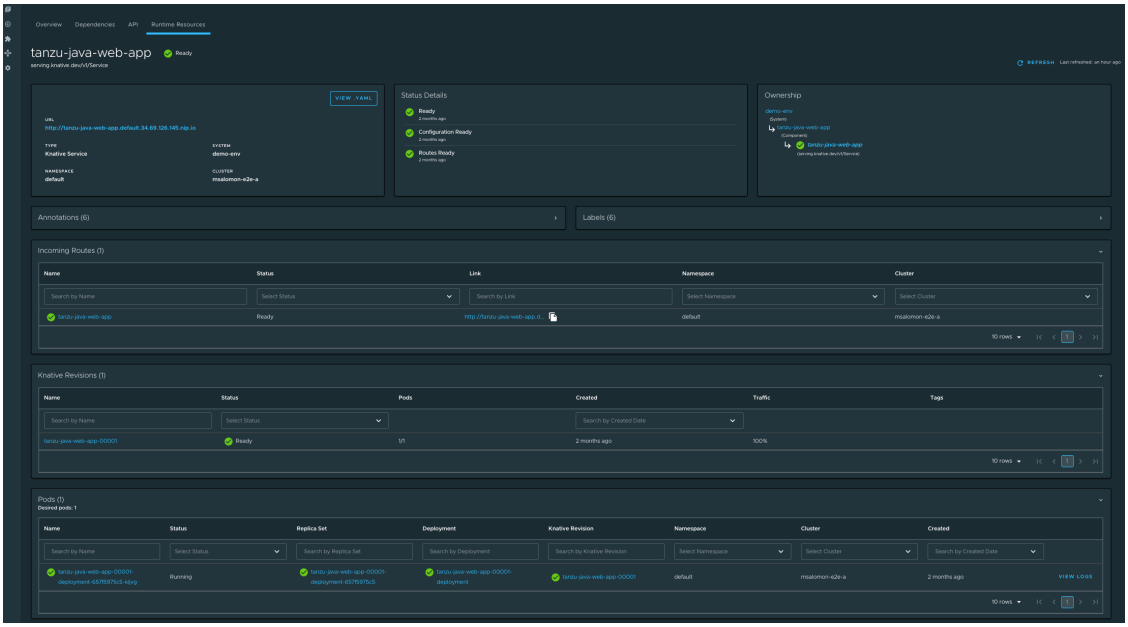
For more information, see Supply Chain Choreographer in Tanzu Developer Portal.

CRDs from Supply Chain are associated with Knative Resources, further down the chain, and built-in resources even further down the chain.



Resources details page

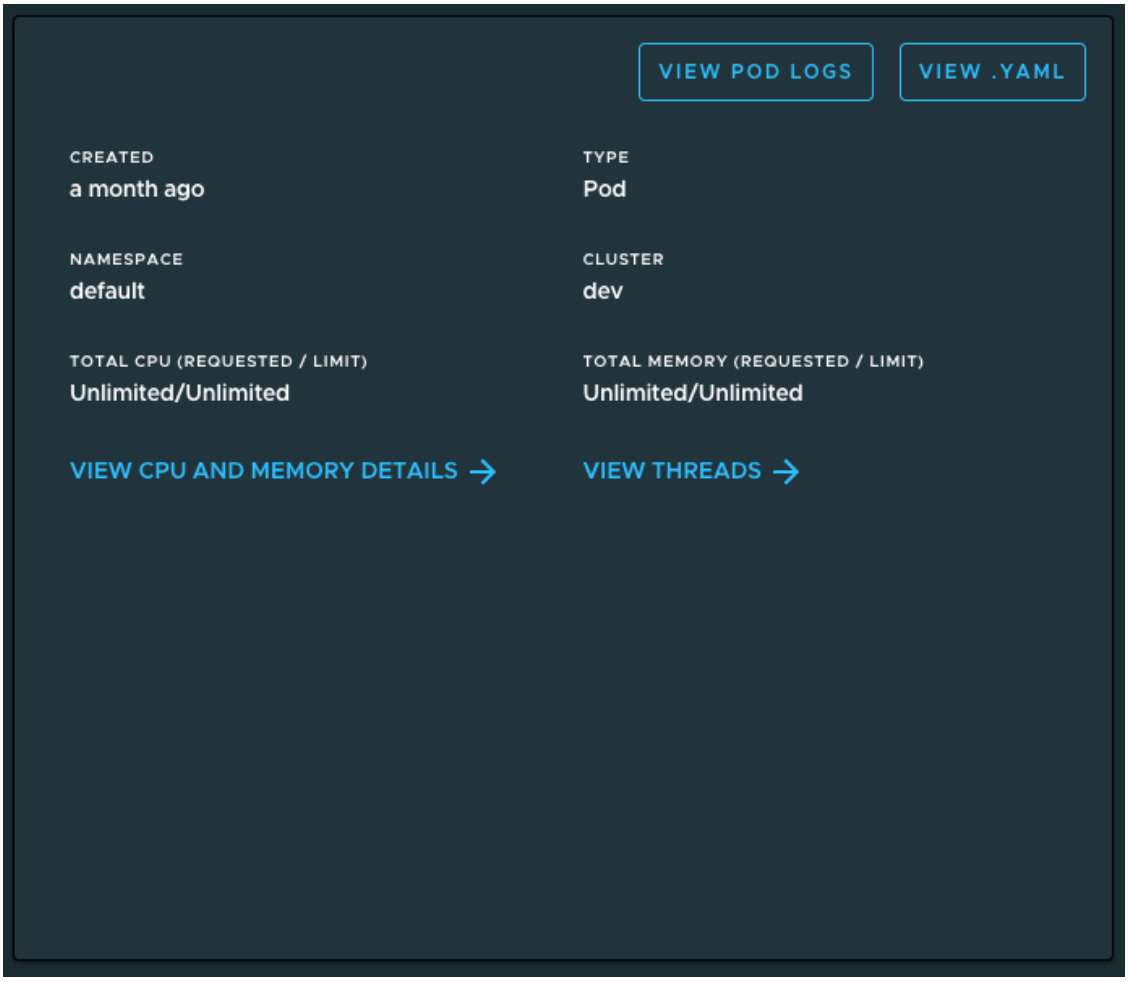
To get more information about a particular workload, select it from the table on the main **Runtime Resources** page to visit a page that provides details about the workload. These details include the workload status, ownership, and resource-specific information.



Overview card

All detail pages provide an overview card with information related to the selected resource. Most of the information feeds from the `metadata` attribute in each object. The following are some attributes that are displayed in the overview card:

- **View Pod Logs** button
- **View .YAML** button
- URL, which is for Knative and Kubernetes service detail pages
- Type
- System
- Namespace
- Cluster



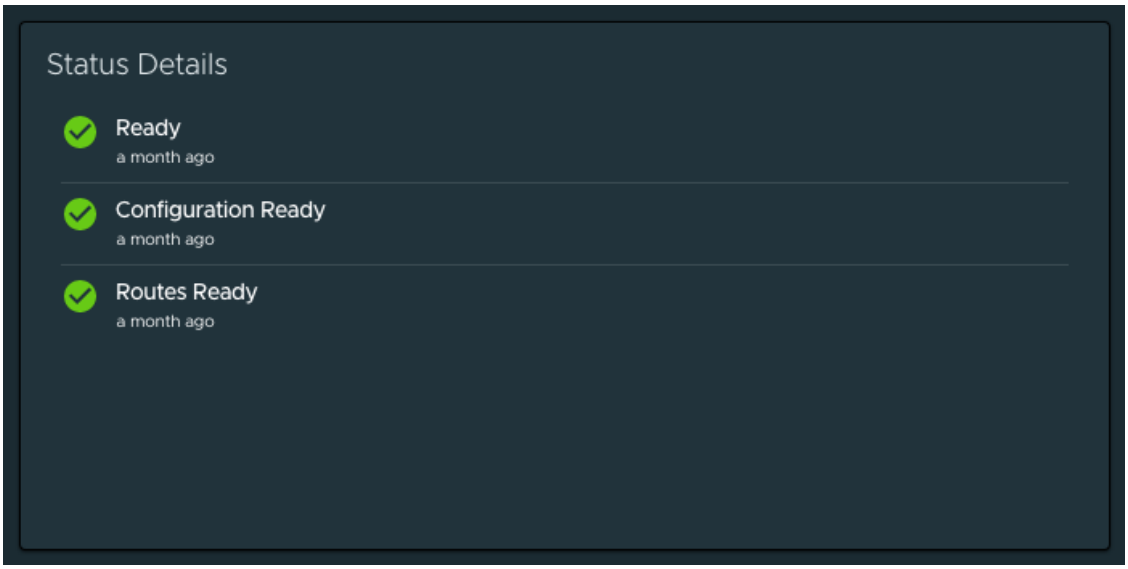
Note

The **VIEW CPU AND MEMORY DETAILS** and **VIEW THREADS** sections are only available for applications supporting Application Live View.

Status card

The status section displays all of the conditions in the resource's attribute `status.conditions`. Not all resources have conditions, and they can vary from one resource to the other.

For more information about object `spec` and `status`, see the Kubernetes documentation.



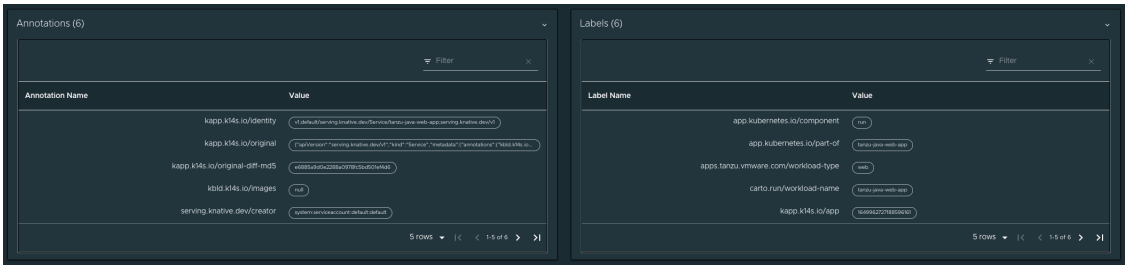
Ownership card

Depending on the resource that you are viewing, the ownership section displays all the resources specified in `metadata.ownerReferences`. You can use this section to navigate between resources.

For more information about owners and dependents, see the Kubernetes documentation.

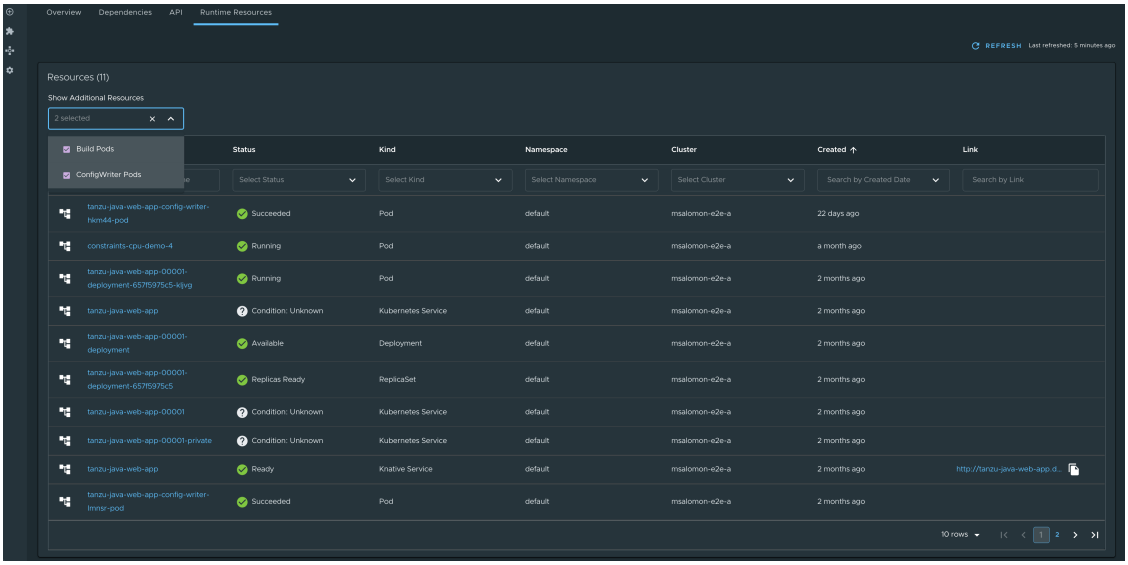
Annotations and Labels

The Annotations and Labels card displays information about `metadata.annotations` and `metadata.labels`.



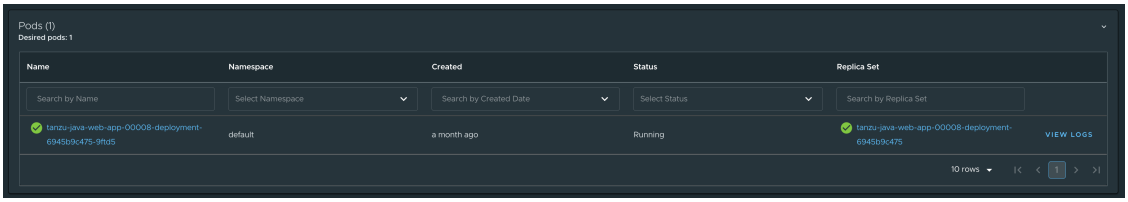
Selecting completed supply chain pods

Completed supply chain pods (build pods and ConfigWriter pods) are hidden by default in the index table. Users can choose to display them from the **Show Additional Resources** drop-down menu above the Resources index table. This drop-down menu is only visible if the resources include Build or ConfigWriter pods.



Navigating to the pod Details page

Users can see the pod table in each resource details page.



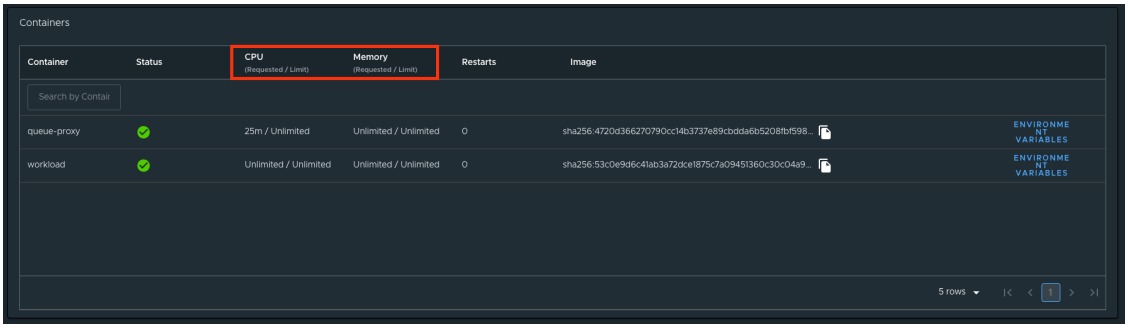
Overview of pod metrics

If you have a metrics server running on your cluster, the overview card displays realtime metrics for pods.

If you do not have a metrics server, the overview card displays the user-configured resource limits on the pod, defined in accordance with the Kubernetes documentation.

For applications built using Spring Boot, you can also monitor the actual real-time resource use using Screenshot of Application Live View for Spring Boot Applications in Tanzu Developer Portal..

Metrics and limits are also displayed for each container on a pod details page. If a particular container's current limit conflicts with a namespace-level LimitRange, a small warning indicator is displayed next to the container limit. Most conflicts are due to creating a container before applying a LimitRange.



Pods display the sum of the limits of all their containers. If a limit is not specified for a container, both the container and its pod are deemed to require unlimited resources.

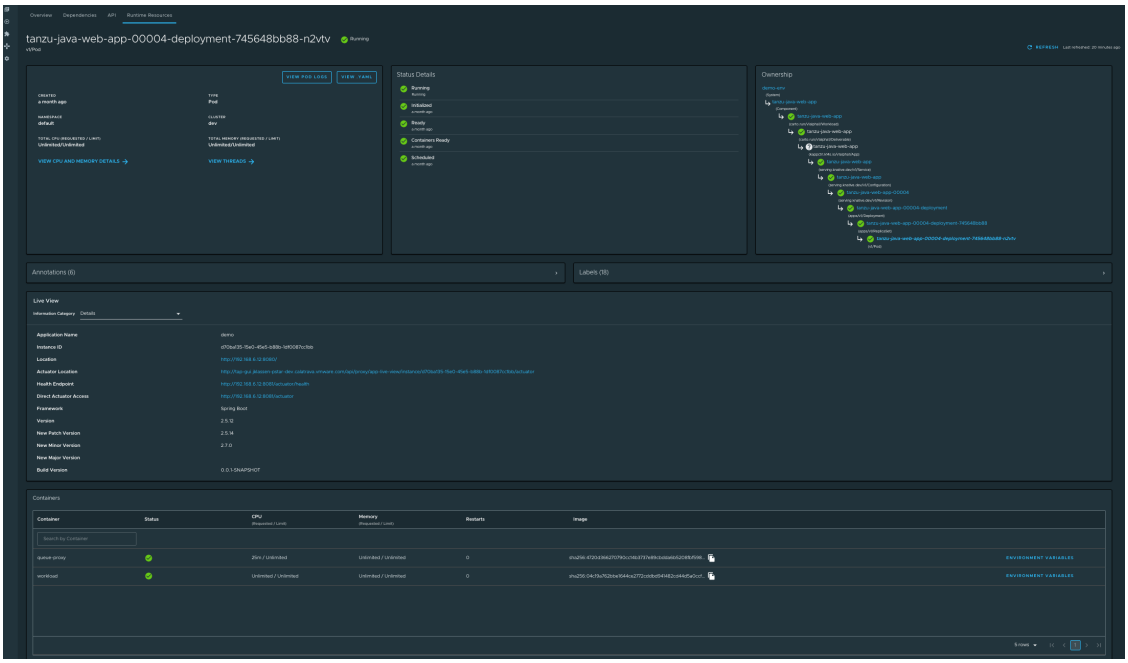
Namespace-level resource limits, such as default memory limits and default CPU limits, are not considered as part of these calculations.

For more information about default memory limits and default CPU limits see the Kubernetes documentation.

These limits apply only for Memory and CPU that a pod or container can use. Kubernetes manages these resource units by using a binary base, which is explained in the Kubernetes documentation.

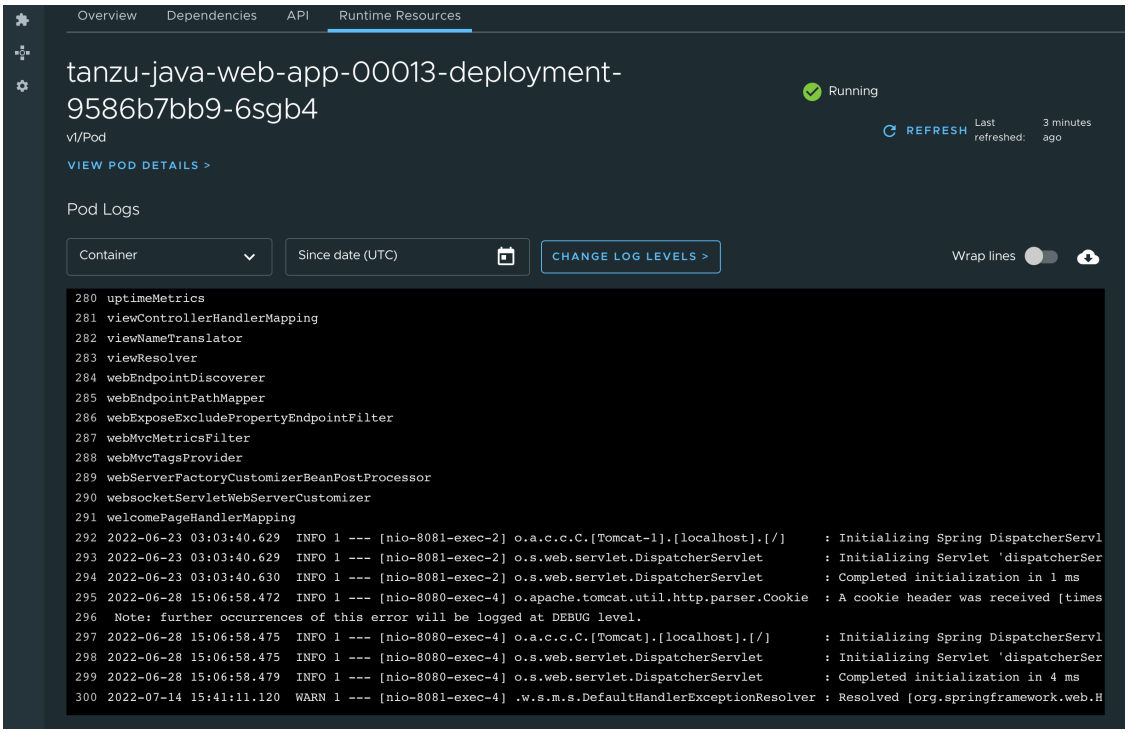
Navigating to Application Live View

To view additional information about your running applications, see the Application Live View section in the **Pod Details** page.



Viewing pod logs

To view logs for a pod, click **View Pod Logs** from the **Pod Details** page. By default, logs for the pod's first container are displayed, dating back to when the pod was created.



Pausing and resuming logs

Log entries are streamed in real time. New entries appear at the bottom of the log content area. Click or scroll the log content area to pause the log stream. Pausing the log stream enables you to focus on specific entries.

To resume the stream, click the **Follow Latest** button that appears after pausing.

Filtering by container

To display logs for a different container, select the container that you want from the **Container** drop-down menu.

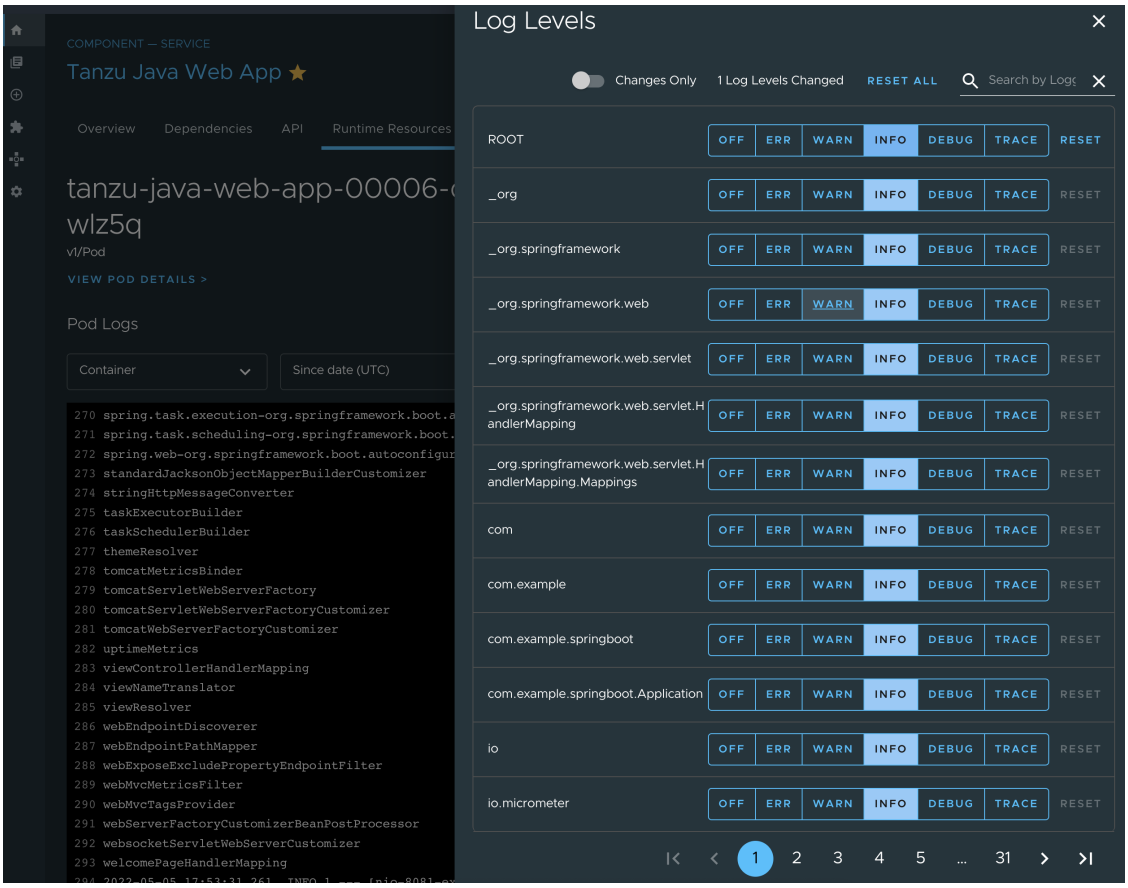
Filtering by date and time

To see logs since a specific date and time, select or type the UTC timestamp in the **Since date** field. If no logs are displayed, adjust the timestamp to an earlier time. If you do not select a timestamp, all logs produced since the pod was created are displayed.

For optimal performance, the pod logs page limits the total log entries displayed to the last 10,000, at most.

Changing log levels

If the pod is associated with an application that supports Application Live View, you can change the application's log levels by clicking the **Change Log Levels** button. You then see a panel that enables you to select levels for each logger associated with your application.



To change the levels for your application, select the desired level for each logger presented, and then click **X** in the upper-right corner of the panel, or press the Escape key, to close the panel.

Because adjusting log levels makes a real-time configuration change to your application, log-level adjustments are only reflected in log entries that your application produces after the change.

If no log entries for the expected levels appear, ensure that:

1. You adjusted the correct application loggers
2. You are viewing logs for the correct container and time frame
3. Your application is currently producing logs at the expected levels

Line wrapping

By default, log entries are not wrapped. To activate or deactivate line wrapping, click the **Wrap lines** toggle.

Downloading logs

To download current log content, click the **Download logs** button.

For optimal performance, the pod logs page limits the total log entries downloaded to the last 10,000, at most.

Connection interruptions

If the log stream connection is interrupted for any reason, such as a network error, a notification appears after the most recent log entry, and the page attempts to reconnect to the log stream. If reconnection fails, an error message displays at the top of the page, and you can click the **Refresh** button at the upper-right of the page to attempt to reconnect.

If you notice frequent disconnections at regular intervals, contact your administrator. Your administrator might need to update the back-end configuration for your installation to allow long-lived HTTP connections to log endpoints (endpoints starting with `BACKEND-HOST/api/k8s-logging/`).

Application Live View in Tanzu Developer Portal

This topic tells you about Application Live View in Tanzu Developer Portal.

Overview

The Application Live View features of Tanzu Application Platform include sophisticated components to give developers and operators a view into their running workloads on Kubernetes.

Application Live View shows an individual running process, for example, a Spring Boot application deployed as a workload resulting in a JVM process running inside of a pod. This is an important concept of Application Live View. Application Live View only recognizes running processes. If there is not a running process inside of a running pod, Application Live View does not show anything.

Under the hood, Application Live View uses the concept of actuators to gather data from those running processes. It visualizes them in a semantically meaningful way and allows users to interact with the inner workings of the running processes within limited boundaries.

The actuator data serves as the source of truth. Application Live View provides a live view of the data from inside of the running processes only. It does not store any of that data for further analysis or historical views.

This easy-to-use interface provides ways to troubleshoot, learn, and maintain an overview of certain aspects of the running processes. It gives a level of control to the users to change some parameters, such as environment properties, without a restart (where the Spring Boot application, for example, supports that).

Entry point to Application Live View plug-in

The Application Live View UI plug-in is part of Tanzu Developer Portal. To use the Application Live View plug-in:

1. Select the relevant component under the **Organization Catalog** in Tanzu Developer Portal.
2. Select the desired service under the **Runtime Resources** tab.
3. Select the desired pod from the **Pods** section under the **Runtime Resources** tab.
4. You can now see all the details, do some lightweight troubleshooting, and interact with the application within certain boundaries under the **Live View** section.

Application Live View in Tanzu Developer Portal

This topic tells you about Application Live View in Tanzu Developer Portal.

Overview

The Application Live View features of Tanzu Application Platform include sophisticated components to give developers and operators a view into their running workloads on Kubernetes.

Application Live View shows an individual running process, for example, a Spring Boot application deployed as a workload resulting in a JVM process running inside of a pod. This is an important concept of Application Live View. Application Live View only recognizes running processes. If there is not a running process inside of a running pod, Application Live View does not show anything.

Under the hood, Application Live View uses the concept of actuators to gather data from those running processes. It visualizes them in a semantically meaningful way and allows users to interact with the inner workings of the running processes within limited boundaries.

The actuator data serves as the source of truth. Application Live View provides a live view of the data from inside of the running processes only. It does not store any of that data for further analysis or historical views.

This easy-to-use interface provides ways to troubleshoot, learn, and maintain an overview of certain aspects of the running processes. It gives a level of control to the users to change some parameters, such as environment properties, without a restart (where the Spring Boot application, for example, supports that).

Entry point to Application Live View plug-in

The Application Live View UI plug-in is part of Tanzu Developer Portal. To use the Application Live View plug-in:

1. Select the relevant component under the **Organization Catalog** in Tanzu Developer Portal.
2. Select the desired service under the **Runtime Resources** tab.
3. Select the desired pod from the **Pods** section under the **Runtime Resources** tab.
4. You can now see all the details, do some lightweight troubleshooting, and interact with the application within certain boundaries under the **Live View** section.

Application Live View for Spring Boot Applications in Tanzu Developer Portal

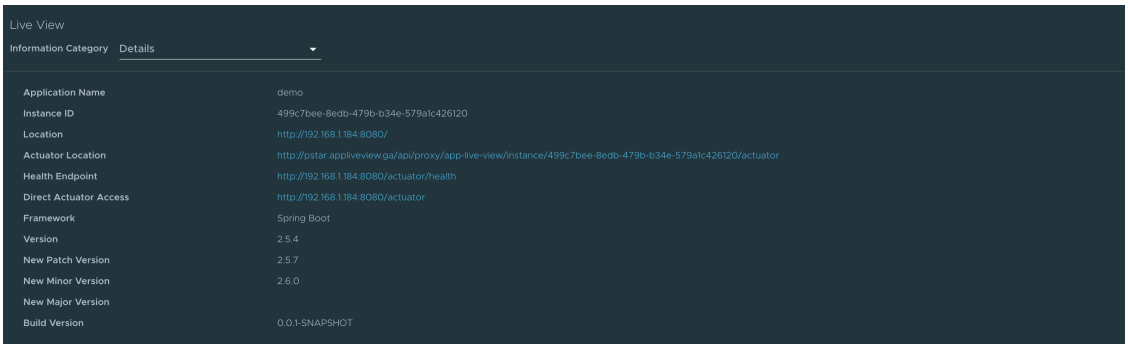
This topic tells you about the Application Live View pages for Spring Boot Applications in Tanzu Developer Portal.

Details page

This is the default page loaded in the **Live View** section. This page gives a tabular overview containing the following information:

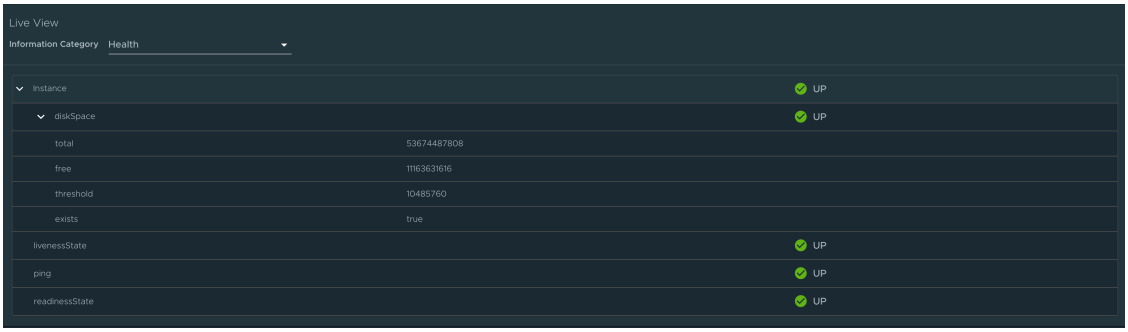
- application name
- instance ID
- location
- actuator location
- health endpoint
- direct actuator access
- framework
- version
- new patch version
- new major version
- build version

You can navigate between **Information Categories** by selecting from the drop-down menu on the top right corner of the page.



Health page

To go to the health page, select the **Health** option from the **Information Category** drop-down menu. The health page provides detailed information about the health of the application. It lists all the components that make up the health of the application such as readiness, liveness, and disk space. It displays the status and details associated with each component.



Environment page

To go to the **Environment** page, select the **Environment** option from the **Information Category** drop-down menu. The Environment page contains details of the applications' environment. It contains properties including, but not limited to, system properties, environment variables, and configuration properties (such as application.properties) in a Spring Boot application.

The page includes the following capabilities for `viewing` configured environment properties:

- The UI has a search feature that enables you to search for a property or values.
- Each property has a search icon at the right corner which helps you quickly see all the occurrences of a specific property key without manually typing in the search box. Clicking the search button locates the property name.
- The **Refresh Scope** button on the top right corner of the page probes the application to refresh all the environment properties.

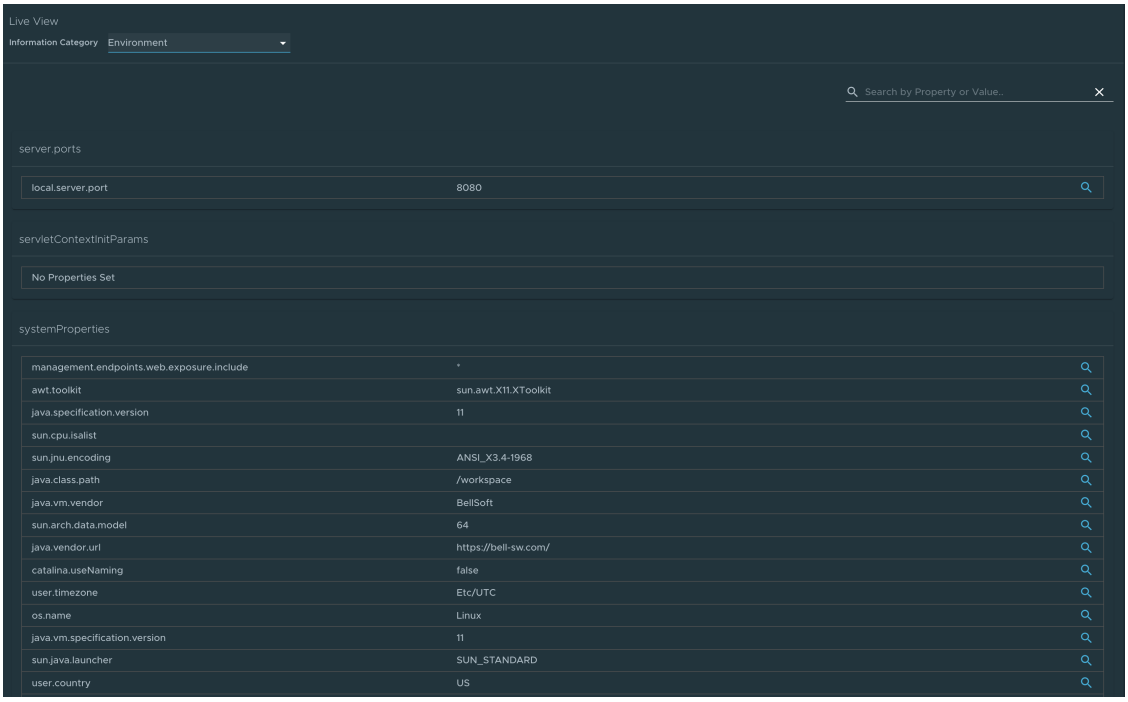
The page also includes the following capabilities for `editing` configured environment properties:

- The UI allows you to edit environment properties and see the live changes in the application. These edits are temporary and go away if the underlying pod is restarted.
- For each configured environment property, you can edit its value by clicking on the **Override** button in the same row. After the value is saved, you can view the message that the property was overridden from the initial value. The updated property is visible in the **Applied Overrides** section at the top of the page. The **Reset** button in the same row resets the environment property to the initial state.
- You can edit or remove the overridden environment variables in the **Applied Overrides** section.

- The **Applied Overrides** section also enables you to add new environment properties to the application.

Note

`management.endpoint.env.post.enabled=true` must be set in the application config properties of the application and a corresponding, editable environment must be present in the application.



Log Levels page

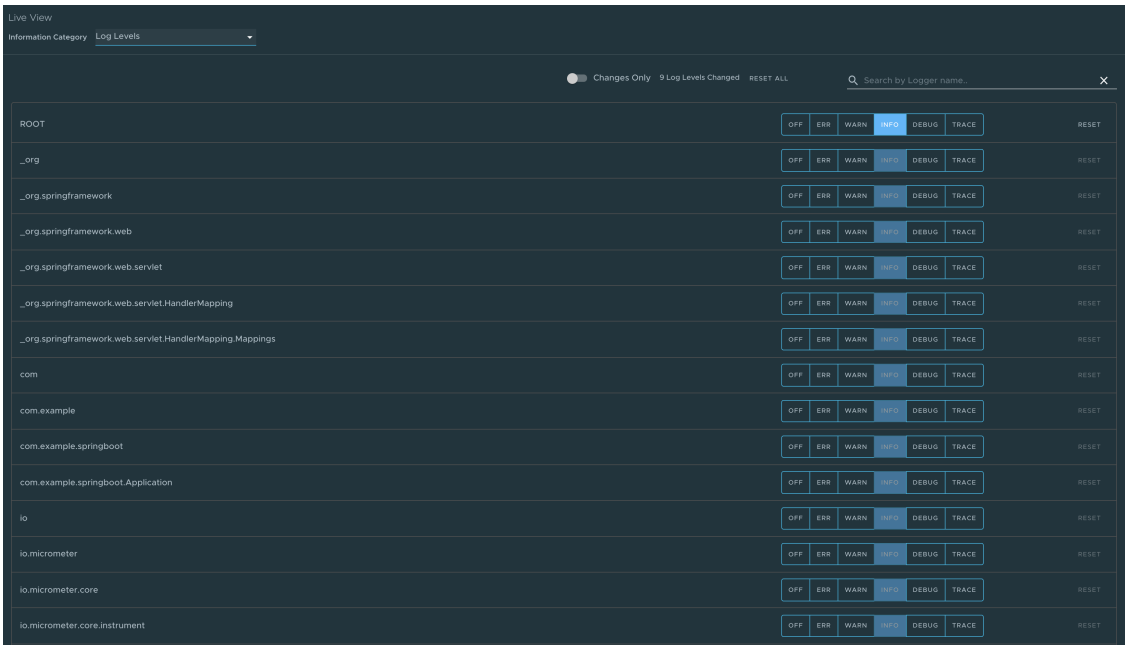
To go to the **Log Levels** page, select the **Log Levels** option from the **Information Category** drop-down menu. The log levels page provides access to the application's loggers and the configuration of their levels.

You can configure the log levels such as INFO, DEBUG, and TRACE in real time from the UI. You can search for a package and edit its respective log level. You can configure the log levels at a specific class and package. They can deactivate all the log levels by modifying the log level of root logger to OFF.

The toggle **Changes Only** displays the changed log levels. Use the search feature to search by logger name. The **Reset** resets the log levels to the original state. The **Reset All** on top right corner of the page resets all the loggers to default state.

Note

Use the UI to change the log levels and see the live changes on the application. These changes are temporary and go away if the underlying pod is restarted.

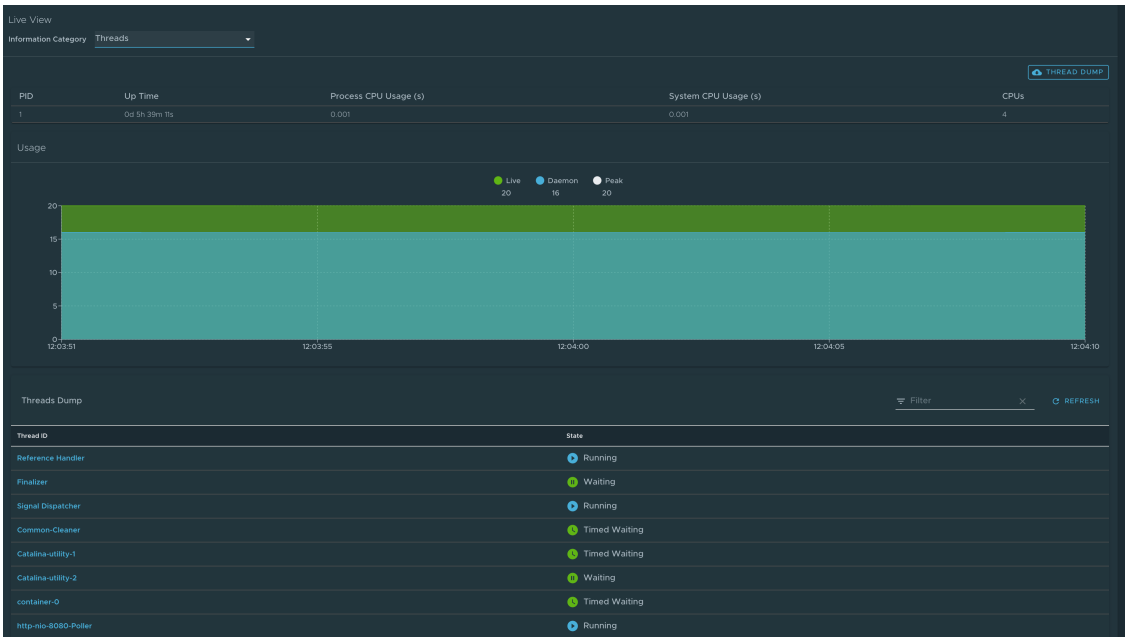


Threads page

To go to the **Threads** page, select the **Threads** option from the **Information Category** drop-down menu.

This page displays all details related to Java Virtual Machine (JVM) threads and running processes of the application. This tracks live threads and daemon threads real-time. It is a snapshot of different thread states. Navigating to a thread state displays all the information about a particular thread and its stack trace.

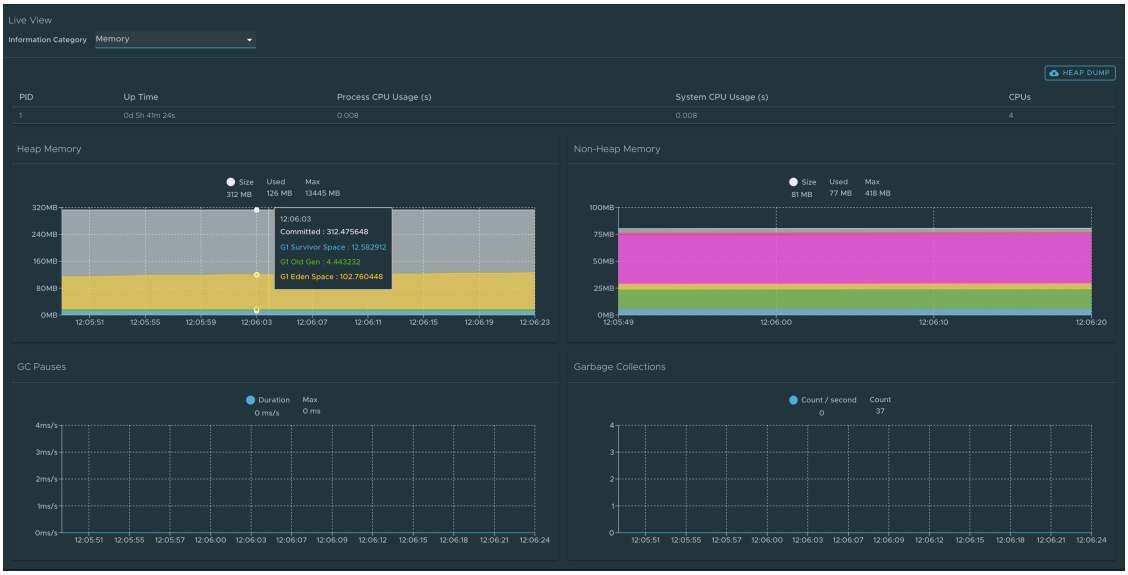
Use the search feature to search for threads by thread ID or state. The refresh icon refreshes to the latest state of the threads. You can view more thread details by clicking on the Thread ID. The page also has a feature to download thread dump data for analysis purposes.



Memory page

To go to the **Memory** page, select the **Memory** option from the **Information Category** drop-down menu.

- The memory page highlights the memory use inside of the JVM. It displays a graphical representation of the different memory regions within heap and non-heap memory. This visualizes data from inside of the JVM (in case of Spring Boot apps running on a JVM) and therefore provides memory insights into the application in contrast to “outside” information about the Kubernetes pod level.
- The real-time graphs displays a stacked overview of the different spaces in memory with the total memory used and total memory size. The page contains graphs to display the GC pauses and GC events.
- The **Heap Dump** at the top-right corner enables you to download heap dump data.



Note

This graphical visualization happens in real time and shows real-time data only. As mentioned at the top, the Application Live View features do not store any information. That means the graphs visualize the data over time only for as long as you stay on that page.

Request Mappings page

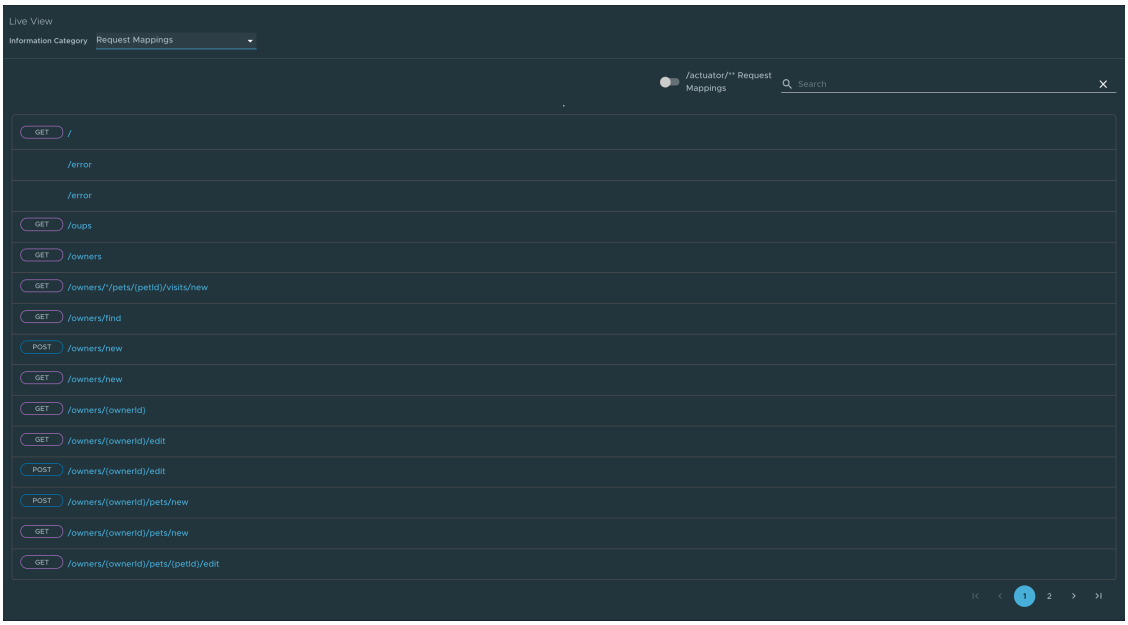
To go to the Request Mappings page, select the **Request Mappings** option from the **Information Category** drop-down menu.

This page provides information about the application’s request mappings. For each mapping, it displays the request handler method. You can view more details of the request mapping, such as the header metadata of the application.

When you click on the request mapping, a side panel appears. This panel contains information about the mapping-media types `Produces` and `Consumes`. The panel also displays the `Handler` class for the request. Use the search feature to search for the request mapping or the method. The toggle **/actuator/** Request Mappings** displays the actuator related mappings of the application.

Note

When application actuator endpoint is exposed on `management.server.port`, the application does not return any actuator request mappings data in the context. The application displays a message when the actuator toggle is enabled.



HTTP Requests page

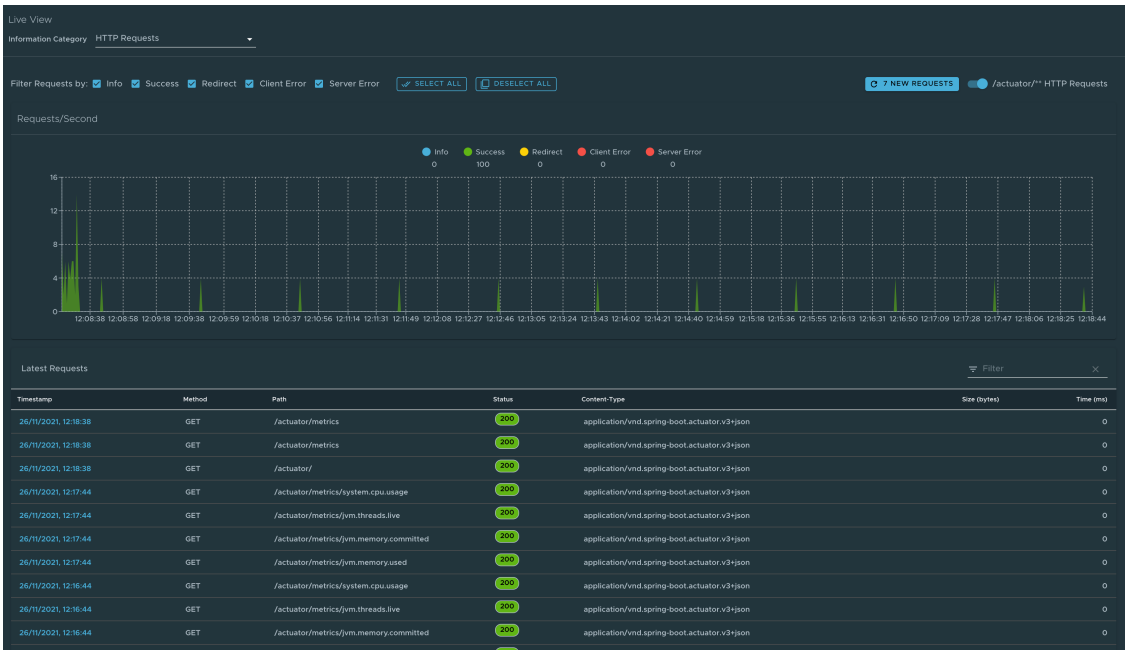
To go to the HTTP Requests page, click **HTTP Requests** from the **Information Category** drop-down menu. The HTTP Requests page provides information about HTTP request-response exchanges to the application.

The graph visualizes the requests per second indicating the response status of all the requests. You can filter the response statuses, which include info, success, redirects, client-errors, and server-errors. The trace data is captured in detail in a tabular format with metrics such as timestamp, method, path, status, content-type, length, time.

The search feature on the table filters the traces based on the search field value. You can view more details of the request, such as method, headers, and response of the application by clicking on the timestamp. The refresh icon above the graph loads the latest traces of the application. The toggle **/actuator/**** on the top right corner of the page displays the actuator related traces of the application.

Note

When application actuator endpoint is exposed on management.server.port, no actuator HTTP Traces data is returned for the application. In this case, a message is displayed when the actuator toggle is enabled.

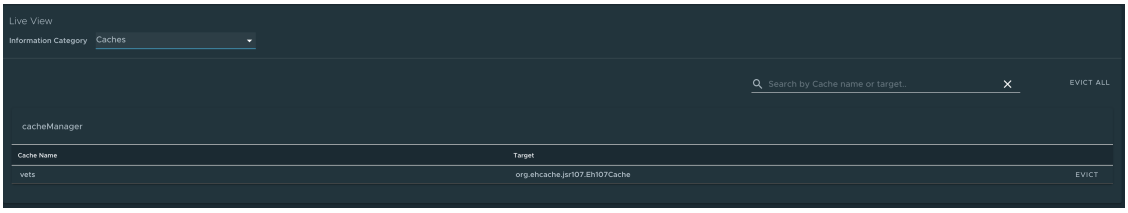


Caches page

To go to the **Caches** page, select the **Caches** option from the **Information Category** drop-down menu.

The Caches page provides access to the application's caches. It gives the details of the cache managers associated with the application including the fully qualified name of the native cache.

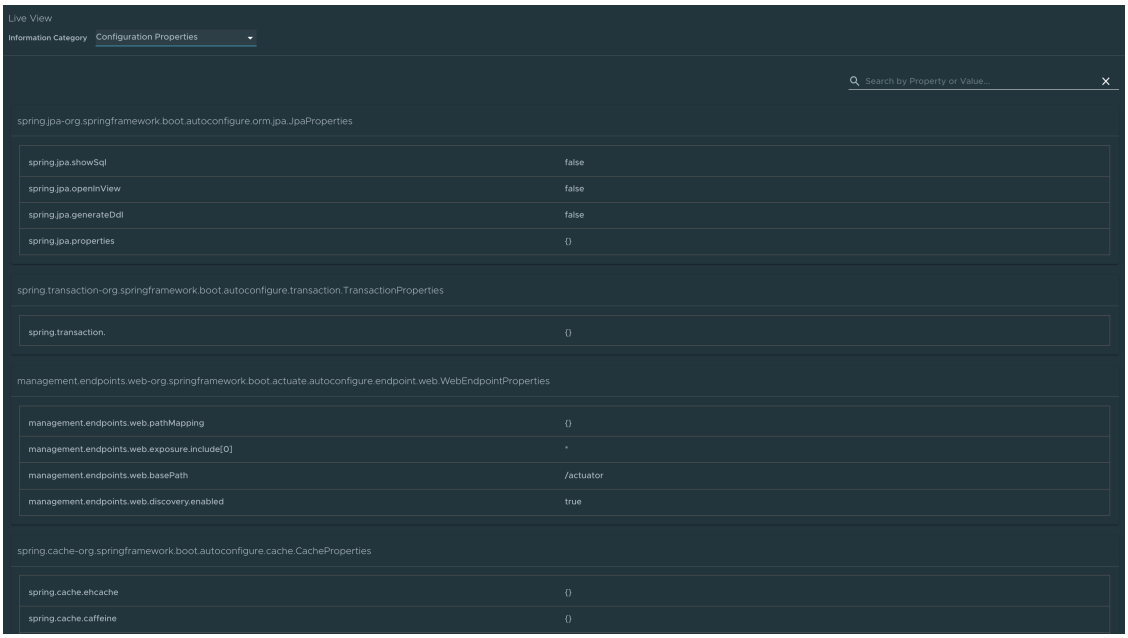
Use the search feature in the Caches Page to search for a specific cache/cache manager. You can clear individual caches by clicking **Evict**. You can clear all the caches completely by clicking **Evict All**. If there are no cache managers for the application, the message `No cache managers available for the application` is displayed.



Configuration Properties page

To go to the **Configuration Properties** page, select the **Configuration Properties** option from the **Information Category** drop-down menu.

The configuration properties page provides information about the configuration properties of the application. In case of Spring Boot, it displays application's `@ConfigurationProperties` beans. It gives a snapshot of all the beans and their associated configuration properties. Use the search feature to search for a property's key/value or the bean name.

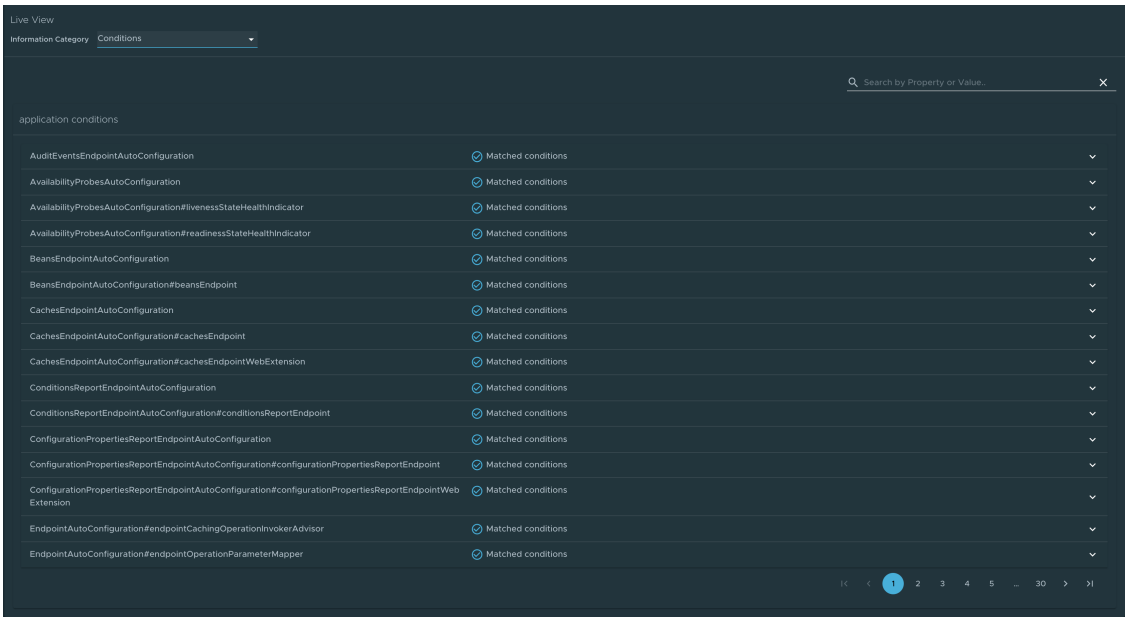


Conditions page

To go to the **Conditions** page, select the **Conditions** option from the **Information Category** drop-down menu. The conditions evaluation report provides information about the evaluation of conditions on configuration and auto-configuration classes.

In the case of Spring Boot, this gives you a view of all the beans configured in the application. When you click on the bean name, the conditions and the reason for the conditional match is displayed.

In the case of non-configured beans, it shows both the matched and unmatched conditions of the bean if any. In addition to this, it also displays names of unconditional auto configuration classes if any. You can use the search feature to filter out the beans and the conditions.

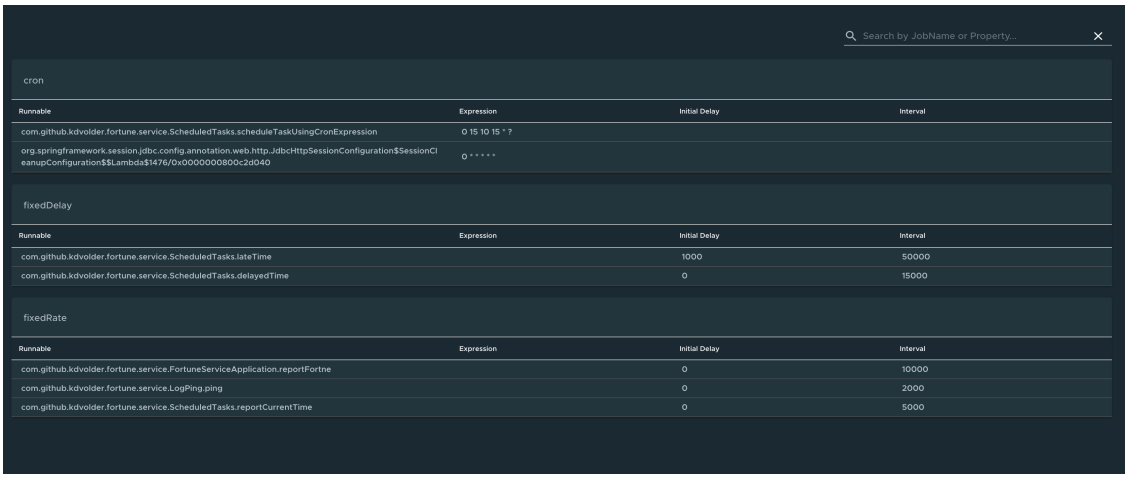


Scheduled Tasks page

To go to the **Scheduled Tasks** page, select the **Scheduled Tasks** option from the **Information Category** drop-down menu.

The scheduled tasks page provides information about the application's scheduled tasks. It includes cron tasks, fixed delay tasks and fixed rate tasks, custom tasks and the properties associated with them.

You can search for a particular property or a task in the search bar to retrieve the task or property details.



Beans page

To go to the **Beans** page, select the **Beans** option from the **Information Category** drop-down menu. The beans page provides information about a list of all application beans and its dependencies. It displays the information about the bean type, dependencies, and its resource. You can search by the bean name or its corresponding fields.

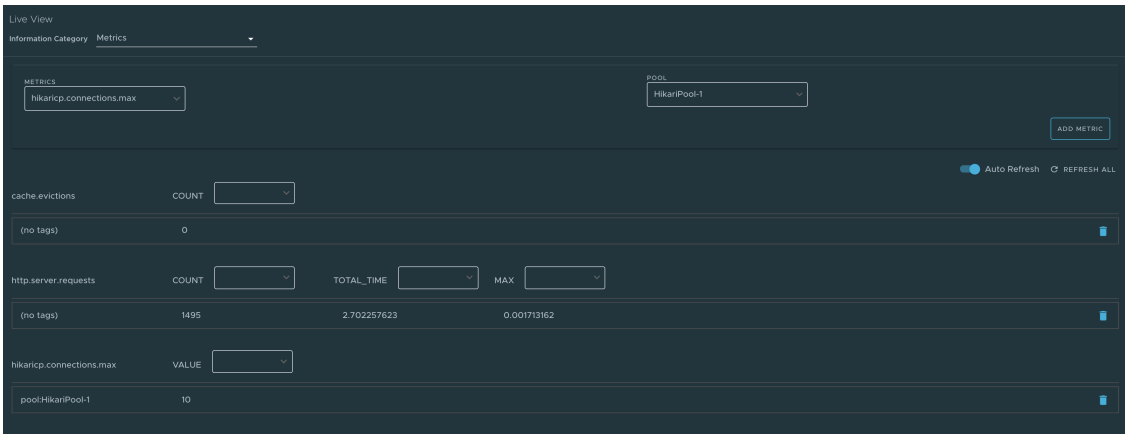
Metrics page

To go to the **Metrics** page, select the **Metrics** option from the **Information Category** drop-down menu.

The metrics page provides access to application metrics information. You can choose from the list of various metrics available for the application, such as `jvm.memory.used`, `jvm.memory.max`, `http.server.request`, and so on.

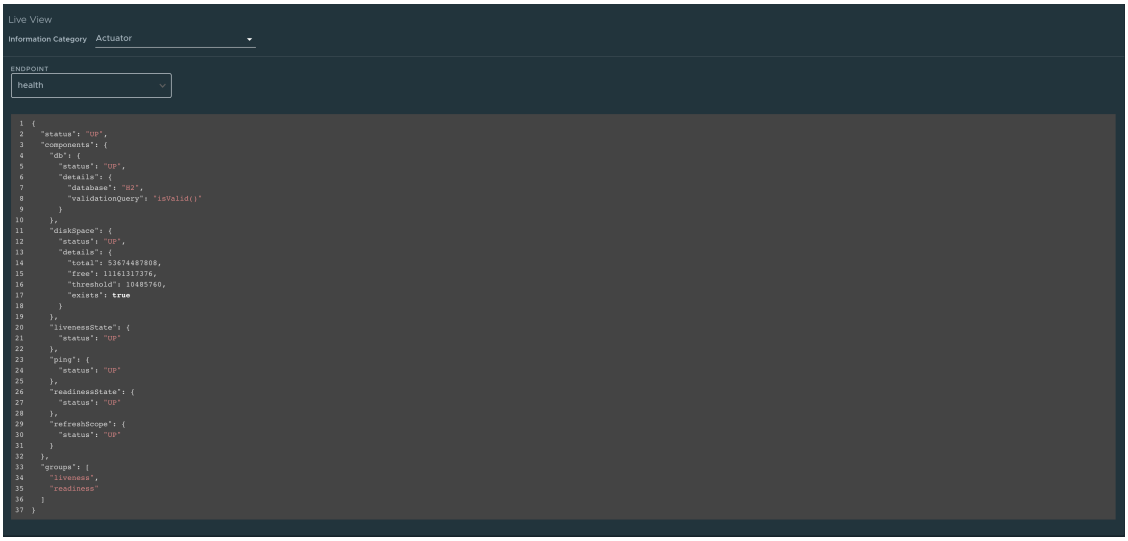
After the metric is chosen, you can view the associated tags. You can choose the value of each tag based on filtering criteria. Clicking **Add Metric** adds the metric to the page which is refreshed every 5 seconds by default.

You can pause the auto refresh feature by deactivating the **Auto Refresh** toggle. You can refresh the metrics manually by clicking **Refresh All**. The format of the metric value can be changed according to your needs. They can delete a particular metric by clicking the minus symbol in the same row.



Actuator page

To go to the **Actuator** page, select the **Actuator** option from the **Information Category** drop-down menu. The actuator page provides a tree view of the actuator data. You can choose from a list of actuator endpoints and parse through the raw actuator data.



Troubleshooting

You might run into cases where a workload running on your cluster does not appear in the Application Live View overview, the detail pages do not load any information while running, or similar issues. If you encounter issues, see Troubleshooting in the Application Live View documentation.

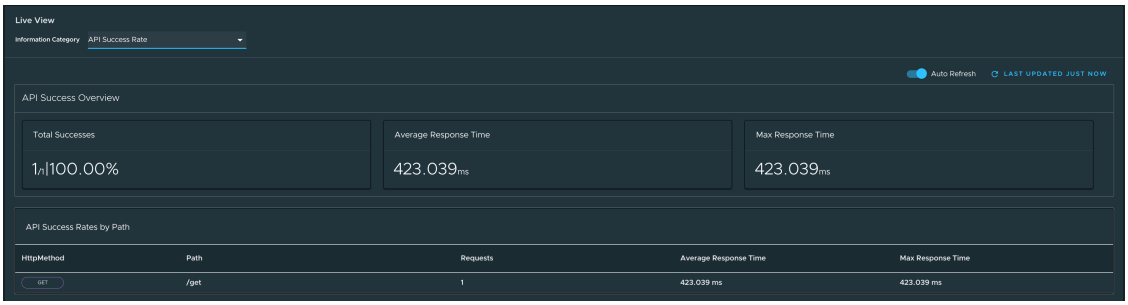
Application Live View for Spring Cloud Gateway applications in Tanzu Developer Portal

This topic tells you about the Application Live View pages for Spring Cloud Gateway applications in Tanzu Developer Portal.

API Success Rate page

To access to the API Success Rate page, select the **API Success Rate** option from the **Information Category** drop-down menu.

The API success rate page displays the total successes, average response time, and maximum response time for the gateway routes. It also displays the details of each successful route path.



API Overview page

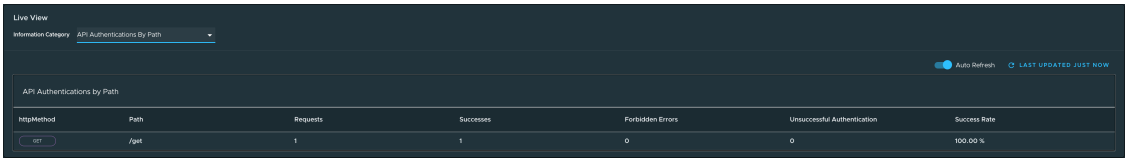
To access the API Overview page, select the **API Overview** option from the **Information Category** drop-down menu.

The API Overview page provides route count, number of successes, errors, and the rate-limited requests. It also provides an **auto refresh** feature to get the updated results. These metrics are depicted in a line graph.

API Authentications By Path page

To access the API Authentications By Path page, select the **API Authentications By Path** option from the **Information Category** drop-down menu.

The API Authentications By Path page displays the total requests, number of successes, and forbidden and unsuccessful authentications grouped by the HTTP method and gateway route path. The page also displays the success rate for each route.



Note

In addition to the preceding three pages, the Spring Boot actuator pages are also displayed.

Troubleshooting

You might run into cases where a workload running on your cluster does not appear in the Application Live View overview, or the detail pages do not load any information while running, or other similar issues. For more information about such issues, see Troubleshooting in the Application Live View documentation.

Application Live View for Steeltoe applications in Tanzu Developer Portal

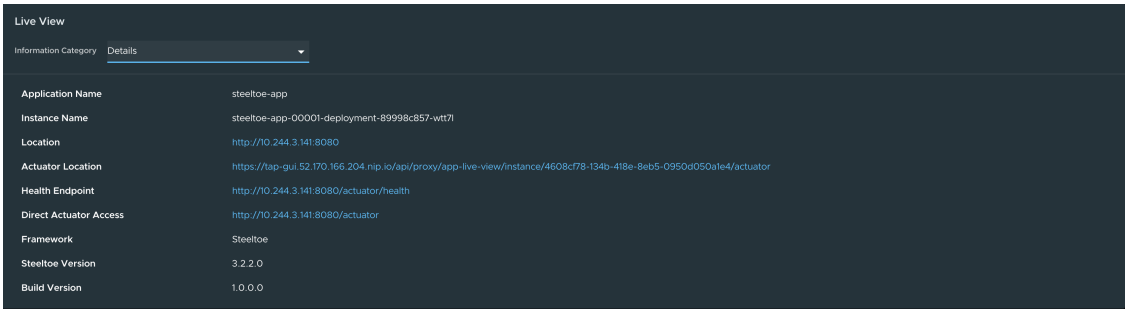
This topic tells you about the Application Live View pages for Steeltoe applications in Tanzu Developer Portal.

Details page

This is the default page loaded in the **Live View** section. This page gives a tabular overview containing the following information:

- Application name
- Instance ID
- Location
- Actuator location
- Health endpoint
- Direct actuator access
- Framework
- Version
- New patch version
- New major version
- Build version

You can navigate between **Information Categories** by selecting from the drop-down menu on the top right corner of the page.



Health page

To access the health page, select the **Health** option from the **Information Category** drop-down menu.

The health page provides detailed information about the health of the application. It lists all the components that make up the health of the application, such as readiness, liveness, and disk space. It displays the status and details associated with each component.

Environment page

To access the **Environment** page, select the **Environment** option from the **Information Category** drop-down menu.

The Environment page contains details of the applications' environment. It contains properties including, but not limited to, system properties, environment variables, and configuration properties (such as `appsettings.json`) in a Steeltoe application.

The page includes the following capabilities for `viewing` configured environment properties:

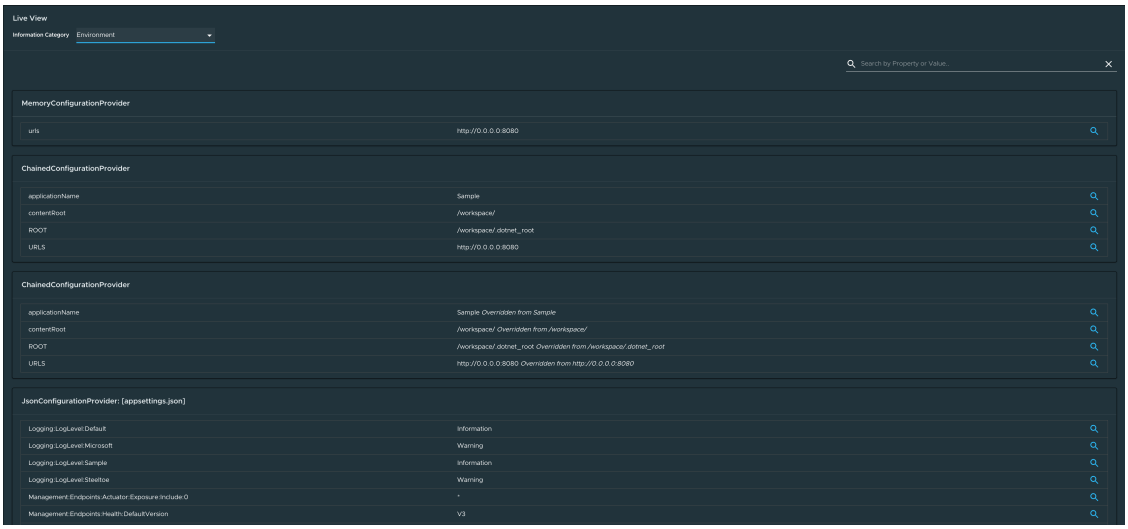
- The UI has a search feature that enables you to search for a property or values.
- Each property has a search icon at the right corner which helps you quickly see all the occurrences of a specific property key without manually typing in the search box. Clicking the search button locates the property name.
- The **Refresh Scope** button on the top right corner of the page probes the application to refresh all the environment properties.

The page also includes the following capabilities for `editing` configured environment properties:

- The UI allows you to edit environment properties and see the live changes in the application. These edits are temporary and go away if the underlying pod is restarted.
- For each of the configured environment properties, you can edit its value by clicking on the **Override** button in the same row. After the value is saved, you can view the message that the property was overridden from the initial value. Also, the updated property is visible in the **Applied Overrides** section at the top of the page. The **Reset** button in the same row resets the environment property to the initial state.
- You can also edit or remove the overridden environment variables in the **Applied Overrides** section.
- The **Applied Overrides** section also enables you to add new environment properties to the application.

Note

The `management.endpoint.env.post.enabled=true` must be set in the application config properties of the application, and a corresponding editable environment must be present in the application.



Log Levels page

To go to the **Log Levels** page, select the **Log Levels** option from the **Information Category** drop-down menu. The **Log Levels** page provides access to the application's loggers and the configuration of the levels.

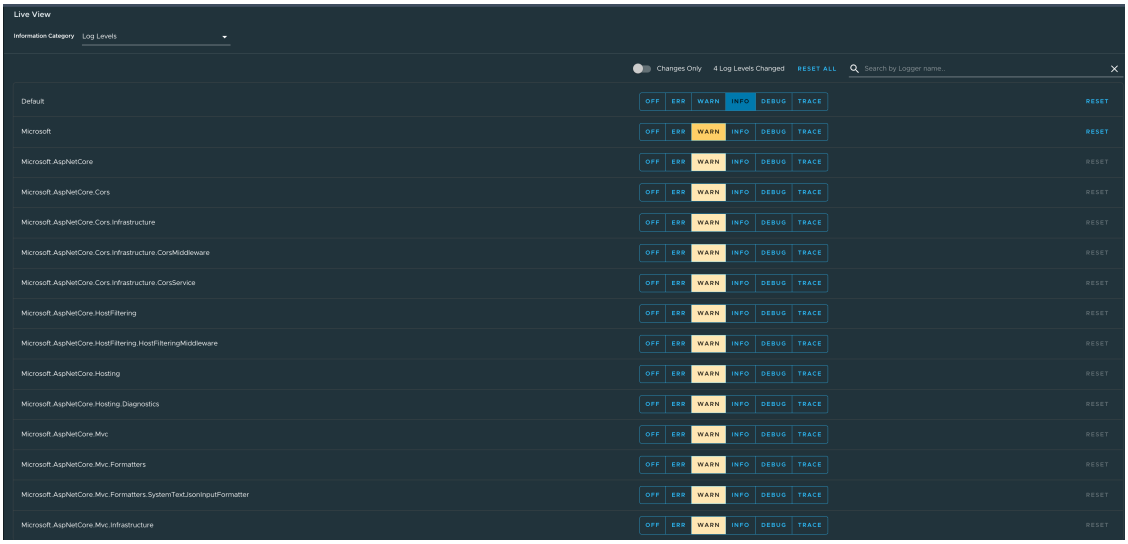
You can:

- Configure log levels, such as **INFO**, **DEBUG**, and **TRACE**, in real time from the UI
- Search for a package and edit its respective log level
- Configure the log levels at a specific class and package
- Deactivate all the log levels by changing the log level of root logger to **OFF**

Use the **Changes Only** toggle to display the changed log levels. Use the search feature to search by logger name. Click **Reset All** to reset all the loggers to the default state.

Note

The UI allows you to change the log levels and see the live changes on the application. These changes are temporary and go away if the underlying pod is restarted.

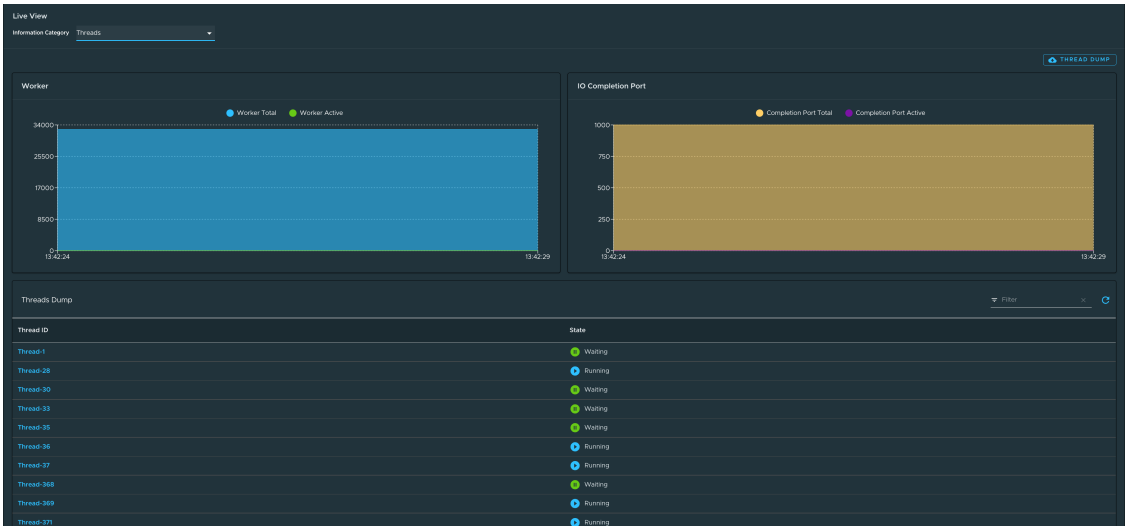


Threads page

To access the **Threads** page, select the **Threads** option from the **Information Category** drop-down menu.

This page displays all details related to CLR threads and running processes of the application. This tracks worker threads and completion port threads in real time. Navigating to a thread state displays all the information about a particular thread and its stack trace.

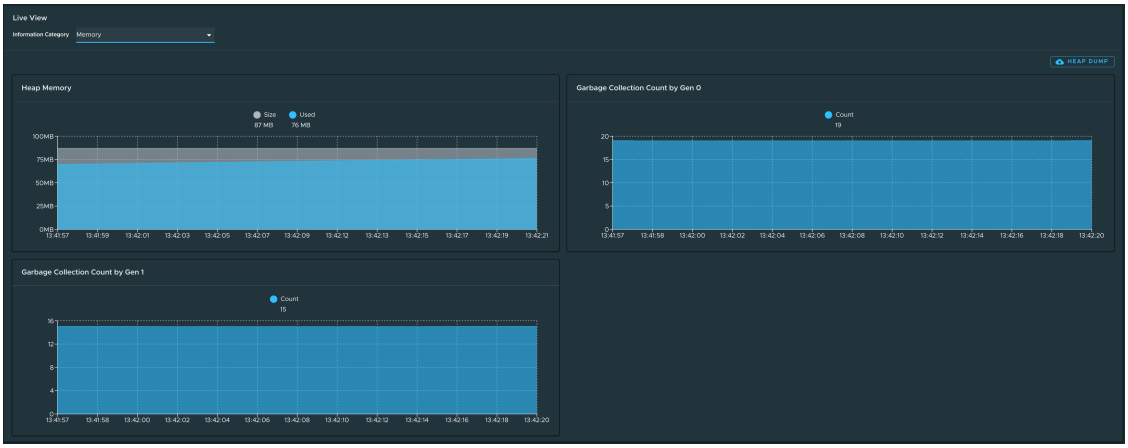
- The refresh icon refreshes to the latest state of the threads.
- To view more thread details, click the thread ID.
- The page has a feature to download the thread dump for analysis.
- The page has a feature to view the CPU stats for a Steeltoe application.



Memory page

To access the **Memory** page, select the **Memory** option from the **Information Category** drop-down menu.

This page displays all details related to used and committed memory of the application. This also displays the garbage collection count by generation (gen0/gen1). The page also has a feature to download the heap dump for analysis. The page also has a feature to view the CPU stats for a Steeltoe application

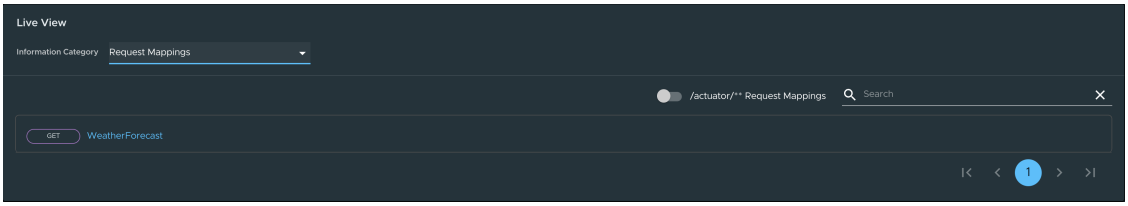


Request Mappings page

To access the **Request Mappings** page, select the **Request Mappings** option from the **Information Category** drop-down menu.

This page provides information about the application's request mappings. For each mapping, the page displays the request handler method. You can view more details of the request mapping, such as the header metadata of the application.

When you click on the request mapping, a side panel appears. This panel contains information about the mapping-media types `Produces` and `Consumes`. The panel also displays the `Handler` class for the request. The search feature enables you to search for the request mapping or the method. The toggle **/actuator/** Request Mappings** displays the actuator-related mappings of the application.



HTTP Requests page

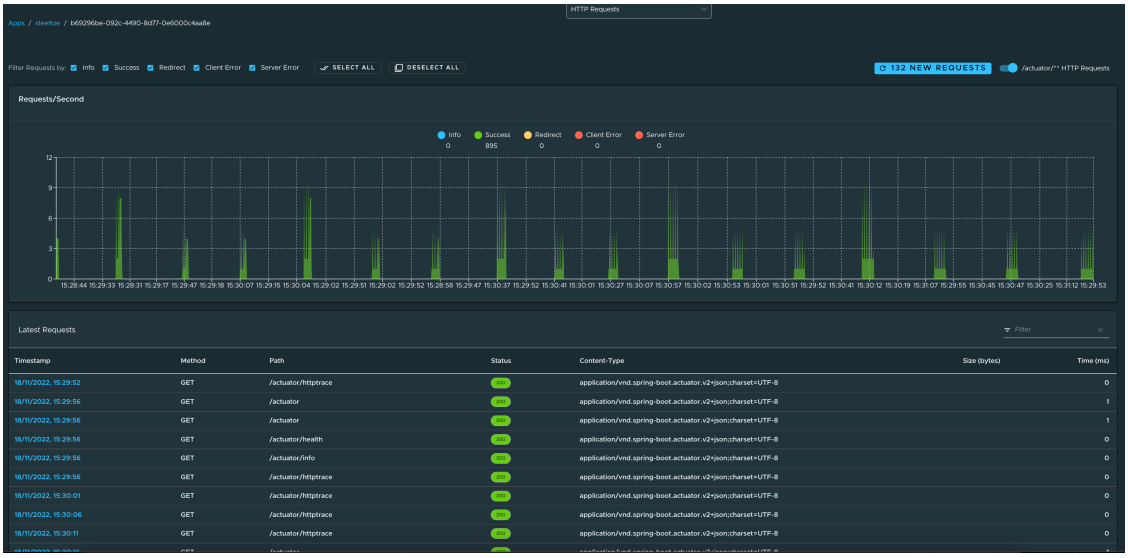
To access the **HTTP Requests** page, select the **HTTP Requests** option from the **Information Category** drop-down menu.

The **HTTP Requests** page provides information about HTTP request-response exchanges to the application.

The graph visualizes the requests per second, which indicates the response status of all the requests. You can filter by the response statuses, which include **info**, **success**, **redirects**, **client-errors**, and **server-errors**. The trace data is captured in detail in a tabular format with metrics, such as **timestamp**, **method**, **path**, **status**, **content-type**, **length**, and **time**.

The search feature on the table filters the traces based on the search text box value. By clicking on the timestamp, you can view more details of the request, such as method, headers, and the response of the application.

The refresh icon above the graph loads the latest traces of the application. The toggle **/actuator/**** on the top-right corner of the page displays the actuator-related traces of the application.



Metrics page

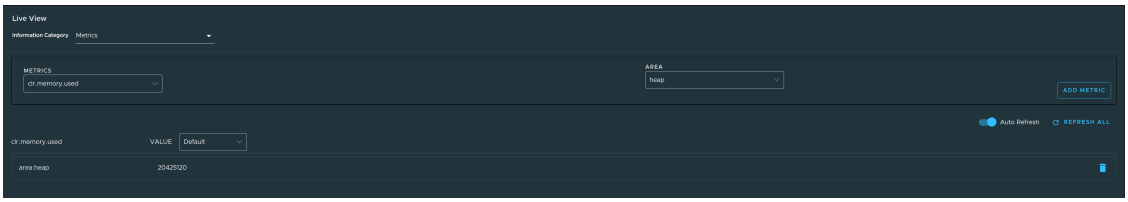
To access the **Metrics** page, select the **Metrics** option from the **Information Category** drop-down menu.

The metrics page provides access to application metrics information. You can choose from the list of various metrics available for the application, such as `clr.memory.used`, `System.Runtime.gc-committed`, `clr.threadpool.active`, and so on.

After you choose the metric, you can view the associated tags. You can choose the value of each of the tags based on filtering criteria. Click **Add Metric** to add the metric to the page. The page is refreshed every 5 seconds by default.

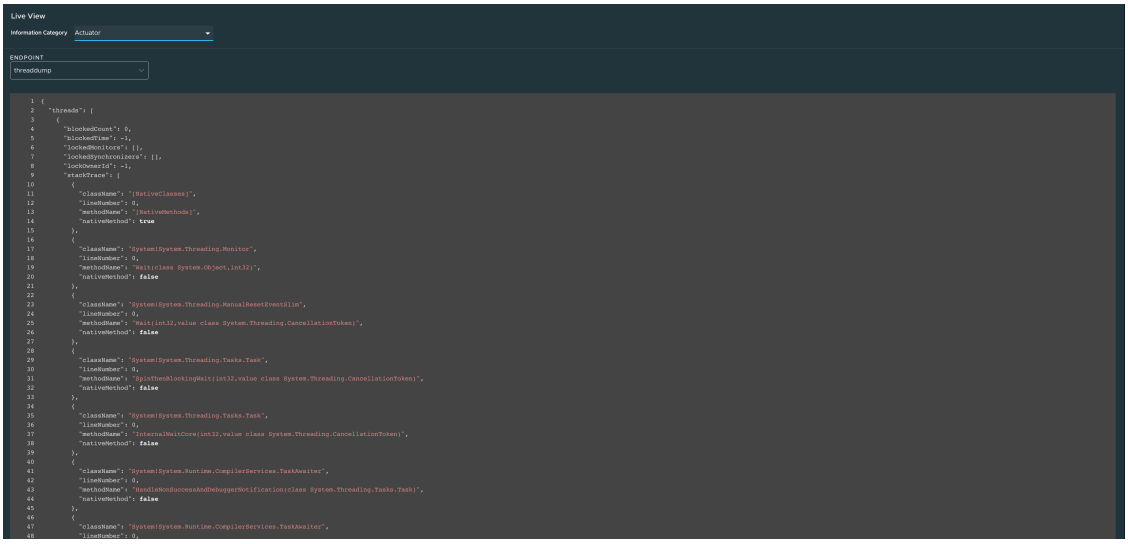
The UI on the **Metrics** page includes features that enable you to:

- Pause the auto refresh feature by deactivating the **Auto Refresh** toggle.
- Refresh the metrics manually by clicking **Refresh All**.
- Change the format of the metric value according to your needs.
- Delete a particular metric by clicking the minus-sign button in the relevant row.



Actuator page

To access the **Actuator** page, select the **Actuator** option from the **Information Category** drop-down menu. The actuator page provides a tree view of the actuator data. You can choose from a list of actuator endpoints and parse through the raw actuator data.



Troubleshooting

You might run into cases where a workload running on your cluster does not appear in the Application Live View overview, or the Details pages do not load any information while running, or other similar issues. For help with troubleshooting common issues, see Troubleshooting.

Application Accelerator in Tanzu Developer Portal

This topic tells you how to use Application Accelerator in Tanzu Developer Portal.

Overview

Application Accelerator for VMware Tanzu helps you bootstrap developing and deploying your applications in a discoverable and repeatable way.

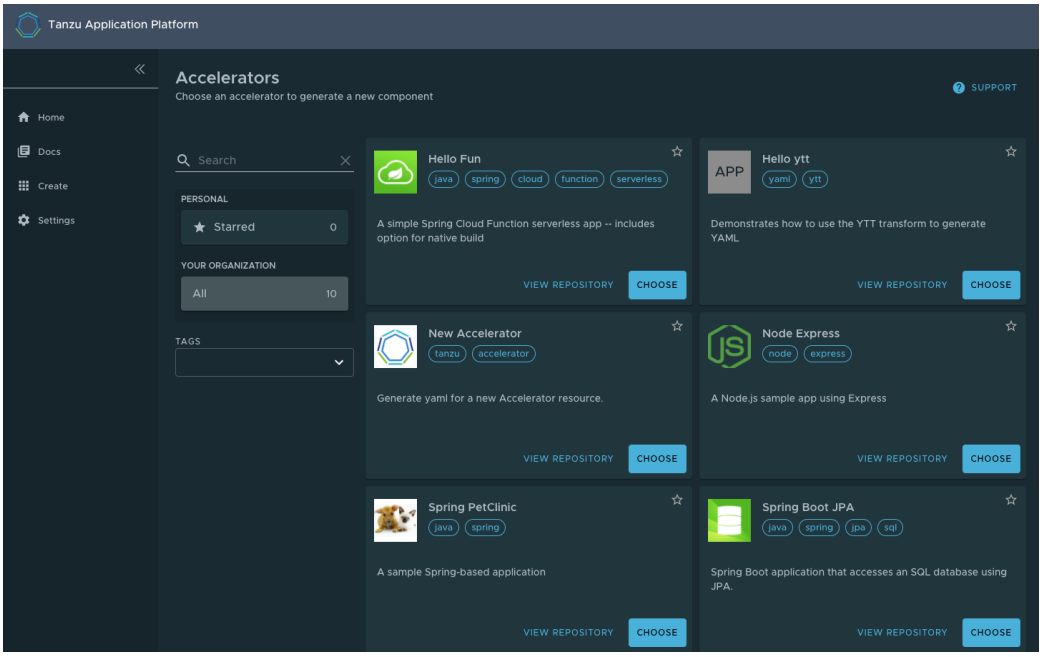
Enterprise architects author and publish accelerator projects that provide developers and operators with ready-made, enterprise-conforming code and configurations. You can then use Application Accelerator to create new projects based on those accelerator projects.

The Application Accelerator UI enables you to discover available accelerators, configure them, and generate new projects to download.

Access Application Accelerator

To open the Application Accelerator UI plug-in and select an accelerator:

1. Within Tanzu Application Platform, click **Create** in the left navigation pane to open the **Accelerators** page.



   Here you can view accelerators already registered with the system. Developers can add new accelerators by registering them with Kubernetes.

2. Every accelerator has a title and short description. Click **VIEW REPOSITORY** to view an accelerator definition. This opens the accelerator's Git repository in a new browser tab.

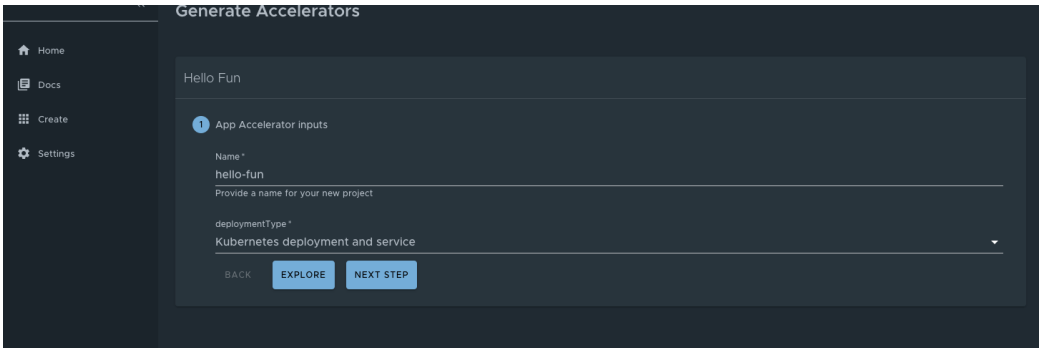
3. Search and filter based on text and tags associated with the accelerators to find the accelerator representing the project you want to create.

4. Click **CHOOSE** for the accelerator you want. This opens the **Generate Accelerators** page.

Configure project generation

To configure how projects are generated:

1. On the **Generate Accelerators** page, add any configuration values needed to generate the project. The application architect defined these values in `accelerator.yaml` in the accelerator definition. Filling some text boxes can cause other text boxes to appear. Fill them all in.

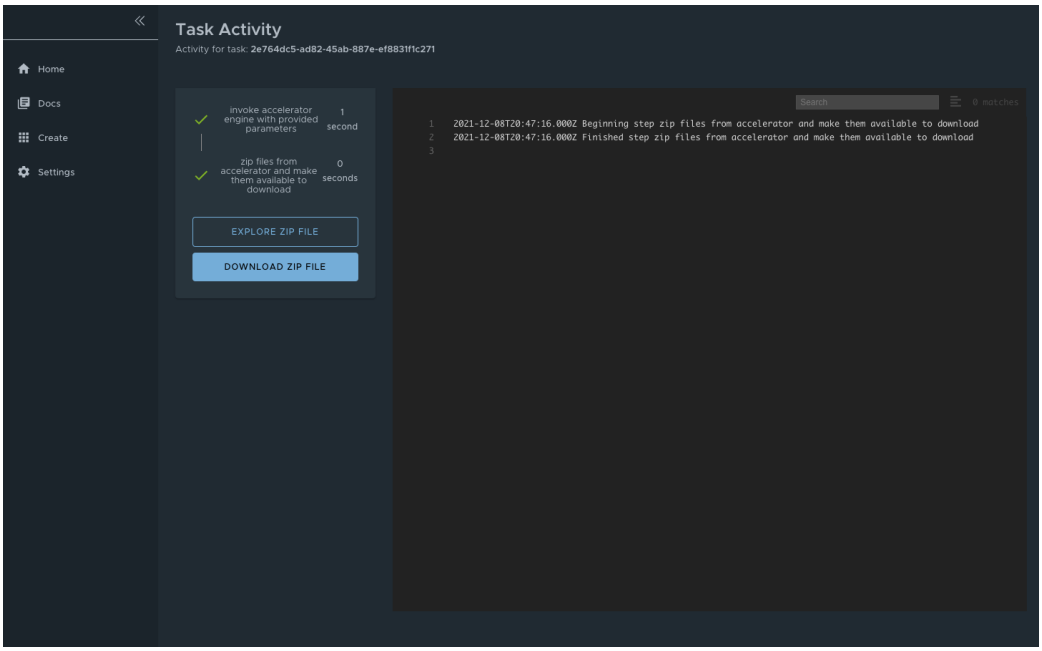


2. Click **EXPLORE** to open the **Explore Project** page and view the project before it is generated.
3. After configuring your project, click **NEXT STEP** to see the project summary page.
4. Review the values you specified for the configurable options.
5. Click **BACK** to make more changes, if necessary. Otherwise, proceed to create the project.

Create the project

To create the project:

1. Click **Create** to start generating your project. See the progress on the **Task Activity** page. A detailed log is displayed on the right.

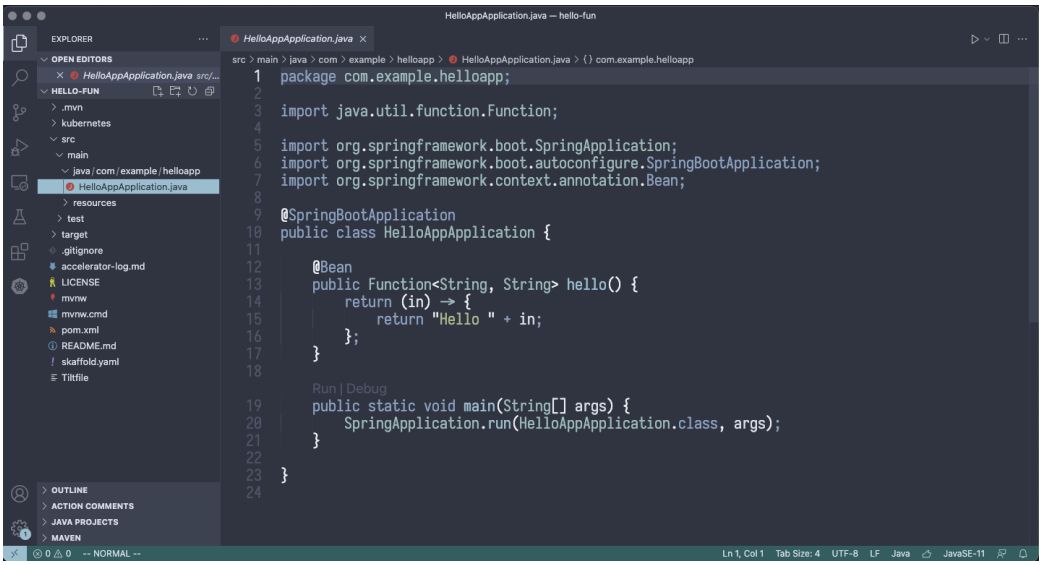


2. After the project is generated, click **EXPLORE ZIP FILE** to open the **Explore Project** page to verify configuration.
3. Click **DOWNLOAD ZIP FILE** to download the project in a ZIP file.

Develop your code

To develop your code:

1. Expand the ZIP file.
2. Open the project in your integrated development environment (IDE).



Next steps

To learn more about Application Accelerator for VMware Tanzu, see the Application Accelerator documentation.

Application Accelerator in Tanzu Developer Portal

This topic tells you how to use Application Accelerator in Tanzu Developer Portal.

Overview

Application Accelerator for VMware Tanzu helps you bootstrap developing and deploying your applications in a discoverable and repeatable way.

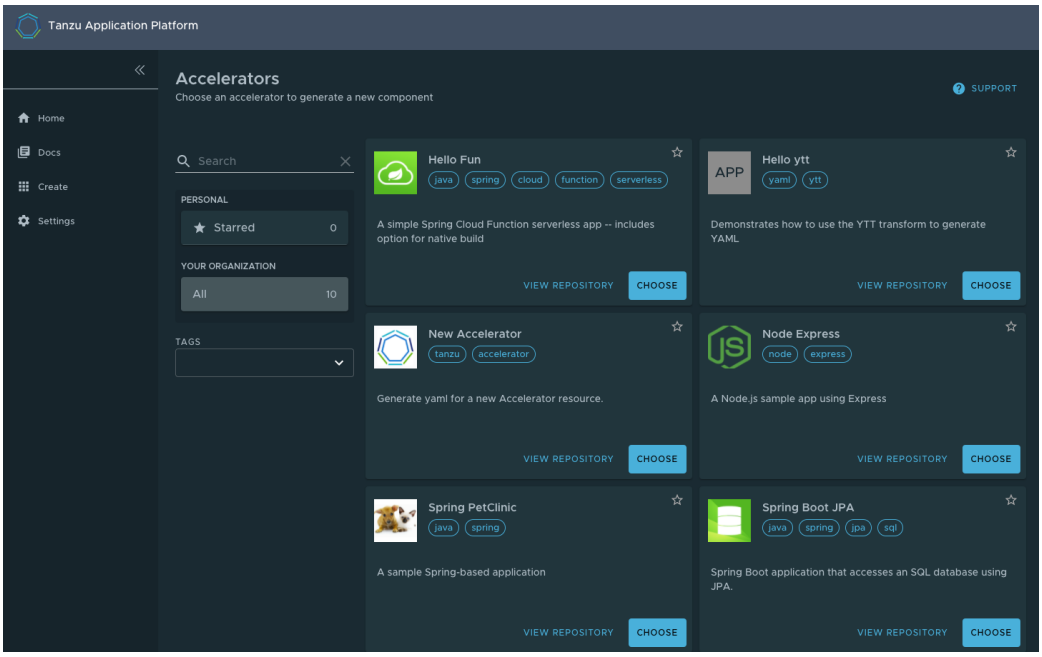
Enterprise architects author and publish accelerator projects that provide developers and operators with ready-made, enterprise-conforming code and configurations. You can then use Application Accelerator to create new projects based on those accelerator projects.

The Application Accelerator UI enables you to discover available accelerators, configure them, and generate new projects to download.

Access Application Accelerator

To open the Application Accelerator UI plug-in and select an accelerator:

1. Within Tanzu Application Platform, click **Create** in the left navigation pane to open the **Accelerators** page.



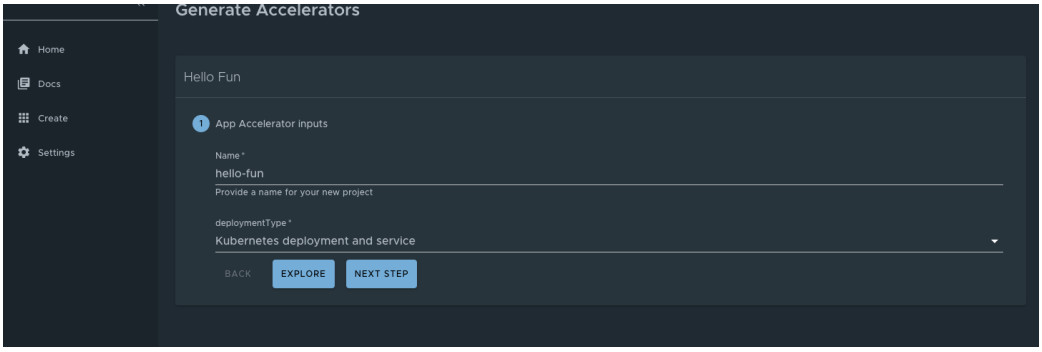
Here you can view accelerators already registered with the system. Developers can add new accelerators by registering them with Kubernetes.

2. Every accelerator has a title and short description. Click **VIEW REPOSITORY** to view an accelerator definition. This opens the accelerator's Git repository in a new browser tab.
3. Search and filter based on text and tags associated with the accelerators to find the accelerator representing the project you want to create.
4. Click **CHOOSE** for the accelerator you want. This opens the **Generate Accelerators** page.

Configure project generation

To configure how projects are generated:

1. On the **Generate Accelerators** page, add any configuration values needed to generate the project. The application architect defined these values in `accelerator.yaml` in the accelerator definition. Filling some text boxes can cause other text boxes to appear. Fill them all in.

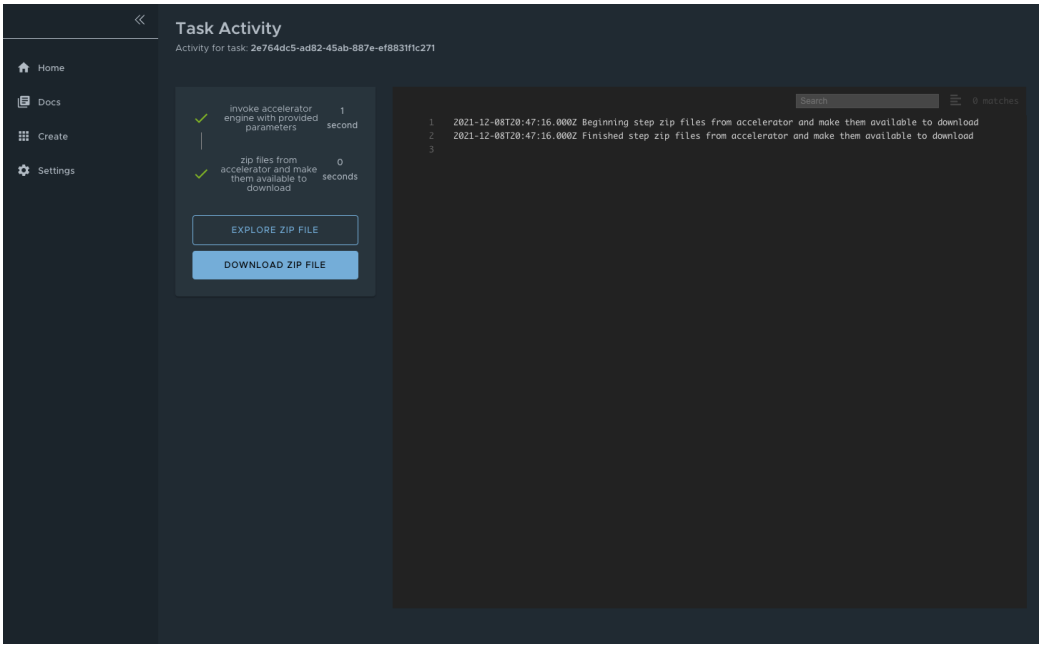


2. Click **EXPLORE** to open the **Explore Project** page and view the project before it is generated.
3. After configuring your project, click **NEXT STEP** to see the project summary page.
4. Review the values you specified for the configurable options.
5. Click **BACK** to make more changes, if necessary. Otherwise, proceed to create the project.

Create the project

To create the project:

1. Click **Create** to start generating your project. See the progress on the **Task Activity** page. A detailed log is displayed on the right.

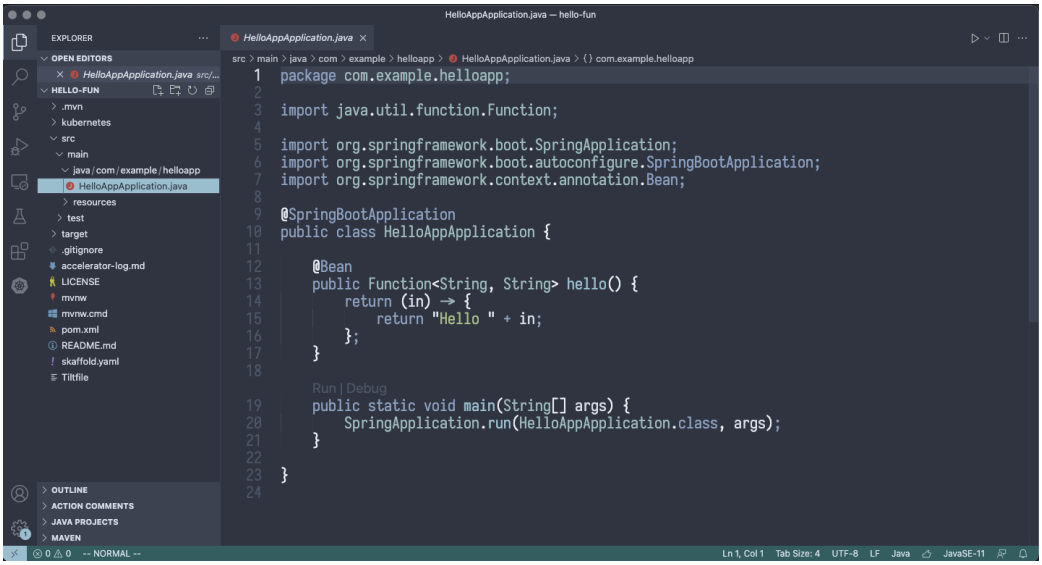


2. After the project is generated, click **EXPLORE ZIP FILE** to open the **Explore Project** page to verify configuration.
3. Click **DOWNLOAD ZIP FILE** to download the project in a ZIP file.

Develop your code

To develop your code:

1. Expand the ZIP file.
2. Open the project in your integrated development environment (IDE).



Next steps

To learn more about Application Accelerator for VMware Tanzu, see the Application Accelerator documentation.

Install Application Accelerator

This topic tells you how to install Application Accelerator from the Tanzu Application Platform (commonly known as TAP) package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Application Accelerator. For more information about profiles, see About Tanzu Application Platform components and profiles.

Prerequisites

Before installing Application Accelerator:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- Install Flux SourceController on the cluster. See Install Flux CD Source Controller.
- Install Source Controller on the cluster. See Install Source Controller.

Install

To install Application Accelerator:

1. List version information for the package by running:

   ```
   tanzu package available list accelerator.apps.tanzu.vmware.com --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package available list accelerator.apps.tanzu.vmware.com --namespace tap-install

   - Retrieving package versions for accelerator.apps.tanzu.vmware.com...
     NAME                                VERSION  RELEASED-AT
     accelerator.apps.tanzu.vmware.com  1.4.0    2022-12-08 12:00:00 -0500 EST
   ```

2. (Optional) View the changes you can make to the default installation settings by running:

   ```
   tanzu package available get accelerator.apps.tanzu.vmware.com/VERSION-NUMBER \
     --values-schema \
     --namespace tap-install
   ```

   Where `VERSION-NUMBER` is the version of the Application Accelerator package listed earlier.

   For example:

   ```
   $ tanzu package available get accelerator.apps.tanzu.vmware.com/1.4.0 \
       --values-schema \
       --namespace tap-install
   ```

For more information about values schema options, see the properties listed in Configure properties and resource use later.

3. Create a file named `app-accelerator-values.yaml` using the following example code:

```
server:
  service_type: "LoadBalancer"
  watched_namespace: "accelerator-system"
samples:
  include: true
```

4. Edit the values in your `app-accelerator-values.yaml` if needed, or leave the default values. You can add values you want from Configure properties and resource use.

5. Install the package by running:

```
tanzu package install app-accelerator \
  --package accelerator.apps.tanzu.vmware.com \
  --version VERSION-NUMBER \
  --namespace tap-install \
  --values-file app-accelerator-values.yaml
```

Where `VERSION-NUMBER` is the version of the Application Accelerator package listed earlier.

For example:

```
$ tanzu package install app-accelerator \
    --package accelerator.apps.tanzu.vmware.com \
    --version 1.4.0 \
    --namespace tap-install \
    --values-file app-accelerator-values.yaml

- Installing package 'accelerator.apps.tanzu.vmware.com'
| Getting package metadata for 'accelerator.apps.tanzu.vmware.com'
| Creating service account 'app-accelerator-tap-install-sa'
| Creating cluster admin role 'app-accelerator-tap-install-cluster-role'
| Creating cluster role binding 'app-accelerator-tap-install-cluster-rolebindin
g'
| Creating secret 'app-accelerator-tap-install-values'
- Creating package resource
- Package install status: Reconciling

 Added installed package 'app-accelerator' in namespace 'tap-install'
```

6. Verify the package install by running:

```
tanzu package installed get app-accelerator -n tap-install
```

For example:

```
$ tanzu package installed get app-accelerator -n tap-install

| Retrieving installation details for cc...
NAME:                    app-accelerator
PACKAGE-NAME:            accelerator.apps.tanzu.vmware.com
PACKAGE-VERSION:         1.4.0
STATUS:                  Reconcile succeeded
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`.

7. To see the IP address for the Application Accelerator API when the `server.service_type` is set to `LoadBalancer`, run:

   ```
   kubectl get service -n accelerator-system
   ```

   This lists an external IP address for use with the `--server-url` Tanzu CLI flag for the Accelerator plug-in `generate` & `generate-from-local` command.

For how to troubleshoot installation issues, see Troubleshoot Application Accelerator.

Configure properties and resource use

When you install the Application Accelerator, you can configure the following optional properties from within your `app-accelerator-values.yaml` configuration file:

Property	Default	Description
`registry.secret_ref`	`registry.tanzu.vmware.com`	The secret used for accessing the registry where the App-Accelerator images are located
`server.service_type`	`ClusterIP`	The service type for the acc-ui-server service including LoadBalancer, NodePort, or ClusterIP
`server.watched_namespace`	`accelerator-system`	The namespace the server watches for accelerator resources
`server.engine_invocation_url`	`http://acc-engine.accelerator-system.svc.cluster.local/invocations`	The URL to use for invoking the accelerator engine
`engine.service_type`	`ClusterIP`	The service type for the acc-engine service including LoadBalancer, NodePort, or ClusterIP
`engine.max_direct_memory_size`	`32M`	The maximum size for the Java -XX:MaxDirectMemorySize setting
`samples.include`	`True`	Option to include the bundled sample Accelerators in the installation
`ingress.include`	`False`	Option to include the ingress configuration in the installation
`ingress.enable_tls`	`False`	Option to include TLS for the ingress configuration
`domain`	`tap.example.com`	Top-level domain to use for ingress configuration, default is `shared.ingress_domain`
`tls.secret_name`	`tls`	The name of the secret
`tls.namespace`	`tanzu-system-ingress`	The namespace for the secret
`telemetry.retain_invocation_events_for_no_days`	`30`	The number of days to retain recorded invocation events resources
`telemetry.record_invocation_events`	`true`	The system records each engine invocation when generating files for an accelerator?
`git_credentials.secret_name`	`git-credentials`	The name to use for the secret storing Git credentials for accelerators
`git_credentials.username`	`null`	The user name to use in secret storing Git credentials for accelerators
`git_credentials.password`	`null`	The password to use in secret storing Git credentials for accelerators

Property	Default	Description
`git_credentials.ca_file`	`null`	The CA certificate data to use in secret storing Git credentials for accelerators
`managed_resources.enable`	`false`	Whether to enable the App used to control managed accelerator resources
`managed_resources.git.url`	none	Required if managed_resources are enabled. Git repository URL containing manifests for managed accelerator resources
`managed_resources.git.ref`	`origin/main`	Required if managed_resources are enabled. Git ref to use for repository containing manifests for managed accelerator resources
`managed_resources.git.sub_path`	`null`	Git subPath to use for repository containing manifests for managed accelerator resources
`managed_resources.git.secret_ref`	`git-credentials`	Secret name to use for repository containing manifests for managed accelerator resources

VMware recommends that you do not override the default setting for `registry.secret_ref`, `server.engine_invocation_url`, or `engine.service_type`. These properties are only used to configure non-standard installations.

The following table is the resource use configurations for the components of Application Accelerator.

Component	Resource requests	Resource limits
acc-controller	CPU: 100m memory: 20Mi	CPU: 100m memory: 30Mi
acc-server	CPU: 100m memory:20Mi	CPU: 100m memory: 30Mi
acc-engine	CPU: 500m memory: 1Gi	CPU: 500m memory: 2Gi

Create an Application Accelerator Git repository during project creation

This topic tells you how to enable and use GitHub repository creation in the Application Accelerator plug-in of Tanzu Developer Portal.

Overview

The Application Accelerator plug-in uses the Backstage GitHub provider integration and the authentication mechanism to retrieve an access token. Then it can interact with the provider API to create GitHub repositories.

Supported Providers

The supported Git providers are GitHub and GitLab.

Note

To create a repository in a self-hosted GitLab, you must add a custom GitLab integration in `tap-values.yaml` as described in the Full Profile sample.

Configure

The following steps describe an example configuration that uses GitHub:

1. Create an **OAuth App** in GitHub based on the configuration described in this Backstage documentation. GitHub Apps are not supported. For more information about creating an OAuth App in GitHub, see the GitHub documentation.

   These values appear in your `app-config.yaml` or `app-config.local.yaml` for local development. For example:

   ```
   auth:
    environment: development
    providers:
      github:
        development:
          clientId: GITHUB-CLIENT-ID
          clientSecret: GITHUB-CLIENT-SECRET
   ```

2. Add a GitHub integration in your `app-config.yaml` configuration. For example:

   ```
   app_config:
    integrations:
      github:
        - host: github.com
   ```

   For more information, see the Backstage documentation.

(Optional) Deactivate Git repository creation

As of Tanzu Application Platform v1.5, you can deactivate Git repository creation by setting the property `customize.features.accelerators.gitRepoCreation` to `false` in `tap-values.yaml`. This also deactivates Git repository creation in the Application Accelerator extension for VS Code.

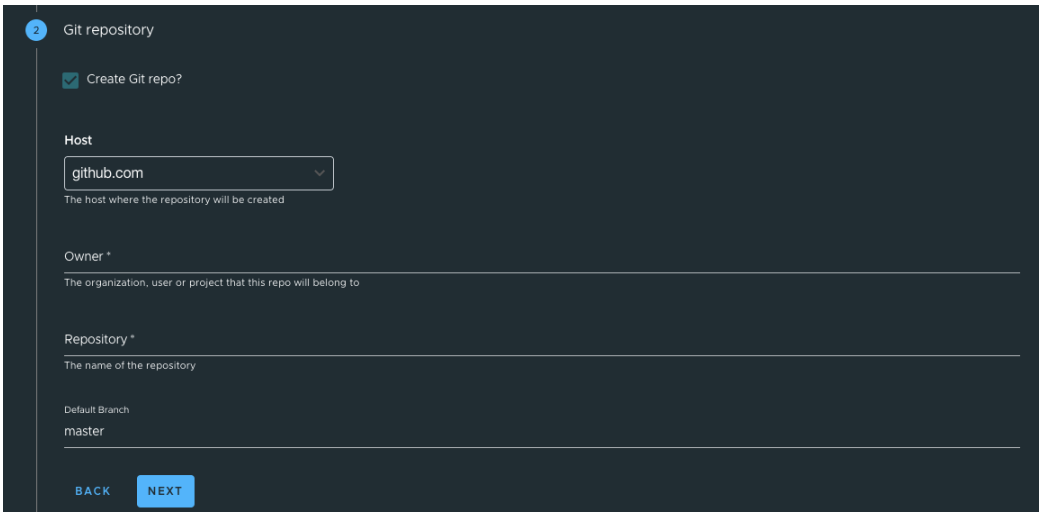
See the following example configuration for deactivating Git repository creation:

```
app_config:
  customize:
    features:
      accelerators:
        gitRepoCreation: false
```

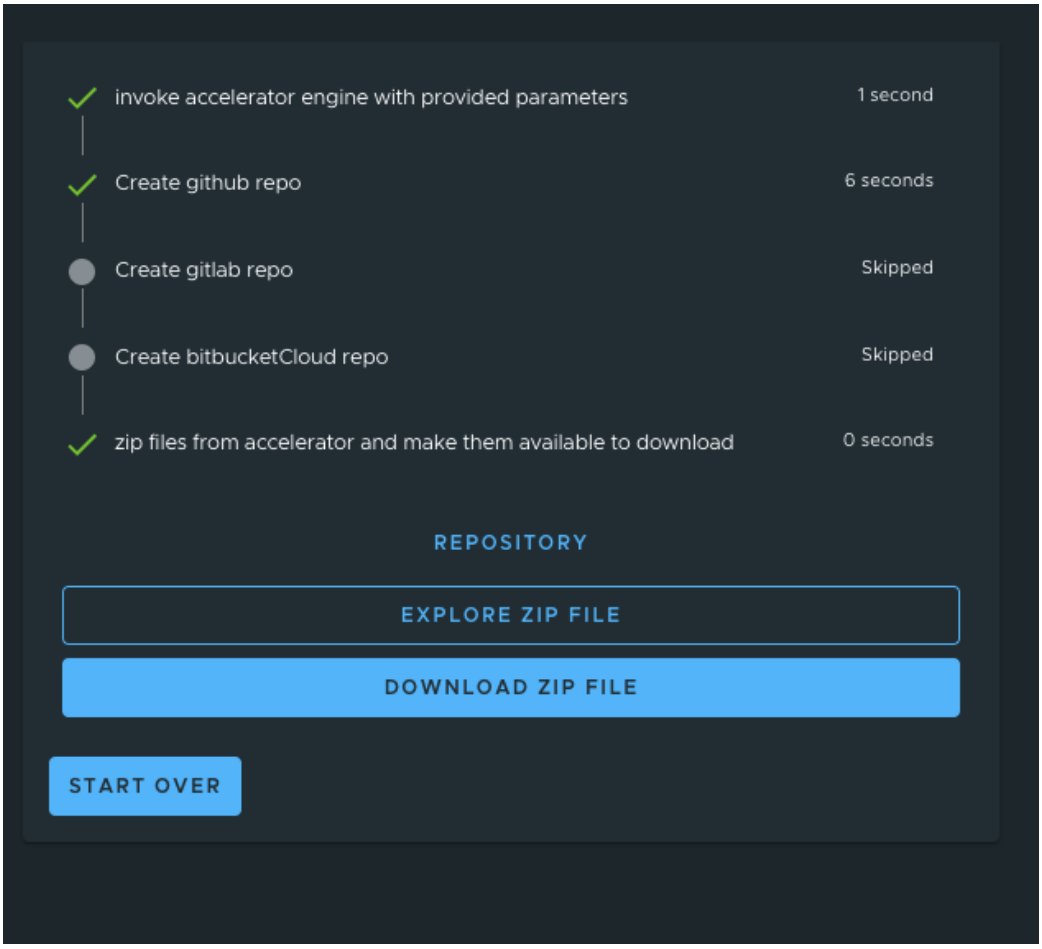
Create a Project

To create a project:

1. Go to Tanzu Developer Portal, access the Accelerators section, and then select an accelerator. The accelerator form now has a second step named **Git repository**.
2. Fill in the accelerator options and click **Next**.
3. Select the **Create Git repo?** check box.
4. Fill in the **Owner**, **Repository**, and **Default Branch** text boxes.



5. After entering the repository name, a dialog box appears that requests GitHub credentials. Log in and then click **Next**.
6. Click **GENERATE ACCELERATOR**. A link to the repository location appears.



API documentation plug-in in Tanzu Developer Portal

This topic gives you an overview of the API documentation plug-in of Tanzu Developer Portal. For more information, see Get started with the API documentation plug-in.

Overview

The API documentation plug-in provides a standalone list of APIs that can be connected to components and systems of the Tanzu Developer Portal software catalog.

Each API entity can reflect the components that provide that API and the list of components that are consumers of that API. Also, an API entity can be associated with systems and appear on the system diagram. To show this dependency, make the `spec.providesApis:` and `spec.consumesApis:` sections of the component definition files reference the name of the API entity.

Here's a sample of how you can add `providesApis` and `consumesApis` to an existing component's catalog definition, linking them together.

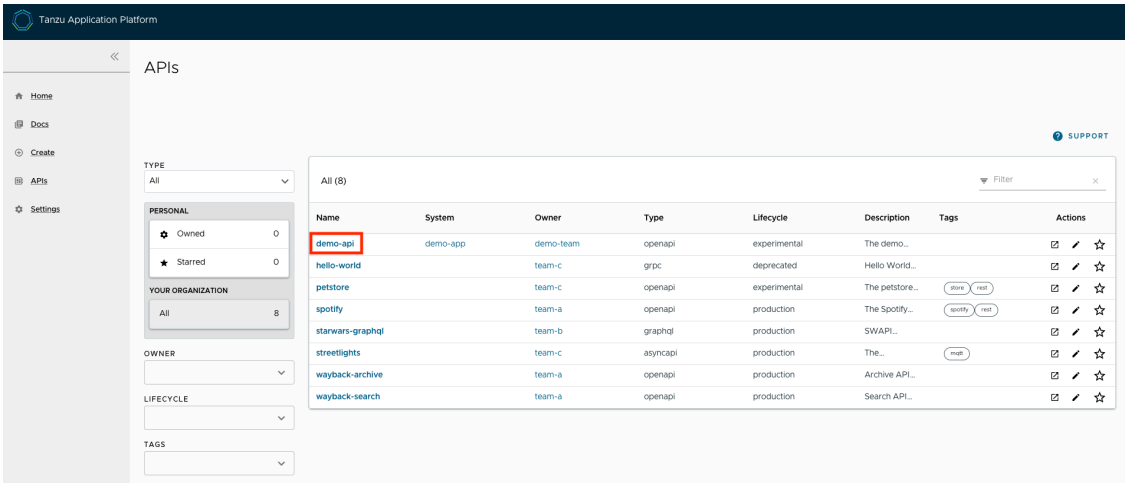
```
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: example-component
  description: Example Component
spec:
  type: service
  lifecycle: experimental
  owner: team-a
  system: example-system
  providesApis: # list of APIs provided by the Component
    - example-api-1
  consumesApis: # list of APIs consumed by the Component
    - example-api-2
```

For more information about the structure of the definition file for an API entity, see the Backstage Kind: API documentation. For more information about the API documentation plug-in, see the Backstage API documentation in GitHub.

Use the API documentation plug-in

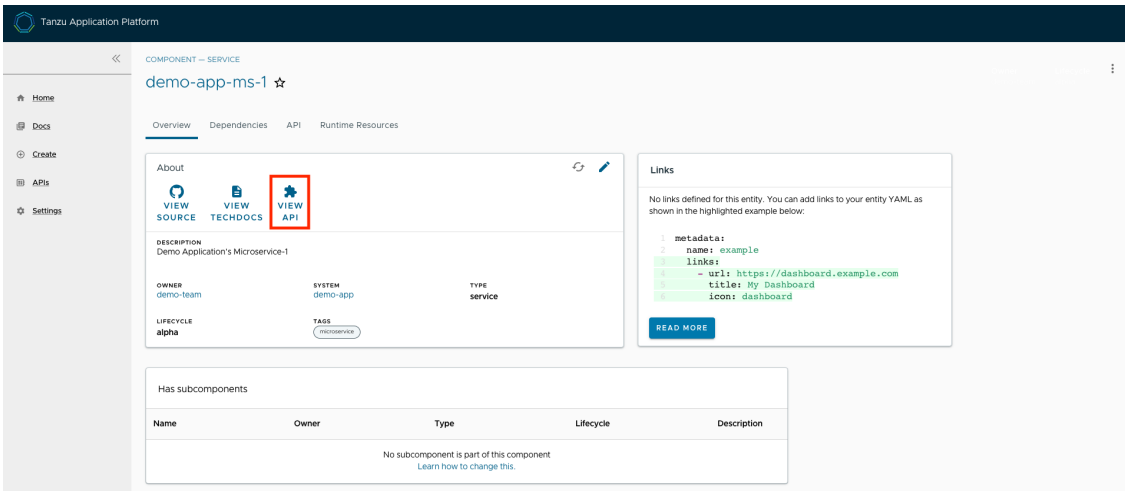
The API documentation plug-in is part of Tanzu Developer Portal.

The first way to use the API documentation plug-in is API-first. Click **APIs** in the left navigation pane of Tanzu Developer Portal. This opens the **API catalog page**.

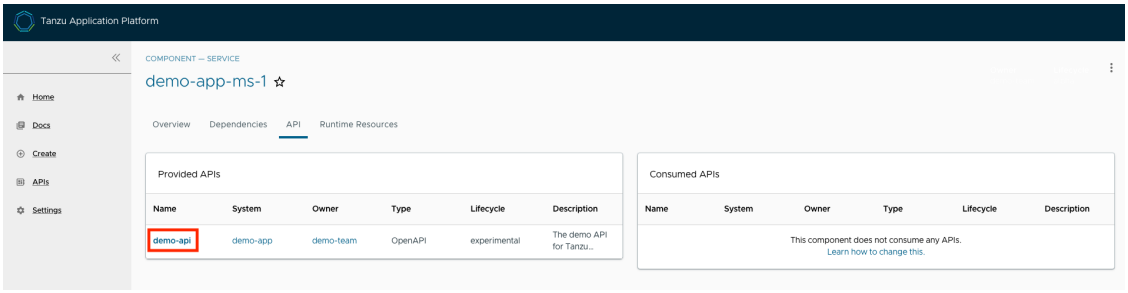


On that page, you can view all the APIs already registered in the catalog regardless of whether they are associated with components or systems.

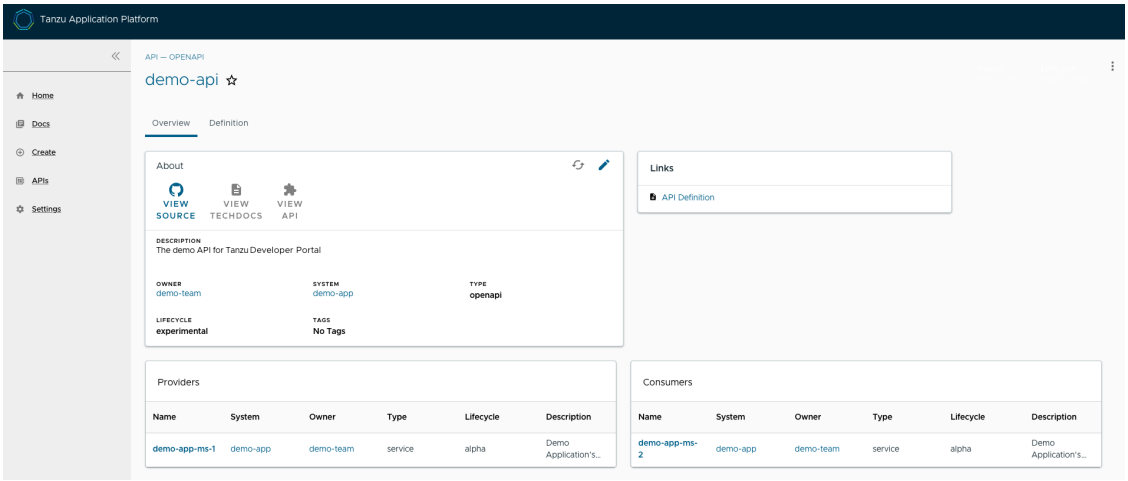
The second way to use the API documentation plug-in is to use components and systems of the software catalog, listed on the home page of Tanzu Developer Portal. If there is an API entity associated with the selected component or system, the **VIEW API** icon is active.



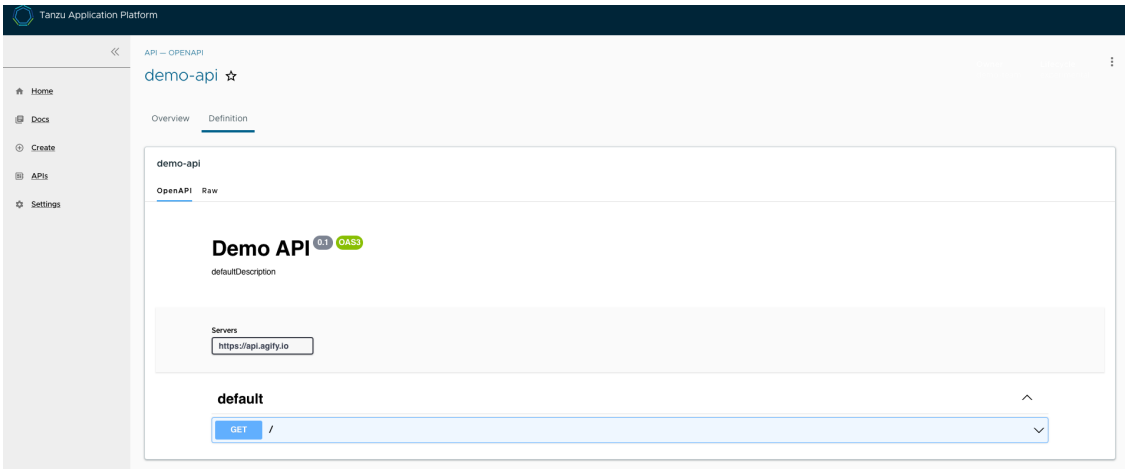
The **VIEW API** tab displays which APIs are being consumed by a component and which APIs are being provided by the component.



Clicking on the API itself takes you to the catalog entry for the API, which the Kind type listed in the upper-left corner denotes. Every API entity has a title and short description, including a reference to the team that owns the definition of that API and the software catalog objects that are connected to it.



Select the **Definition** tab on the top of the API page to see the definition of that API in human-readable and machine-readable format.



The API documentation plug-in supports the following API formats:

- OpenAPI 2 & 3
- AsyncAPI
- GraphQL
- Plain (to support any other format)

Create a new API entry

You can create a new API entry manually or automatically.

Manually create a new API entry

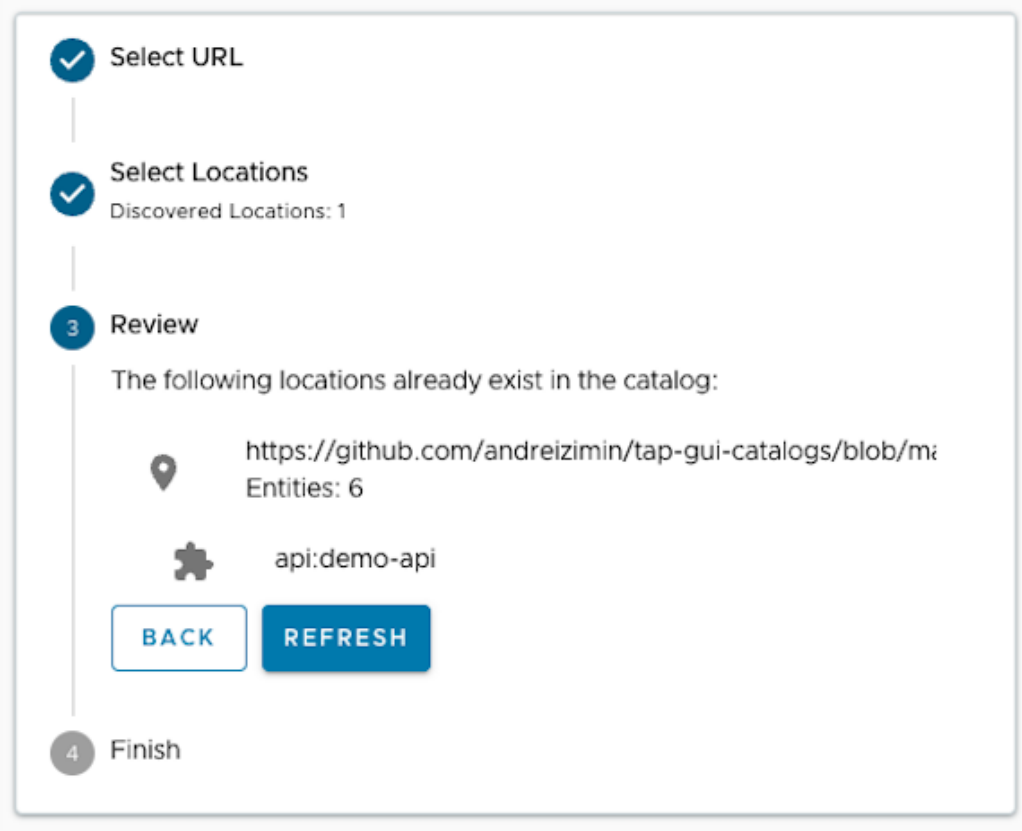
Manually creating a new API entity is similar to registering any other software catalog entity. To manually create a new API entity:

1. Click the **Home** button on the left navigation pane to access the home page of Tanzu Developer Portal.
2. Click **REGISTER ENTITY**.
3. **Register an existing component** prompts you to type a repository URL. Paste the link to the `catalog-info.yaml` file of your choice that contains the definition of your API entity. For example, you can copy the following YAML content and save it as `catalog-info.yaml` on a Git repository of your choice.

```
apiVersion: backstage.io/v1alpha1
kind: API
metadata:
name: demo-api
description: The demo API for Tanzu Developer Portal
links:
  - url: https://api.agify.io
    title: API Definition
    icon: docs
spec:
type: openapi
lifecycle: experimental
owner: demo-team
system: demo-app # Or specify system name of your choice
definition: |
  openapi: 3.0.1
  info:
    title: defaultTitle
    description: defaultDescription
```

```
  version: '0.1'
servers:
  - url: https://api.agify.io
paths:
  /:
    get:
      description: Auto generated using Swagger Inspector
      parameters:
        - name: name
          in: query
          schema:
            type: string
          example: type_any_name
      responses:
        '200':
          description: Auto generated using Swagger Inspector
          content:
            application/json; charset=utf-8:
              schema:
                type: string
              examples: {}
```

4. Click **ANALYZE** and then review the catalog entities to be added.



5. Click **IMPORT**.
6. Click **APIs** on the left navigation pane to view entries on the **API** page.

Automatically create a new API entry

Tanzu Application Platform v1.3 introduced a feature called **API Auto Registration** that can automatically register your APIs. For more information, see API Auto Registration.

API documentation plug-in in Tanzu Developer Portal

This topic gives you an overview of the API documentation plug-in of Tanzu Developer Portal. For more information, see Get started with the API documentation plug-in.

Overview

The API documentation plug-in provides a standalone list of APIs that can be connected to components and systems of the Tanzu Developer Portal software catalog.

Each API entity can reflect the components that provide that API and the list of components that are consumers of that API. Also, an API entity can be associated with systems and appear on the system diagram. To show this dependency, make the `spec.providesApis:` and `spec.consumesApis:` sections of the component definition files reference the name of the API entity.

Here's a sample of how you can add `providesApis` and `consumesApis` to an existing component's catalog definition, linking them together.

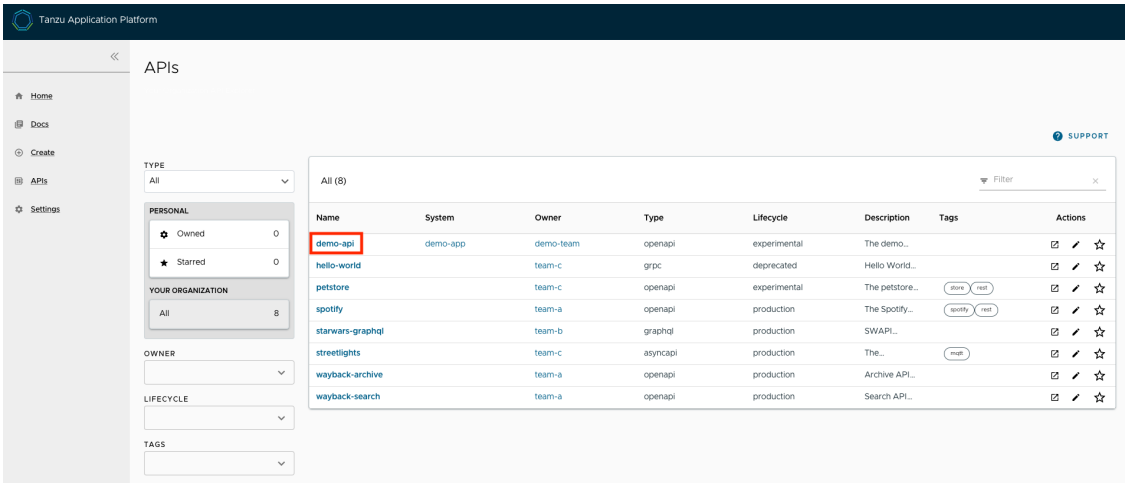
```
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: example-component
  description: Example Component
spec:
  type: service
  lifecycle: experimental
  owner: team-a
  system: example-system
  providesApis: # list of APIs provided by the Component
    - example-api-1
  consumesApis: # list of APIs consumed by the Component
    - example-api-2
```

For more information about the structure of the definition file for an API entity, see the Backstage Kind: API documentation. For more information about the API documentation plug-in, see the Backstage API documentation in GitHub.

Use the API documentation plug-in

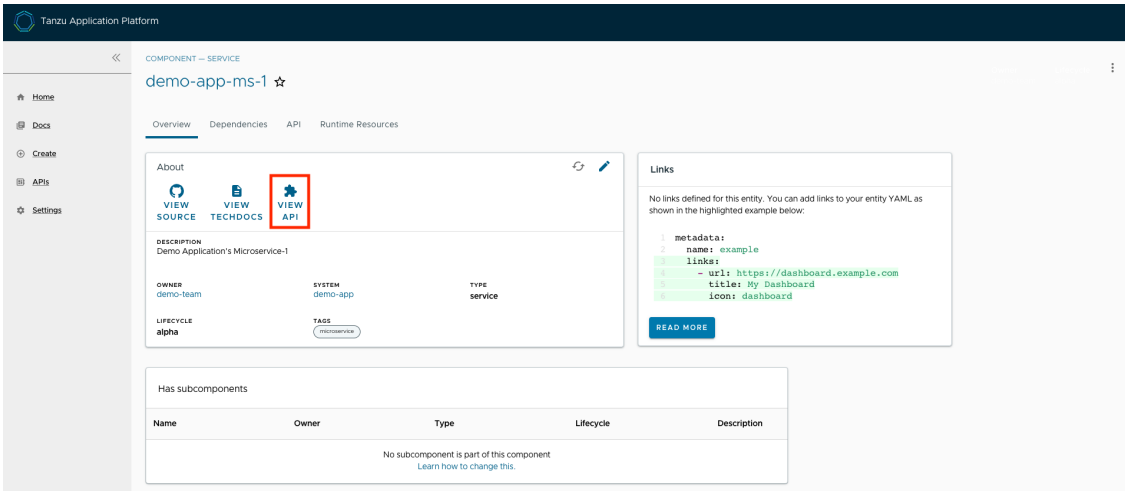
The API documentation plug-in is part of Tanzu Developer Portal.

The first way to use the API documentation plug-in is API-first. Click **APIs** in the left navigation pane of Tanzu Developer Portal. This opens the **API catalog page**.

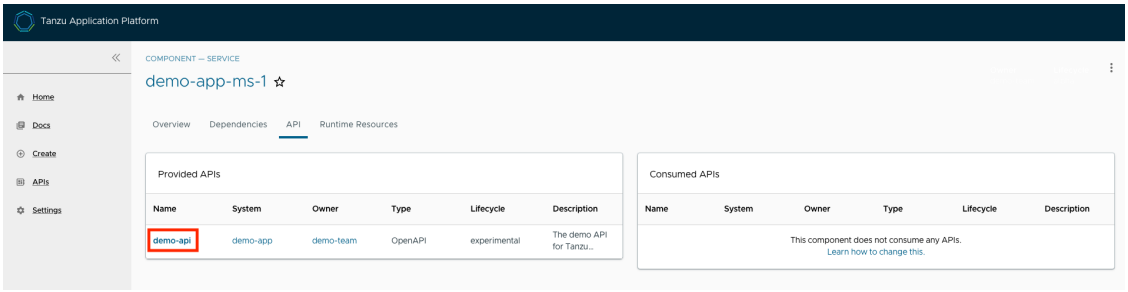


On that page, you can view all the APIs already registered in the catalog regardless of whether they are associated with components or systems.

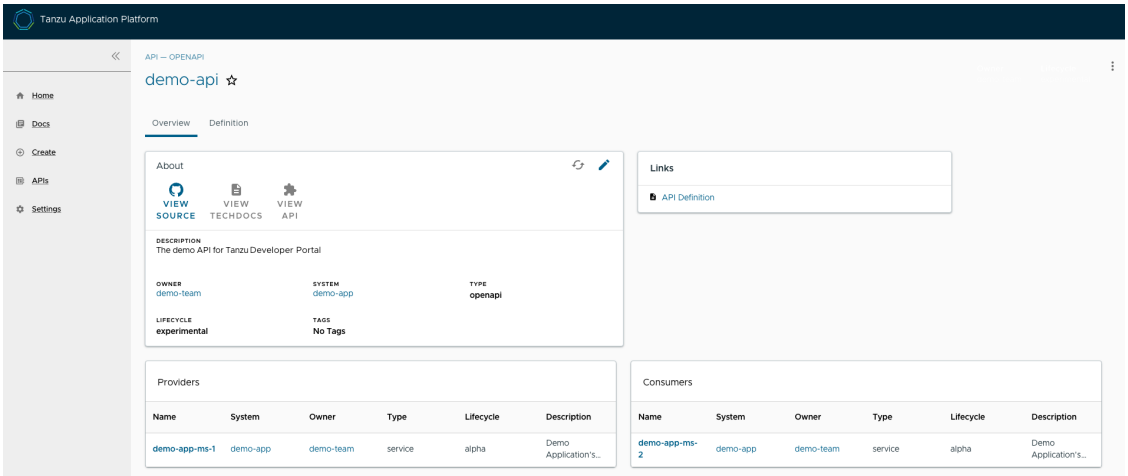
The second way to use the API documentation plug-in is to use components and systems of the software catalog, listed on the home page of Tanzu Developer Portal. If there is an API entity associated with the selected component or system, the **VIEW API** icon is active.



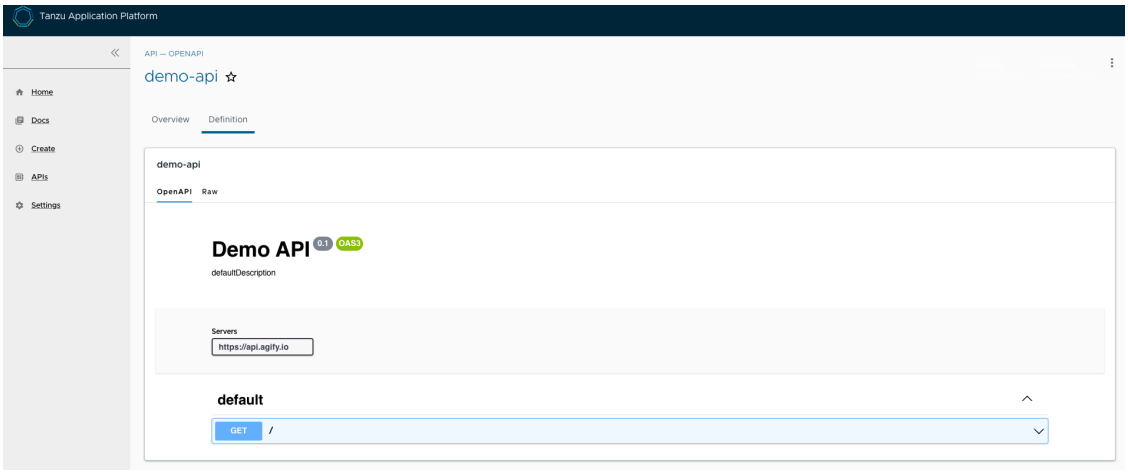
The **VIEW API** tab displays which APIs are being consumed by a component and which APIs are being provided by the component.



Clicking on the API itself takes you to the catalog entry for the API, which the Kind type listed in the upper-left corner denotes. Every API entity has a title and short description, including a reference to the team that owns the definition of that API and the software catalog objects that are connected to it.



Select the **Definition** tab on the top of the API page to see the definition of that API in human-readable and machine-readable format.



The API documentation plug-in supports the following API formats:

- OpenAPI 2 & 3
- AsyncAPI
- GraphQL
- Plain (to support any other format)

Create a new API entry

You can create a new API entry manually or automatically.

Manually create a new API entry

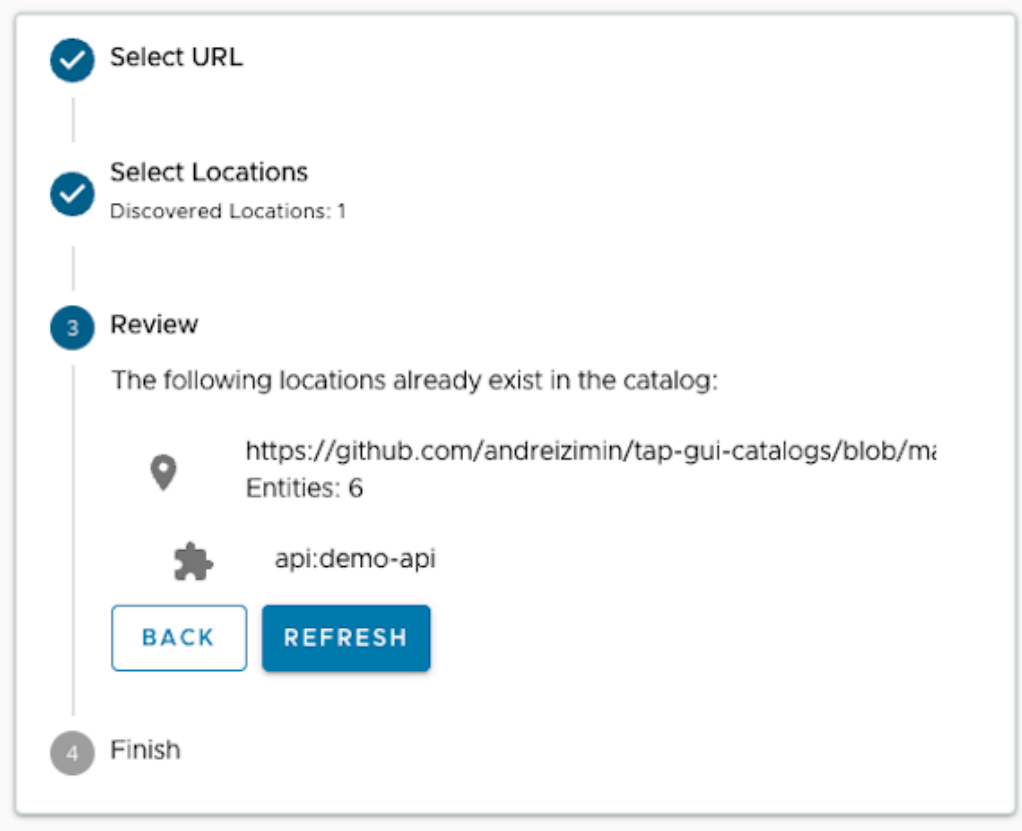
Manually creating a new API entity is similar to registering any other software catalog entity. To manually create a new API entity:

1. Click the **Home** button on the left navigation pane to access the home page of Tanzu Developer Portal.
2. Click **REGISTER ENTITY**.
3. **Register an existing component** prompts you to type a repository URL. Paste the link to the `catalog-info.yaml` file of your choice that contains the definition of your API entity. For example, you can copy the following YAML content and save it as `catalog-info.yaml` on a Git repository of your choice.

```
apiVersion: backstage.io/v1alpha1
kind: API
metadata:
name: demo-api
description: The demo API for Tanzu Developer Portal
links:
  - url: https://api.agify.io
    title: API Definition
    icon: docs
spec:
type: openapi
lifecycle: experimental
owner: demo-team
system: demo-app # Or specify system name of your choice
definition: |
  openapi: 3.0.1
  info:
    title: defaultTitle
    description: defaultDescription
```

```
  version: '0.1'
servers:
  - url: https://api.agify.io
paths:
  /:
    get:
      description: Auto generated using Swagger Inspector
      parameters:
        - name: name
          in: query
          schema:
            type: string
          example: type_any_name
      responses:
        '200':
          description: Auto generated using Swagger Inspector
          content:
            application/json; charset=utf-8:
              schema:
                type: string
              examples: {}
```

4. Click **ANALYZE** and then review the catalog entities to be added.



5. Click **IMPORT**.
6. Click **APIs** on the left navigation pane to view entries on the **API** page.

Automatically create a new API entry

Tanzu Application Platform v1.3 introduced a feature called **API Auto Registration** that can automatically register your APIs. For more information, see API Auto Registration.

Get started with the API documentation plug-in

This topic tells you how to get started with the API documentation plug-in in Tanzu Developer Portal.

API entries

This section describes API entities, how to add them, and how to update them.

About API entities

The list of API entities is visible on the left side navigation pane of Tanzu Developer Portal. It is also visible on the Overview page of specific components on the home page. APIs are a definition of the interface between components.

Their definition is provided in raw machine-readable and human-readable formats. For more information, see the API plug-in documentation.

Add a demo API entity to the Tanzu Developer Portal software catalog

To add a demo API entity and its related Catalog objects, follow the steps used for registering any other software catalog entity:

1. Go to the Home page of Tanzu Developer Portal by clicking **Home** on the left-side navigation pane.
2. Click **REGISTER ENTITY**.
3. In the repository URL text box, type the link to the `catalog-info.yaml` file of your choice or use the following sample definition.
4. Save this code block as `catalog-info.yaml`.
5. Upload it to the Git repository of your choice and copy the link to `catalog-info.yaml`. This demo setup includes a domain named `demo-domain` with a single system named `demo-system`. This systems consists of two microservices (`demo-app-ms-1` and `demo-app-ms-1`) and one API named `demo-api` that `demo-app-ms-1` provides and that `demo-app-ms-2` consumes.

```
apiVersion: backstage.io/v1alpha1
kind: Domain
metadata:
  name: demo-domain
  description: Demo Domain for Tanzu Application Platform
  annotations:
    'backstage.io/techdocs-ref': dir:.
spec:
  owner: demo-team

---

apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: demo-app-ms-1
  description: Demo Application's Microservice-1
  tags:
    - microservice
  annotations:
    'backstage.io/kubernetes-label-selector': 'app.kubernetes.io/part-of=demo-a
pp-ms-1'
    'backstage.io/techdocs-ref': dir:.
spec:
  type: service
```

```
  providesApis:
   - demo-api
  lifecycle: alpha
  owner: demo-team
  system: demo-app

---

apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: demo-app-ms-2
  description: Demo Application's Microservice-2
  tags:
    - microservice
  annotations:
    'backstage.io/kubernetes-label-selector': 'app.kubernetes.io/part-of=demo-a
pp-ms-2'
    'backstage.io/techdocs-ref': dir:.
spec:
  type: service
  consumesApis:
   - demo-api
  lifecycle: alpha
  owner: demo-team
  system: demo-app

---

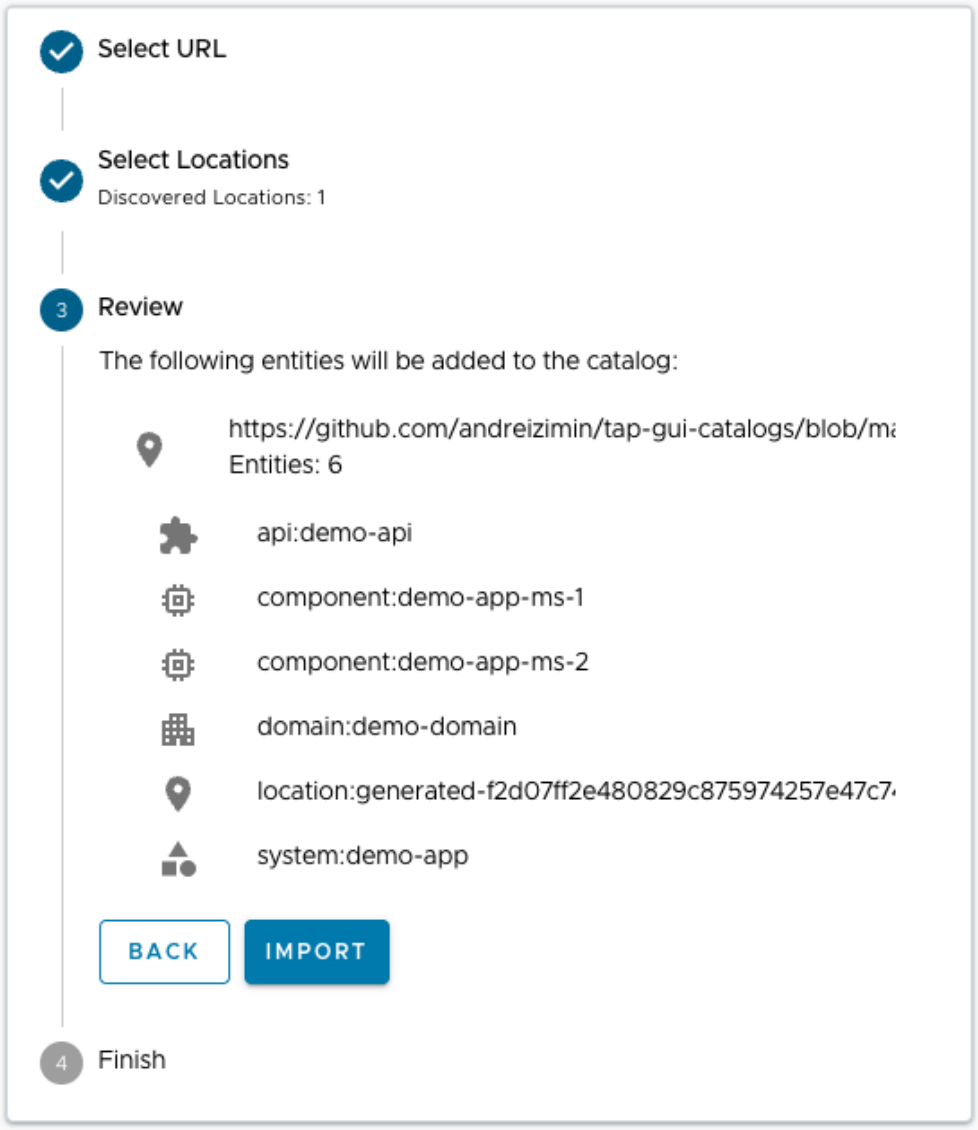
apiVersion: backstage.io/v1alpha1
kind: System
metadata:
  name: demo-app
  description: Demo Application for Tanzu Application Platform
  annotations:
    'backstage.io/techdocs-ref': dir:.
spec:
  owner: demo-team
  domain: demo-domain

---

apiVersion: backstage.io/v1alpha1
kind: API
metadata:
  name: demo-api
  description: The demo API for Tanzu Developer Portal
  links:
    - url: https://api.agify.io
      title: API Definition
      icon: docs
spec:
  type: openapi
  lifecycle: experimental
  owner: demo-team
  system: demo-app # Or specify system name of your choice
  definition: |
    openapi: 3.0.1
    info:
      title: Demo API
      description: defaultDescription
      version: '0.1'
    servers:
      - url: https://api.agify.io
    paths:
      /:
```

```
    get:
      description: Auto generated using Swagger Inspector
      parameters:
        - name: name
          in: query
          schema:
            type: string
          example: type_any_name
      responses:
        '200':
          description: Auto generated using Swagger Inspector
          content:
            application/json; charset=utf-8:
              schema:
                type: string
              examples: {}
```

6. Paste the link to `catalog-info.yaml` and click **ANALYZE**.
7. Review the catalog entities and click **IMPORT**.



8. Go to the **API** page by clicking **APIs** on the left side navigation pane. The catalog changes and entries are visible for further inspection. If you select the system **demo-app**, the diagram appears as follows:

Update your demo API entry

To update your demo API entry, click **demo-api** in the list of available APIs in your software catalog and click the **Edit** icon on the Overview page.

It opens the source `catalog-info.yaml` file that you can edit. For example, you can change the `spec.paths.parameters.example` from `type_any_name` to `Tanzu` and then save your changes.

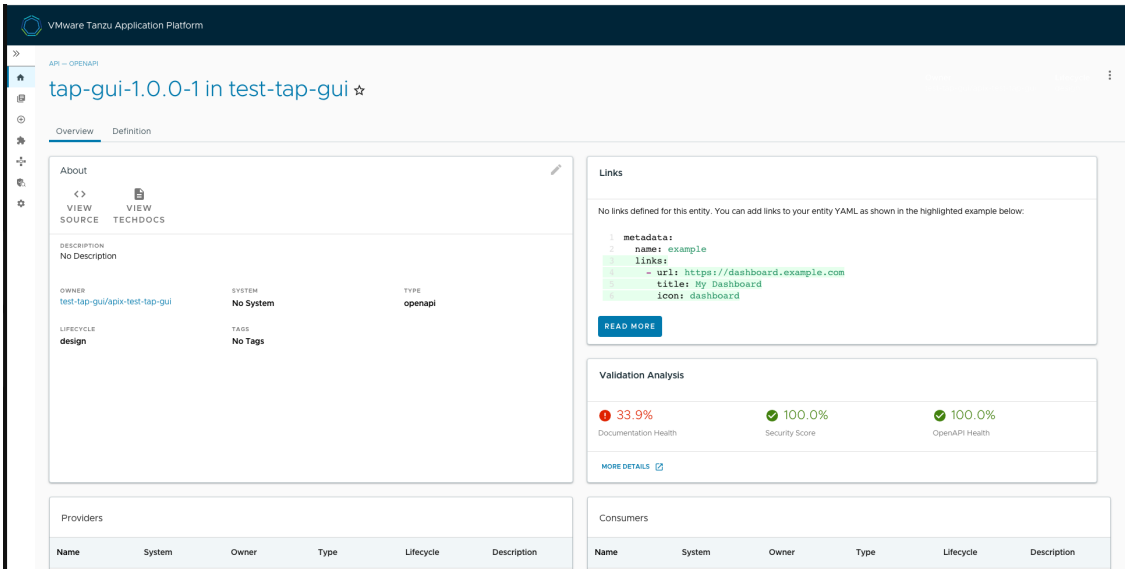
After making any edits, Tanzu Developer Portal re-renders the API entry with the next refresh cycle.

Validation Analysis of API specifications

This section describes the Validation Analysis card, the data format needed to populate the card, and how to get automatic scores for your OpenAPI entities.

About the Validation Analysis card

When viewing entities of the kind `API` on the Overview tab, you see the Validation Analysis card that displays the health of an API through various scoring parameters.



To display the health scores, an API entity must contain the following metadata structure:

```
apiVersion: backstage.io/v1alpha1
kind: API
metadata:
  name: NAME
  description: DESCRIPTION
  apiscores:
    scores:
    - id: documentationReport
      label: "Documentation Health"
      value: 34.375
      valueType: percentage
      status: failed
    - id: securityReport
      label: "Security Score"
      value: 70.0
      valueType: percentage
      status: warning
    - id: openApiHealthReport
      label: "OpenAPI Health"
      value: 89.0625
      valueType: percentage
      status: passed
    scoreDetailsURL:  VALIDATION-REPORT-URL-FOR-MORE-DETAILS
# Other API Entity parameters
```

If an API entity follows this schema, the Validation Analysis card displays helpful information about the API.

```
    - id:        # Unique ID
      label:     # Descriptive label displayed as a title over the numerical value
      value:     # Any number value
      valueType: # One of the types (percentage or other). Displays the % symbol or di
splays nothing.
      status:    # One of the statuses (passed, warning, or failed). Displays the numb
er in green, yellow, or red.
```

DORA metrics in Tanzu Developer Portal

This topic tells you about viewing DevOps Research and Assessment (DORA) metrics in Tanzu Developer Portal.

Overview

DORA is a research program for studying the capabilities that drive software delivery and operations performance. DORA helps teams apply these capabilities to improve organizational performance. For more information about DORA, see the DORA website.

DORA metrics are a set of key performance indicators (KPIs) that DORA has developed to measure the effectiveness of an organization's DevOps practices. These metrics help organizations to assess their software development and delivery processes and identify areas for improvement.

Collecting DORA metrics can be challenging because it involves:

- Gathering data from various sources and tools
- Ensuring data accuracy and consistency
- Dealing with organizational and cultural resistance to measurement and improvement

Tanzu Application Platform is uniquely positioned to provide DORA metrics through its integrated supply chain. This supply chain offers end-to-end visibility and control over the entire development

and deployment process, enabling comprehensive measurement and optimization of DevOps practices.

DORA metrics

DORA metrics include:

- Deployment Frequency, which measures how often code changes are deployed to an environment. High deployment frequency is often associated with a mature DevOps culture.
- Lead Time for Changes, which measures the time it takes to go from code committed to code successfully running in an environment. Shortening this lead time is often a goal of DevOps practices.
- Change Failure Rate, which measures the rate at which changes to the production environment cause failures or incidents. Lower failure rates indicate a reliable software delivery process.
- Mean Time to Recovery (MTTR), which measures how quickly an organization can recover from incidents or outages in production. A lower MTTR suggests that an organization resolves issues quickly.

Supported DORA metrics

This table shows supported DORA metrics in the Tanzu Application Platform v1.9.1 DORA plug-in. Support for more metrics is planned for later DORA plug-in versions.

DORA metric	Tanzu Application Platform v1.9.1 DORA plug-in support
Deployment Frequency	Yes
Lead Time for Changes	Yes
Change Failure Rate	No
Mean Time to Recovery	No

Use the DORA plug-in

To use the DORA plug-in:

1. Select the component you want to view DORA metrics for.
2. Click the **DORA** tab in the navigation list.

DORA metric calculation

By default, DORA metrics are calculated from the average number of deployments to all environments in the last 7 days. Two filtering options are available from drop-down menus:

- By date range, based on a predefined set of choices
- By environment, based on the location labels configured by a platform engineer

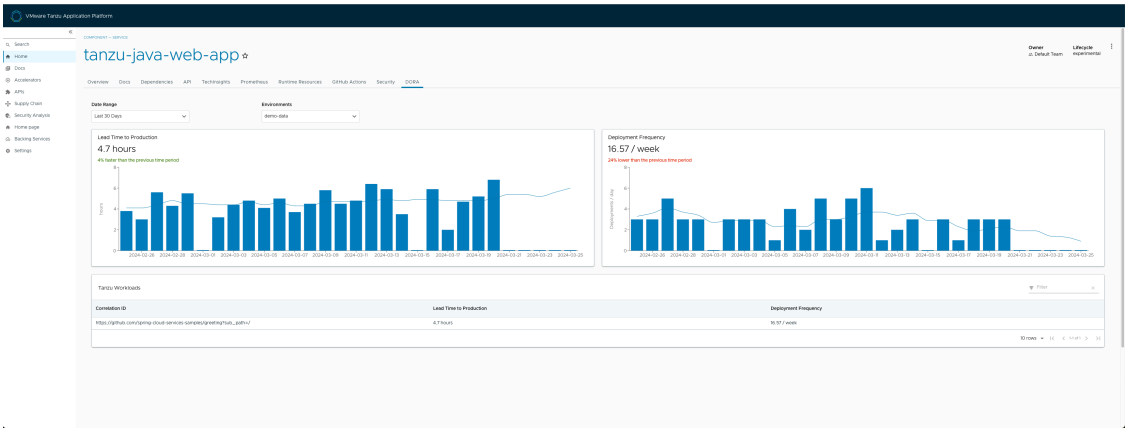
For more information about location configuration, see Configure Artifact Metadata Repository.

DORA graphs display trends of Lead Time and Deployment Frequency metrics over time.

The Lead Time graph displays in bars the daily average time taken for a code change to go from a commit to a running container. The trend line illustrates the average lead time for deployments made in the last 7 days.

The Deployment Frequency graph displays the frequency of code changes deployed to production, depicted in daily bars and a corresponding 7-day moving average trend line.

Tanzu workloads have a correlation ID that groups all the artifacts together.



Security Analysis in Tanzu Developer Portal

This topic tells you about the Security Analysis plug-in in Tanzu Developer Portal.

Overview

The Security Analysis plug-in summarizes vulnerability data across all workloads running in Tanzu Application Platform, enabling faster identification and remediation of CVEs.

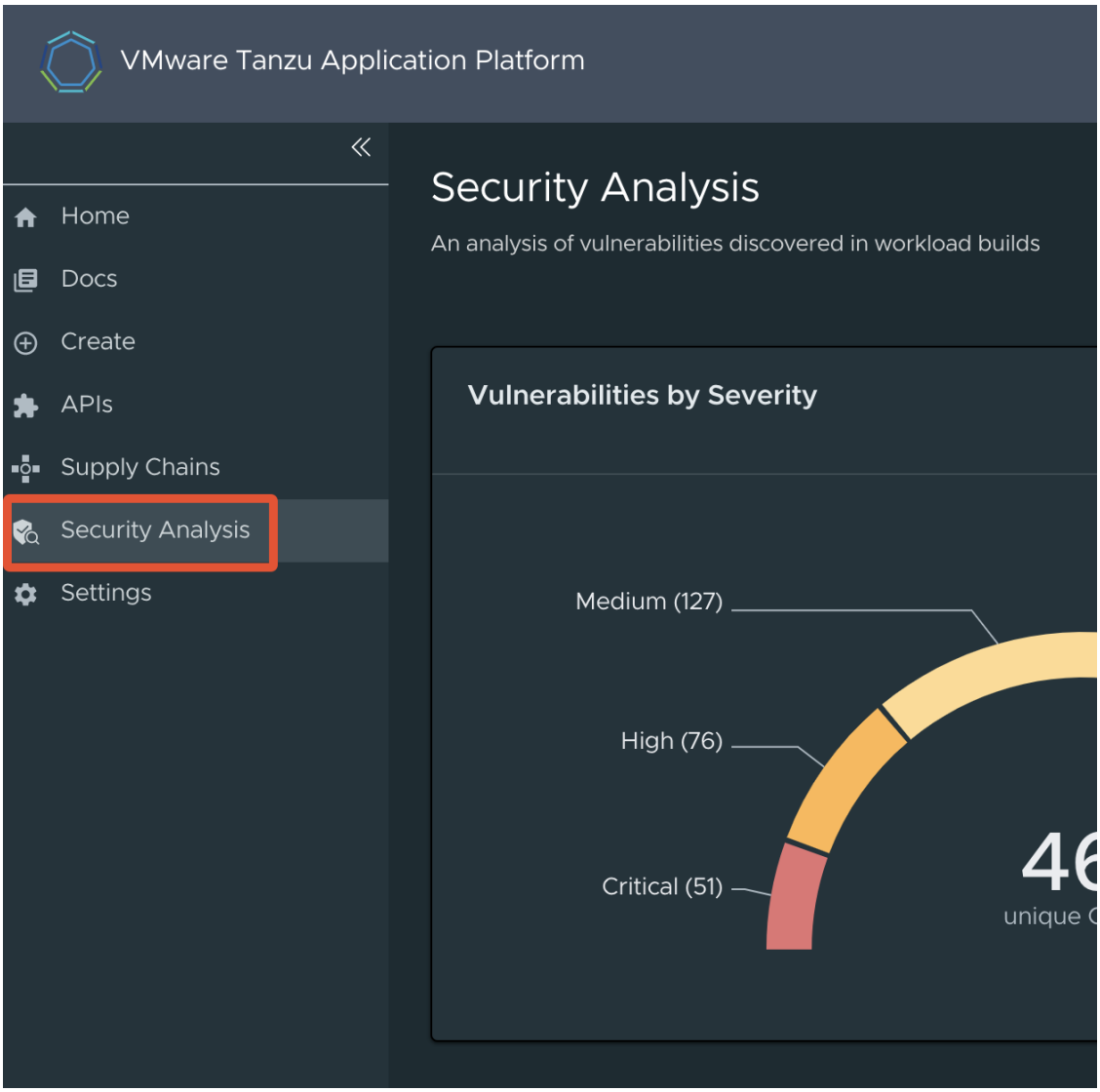
Installing and configuring

The Security Analysis plug-in is installed by default. It is tightly coupled with the Supply Chain Choreographer plug-in. After installing and configuring the Supply Chain Choreographer GUI plug-in, there is no additional configuration needed for the Security Analysis plug-in.

The Security Analysis plug-in is part of the Tanzu Application Platform Full and View profiles.

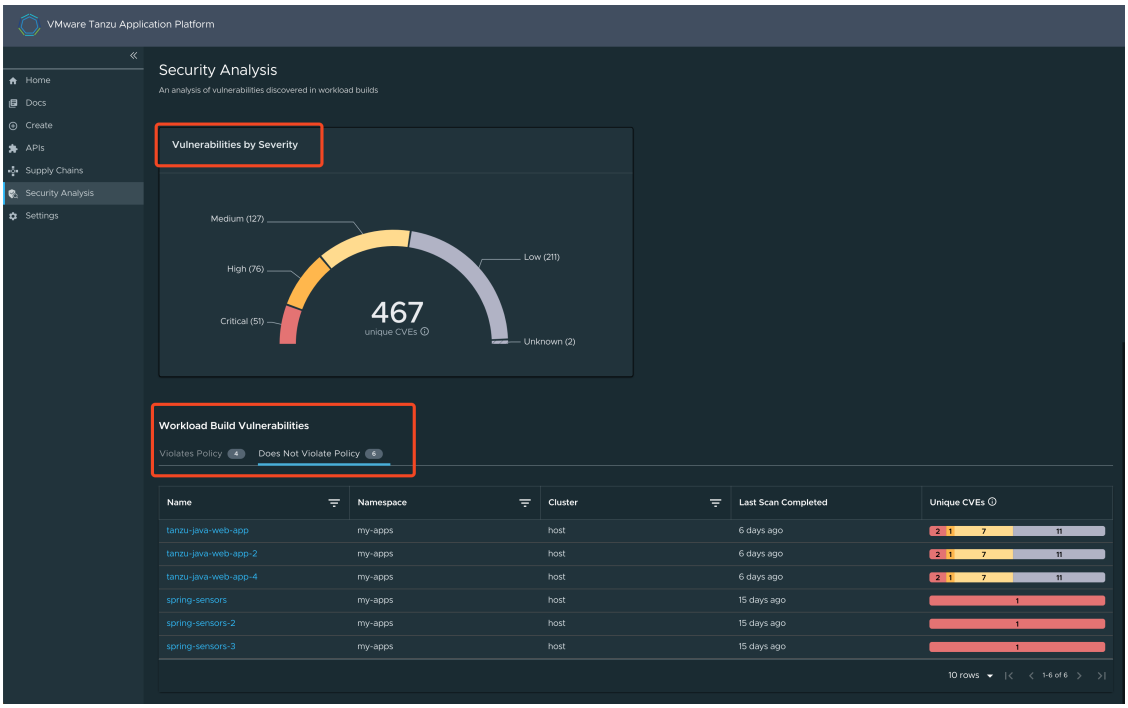
Accessing the plug-in

The Security Analysis plug-in is always accessible from the left navigation pane. Click the **Security Analysis** button to open the **Security Analysis** dashboard.



Viewing vulnerability data

The **Security Analysis** dashboard provides a summary of all vulnerabilities across all clusters for single-cluster and multicluster deployments.



The **Vulnerabilities by Severity** widget quickly counts the number of critical, high, medium, low, and unknown severity CVEs, based on the CVSS severity rating of each CVE.

It includes a sum of all workloads' source and image scan vulnerabilities. For example, if CVE-123 exists in the latest source scans and image scans of Workload ABC and Workload DEF, it is counted four times.

Note

The sum includes any CVEs on the allowlist (ignoreCVEs).

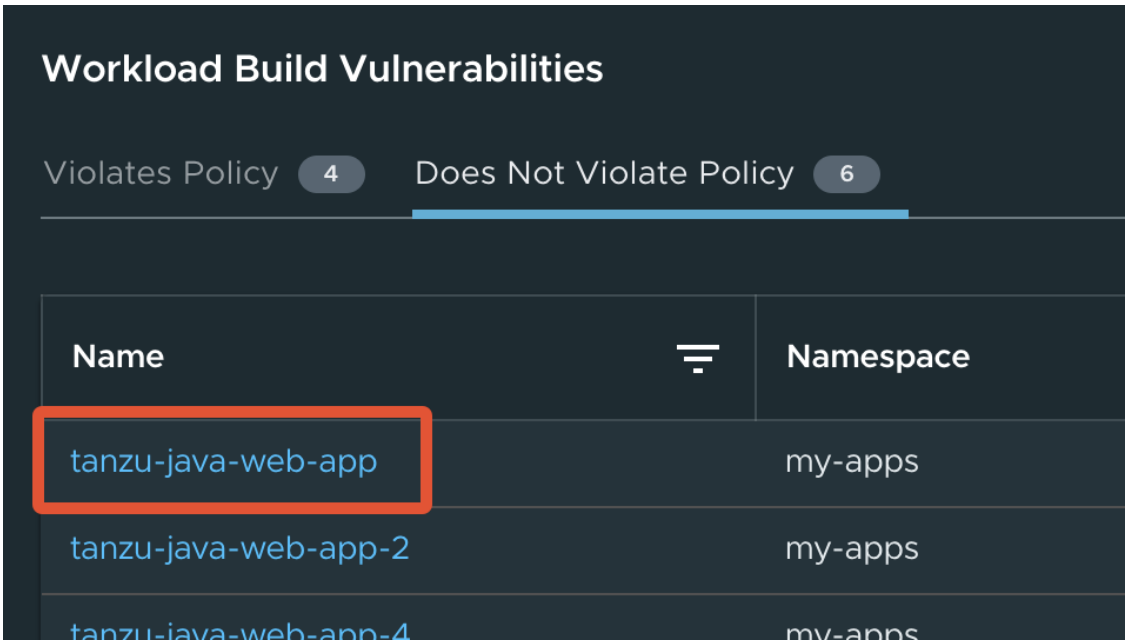
The **Workload Build Vulnerabilities** tables, with the **Violates Policy** tab and **Does Not Violate** tab, separate workloads based on the scan policy. For more information, see Enforce compliance policy using Open Policy Agent The Unique CVEs column uses the same sum logic as described earlier, but for individual workloads.

The sum of a workload's CVEs might not match the Supply Chain Choreographer's Vulnerability Scan Results. The data on this dashboard is based on `kubectl describe` for `SourceScan` and `ImageScan`. The data on the Supply Chain Choreographer's Vulnerability Scan Results is based on Metadata Store data.

Only vulnerability scans associated with a Cartographer workload appear. Use tanzu insight to view results for non-workload scan results.

Viewing CVE and package details

The Security Analysis plug-in has a **CVE** page and a **Package** page. These are accessed by clicking on a workload name, which opens the Supply Chain Choregrapher plug-in. Clicking on the CVE or Package name opens the **CVE** or **Package** page, respectively.



The **CVE** page contains basic information about the vulnerability and includes a table with all affected packages and versions.

The **Package** page contains basic information about a package and includes a table with all CVEs and the affected package versions.

Supply Chain Choreographer in Tanzu Developer Portal

This topic tells you about Supply Chain Choreographer in Tanzu Developer Portal.

Overview

The Supply Chain Choreographer (SCC) plug-in enables you to visualize the execution of a workload by using any of the installed Out-of-the-Box supply chains. For more information about the Out-of-the-Box (OOTB) supply chains that are available in Tanzu Application Platform, see Supply Chain Choreographer for Tanzu.

Prerequisites

To use Supply Chain Choreographer in Tanzu Developer Portal you must have:

- One of the following installed on your cluster:
  - Tanzu Application Platform Full profile
  - Tanzu Application Platform View profile
  - Tanzu Developer Portal package and a metadata store package
- One of the following installed on the target cluster where you want to deploy your workload:
  - Tanzu Application Platform Run profile
  - Tanzu Application Platform Full profile

For more information, see Overview of multicluster Tanzu Application Platform

Enable CVE scan results

To see CVE scan results within Tanzu Developer Portal, connect Tanzu Developer Portal to the Tanzu Application Platform component Supply Chain Security Tools - Store (SCST - Store).

Automatically connect Tanzu Developer Portal to SCST - Store

Tanzu Developer Portal has automation for enabling connection between Tanzu Developer Portal and SCST - Store. This automation is active by default and requires no configuration.

Important

There is a known issue with the automatic configuration breaking the SBOM download feature introduced in Tanzu Application Platform v1.6. To fix this issue, edit `tap-values.yaml` as described in Troubleshooting.

To deactivate this automation, add the following block to the Tanzu Developer Portal section within `tap-values.yaml`:

```
# ...
tap_gui:
  # ...
  metadataStoreAutoconfiguration: false
```

This file change creates a service account for the connection with privileges scoped only to Metadata Store. In addition, it mounts the token of the service account into the Tanzu Developer Portal pod and produces for you the `app_config` section necessary for Tanzu Developer Portal to communicate with SCST - Store.

Troubleshooting

For debugging the automation, or for verifying that the automation is active, you must know which resources are created. The following commands display the different Kubernetes resources that are created when `tap_gui.metadataStoreAutoconfiguration` is set to `true`:

```
$ kubectl -n tap-gui get serviceaccount metadata-store
NAME             SECRETS   AGE
metadata-store   1         AGE-VALUE
```

```
$ kubectl -n tap-gui get secret metadata-store-access-token
NAME                          TYPE                                  DATA   AGE
metadata-store-access-token   kubernetes.io/service-account-token   3      AGE-VALUE
```

```
$ kubectl -n tap-gui get clusterrole metadata-store-reader
NAME                     CREATED AT
metadata-store-reader    CREATED-AT-TIME
```

```
$ kubectl -n tap-gui get clusterrolebinding read-metadata-store
NAME                   ROLE                                  AGE
read-metadata-store    ClusterRole/metadata-store-reader     AGE-VALUE
```

There is another condition that impacts whether the automation creates the necessary service account. If your configuration includes a `/metadata-store` block, the automation doesn't create the Kubernetes resources for use in `autoconfiguration` and the automation doesn't overwrite the proxy block that you provide. To use the automation, you must delete the block at `tap_gui.app_config.proxy["/metadata-store"]`.

For example, a `tap-values.yaml` file with the following content does not create additional Kubernetes resources as described earlier:

```
# ...
tap_gui:
  # ...
  app_config:
    # ...
    proxy:
      '/metadata-store':
        target: SOMETHING
```

Manually connect Tanzu Developer Portal to the Metadata Store

To manually enable CVE scan results:

1. Obtain the read-write token, which is created by default when installing Tanzu Application Platform. Alternatively, create an additional read-write service account.
2. Add this proxy configuration to the `tap-gui:` section of `tap-values.yaml`:

   ```
   tap_gui:
     app_config:
       proxy:
         /metadata-store:
           target: https://metadata-store-app.metadata-store:8443/api/v1
           changeOrigin: true
           secure: false
           allowedHeaders: ['Accept', 'Report-Type-Format']
           headers:
             Authorization: "Bearer ACCESS-TOKEN"
             X-Custom-Source: project-star
   ```

   Where `ACCESS-TOKEN` is the token you obtained after creating a read-write service account.

Important

The `Authorization` value must start with the word `Bearer`.

Enable GitOps Pull Request Flow

Set up for GitOps and pull requests to enable the supply chain box-and-line diagram to show **Approve a Request** in the **Config Writer** stage details view when the **Config Writer** stage is clicked. For more information, see GitOps vs. RegistryOps.

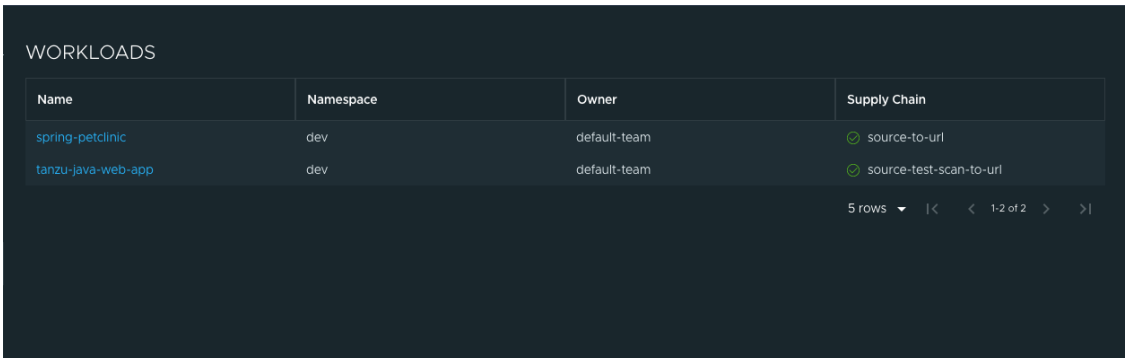
Supply Chain Visibility

Before using the Supply Chain Visibility (SCC) plug-in to visualize a workload, you must create a workload.

The workload must have the `app.kubernetes.io/part-of` label specified, whether you manually create the workload or use one supplied with the OOTB supply chains.

Use the left sidebar navigation to access your workload and visualize it in the supply chain that is installed on your cluster.

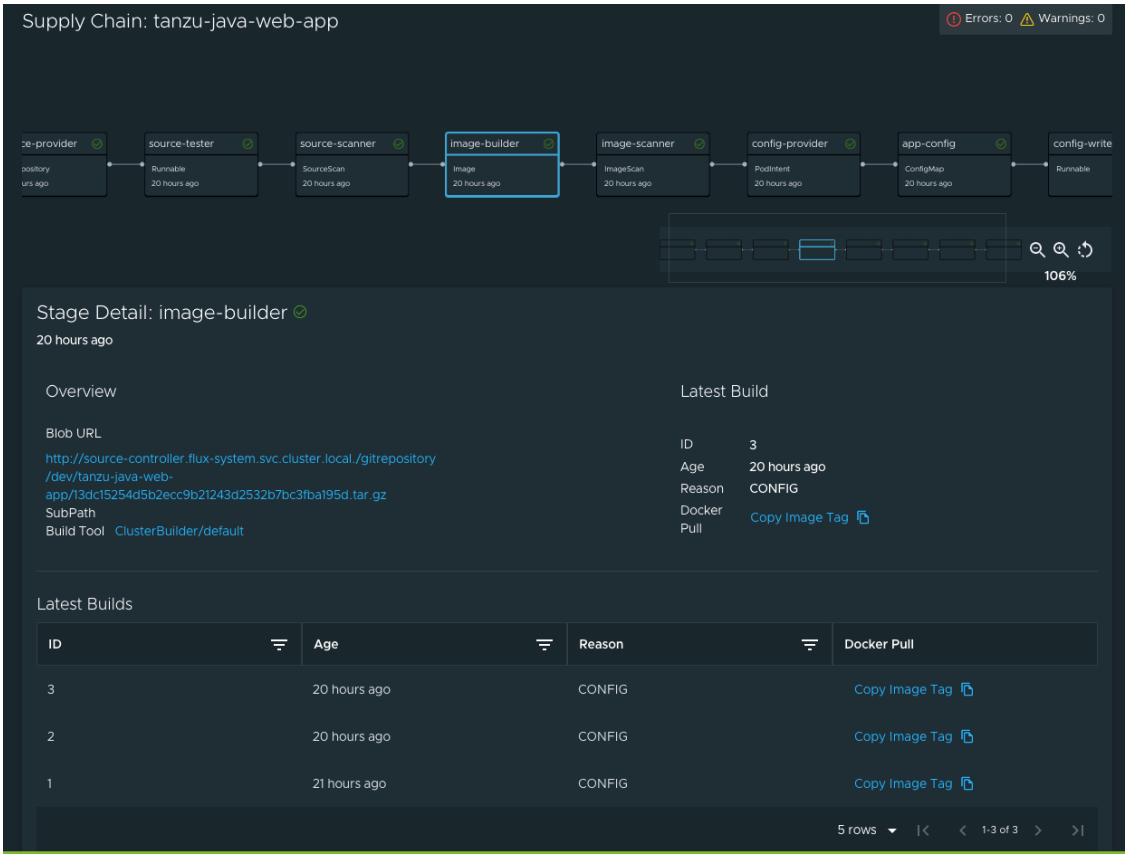
The example workload described in this topic is named `tanzu-java-web-app`.



Click **tanzu-java-web-app** in the **WORKLOADS** table to navigate to the visualization of the supply chain.

There are two sections within this view:

- The box-and-line diagram at the top shows all the configured custom resource definitions (CRDs) that this supply chain uses, and any artifacts that the supply chain's execution outputs
- The **Stage Detail** section at the bottom shows source data for each part of the supply chain that you select in the diagram view



When a workload is deployed to a cluster that has the `deliverable` package installed, a new section appears in the supply chain that shows **Pull Config** boxes and **Delivery** boxes.

When you have a `Pull Request` configured in your environment, access the merge request from the supply chain by clicking **APPROVE A REQUEST**. This button is displayed after you click **Config Writer** in the supply chain diagram.

View Vulnerability Scan Results

Click the **Source Scan** stage or **Image Scan** stage to view vulnerability source scans and image scans for workload builds. The data is from Supply Chain Security Tools - Store.

CVE issues represent any vulnerabilities associated with a package or version found in the source code or image, including vulnerabilities from past scans.

Note

For example, the `log4shell` package is found in image ABC on 1 January without any CVEs. On 15 January, the log4j CVE issue is found while scanning image DEF. If a user returns to the **Image Scan** stage for image ABC, the log4j CVE issue appears and is associated with the `log4shell` package.

Triage vulnerabilities (alpha)

This feature enables you to store analysis data for each of the vulnerabilities found in the scan.

Caution

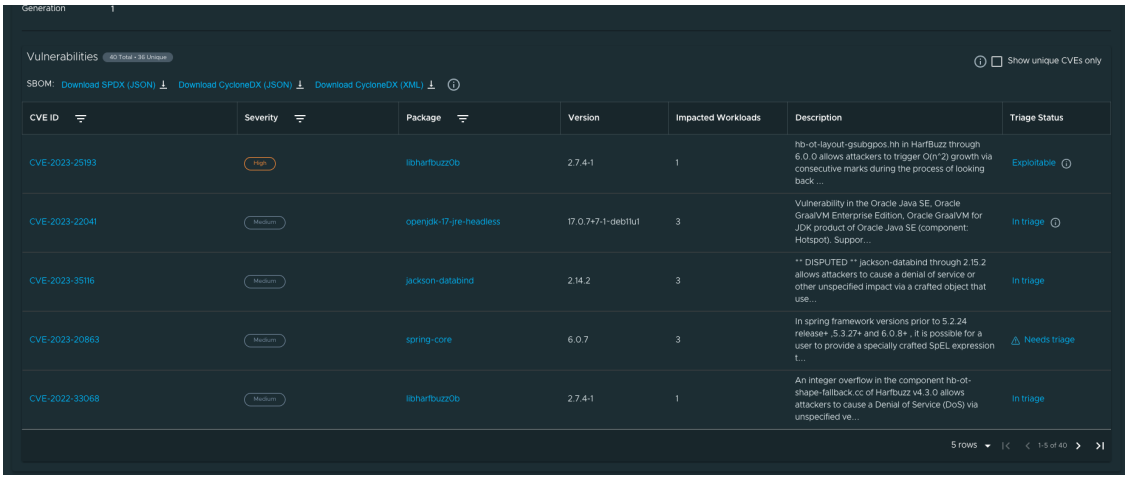
The capability to triage scan results in Tanzu Developer Portal is in the alpha stage, which means that it is still in early development and is subject to change at any point. You might encounter unexpected behavior from it.

The feature is turned off by default in Tanzu Developer Portal. To enable the feature, add the following YAML to your configuration section within the `tap-values.yaml` file:

```
# tap-values.yaml

tap_gui:
  app_config:
    customize:
      features:
        supplyChain:
          enableTriageUI: true
```

When you select a scan stage, the system shows a table with vulnerabilities and a **Triage Status** column where you can see the latest status stored for each vulnerability.



The triage panel enables you to select a status, justification, and resolutions from a set of options, and has a text box to add extra details for the analysis. After you submit this information, the status is updated on the table and the latest analysis is visible the next time you open the panel.

Needs triage is the default status for all vulnerabilities. After you submit an analysis, the status changes and the information button next to the status shows you the stored vulnerability analysis.

Support for CRDs

Tanzu Developer Portal v1.7.0 and later introduced support for CRDs. The following example illustrates the creation of a basic custom resource definition (CRD), which is used as part of a supply chain and its visualization in the workload:

To define and use a CRD in a supply chain:

1. Define the CRD.
2. Set CRD permissions.
3. Define the supply chain.
4. Define the `ClusterTemplate`.
5. Create the workload.
6. Visualize the workload.

Define the CRD

To define a CRD:

1. Create a new YAML file with the name `NAME-crd.yaml`. For example, `rockets-crd.yaml`.
2. Add the following basic structure to the YAML file:

   ```
   apiVersion: apiextensions.k8s.io/v1
   kind: CustomResourceDefinition
   metadata:
    name: ...
   spec: ...
   ```

   At its most basic, a CRD must have `apiVersion`, `kind`, `metadata`, and `spec`.

   The `apiVersion` for a CRD must always be `apiextensions.k8s.io/v1` and the `kind` must be `CustomResourceDefinition`.

3. Add values for `group` and `name`. The value for `group` is usually expressed in a domain URL format, such as `company.com`, and the `name` value can be anything.

The following example uses `spaceagency.com` for the `group` and `rockets` for the `name`.

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: rockets.spaceagency.com
spec:
  group: spaceagency.com
  scope: Namespaced
  names:
    plural: rockets
    singular: rocket
    kind: Rocket
    shortNames:
      - roc
```

Ensure that the name used in `metadata.name` follows the format `PLURAL-NAME.GROUP`. In this example it is `rockets.spaceagency.com`.

4. Add properties in the `spec` section under a list called `versions`. Make each version an object with a `name` and a `schema` for the version, so that the CRD looks like the following example:

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata: ...
spec:
  ...
  versions:
  - name: v1
    served: true
    storage: true
    schema:
      openAPIV3Schema:
        type: object
        properties:
          spec:
            type: object
            properties:
              type:
                type: string
              fuel:
                type: string
              payloadCapacity:
                type: string
```

The `versions` property also has mandatory `served` and `storage` properties. For more information about `served` and `storage`, and CRDs in general, see the Kubernetes documentation.

For this example the `schema` is an `openAPIV3Schema` object. For more information, see the OpenAPI Specification in GitHub.

The `openAPIV3Schema` object lists the attributes that the instances will have and their types. In this example there are 3 attributes (`type`, `fuel`, and `payloadCapacity`), and all of them are strings.

5. Verify that your CRD looks like this finished example:

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: rockets.spaceagency.com
spec:
  group: spaceagency.com
```

```
  scope: Namespaced
  names:
    plural: rockets
    singular: rocket
    kind: Rocket
    shortNames:
      - roc
  versions:
  - name: v1
    served: true
    storage: true
    schema:
      openAPIV3Schema:
        type: object
        properties:
          spec:
            type: object
            properties:
              type:
                type: string
              fuel:
                type: string
              payloadCapacity:
                type: string
```

(Optional) Add custom data to display in the SCC UI

You can display custom data in the SCC plug-in UI by using the Printer Column feature. For more information, see the Kubernetes documentation.

Note

You must have a service account with permissions to view the CRD, which is where printer column data is maintained. When using a service account without this permission, you don't see any printer column data and you don't see any warning that the data is missing.

A printer column is a list that is specified as part of a `version` object. Each list item specifies the following for printing:

- A column name
- The type of the value
- A JSON path, relative to the CRD itself, that shows where to get the value

The following example has 3 printer columns for displaying the `.spec.type`, `.spec.fuel`, and `.spec.payloadCapacity` attributes:

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: rockets.spaceagency.com
spec:
  group: spaceagency.com
  scope: Namespaced
  names:
    plural: rockets
    singular: rocket
    kind: Rocket
    shortNames:
      - roc
```

```
versions:
- name: v1
  served: true
  storage: true
  schema:
    openAPIV3Schema:
      type: object
      properties:
        spec:
          type: object
          properties:
            type:
              type: string
            fuel:
              type: string
            payloadCapacity:
              type: string
  additionalPrinterColumns:
  - name: Type
    type: string
    jsonPath: .spec.type
  - name: Fuel
    type: string
    jsonPath: .spec.fuel
  - name: Payload Capacity
    type: string
    jsonPath: .spec.payloadCapacity
```

Set resource permissions

To use resources in a supply chain, set resource permissions:

1. Create a new YAML file named `permissions.yaml`.
2. Add a `Role` to define which actions (`verbs`) are allowed on this resource:

   ```
   apiVersion: rbac.authorization.k8s.io/v1
   kind: Role
   metadata:
     namespace: NAMESPACE
     name: NAME
   rules:
   - apiGroups: ["API-GROUPS"]
     resources: ["RESOURCES-NAME"]
     verbs:
     - get
     - list
     - watch
     - create
     - patch
     - update
     - delete
     - deletecollection
   ```

   Where:

   - `NAMESPACE` is the namespace in which the rules apply. For example, `my-apps`.
   - `NAME` is the name for the role. For example, `rocket-reader`.
   - `API-GROUPS` is the name of the groups. For example, `spaceagency.com` or `apiextensions.k8s.io/v1`.
   - `RESOURCES-NAME` is the name of the resources. For example, `rockets` or `customresourcedefinitions`.

3. Add a `RoleBinding` to bind this new `Role` to the `serviceAccount` that you typically use:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: BINDING-NAME
  namespace: NAMESPACE
subjects:
- kind: ServiceAccount
  name: default
  namespace: my-apps
roleRef:
  kind: Role
  name: ROLE-REFERENCE-NAME
  apiGroup: rbac.authorization.k8s.io
```

Where:

- `BINDING-NAME` is the binding name. For example, `rocket-reader-binding`.
- `NAMESPACE` is the namespace. For example, `my-apps`.
- `ROLE-REFERENCE-NAME` is the role reference name. For example, `rocket-reader`.

In this binding you associate the `default` service account that you use with the new role you created. The two definitions together look like this example:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: my-apps
  name: rocket-reader
rules:
- apiGroups: ["spaceagency.com"]
  resources: ["rockets"]
  verbs:
  - get
  - list
  - watch
  - create
  - patch
  - update
  - delete
  - deletecollection
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: rocket-reader-binding
  namespace: my-apps
subjects:
- kind: ServiceAccount
  name: default
  namespace: my-apps
roleRef:
  kind: Role
  name: rocket-reader
  apiGroup: rbac.authorization.k8s.io
```

Important

If you defined `additionalPrinterColumns` in your CRD, you must grant permissions for both the group you defined in the CRD and to the

`apiextensions.k8s.io/v1` group that contains the definition of your resource.

The finished definitions look like the following example:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: my-apps
  name: rocket-reader
rules:
- apiGroups: ["spaceagency.com", "apiextensions.k8s.io/v1"]
  resources: ["rockets", "customresourcedefinitions"]
  verbs:
  - get
  - list
  - watch
  - create
  - patch
  - update
  - delete
  - deletecollection
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: rocket-reader-binding
  namespace: my-apps
subjects:
- kind: ServiceAccount
  name: default
  namespace: my-apps
roleRef:
  kind: Role
  name: rocket-reader
  apiGroup: rbac.authorization.k8s.io
```

Define the supply chain

Now that you have a CRD and the permissions for it, you can define a supply chain that uses this CRD as one of its resources.

The following example is a simple supply chain that has only one stage that uses an instance of the CRD. You can create the supply chain by downloading another supply chain and editing it.

1. List the existing supply chains in your cluster by running:

   ```
   kubectl get ClusterSupplyChain -n NAMESPACE
   ```

   Where `NAMESPACE` is your namespace. For example, `my-apps`.

2. Download one of them to a file by running:

   ```
   kubectl get ClusterSupplyChain SUPPLY-CHAIN -n NAMESPACE -oyaml >> ~/supply-cha
   in.yaml
   ```

   Where:

   - `SUPPLY-CHAIN` is the name a supply chain you discovered earlier.
   - `NAMESPACE` is your namespace.

   For example:

```
$ kubectl get ClusterSupplyChain source-test-scan-to-url -n my-apps -oyaml >>
~/supply-chain.yaml
```

3. Edit the downloaded supply-chain definition as follows:

```
apiVersion: carto.run/v1alpha1
kind: ClusterSupplyChain
metadata:
  name: SUPPLY-CHAIN-NAME
spec:
  resources:
  - name: RESOURCES-NAME
    templateRef:
      kind: ClusterTemplate
      name: TEMPLATE-REFERENCE-NAME
  selector:
    apps.tanzu.vmware.com/LABEL: "true"
  selectorMatchExpressions:
  - key: apps.tanzu.vmware.com/workload-type
    operator: In
    values:
    - web
    - server
    - worker
```

Where:

- `SUPPLY-CHAIN-NAME` is the supply-chain name. For example, `source-scan-test-scan-to-url-rockets`.
- `RESOURCES-NAME` is the resources name. For example, `rocket-provider`.
- `TEMPLATE-REFERENCE-NAME` is the template reference name. For example, `rocket-source-template`.
- `apps.tanzu.vmware.com/LABEL:` is a label that must be present, when creating the workload, to use this supply chain. For example, `apps.tanzu.vmware.com/has-rockets: "true"`.

`apiVersion` and `kind` stay the same. `metadata.name` is created for this new supply chain.

The `spec.selector` field states which label selector is used to select this supply chain when creating a workload.

4. Save the supply-chain YAML file as `NAME-supply-chain.yaml`. For example, `rocket-supply-chain.yaml`.

Define the ClusterTemplate

In this procedure you define, for the `resources` field, a single resource that uses an instance of your CRD.

This example supply chain has just a single resource (stage), which is named `rocket-provider`. The supply chain uses a `templateRef`, of the kind `ClusterTemplate`, which is named `rocket-source-template`.

At its most basic, a supply chain's resource is an object consisting of a `name` and a `templateRef` pointing to an existing `ClusterTemplate`.

To define a new `ClusterTemplate`:

1. List existing `ClusterTemplate` resources by running:

```
kubectl get ClusterTemplates -n NAMESPACE
```

Where `NAMESPACE` is your namespace

2. Download a `ClusterTemplate` resource that you found by running:

```
kubectl get ClusterTemplates TEMPLATE-NAME -n NAMESPACE -oyaml >> ~/cluster-tem
plate.yaml
```

Where:

- `TEMPLATE-NAME` is the name of the `ClusterTemplate` resource you found. For example, `config-writer-template`.
- `NAMESPACE` is your namespace. For example, `my-apps`.

For example:

```
$ kubectl get ClusterTemplates config-writer-template -n my-apps -oyaml >> ~/cl
uster-template.yaml
```

3. Verify that the file, when cleaned up, looks similar to the following:

```
apiVersion: carto.run/v1alpha1
kind: ClusterTemplate
metadata:
  name: rocket-source-template
spec:
  lifecycle: mutable
  ytt: |
    #@ load("@ytt:data", "data")

    #@ def merge_labels(fixed_values):
    #@   labels = {}
    #@   if hasattr(data.values.workload.metadata, "labels"):
    #@     exclusions = ["kapp.k14s.io/app", "kapp.k14s.io/association"]
    #@     for k,v in dict(data.values.workload.metadata.labels).items():
    #@       if k not in exclusions:
    #@         labels[k] = v
    #@       end
    #@     end
    #@   end
    #@   labels.update(fixed_values)
    #@   return labels
    #@ end

    ---
    apiVersion: spaceagency.com/v1
    kind: Rocket
    metadata:
      name: falcon9
      labels: #@ merge_labels({ "app.kubernetes.io/component": "rocket" })
    spec:
      type: Falcon 9
      fuel: RP-1/LOX
      payloadCapacity: 22000 kg
```

`metadata.name` matches the name specified in the supply-chain resource.

An instance of the new CRD is used in the `spec` of this resource through the `ytt` field. ytt is a templating language that can output resource instances.

A function is retained that takes in labels from the workload and propagates them to the resource. This function is not essential, but is usually performed to propagate important labels from the workload down to the individual resources.

4. Save the file as `NAME-cluster-template.yaml`. For example, `rocket-cluster-template.yaml`.

You have now completed all the necessary definitions to create a workload that uses this new supply chain and the new CRD.

Create the workload

Now that you have all of the resources, apply them to a cluster and then create a workload:

1. Apply your CRD by running:

   ```
   kubectl apply -f rockets-crd.yaml
   ```

2. Apply the resource permissions by running:

   ```
   kubectl apply -f permissions.yaml
   ```

3. Apply the cluster template by running:

   ```
   kubectl apply -f rocket-cluster-template.yaml
   ```

4. Apply the supply chain by running:

   ```
   kubectl apply -f rocket-supply-chain.yaml
   ```

   The cluster now has all the necessary resource definitions to create a workload by using the new supply chain, which, in turn, uses an instance of the new resource.

5. Create the workload by running:

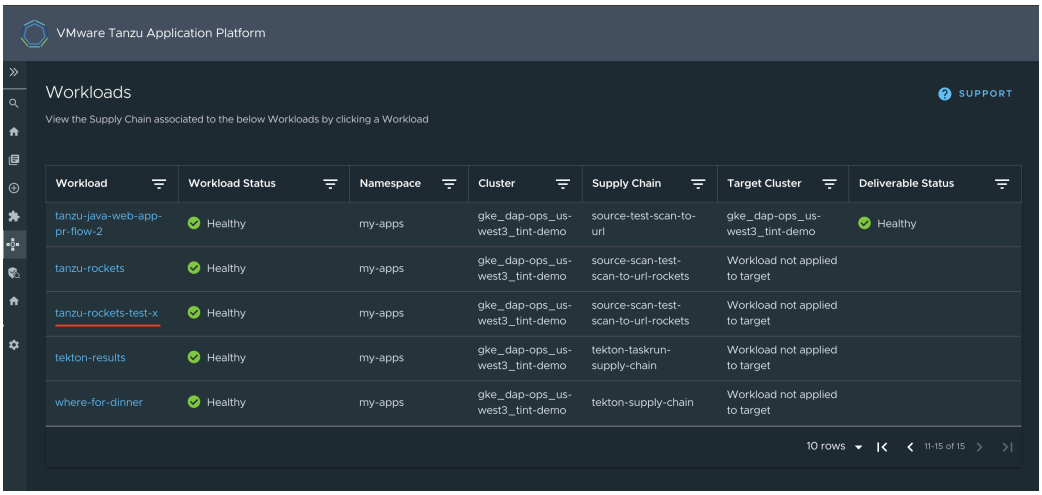
   ```
   tanzu apps workload create tanzu-rockets-test-x \
   --type web \
   --label app.kubernetes.io/part-of=tanzu-rockets \
   --label apps.tanzu.vmware.com/has-rockets=true \
   --yes \
   --namespace my-apps
   ```

   The label `apps.tanzu.vmware.com/has-rockets=true` is explicitly set. The `selector` property, specified when defining the new supply chain, ties the new supply chain with this particular workload.

Visualize the workload

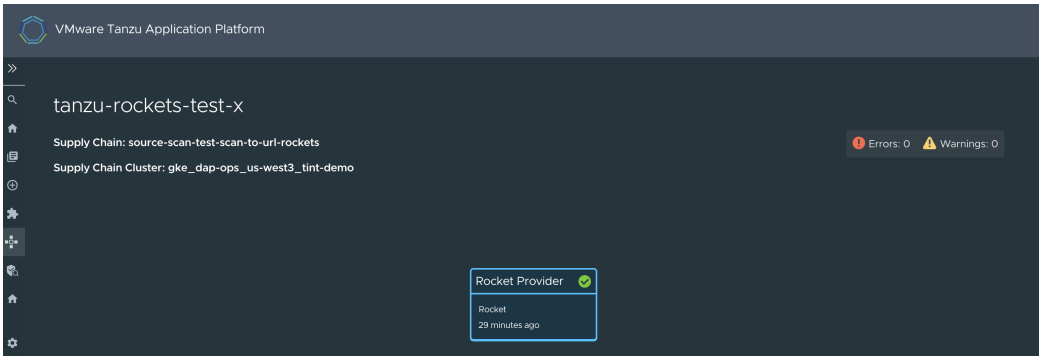
To see the workload rendered through the Supply Chain plug-in:

1. Go to the supply chain plug-in section in Tanzu Developer Portal and locate the workload among the listed ones:

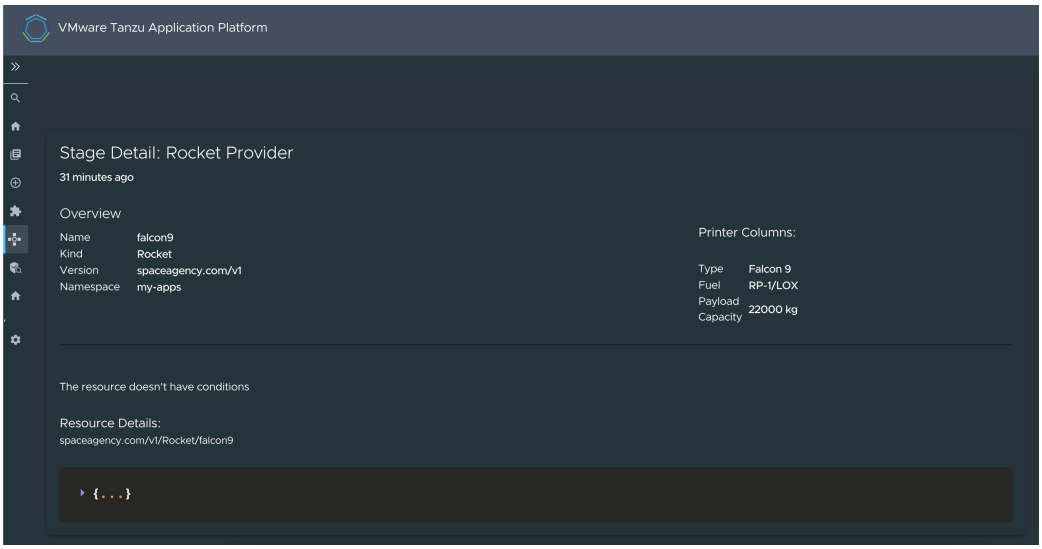


The workload `tanzu-rockets-x` is **Healthy**. The **Supply Chain** column shows that it is using the `source-scan-test-scan-to-url-rockets` supply chain.

2. Click on it to see its details. The **Workload** graph appears. Given that the supply chain `source-scan-test-scan-to-url-rockets` only specified one `resource`, you see a simple single-stage graph.

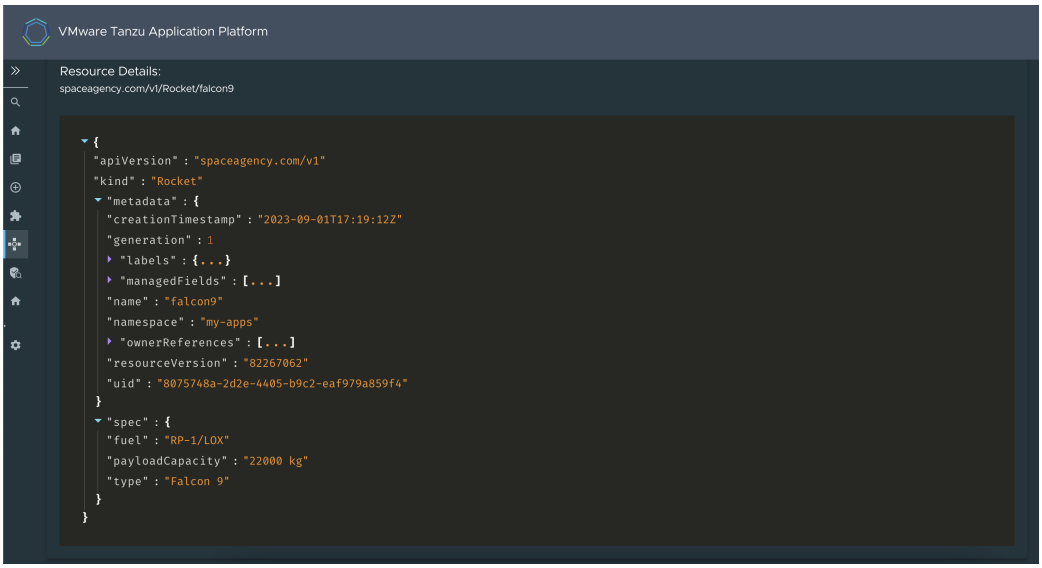


3. Scroll down the screen to see the details associated with the stage.



The printer columns that you defined in the CRD are now rendered in the **Overview** section. This happens with any CRD that you define and that includes the `additionalPrinterColumns` definition.

4. Go to the end of the section to see the full resource in JSON format.



Overview of validated plug-ins for Tanzu Developer Portal

Validated plug-ins are Backstage plug-ins or community plug-ins that VMware has validated for use with Tanzu Developer Portal. You don't need to create custom wrappers to integrate these plug-ins with Tanzu Developer Portal.

Validated plug-ins are listed in the side navigation pane.

Overview of validated plug-ins for Tanzu Developer Portal

Validated plug-ins are Backstage plug-ins or community plug-ins that VMware has validated for use with Tanzu Developer Portal. You don't need to create custom wrappers to integrate these plug-ins with Tanzu Developer Portal.

Validated plug-ins are listed in the side navigation pane.

GitHub Actions in Tanzu Developer Portal

This topic tells you about the GitHub Actions validated front-end plug-in.

The GitHub Actions front-end plug-in visualizes your GitHub Actions integrations. For more information, see the GitHub Actions Backstage documentation.

Add and configure the plug-in

To add the plug-in to your customized Tanzu Developer Portal and configure the plug-in, see the following sections.

Add the plug-in

To add the plug-in to your customized Tanzu Developer Portal, add the front-end plug-in to your `tdp-config.yaml` file:

```
app:
  plugins:
    ...
    - name: '@vmware-tanzu/tdp-plugin-github-actions'
      version: 'VERSION'
    ...
```

Where `VERSION` is the latest version. For example, `^0.0.2`.

Configure the plug-in

To configure the plug-in, update the `app_config` section of your `tap-values.yaml` file to include a GitHub integration. For example:

```
tap_gui:
  # ... existing configuration
  app_config:
    # ... existing configuration
    integrations:
      github:
        - host: 'GITHUB-HOST.com'
          apiBaseUrl: 'https://api.GITHUB-HOST.com'
```

Where `GITHUB-HOST` is your GitHub domain

Grafana in Tanzu Developer Portal

This topic tells you about the Grafana validated front-end plug-in.

The Grafana front-end plug-in visualizes Grafana information in a component **Overview** tab. For more information, see the Grafana plug-in documentation.

Add and configure the plug-in

To add the plug-in to your customized Tanzu Developer Portal and configure the plug-in, see the following sections.

Add the plug-in

To add the plug-in to your custom Tanzu Developer Portal, add the front-end plug-in to your `tdp-config.yaml` file:

```
app:
  plugins:
    ...
    - name: '@vmware-tanzu/tdp-plugin-backstage-grafana'
      version: 'VERSION'
    ...
```

Where `VERSION` is the latest version. For example, `^0.0.2`.

Configure the plug-in

To configure the plug-in, update the `app_config` section of your `tap-values.yaml` file to include a `proxy` entry and `grafana` configuration. For example:

```
tap_gui:
  # ... existing configuration
  app_config:
    # ... existing configuration
    proxy:
      # ... existing configuration
      '/grafana/api':
        # May be a public or an internal DNS
        target: https://test.grafana.net/
        headers:
          Authorization: 'Bearer ${GRAFANA-TOKEN}'
```

```
    grafana:
      # Publicly accessible domain
      domain: https://test.grafana.net/
      # Is unified alerting enabled in Grafana?
      # See: https://grafana.com/blog/2021/06/14/the-new-unified-alerting-system-for-g
rafana-everything-you-need-to-know/
      # Optional. Default: false
      unifiedAlerting: true
```

Where `GRAFANA-TOKEN` is a valid token for your Grafana instance.

Add the annotations entry for Grafana to the catalog entity to display the Grafana alerts and dashboard lists on the component **Overview** tab.

```
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: grafana-example
  description: An example for Grafana integration.
  annotations:
    grafana/dashboard-selector: 'general'
    grafana/alert-label-selector: 'type=http-requests'
```

Home in Tanzu Developer Portal

This topic tells you about the Home front-end plug-in.

The Home front-end plug-in makes it possible to create a custom home page for your Tanzu Developer Portal to conveniently surface important information. For more information, see the Home plug-in documentation.

Add and configure the plug-in

To add the plug-in to your customized Tanzu Developer Portal and configure the plug-in, see the following sections.

Add the plug-in

To add the plug-in to your customized Tanzu Developer Portal, add the front-end plug-in to your `tdp-config.yaml` file:

```
app:
  plugins:
    ...
    - name: '@vmware-tanzu/tdp-plugin-home'
      version: 'VERSION'
    ...
```

Where `VERSION` is the latest version. For example, `^0.0.2`.

Configure the plug-in

To configure the plug-in, update the `app_config` section of your `tap-values.yaml` file. All of the values are optional. For example:

```
tap_gui:
  # ... existing configuration
  app_config:
```

```
    # ... existing configuration
    customize:
      # ... existing configuration
      features:
        # ... existing configuration
        home:
          # Activate or deactivate the plugin? Default: true
          enabled: true
          # Show or hide the sidebar entry. Default: true
          showInSidebar: true
          # base64 encoded SVG image.
          logo: 'BASE64-ENCODED-LOGO-STRING'
          welcomeMessage: string
          quickLinks:
            url: string
            label: string
            # base64 encoded SVG image.
            icon: 'BASE64-ENCODED-ICON-STRING'
```

Where `BASE64-ENCODED-LOGO-STRING` and `BASE64-ENCODED-ICON-STRING` are Base64-encoded SVG files.

Jira in Tanzu Developer Portal

This topic tells you about the Jira validated front-end plug-in.

The Jira front-end plug-in visualizes your JIRA activity stream in a component **Overview** tab. For more information, see the Jira plug-in documentation.

Add and configure the plug-in

To add the plug-in to your customized Tanzu Developer Portal and configure the plug-in, see the following sections.

Add the plug-in

To add the plug-in to your customized Tanzu Developer Portal, add the front-end plug-in to your `tdp-config.yaml` file:

```
app:
  plugins:
    ...
    - name: '@vmware-tanzu/tdp-plugin-backstage-jira'
      version: 'VERSION'
    ...
```

Where `VERSION` is the latest version. For example, `^0.0.2`.

Configure the plug-in

To configure the Jira plug-in, update the `app_config` section of your `tap-values.yaml` file to include a proxy entry. For example:

```
tap_gui:
  # ... existing configuration
  app_config:
    # ... existing configuration
    proxy:
      # ... existing configuration
      '/jira/api':
```

```
        target: JIRA-URL
        headers:
          Authorization:
            $env: JIRA-TOKEN
          Accept: 'application/json'
          Content-Type: 'application/json'
          X-Atlassian-Token: 'no-check'
          # This is a workaround since Jira APIs reject browser origin requests.
          # Any dummy string without whitespace works.
          User-Agent: 'AnyRandomString'
```

Where:

- `JIRA-URL` is the URL for your Jira instance
- `JIRA-TOKEN` is a valid token for your Jira instance

Add the `annotations` entry for Jira to the catalog entity to display the Jira project results on the component overview tab.

Catalog

```
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: jira-overview-card
  annotations:
    jira/project-key: YOUR_PROJECT_KEY
```

Prometheus in Tanzu Developer Portal

This topic tells you about the Prometheus front-end plug-in.

The Prometheus front-end plug-in displays Prometheus information in your Tanzu Developer Portal. For more information, see the Prometheus plug-in documentation.

Add and configure the plug-in

To add the plug-in to your customized Tanzu Developer Portal and configure the plug-in, see the following sections.

Add the plug-in

To add the plug-in to your customized Tanzu Developer Portal, add the front-end plug-in to your `tdp-config.yaml` file:

```
app:
  plugins:
    ...
    - name: '@vmware-tanzu/tdp-plugin-prometheus'
      version: 'VERSION'
    ...
```

Where `VERSION` is the latest version. For example, `^0.0.2`.

Configure the plug-in

To configure the Prometheus plug-in, update the `app_config` section of your `tap-values.yaml` file to include a `proxy` entry and `prometheus` configuration. For example:

```
tap_gui:
  # ... existing configuration
  app_config:
    # ... existing configuration
    proxy:
      # ... existing configuration
      '/prometheus/api':
        # url to the api and path of your hosted prometheus instance
        target: http://localhost:9090/api/v1/
        changeOrigin: true
        secure: false
        headers:
          Authorization: AUTH-TOKEN

    # Defaults to /prometheus/api and can be omitted if proxy is configured for that u
rl
    prometheus:
      proxyPath: /prometheus/api
      uiUrl: http://localhost:9090
```

Where `PROMETHEUS-AUTH-TOKEN` is a valid token for your secure `prometheus` instance

Snyk in Tanzu Developer Portal

This topic tells you about the Snyk front-end plug-in.

The Snyk front-end plug-in displays security vulnerabilities from snyk.io.

The plug-in shows an overview of the vulnerabilities found by Snyk on the **Overview** tab of an entity. The plug-in also adds a tab to the entity view, which shows all details related to the scan. For more information, see the Snyk Backstage documentation.

Add the plug-in

To add the plug-in to your customized Tanzu Developer Portal, add the front-end plug-in to your `tdp-config.yaml` file:

```
app:
  plugins:
    ...
    - name: '@vmware-tanzu/tdp-plugin-snyk'
      version: 'VERSION'
    ...
```

Where `VERSION` is the latest version. For example, `^0.0.2`.

SonarQube in Tanzu Developer Portal

This topic tells you about the SonarQube validated front-end and back-end plug-ins.

The SonarQube front-end plug-in displays static analysis code quality statistics. For more information, see the SonarQube Backstage documentation.

Add and configure the plug-in

To add the plug-in to your customized Tanzu Developer Portal and configure the plug-in, see the following sections.

Add the plug-in

To add the plug-in to your customized Tanzu Developer Portal, add the front-end and back-end plug-ins to your `tdp-config.yaml` file:

```
app:
  plugins:
    ...
    - name: '@vmware-tanzu/tdp-plugin-backstage-sonarqube'
      version: 'VERSION'
    ...
backend:
  plugins:
    ...
    - name: '@vmware-tanzu/tdp-plugin-backstage-sonarqube-backend'
      version: 'VERSION'
    ...
```

Where `VERSION` is the latest version for each plug-in. For example, `^0.0.2` and `^0.0.3`.

Configure the plug-in

To connect your SonarQube plug-in to a running instance, update the `app_config` section of your `tap-values.yaml` file. For example:

```
tap_gui:
  # ... existing configuration
  app_config:
    # ... existing configuration
    sonarqube:
      baseUrl: 'SONARQUBE-URL'
      apiKey: 'SONARQUBE-API-KEY'
```

Where:

- `SONARQUBE-URL` is the URL of your SonarQube instance
- `SONARQUBE-API-KEY` is a valid API key for connecting to the SonarQube instance

Add the `annotations` entry for SonarQube to the catalog entity to display the SonarQube project results on the component overview tab.

```
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: backstage
  annotations:
    sonarqube.org/project-key: YOUR_INSTANCE_NAME/YOUR_PROJECT_KEY
```

For a more detailed explanation of SonarQube configuration, see SonarQube Backstage documentation.

Stack Overflow in Tanzu Developer Portal

This topic tells you about the Stack Overflow front-end plug-in.

The Stack Overflow front-end plug-in provides Stack Overflow functions in Tanzu Developer Portal. For more information, see the Backstage Stack Overflow documentation.

Add and configure the plug-in

To add the plug-in to your customized Tanzu Developer Portal and configure the plug-in, see the following sections.

Add the plug-in

To add the plug-in to your customized Tanzu Developer Portal, add the front-end plug-in to your `tdp-config.yaml` file:

```
app:
  plugins:
    ...
    - name: '@vmware-tanzu/tdp-plugin-stack-overflow'
      version: 'VERSION'
    ...
```

Where `VERSION` is the latest version. For example, `^0.0.2`.

Configure the plug-in

To configure your Stack Overflow plug-in, update the `app_config` section of your `tap-values.yaml` file. For example:

```
tap_gui:
  # ... existing configuration
  app_config:
    # ... existing configuration
    stackoverflow:
      baseUrl: https://api.stackexchange.com/2.2 # alternative: your internal stack ov
erflow instance
```

TechInsights in Tanzu Developer Portal

This topic tells you about the TechInsights front-end and back-end plug-ins.

The TechInsights front-end plug-in visualizes entity checks defined by the back-end plug-in. The TechInsights back-end plug-in performs entity checks and provides an API for the front-end plug-in. For more information about the TechInsights plug-in, see the Backstage TechInsights documentation.

Add the plug-in

To add the plug-in to your customized Tanzu Developer Portal, add the front-end and back-end plug-ins to your `tdp-config.yaml` file:

```
app:
  plugins:
    ...
    - name: '@vmware-tanzu/tdp-plugin-techinsights'
      version: 'VERSION'
    ...
backend:
  plugins:
    ...
    - name: '@vmware-tanzu/tdp-plugin-techinsights-backend'
      version: 'VERSION'
    ...
```

Where `VERSION` is the latest version for each plug-in. For example, `^0.0.2`.

Overview of enabling TLS for Tanzu Developer Portal

Many users want inbound traffic to Tanzu Developer Portal to be properly encrypted. These topics tell you how to enable TLS encryption either with an existing certificate or by using the included cert-manager instance.

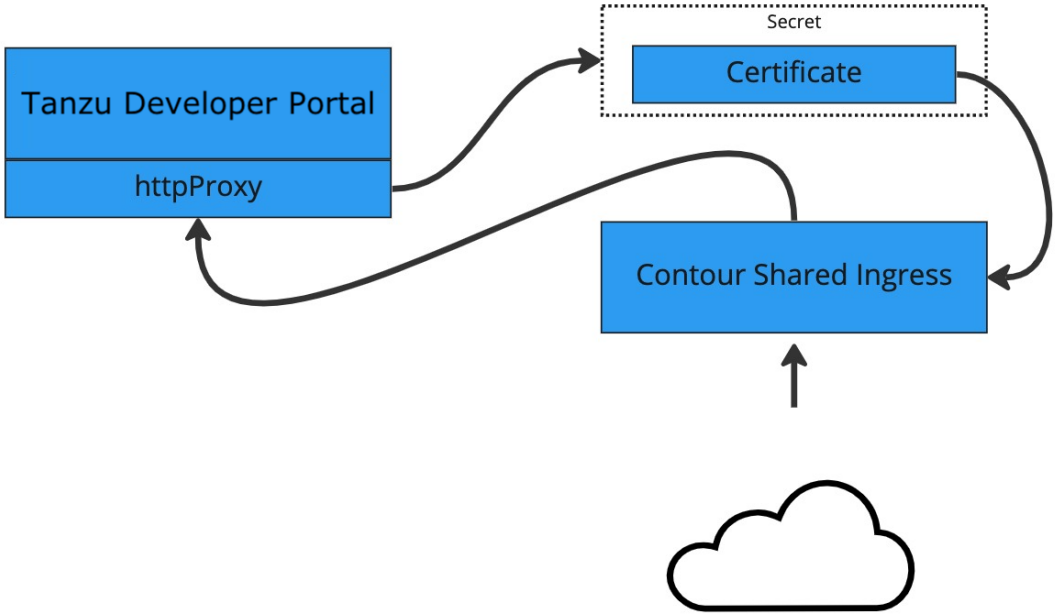
Concepts

The two key concepts are certificate delegation and the relationship between cert-manager, certificates, and ClusterIssuers.

Certificate delegation

Tanzu Developer Portal uses the established shared Contour ingress for TLS termination.

This enables you to store the certificate in a Kubernetes `secret` and then pass that `secret` and `namespace` to the `httpProxy` that was created during installation. To do this, see Configuring a TLS certificate by using an existing certificate.

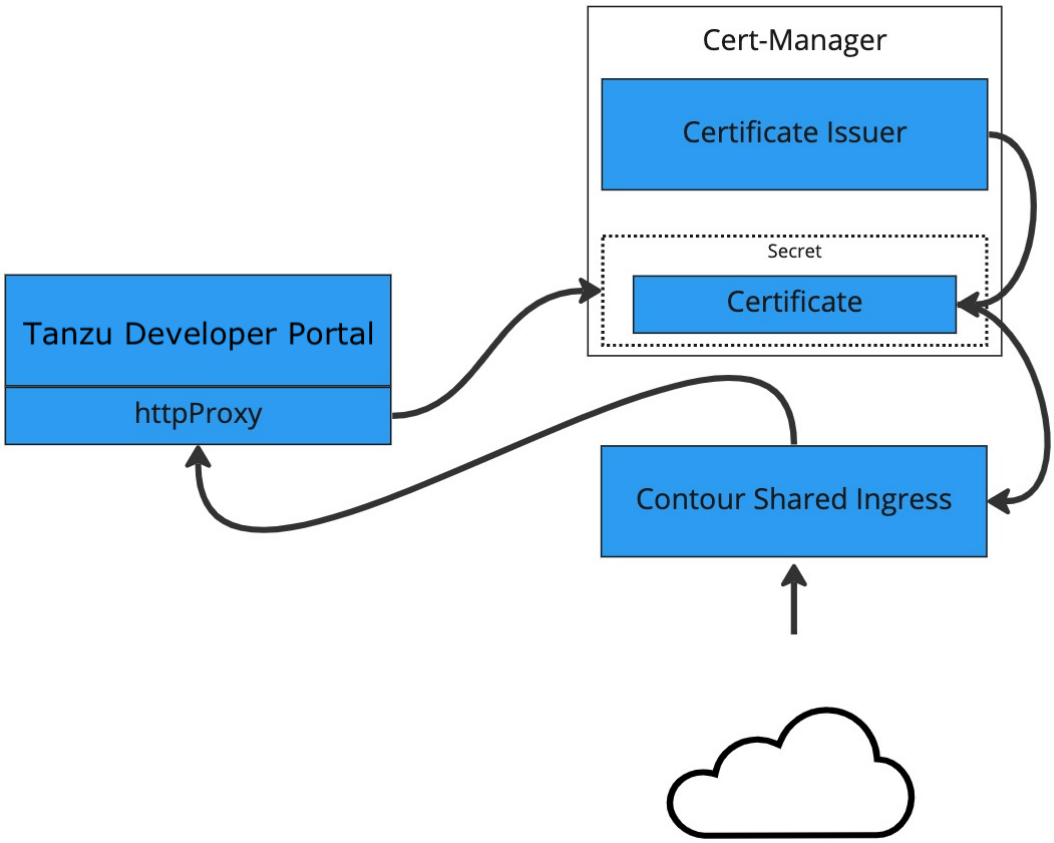


cert-manager, certificates, and ClusterIssuers

Tanzu Developer Portal can also use the `cert-manager` package that is installed when the profile was installed.

This tool allows cert-manager to automatically acquire a certificate from a `clusterIssuer` entity.

This external entity can be an external certificate authority, such as Let's Encrypt, or a self-signed certificate.



Guides

The following topics describe different ways to configure TLS:

- Configuring a TLS certificate by using an existing certificate
- Configuring a TLS certificate by using a self-signed certificate
- Configuring a TLS certificate by using cert-manager and a ClusterIssuer

Overview of enabling TLS for Tanzu Developer Portal

Many users want inbound traffic to Tanzu Developer Portal to be properly encrypted. These topics tell you how to enable TLS encryption either with an existing certificate or by using the included cert-manager instance.

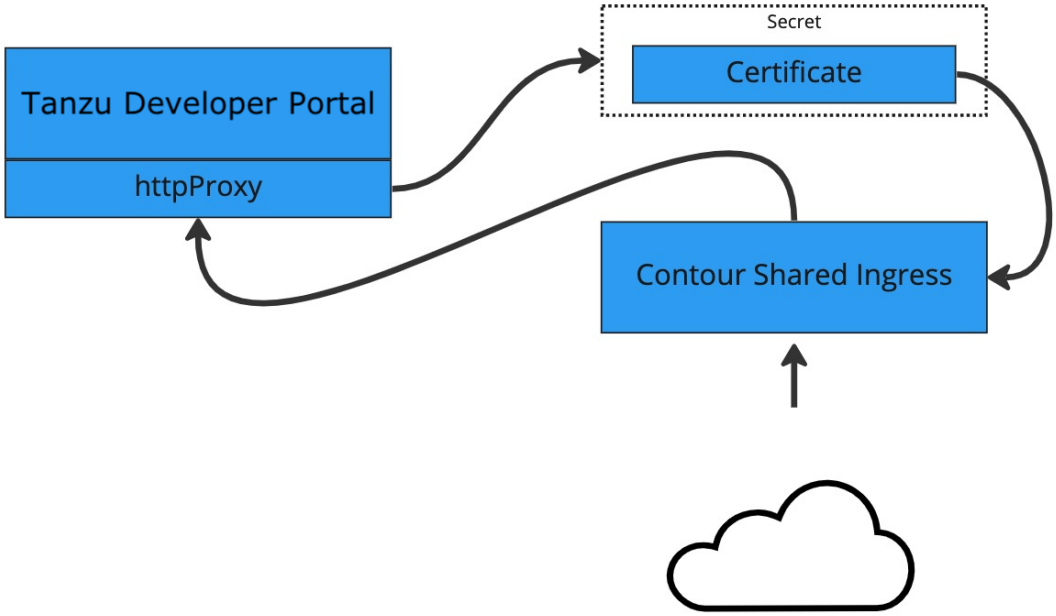
Concepts

The two key concepts are certificate delegation and the relationship between cert-manager, certificates, and ClusterIssuers.

Certificate delegation

Tanzu Developer Portal uses the established shared Contour ingress for TLS termination.

This enables you to store the certificate in a Kubernetes `secret` and then pass that `secret` and `namespace` to the `httpProxy` that was created during installation. To do this, see Configuring a TLS certificate by using an existing certificate.

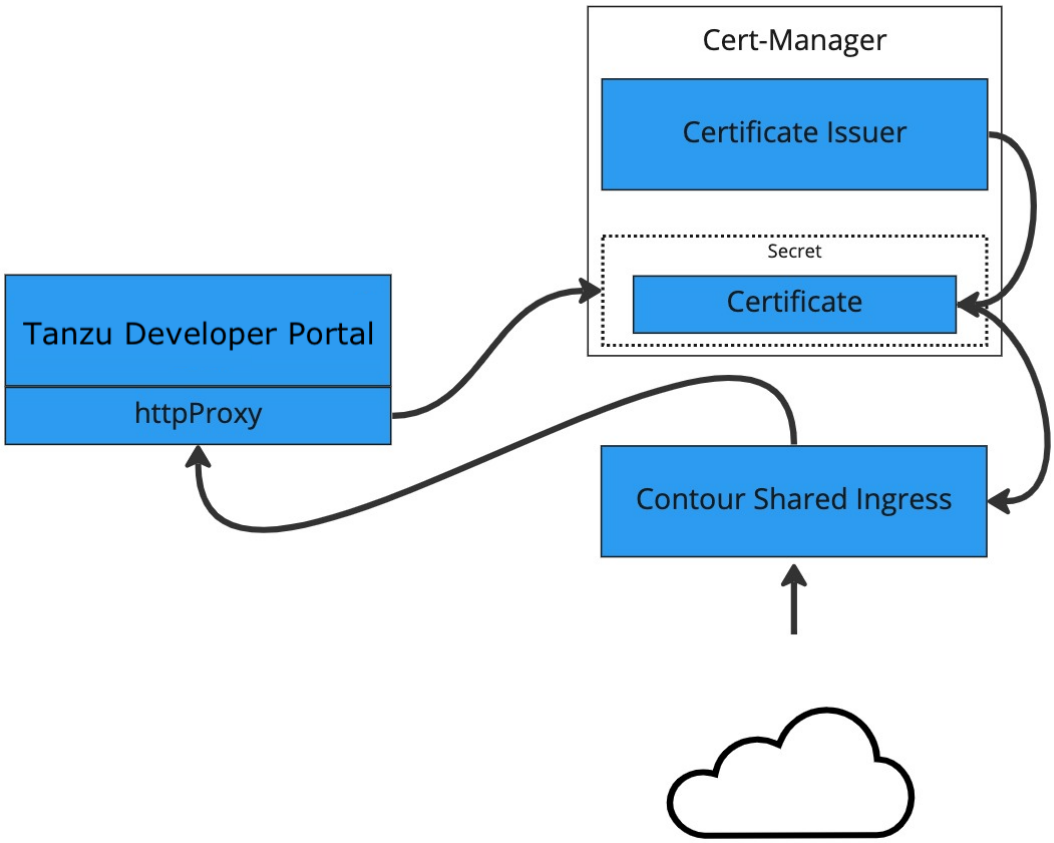


cert-manager, certificates, and ClusterIssuers

Tanzu Developer Portal can also use the `cert-manager` package that is installed when the profile was installed.

This tool allows cert-manager to automatically acquire a certificate from a `clusterIssuer` entity.

This external entity can be an external certificate authority, such as Let's Encrypt, or a self-signed certificate.



Guides

The following topics describe different ways to configure TLS:

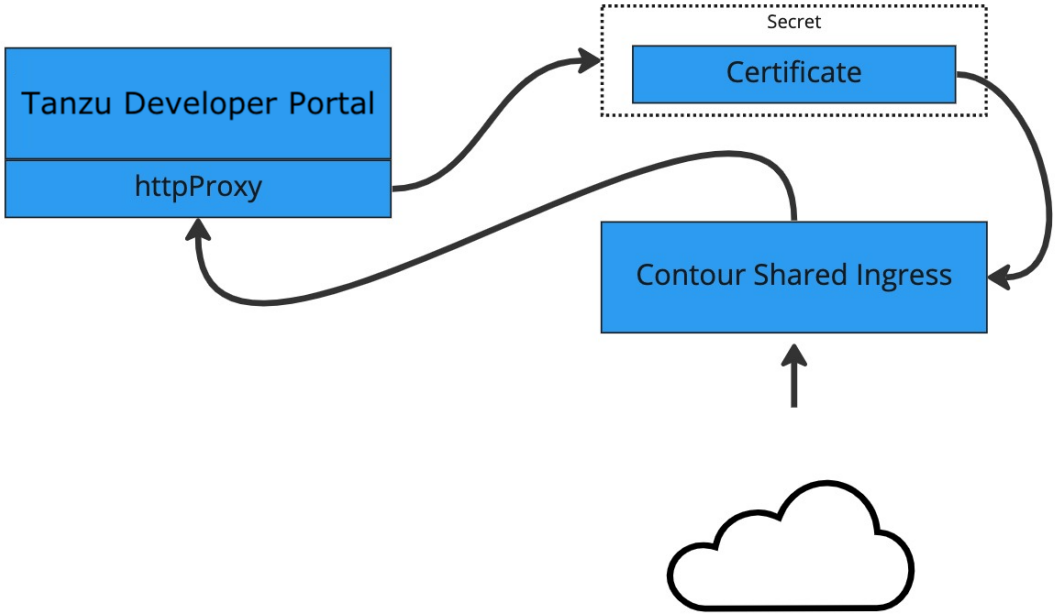
- Configuring a TLS certificate by using an existing certificate
- Configuring a TLS certificate by using a self-signed certificate
- Configuring a TLS certificate by using cert-manager and a ClusterIssuer

Configure a TLS certificate by using an existing certificate

This topic tells you how to use the certificate information from your external certificate authority to encrypt inbound traffic to Tanzu Developer Portal.

Prerequisites

Your certificate authority gave you a certificate file, of the form `CERTIFICATE-FILE-NAME.crt`, and a signing key, of the form `KEY-FILE-NAME.key`. Ensure that these files are present on the host from which you run the CLI commands.



Procedure

To configure Tanzu Developer Portal with an existing certificate:

1. Create the Kubernetes secret by running:

   ```
   kubectl create secret tls tap-gui-cert --key="KEY-FILE-NAME.key" --cert="CERTIF
   ICATE-FILE-NAME.crt" -n tap-gui
   ```

   Where:

   - `KEY-FILE-NAME` is the name of the `key` file that your certificate issuer gave you
   - `CERTIFICATE-FILE-NAME` is the name of the `crt` file that your certificate issuer gave you

2. Configure Tanzu Developer Portal to use the newly created secret. Do so by editing the `tap-values.yaml` file that you used during installation to include the following under the `tap-gui` section:

   - A top-level `tls` key with subkeys for `namespace` and `secretName`
   - A namespace referring to the namespace used earlier
   - A secret name referring to the `secretName` value defined earlier

   Example:

   ```
   tap_gui:
    tls:
      namespace: tap-gui
      secretName: tap-gui-cert
   # Additional configuration below this line as needed
   ```

3. Update the Tanzu Application Platform package with the new values in `tap-values.yaml` by running:

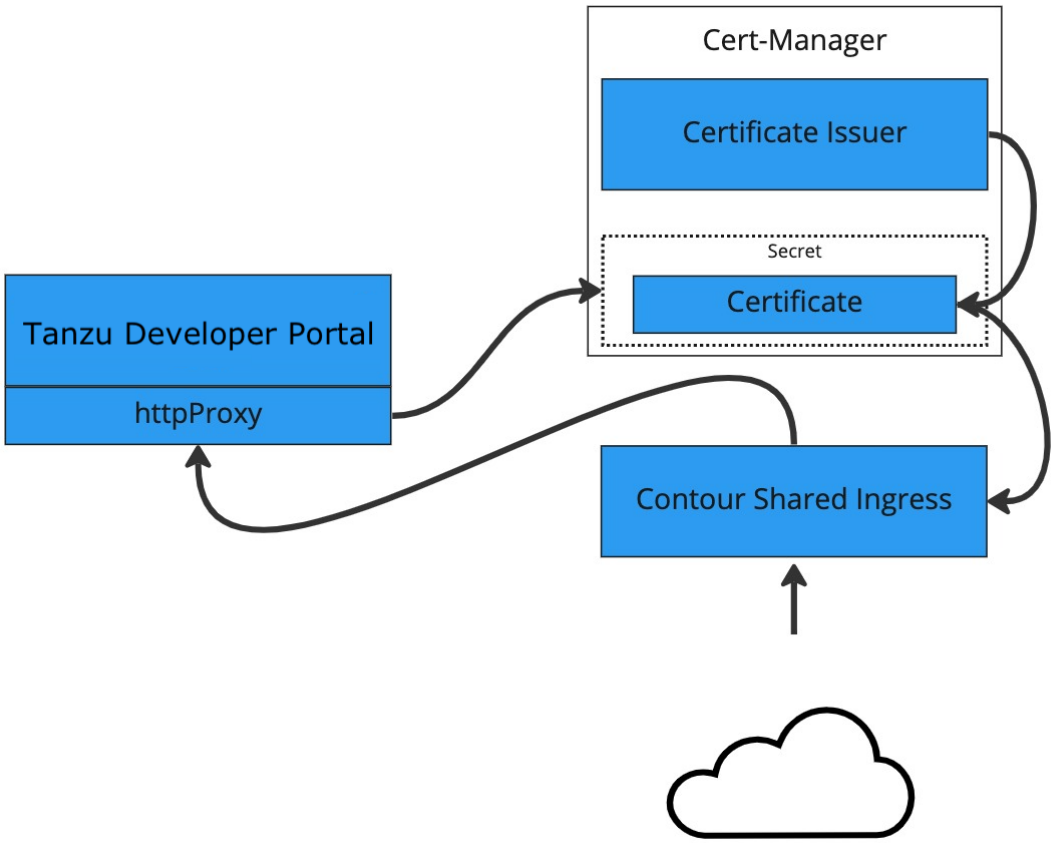
```
tanzu package installed update tap -p tap.tanzu.vmware.com -v TAP-VERSION  --va
lues-file tap-values.yaml -n tap-install
```

Where `TAP-VERSION` is the version number that matches the values you used when you installed your profile.

Configure a TLS certificate by using a self-signed certificate

This topic tells you how to use cert-manager to create a self-signed certificate issuer and then generate a certificate for Tanzu Developer Portal to use based on that issuer.

Some browsers and corporate policies do not allow you to visit webpages that have self-signed certificates. You might need to navigate through a series of error messages to visit the page.



Prerequisite

Install a Tanzu Application Platform profile that includes cert-manager. Verify you did this by running the following command to detect the cert-manager namespace:

```
kubectl get ns
```

Procedure

To configure a self-signed TLS certificate for Tanzu Developer Portal:

1. Create a `certificate.yaml` file that defines an issuer and a certificate. For example:

```
apiVersion: cert-manager.io/v1
kind: Issuer
```

```
metadata:
 name: ca-issuer
 namespace: tap-gui
spec:
 selfSigned: {}
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
 name: tap-gui-cert
 namespace: tap-gui
spec:
 secretName: tap-gui-cert
 dnsNames:
 - tap-gui.INGRESS-DOMAIN
 issuerRef:
   name: ca-issuer
```

Where `INGRESS-DOMAIN` is your domain value that matches the values you used when you installed the profile.

2. Add the issuer and certificate to your cluster by running:

```
kubectl apply -f certificate.yaml
```

3. Configure Tanzu Developer Portal to use the newly created certificate. Update the `tap-values.yaml` file used during installation to include the following under the `tap-gui` section:
    - A top-level `tls` key with subkeys for `namespace` and `secretName`
    - A namespace referring to the namespace containing the `Certificate` object mentioned earlier
    - A secret name referring to the `secretName` value defined in your `Certificate` resource earlier

    Example:

```
tap_gui:
 tls:
   namespace: tap-gui
   secretName: tap-gui-cert
# Additional configuration below this line as needed
```

4. Update the Tanzu Application Platform package with the new values in `tap-values.yaml`:

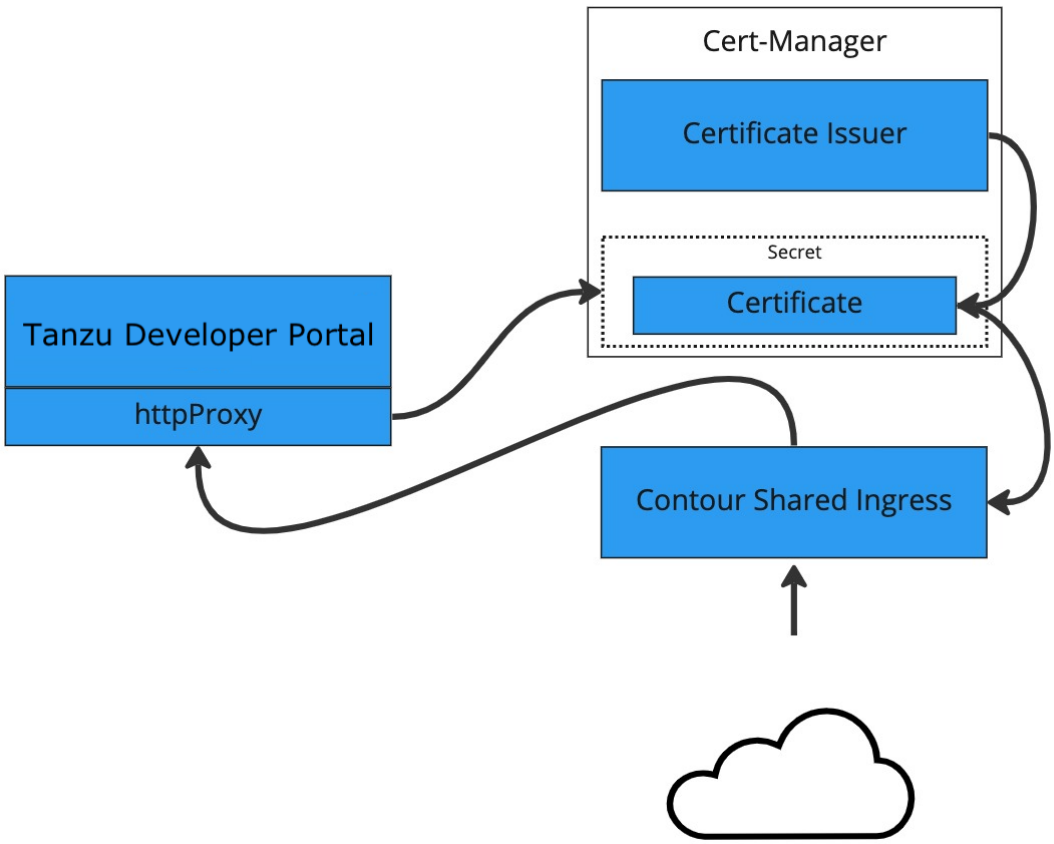
```
tanzu package installed update tap -p tap.tanzu.vmware.com -v TAP-VERSION  --va
lues-file tap-values.yaml -n tap-install
```

Where `TAP-VERSION` is the version that matches the values you used when you installed the profile.

Configure a TLS certificate by using cert-manager and a ClusterIssuer

This topic tells you how to use cert-manager to create a certificate issuer and then generate a certificate for Tanzu Developer Portal to use based on that issuer.

This topic uses the free certificate issuer Let's Encrypt. You can use other certificate issuers compatible with cert-manager in a similar fashion.



Prerequisites

Fulfil these prerequisites:

- Install a Tanzu Application Platform profile that includes cert-manager. Verify you did this by running the following command to detect the cert-manager namespace:

  ```
  kubectl get ns
  ```

- Obtain a domain name that you control or own and have proof that you control or own it. In most cases, this domain name is the one you used for the `INGRESS-DOMAIN` values when you installed Tanzu Application Platform and Tanzu Developer Portal.
- If cert-manager cannot perform the challenge to verify your domain's compatibility, you must do so manually. For more information, see How It Works and Getting Started in the Let's Encrypt documentation.
- Ensure that your domain name is pointed at the shared Contour ingress for the installation. Find the IP address by running:

  ```
  kubectl -n tanzu-system-ingress get services envoy -o jsonpath='{.status.loadBa
  lancer.ingress[0].ip}'
  ```

Procedure

To configure a self-signed TLS certificate for Tanzu Developer Portal:

1. Create a `certificate.yaml` file that defines an issuer and a certificate. For example:

```
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-http01-issuer
  namespace: cert-manager
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    email: EMAIL-ADDRESS
    privateKeySecretRef:
      name: letsencrypt-http01-issuer
    solvers:
    - http01:
        ingress:
          class: contour
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  namespace: cert-manager
  name: tap-gui
spec:
  commonName: tap-gui.INGRESS-DOMAIN
  dnsNames:
    - tap-gui.INGRESS-DOMAIN
  issuerRef:
    name: letsencrypt-http01-issuer
    kind: ClusterIssuer
  secretName: tap-gui
```

Where:

- `EMAIL-ADDRESS` is the email address that Let's Encrypt shows as responsible for this certificate
- `INGRESS-DOMAIN` is your domain value that matches the values you used when you installed the profile

2. Add the issuer and certificate to your cluster by running:

```
kubectl apply -f certificate.yaml
```

By applying the certificate, cert-manager attempts to perform an HTTP01 challenge by creating an Ingress resource specifically for the challenge. This is automatically removed from your cluster after the challenge is completed. For more information about how this works, and when it might not, see the cert-manager documentation.

3. Validate the certificate was created and is ready by running:

```
kubectl get certs -n cert-manager
```

Wait a few moments for this to take place, if need be.

4. Configure Tanzu Developer Portal to use the newly created certificate. To do so, update the `tap-values.yaml` file that you used during installation to include the following items under the `tap-gui` section:

   - A top-level `tls` key with subkeys for `namespace` and `secretName`
   - A namespace referring to the namespace containing the `Certificate` object from earlier
   - A secret name referring to the `secretName` value defined in your `Certificate` resource earlier

Example:

```
tap_gui:
 tls:
   namespace: cert-manager
   secretName: tap-gui
# Additional configuration below this line as needed
```

5. Update the Tanzu Application Platform package with the new values in `tap-values.yaml` by running:

   ```
   tanzu package installed update tap -p tap.tanzu.vmware.com -v TAP-VERSION  --va
   lues-file tap-values.yaml -n tap-install
   ```

   Where `TAP-VERSION` is the version that matches the values you used when you installed the profile.

Configure a corporate HTTP or HTTPS proxy

You can configure Tanzu Developer Portal to route its traffic through a specified HTTP or HTTPS proxy. This includes all outgoing requests that Backstage or Tanzu Developer Portal makes. This topic shows you how to configure proxy settings through the values file.

Proxy variables

Tanzu Developer Portal uses two optional variables to support HTTP and HTTPS proxy configuration:

- `HTTP_PROXY` is a host or IP address of a server that is a proxy. All outgoing requests, whether they are HTTP or HTTPS, are sent to this proxy instead of their intended destination. This value must include the proxy name or IP address and the port.
- `NO_PROXY` is a comma-separated list of hosts or IP addresses. When both this variable and `HTTP_PROXY` is set, traffic is directed through `HTTP_PROXY` unless it matches the host (or IP address) of one of the entries in this list.

If any plug-ins need access to resources that can only be accessed through a proxy then you add the `HTTP_PROXY` value. For resources that are reachable without going through the proxy server, their domains are listed in the `NO_PROXY` variable.

Define the proxy server

To define the proxy server in `tap-values.yaml` in a Tanzu Application Platform installation, add the following example section and edit it as needed:

```
tap_gui:
  HTTP_PROXY: http://foo:bar@127.0.0.1:8888
  NO_PROXY: 'bar.com,baz.com'
```

`NO_PROXY` default values

Tanzu Developer Portal applies default values for `NO_PROXY` to its Kubernetes manifests. These default values are implemented any time you use the `HTTP_PROXY` variable, even when you have not specified `NO_PROXY`. When you do specify a value for `NO_PROXY`, your input is prepended to the default values with a comma.

`tap-values.yaml` snippet example:

```
tap_gui:
  HTTP_PROXY: http://foo:bar@127.0.0.1:8888
  NO_PROXY: 'bar.com,baz.com'
```

Command output example:

```
$ kubectl -n tap-gui get deployment server -o jsonpath="{.spec.template.spec.container
s[0].env[?(@.name=='NO_PROXY')].value}"
bar.com,baz.com,.local,.local.,localhost,.metadata-store,.accelerator-system,$(KUBERNE
TES_SERVICE_HOST)
```

The list of default values is as follows and only affects requests within the cluster:

```
.local,.local.,localhost,.metadata-store,.accelerator-system,$(KUBERNETES_SERVICE_HOS
T)
```

See which values are in use on your installation

To see which values are in use on your installation, run:

```
kubectl -n tap-gui get deployment server -o jsonpath="{.spec.template.spec.containers
[0].env[?(@.name=='NO_PROXY')].value}"
```

Override the default values

Currently the only way to override the list of default values is to use an overlay and replace the entire list. For example:

```
apiVersion: v1
kind: Secret
metadata:
  name: no-proxy-overlay
  namespace: tap-install
stringData:
  custom-app-image-overlay.yaml: |
    #@ load("@ytt:overlay", "overlay")

    #! makes an assumption that tap-gui is deployed in the namespace: "tap-gui"

    #@overlay/match by=overlay.subset({"kind": "Deployment", "metadata": {"name": "ser
ver", "namespace": "tap-gui"}}), expects="1+"
    ---
    spec:
      template:
        spec:
          containers:
            #@overlay/match by=overlay.subset({"name": "backstage"}),expects="1+"
            #@overlay/match-child-defaults missing_ok=True
            - env:
                - name: NO_PROXY
                  value: 'MY-VALUES'
```

For more information about how to apply overlays, see Customize your package installation.

Upgrade Tanzu Developer Portal

This topic tells you how to upgrade Tanzu Developer Portal outside of a Tanzu Application Platform profile installation. If you installed Tanzu Application Platform through a profile, see Upgrading Tanzu Application Platform instead.

Considerations

As part of the upgrade, Tanzu Application Platform updates its container with the new version.

As a result, if you installed Tanzu Developer Portal without the support of a backing database, you lose your in-memory data for any manual component registrations when the container restarts. While the update is pulling the new pod from the registry, users might experience a short UI interruption and might need to re-authenticate because the in-memory session data is rebuilt.

Upgrade within a Tanzu Application Platform profile

If you installed Tanzu Developer Portal as part of a Tanzu Application Platform profile, see Upgrading Tanzu Application Platform.

Upgrade Tanzu Developer Portal individually

These steps only apply to installing Tanzu Developer Portal individually, not as part of a Tanzu Application Platform profile.

To upgrade Tanzu Developer Portal outside of a Tanzu Application Platform profile:

1. Ensure that your repository has access to the new version of the package by running:

   ```
   tanzu package available list tap-gui.tanzu.vmware.com -n tap-install
   ```

   For example:

   ```
   $ tanzu package available list tap-gui.tanzu.vmware.com -n tap-install
   - Retrieving package versions for tap-gui.tanzu.vmware.com...
     NAME                       VERSION  RELEASED-AT
     tap-gui.tanzu.vmware.com  1.0.1    2021-12-22 17:45:51 +0000 UTC
     tap-gui.tanzu.vmware.com  1.0.2    2022-01-25 01:57:19 +0000 UTC
   ```

2. Perform the package upgrade by using the targeted package update version. Run:

   ```
   tanzu package installed update tap-gui -p tap-gui.tanzu.vmware.com -v VERSION
   --values-file \
   TAP-GUI-VALUES.yaml -n tap-install
   ```

   Where:

   - `VERSION` is the target version of Tanzu Developer Portal that you want.
   - `TAP-GUI-VALUES` is the configuration values file that contains the configuration used when you installed Tanzu Developer Portal.

3. Verify that you upgraded your application by running:

   ```
   tanzu package installed get tap-gui -n tap-install
   ```

Troubleshoot Tanzu Developer Portal

This topic tells you how to troubleshoot issues encountered when installing Tanzu Developer Portal. The topic is divided into sections:

- General issues
- Runtime Resources tab issues
- Accelerators page issues

- Supporting ImageVulnerabilityScan issues
- Security Analysis plug-in issues
- Supply Chain Choreographer plug-in issues

General issues

The following are general issues:

Tanzu Developer Portal reports that the port range is not valid

Symptom

You provided a full URL in a `backend.reading.allow` entry, as in this example `tap-values.yaml` snippet:

```
tap_gui:
  app_config:
    backend:
      reading:
        allow:
          - host: http://gitlab.example.com/some-group/some-repo/-/blob/main/catalog-i
nfo.yaml
```

and you see the following error message:

```
Backend failed to start up, Error: Port range is not valid: //gitlab.example.com/some-
group/some-repo/-/blob/main/catalog-info.yaml
```

Cause

Tanzu Developer Portal expects a host name to be passed into the field `backend.reading.allow[].host`.

Solution

Edit your `tap-values.yaml` file as in the following example:

```
tap_gui:
  app_config:
    backend:
      reading:
        allow:
          - host: gitlab.example.com
            paths: ['/some-group/some-repo/']
```

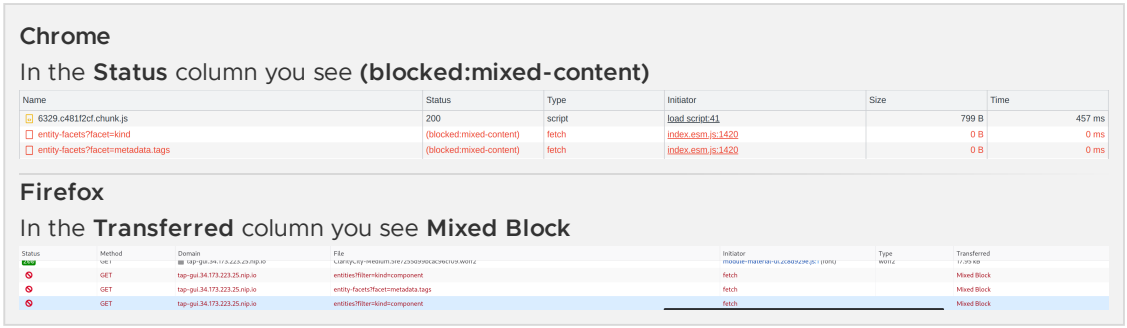
Tanzu Developer Portal does not load the catalog

Symptom

You are able to visit Tanzu Developer Portal, but it does not load the catalog and you see the following error message.

```
> Error: Could not fetch catalog entities.
> TypeError: Failed to fetch
```

When viewing your network tab you see that your browser has not downloaded mixed content. This might look different on different browsers.



Cause

As of Tanzu Application Platform v1.5, Tanzu Developer Portal provides TLS connections by default. Because of this, if you visit a Tanzu Developer Portal site your connection is automatically upgraded to https.

You might have manually set the fields `app.baseUrl`, `backend.baseUrl`, and `backend.cors.origin` in your `tap-values.yaml` file. Tanzu Developer Portal uses the `baseUrl` to determine how to create links to fetch from its APIs. The combination of these two factors causes your browser to attempt to fetch mixed content.

Solution

The solution is to delete these fields or update your values in `tap-values.yaml` to reflect that your Tanzu Developer Portal instance is serving https, as in the following example:

```
tap_gui:
  app_config:
    app:
      baseUrl: https://tap-gui.INGRESS-DOMAIN/
    backend:
      baseUrl: https://tap-gui.INGRESS-DOMAIN/
      cors:
        origin: https://tap-gui.INGRESS-DOMAIN/
```

Where `INGRESS-DOMAIN` is the ingress domain you have configured for Tanzu Application Platform.

The installer determines acceptable values based on your `tap_gui.ingressDomain` or `shared.ingress_domain` and the TLS status of the installation.

Updating a supply chain causes an error (`Can not create edge...`)

Symptom

Updating a supply chain causes an error (`Can not create edge...`) when an existing workload is clicked in the Workloads table and that supply chain is no longer present.

Solution

Recreate the same workload to execute through the new or updated supply chain.

Catalog not found

Symptom

When you pull up Tanzu Developer Portal, you get the error `Catalog Not Found`.

Cause

The catalog plug-in can't read the Git location of your catalog definition files.

Solution

1. Ensure you have built your own Backstage-compatible catalog or that you have downloaded one of the Tanzu Developer Portal catalogs from VMware Tanzu Network.
2. Ensure you defined the catalog in the values file that you input as part of installation. To update this location, change the definition file:
    - Change the Tanzu Application Platform profile file if installed by using a profile.
    - Change the standalone Tanzu Developer Portal values file if you're only installing that package on its own.

    ```
    namespace: tap-gui
    service_type: SERVICE-TYPE
    app_config:
      catalog:
        locations:
          - type: url
            target: https://GIT-CATALOG-URL/catalog-info.yaml
    ```

3. Provide the proper integration information for the Git location you specified earlier.

    ```
    namespace: tap-gui
    service_type: SERVICE-TYPE
    app_config:
      app:
        baseUrl: https://EXTERNAL-IP:PORT
      integrations:
        gitlab: # Other integrations available
          - host: GITLAB-HOST
            apiBaseUrl: https://GITLAB-URL/api/v4
            token: GITLAB-TOKEN
    ```

You can substitute for other integrations as defined in the Backstage documentation.

No configured authentication provider

Symptom

When you load Tanzu Developer Portal in a browser, the following message appears:

```
No configured authentication providers. Please configure at least one.
```

Cause

You have not configured any authentication providers and you have not allowed guest access.

Solution

Configure an authentication provider or allow guest access.

Issues updating the values file

Symptom

After updating the configuration of Tanzu Developer Portal, either by using a profile or as a standalone package installation, you don't know whether the configuration has reloaded.

Solution

1. Get the name you need by running:

   ```
   kubectl get pods -n tap-gui
   ```

   For example:

   ```
   $ kubectl get pods -n tap-gui
   NAME                       READY   STATUS    RESTARTS   AGE
   server-6b9ff657bd-hllq9    1/1     Running   0          13m
   ```

2. Read the log of the pod to see if the configuration reloaded by running:

   ```
   kubectl logs NAME -n tap-gui
   ```

   Where `NAME` is the value you recorded earlier, such as `server-6b9ff657bd-hllq9`.

3. Search for a line similar to this one:

   ```
   2021-10-29T15:08:49.725Z backstage info Reloaded config from app-config.yaml, a
   pp-config.yaml
   ```

4. If need be, delete and re-instantiate the pod.

   Caution

   Depending on your database configuration, deleting, and re-instantiating the pod might cause the loss of user preferences and manually registered entities. If you have configured an external PostgreSQL database, `tap-gui` pods are not stateful. In most cases, state is held in ConfigMaps, Secrets, or the database. For more information, see Configuring the Tanzu Developer Portal database and Register components.

   To delete and re-instantiate the pod, run:

   ```
   kubectl delete pod -l app=backstage -n tap-gui
   ```

Pull logs from Tanzu Developer Portal

Symptom

You have a problem with Tanzu Developer Portal, such as `Catalog: Not Found`, and don't have enough information to diagnose it.

Solution

Get timestamped logs from the running pod and review the logs:

1. Pull the logs by using the pod label by running:

```
kubectl logs -l app=backstage -n tap-gui
```

2. Review the logs.

Ad-blocking software interference

Symptom

One or both of the following is true:

- You cannot access some, or all, of Tanzu Developer Portal.
- Telemetry collection within the VMware Customer Experience Improvement Program is failing.

Cause

Your ad-blocking browser extension or standalone ad-blocking software is causing interference.

Solution

Add Tanzu Developer Portal to your ad-blocking allowlist. Alternatively, deactivate the ad-blocking software or turn off Pendo telemetry collection.

Runtime Resources tab issues

Here are some common troubleshooting steps for errors presented in the **Runtime Resources** tab.

Error communicating with Tanzu Application Platform web server

Symptom

When accessing the **Runtime Resource Visibility** tab, the system displays `Error communicating with TAP GUI back end.`

Causes

- An interrupted Internet connection
- Error with the back end service

Solution

1. Confirm that you have Internet access.
2. Confirm that the back-end service is running correctly.
3. Confirm the cluster configuration is correct.

No data available

Symptom

When accessing the **Runtime Resource Visibility** tab, the system displays

```
One or more resources are missing. This could be due to a label mismatch. \
Please make sure your resources have the label(s) "LABEL_SELECTOR".
```

Cause

No communications error has occurred, but no resources were found.

Solution

Confirm that you are using the correct label:

1. Verify the Component definition includes the annotation `backstage.io/kubernetes-label-selector`.
2. Confirm your Kubernetes resources correspond to that label drop-down menu.

Errors retrieving resources

Symptom

When opening the **Runtime Resource Visibility** tab, the system displays `One or more resources might be missing because of cluster query errors.`

The reported errors might not indicate a real problem. A build cluster might not have runtime CRDs installed, such as Knative Service, and a run cluster might not have build CRDs installed, such as a Cartographer workload. In these cases, 403 and 404 errors might be false positives.

You might receive the following error messages because these resources might not be required for your specific Tanzu Application Platform profile. These error messages are known issues:

- `Access error when querying cluster CLUSTER_NAME for resource KUBERNETES_RESOURCE_PATH (status: 401). Contact your administrator.`
- `Access error when querying cluster CLUSTER_NAME for resource KUBERNETES_RESOURCE_PATH (status: 403). Contact your administrator.`
- `Knative is not installed on CLUSTER_NAME (status: 404). Contact your administrator.`
- `Error when querying cluster CLUSTER_NAME for resource KUBERNETES_RESOURCE_PATH (status: 404). Contact your administrator.`

Accelerators page issues

Here are some common troubleshooting steps for errors displayed on the **Accelerators** page.

No accelerators

Symptom

When the `app_config.backend.reading.allow` section is configured in the `tap-values.yaml` file during the `tap-gui` package installation, there are no accelerators on the **Accelerators** page.

Cause

This section in `tap-values.yaml` overrides the default configuration that gives Tanzu Developer Portal access to the accelerators.

Solution

As a workaround, provide a value for Application Accelerator in this section. For example:

```
app_config:
  # Existing tap-values yaml above
  backend:
    reading:
      allow:
      - host: acc-server.accelerator-system.svc.cluster.local
```

Supporting ImageVulnerabilityScan issues

The following troubleshooting issues concern `ImageVulnerabilityScan`.

No Vulnerability data

Symptom

SCST - Scan 2.0 is enabled and there is no vulnerability data on the Security Analysis and Supply Chain Choreographer dashboards.

Cause

Tanzu Developer Portal lacks the required access to detect the new `ImageVulnerabilityScan` custom resource.

Solution

As a workaround, you can apply a ytt overlay to add permissions to the Kubernetes role that is used to access the `ImageVulnerabilityScan` resource. For more information about how overlays work with Tanzu Application Platform, see Customize your package installation.

1. Create a `secret.yml` file with a `Secret` that contains your ytt overlay. For example:

   ```
   kind: Secret
   metadata:
     name: add-tap-portal-ivs-permissions
     namespace: tap-install
     annotations:
       kapp.k14s.io/change-group: "tap-overlays"
   type: Opaque
   stringData:
     add-tap-portal-ivs-permissions.yaml: |
       #@ load("@ytt:overlay", "overlay")
       #@overlay/match by=overlay.subset({"metadata":{"name":"k8s-reader"}, "kin
   d": "ClusterRole"})
       ---
       rules:
         #@overlay/append
           - apiGroups:
             - app-scanning.apps.tanzu.vmware.com
             resources:
             - imagevulnerabilityscans
             verbs:
             - get
             - watch
             - list
   ```

2. Apply the `Secret` to your cluster by running:

   ```
   kubectl apply -f secret.yml
   ```

3. Update your values file to include a `package_overlays` field:

```
package_overlays:
- name: tap-gui
secrets:
- name: add-tap-portal-ivs-permissions
```

4. Update Tanzu Developer Portal with the new Tanzu Application Platform values by running:

```
tanzu package installed update tap -p tap.tanzu.vmware.com -n tap-install --val
ues-file tap-values.yaml
```

Scanner name not shown in Tanzu Developer Portal for SCST - Scan 2.0

Symptom

SCST - Scan 2.0 is enabled and the scanner name in the Security Analysis and Supply Chain Choreographer dashboards does not appear.

Cause

The `ImageVulnerabilityScan` custom resource lacks the required annotation for Tanzu Developer Portal to detect it.

Solution

Add the `app-scanning.apps.tanzu.vmware.com/scanner-name` annotation to `ImageVulnerabilityScan`:

```
apiVersion: app-scanning.apps.tanzu.vmware.com/v1alpha1
kind: ImageVulnerabilityScan
metadata:
  annotations:
    app-scanning.apps.tanzu.vmware.com/scanner-name: SCANNER-NAME
```

Where `SCANNER-NAME` is the name that is reported in Tanzu Developer Portal.

Security Analysis plug-in issues

These are troubleshooting issues for the Security Analysis plug-in.

Empty Impacted Workloads table

Symptom

The Impacted Workloads table is empty on the **CVE and Package Details** pages.

Cause

The relevant CVE belongs to a workload that has only completed one type of vulnerability scan (either image or source).

Solution

A fix is planned for Tanzu Developer Portal v1.5.1.

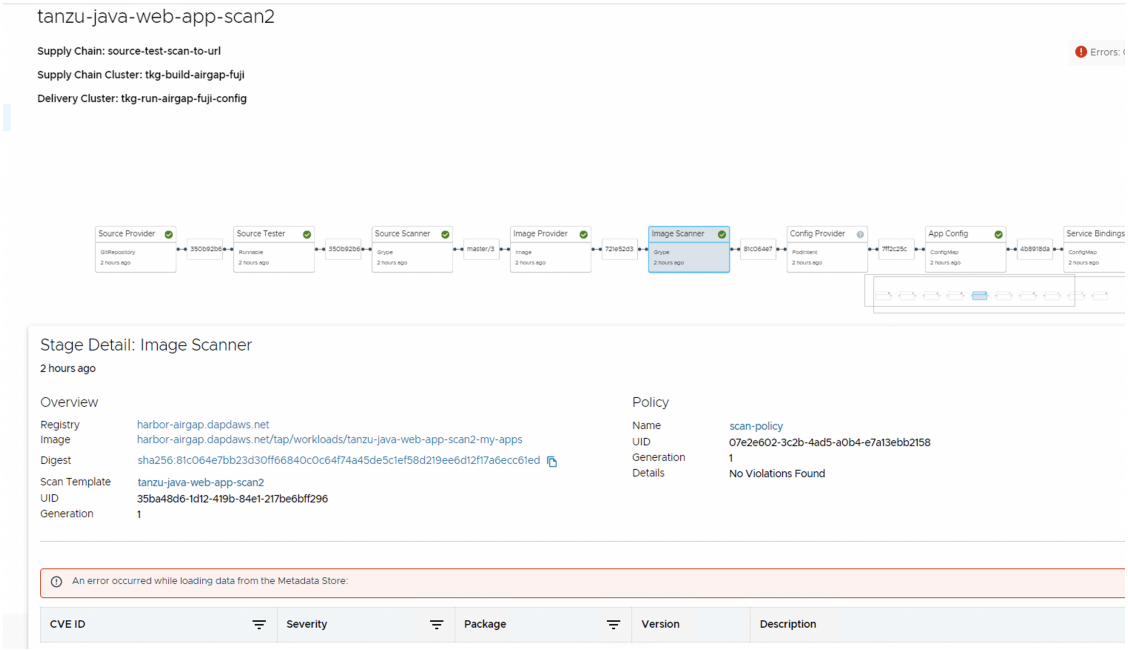
Supply Chain Choreographer plug-in issues

These are troubleshooting issues for the Supply Chain Choreographer plug-in.

An error occurred while loading data from the Metadata Store

Symptom

In the Supply Chain Choreographer plug-in, you see the error message `An error occurred while loading data from the Metadata Store`.



Cause

There are multiple potential causes. The most common cause is `tap-values.yaml` missing the configuration that enables Tanzu Developer Portal to communicate with Supply Chain Security Tools - Store.

Solution

See Supply Chain Choreographer - Enable CVE scan results for the necessary configuration to add to `tap-values.yaml`. After adding the configuration, update your Tanzu Application Platform deployment or Tanzu Developer Portal deployment with the new values.

SBOMs do not download when automatically configuring Tanzu Developer Portal for SCST - Store

Symptom

The auto configuration between Tanzu Developer Portal and SCST - Store prevents the SBOM feature from working. The SBOM feature was introduced in Tanzu Application Platform v1.6.

Cause

Backstage expects `allowedHeaders` values for SCST - Store to reply with proper XML and JSON SBOM responses. The `allowedHeaders` values are missing.

Solution

Edit your `tap-values.yaml` file so that `allowedHeaders` and the accompanying values are included:

```
tap_gui:
  app_config:
    proxy:
      /metadata-store:
        target: https://metadata-store-app.metadata-store:8443/api/v1
        changeOrigin: true
        secure: false
        allowedHeaders: ['Accept', 'Report-Type-Format']
        headers:
          Authorization: "Bearer ACCESS-TOKEN"
          X-Custom-Source: project-star
```

Where `ACCESS-TOKEN` is the token you obtained after creating a read-write service account. For more information, see Manually connect Tanzu Developer Portal to Metadata Store.

Overview of Tanzu Developer Tools for IntelliJ

Tanzu Developer Tools for IntelliJ is the official VMware Tanzu IDE extension for IntelliJ IDEA. The extension helps you develop with Tanzu Application Platform and enables you to rapidly iterate on your workloads on supported Kubernetes clusters that have Tanzu Application Platform installed.

Tanzu Developer Tools for IntelliJ currently supports Java applications on macOS and Windows.

Extension features

This extension gives the following features:

- **Deploy applications directly from IntelliJ:**

  Rapidly iterate on your applications on Tanzu Application Platform and deploy them as workloads directly from within IntelliJ.

- **See code updates running on-cluster in seconds:**

  With the use of Live Update facilitated by Tilt, deploy your workload once, save changes to the code and then, seconds later, see those changes reflected in the workload running on the cluster.

- **Debug workloads directly on the cluster:**

  Debug your application in a production-like environment by debugging on your Kubernetes cluster that has Tanzu Application Platform. An environment's similarity to production relies on keeping dependencies updated, among other variables.

- **See workloads running on the cluster:**

  From the Workloads panel you can see any workload found within the cluster and namespace specified in the current kubectl context.

- **Work with microservices in a Java monorepo:**

  Tanzu Developer Tools for IntelliJ v1.3 and later supports working with a monorepo containing multiple modules that represent different microservices. This makes it possible to deploy, debug, and Live Update multiple workloads simultaneously from the same IntelliJ multimodule project. For more information about projects with multiple modules, see the IntelliJ documentation. For more information about a typical monorepo setup, see Working with microservices in a monorepo.

- **Connect to a sandbox cluster:**

  You can connect to a Tanzu Application Platform sandbox instance from Tanzu Developer Tools for IntelliJ to try Tanzu Application Platform and see its benefits. To use Developer Sandbox and connect to it from your IDE, see the Tanzu Academy website.

Note

The new variation of Out of the Box (OOTB) Basic supply chains, which outputs Carvel packages to enable configuring multiple runtime environments, is not currently supported. For more information about the variation, see Carvel Package Supply Chains.

Next steps

Follow the steps to install the extension.

Overview of Tanzu Developer Tools for IntelliJ

Tanzu Developer Tools for IntelliJ is the official VMware Tanzu IDE extension for IntelliJ IDEA. The extension helps you develop with Tanzu Application Platform and enables you to rapidly iterate on your workloads on supported Kubernetes clusters that have Tanzu Application Platform installed.

Tanzu Developer Tools for IntelliJ currently supports Java applications on macOS and Windows.

Extension features

This extension gives the following features:

- **Deploy applications directly from IntelliJ:**

  Rapidly iterate on your applications on Tanzu Application Platform and deploy them as workloads directly from within IntelliJ.

- **See code updates running on-cluster in seconds:**

  With the use of Live Update facilitated by Tilt, deploy your workload once, save changes to the code and then, seconds later, see those changes reflected in the workload running on the cluster.

- **Debug workloads directly on the cluster:**

  Debug your application in a production-like environment by debugging on your Kubernetes cluster that has Tanzu Application Platform. An environment's similarity to production relies on keeping dependencies updated, among other variables.

- **See workloads running on the cluster:**

  From the Workloads panel you can see any workload found within the cluster and namespace specified in the current kubectl context.

- **Work with microservices in a Java monorepo:**

  Tanzu Developer Tools for IntelliJ v1.3 and later supports working with a monorepo containing multiple modules that represent different microservices. This makes it possible to deploy, debug, and Live Update multiple workloads simultaneously from the same IntelliJ multimodule project. For more information about projects with multiple modules, see the IntelliJ documentation. For more information about a typical monorepo setup, see Working with microservices in a monorepo.

- **Connect to a sandbox cluster:**

  You can connect to a Tanzu Application Platform sandbox instance from Tanzu Developer Tools for IntelliJ to try Tanzu Application Platform and see its benefits. To use Developer Sandbox and connect to it from your IDE, see the Tanzu Academy website.

> **Note**
>
> The new variation of Out of the Box (OOTB) Basic supply chains, which outputs Carvel packages to enable configuring multiple runtime environments, is not currently supported. For more information about the variation, see Carvel Package Supply Chains.

Next steps

Follow the steps to install the extension.

Install Tanzu Developer Tools for IntelliJ

This topic explains how to install the VMware Tanzu Developer Tools for IntelliJ IDE extension. The extension currently only supports Java applications on macOS and Windows.

Prerequisites

Before installing the extension, you must have:

- IntelliJ
- kubectl
- Tilt v0.30.12 or later
- Tanzu CLI and plug-ins
- A cluster with the Tanzu Application Platform Full profile or Iterate profile

> **Note**
>
> If you are an app developer, someone else in your organization might have already set up the Tanzu Application Platform environment.

Install

To install VMware Tanzu Developer Tools for IntelliJ:

1. Open IntelliJ.
2. Open the command palette, enter `Plugins`, and then click **Plugins**.
3. Select the **Marketplace** tab in the **Plugins Settings** dialog box.
4. In the search box enter `Tanzu`.
5. Click **Tanzu Developers Tools** and then click **Install**.

Update

To update to a later version, repeat the steps in the Install section. You do not need to uninstall the current version.

Uninstall

To uninstall the VMware Tanzu Developer Tools for IntelliJ:

1. Open the **Preferences** pane and then go to **Plugins**.
2. Select the extension, click the gear icon, and then click **Uninstall**.
3. Restart IntelliJ.

Next steps

Proceed to Getting started.

Get Started with Tanzu Developer Tools for IntelliJ

This topic guides you through getting started with Tanzu Developer Tools for IntelliJ.

Prerequisite

Install Tanzu Developer Tools for IntelliJ.

Configure Local Source Proxy or use a source image registry

Configure Local Source Proxy if you're able to do so. For more information, see the Local Source Proxy documentation.

If you cannot use Local Source Proxy, use a source image registry. Before deploying a workload, you must authenticate with an image registry to store your source code. You can use the Docker CLI to authenticate or you can set environment variables that the Tanzu CLI can use to authenticate.

Docker CLI
To authenticate by using the Docker CLI, run:

```
docker login $REGISTRY_HOSTNAME -u $REGISTRY_USERNAME -p $REGISTRY_PASSWORD
```

Tanzu CLI
To authenticate by using the Tanzu CLI, export environment variables by running:

```
export TANZU_APPS_REGISTRY_CA_CERT=PATH-TO-CA-CERT.nip.io.crt
export TANZU_APPS_REGISTRY_PASSWORD=USERNAME
export TANZU_APPS_REGISTRY_USERNAME=PASSWORD
```

`CA_CERT` is only needed for a custom or private registry.

For more information, see Workload creation fails due to authentication failure in Docker Registry.

Run Tanzu Developer Tools for IntelliJ

Run IntelliJ from a CLI, instead of through your operating system GUI, to avoid restricting the set of environment variables the app receives. This is especially relevant for macOS.

Limited environment variables can cause problems with cluster authentication for Tanzu Developer Tools for IntelliJ. For example, a common situation is that a sanitized `PATH` does not provide access to the `gke-cloud-auth-plugin` installed on your system. This makes Tanzu Developer Tools for IntelliJ unable to authenticate and access your GKE cluster.

This situation is complex and different things can go wrong depending on:

- Precisely how you installed various cloud-related CLI tools
- How you set environment variables
- Your OS version
- Which cloud provider and authentication method you are using

All of these problems are most easily avoided by running IntelliJ from a CLI. Run IntelliJ from a CLI in macOS by running:

```
open /Applications/IntelliJ\ IDEA.app
```

Set up Tanzu Developer Tools for IntelliJ

The extension makes use of the following files within your project:

- `workload.yaml`
- `catalog-info.yaml`
- `Tiltfile`
- `.tanzuignore`

You can create these files by using the instructions in this topic, or use the files in the View an example project section.

There are two ways to create these files:

- Using the code snippets that Tanzu Developer Tools provide, which create templates in empty files that you then fill in with the required information.
- Writing the files manually.

Create the `workload.yaml` file

You must include a file named `workload.yaml` in your project. For example, `my-project/config/workload.yaml`.

`workload.yaml` provides instructions to Supply Chain Choreographer about how to build and manage a workload. For more information, see Supply Chain Choreographer for Tanzu.

The Tanzu Developer Tools for IntelliJ extension requires only one `workload.yaml` file per project. `workload.yaml` must be a single-document YAML file, not a multi-document YAML file.

To create a `workload.yaml` file by using code snippets:

1. Right-click the IntelliJ project explorer and then click **New**.
2. Select the Tanzu workload.
3. Add the filename `workload`.
4. Fill in the template.

See the following `workload.yaml` example:

```
apiVersion: carto.run/v1alpa1
kind: Workload
metadata:
 name: APP-NAME
 labels:
   apps.tanzu.vmware.com/workload-type: WORKLOAD-TYPE
   app.kubernetes.io/part-of: APP-NAME
spec:
 source:
   git:
     url: GIT-SOURCE-URL
     ref:
       branch: GIT-BRANCH-NAME
```

Where:

- `APP-NAME` is the name of your application. For example, `my app`.
- `WORKLOAD-TYPE` is the type of workload for your app. For example, `web`. For more information, see Workload types.
- `GIT-SOURCE-URL` is the Git source code URL for your app. For example, `github.com/mycompany/myapp`.
- `GIT-BRANCH-NAME` is the branch of the Git source code you want to use. For example, `main`.

Alternatively you can use the Tanzu CLI to create a `workload.yaml` file. For more information, see Create or update a workload.

Create the `catalog-info.yaml` file

You must include a file named `catalog-info.yaml` in your project. For example, `my-project/catalog/catalog-info.yaml`.

`catalog-info.yaml` enables the workloads created with Tanzu Developer Tools for IntelliJ to be visible in Tanzu Developer Portal. For more information, see Overview of Tanzu Developer Portal.

To create a `catalog-info.yaml` file by using the code snippets:

1. Right-click the IntelliJ project explorer and then click **New**.
2. Select the Tanzu Catalog.
3. Add the filename `catalog-info`.
4. Fill in the template.

See the following `workload.yaml` example:

```
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
 name: APP-NAME
 description: APP-DESCRIPTION
 tags:
   - tanzu
 annotations:
   'backstage.io/kubernetes-label-selector': 'app.kubernetes.io/part-of=APP-NAME'
spec:
 type: service
 lifecycle: experimental
 owner: default-team
```

Where:

- `APP-NAME` is the name of your application.
- `APP-DESCRIPTION` is a description of your application.

Create the Tiltfile file

In your project you must include a file named `Tiltfile` with no extension (no filetype), such as `my-project/Tiltfile`.

The `Tiltfile` provides the configuration for Tilt to enable your project to Live Update on the Tanzu Application Platform-enabled Kubernetes cluster. For more information, see the Tilt documentation.

The Tanzu Developer Tools for IntelliJ extension requires only one Tiltfile per project.

The following is an example `Tiltfile`:

```
LOCAL_PATH = os.getenv("LOCAL_PATH", default='.')
NAMESPACE = os.getenv("NAMESPACE", default='default')

k8s_custom_deploy(
   'APP-NAME',
   apply_cmd="tanzu apps workload apply -f PATH-TO-WORKLOAD-YAMl --live-update" +
       " --local-path " + LOCAL_PATH +
       " --namespace " + NAMESPACE +
       " --yes >/dev/null" +
       " && kubectl get workload APP-NAME --namespace " + NAMESPACE + " -o yaml",
   delete_cmd="tanzu apps workload delete -f PATH-TO-WORKLOAD-YAML --namespace " + NAM
ESPACE + " --yes" ,
   deps=['pom.xml', './target/classes'],
   container_selector='workload',
   live_update=[
       sync('./target/classes', '/workspace/BOOT-INF/classes')
   ]
)

k8s_resource('APP-NAME', port_forwards=["8080:8080"],
   extra_pod_selectors=[{'carto.run/workload-name': 'APP-NAME', 'app.kubernetes.io/com
ponent': 'run'}])
allow_k8s_contexts('CONTEXT-NAME')
```

Where:

- `APP-NAME` is the name of your application.
- `PATH-TO-WORKLOAD-YAML` is the local file system path to your `workload.yaml` file. For example, `config/workload.yaml`.
- `CONTEXT-NAME` is the name of your current Kubernetes context. If your Tanzu Application Platform-enabled Kubernetes cluster is running on your local machine, you can remove the entire `allow_k8s_contexts` line. For more information about this line, see the Tilt documentation.

If you want to compile the source image from a local directory other than the project directory, change the value of `local_path`. For more information, see local path in the glossary.

Create the `.tanzuignore` file

In your project, you can include a file named `.tanzuignore` with no file extension. For example, `my-project/.tanzuignore`.

When working with local source code, `.tanzuignore` excludes files from the source code that are uploaded within the image. It has syntax similar to the `.gitignore` file.

For an example, see the `.tanzuignore` file in GitHub that is used for the sample Tanzu Java web app. You can use the file as it is or edit it for your needs.

View an example project

Before you begin, you need a container image registry to use the sample application. There are two ways to view a sample application that demonstrates the necessary configuration files.

Use Application Accelerator
If your company has configured Application Accelerator, you can obtain the sample application there if it was not removed. To view the example using Application Accelerator:

1. Open Application Accelerator. The Application Accelerator location varies based on where your company placed it. Contact the appropriate team to learn its location.
2. Search for `Tanzu Java Web App` in the Application Accelerator.
3. Add the required configuration information and generate the application.
4. Unzip the application and open the directory in IntelliJ.

Clone from GitHub
To clone the example from GitHub:

1. Use `git clone` to clone the application-accelerator-samples repository from GitHub.
2. Go to the `tanzu-java-web-app` directory.
3. Open the `Tiltfile` and replace `your-registry.io/project` with your registry.

Next steps

Use Tanzu Developer Tools for IntelliJ.

Use Tanzu Developer Tools for IntelliJ

Ensure that the project you want to use the Tanzu Developer Tools for IntelliJ extension with has the required files specified in Getting started.

The extension requires only one Tiltfile and one `workload.yaml` file per project. `workload.yaml` must be a single-document YAML file, not a multi-document YAML file.

Workload Actions

The extension enables you to apply, debug, and Live Update your application on a Kubernetes cluster that has Tanzu Application Platform. The developer sandbox experience enables developers to Live Update their code and simultaneously debug the updated code, without having to deactivate Live Update when debugging.

Apply a workload

The extension enables you to apply workloads on your Kubernetes cluster that has Tanzu Application Platform.

To apply a workload:

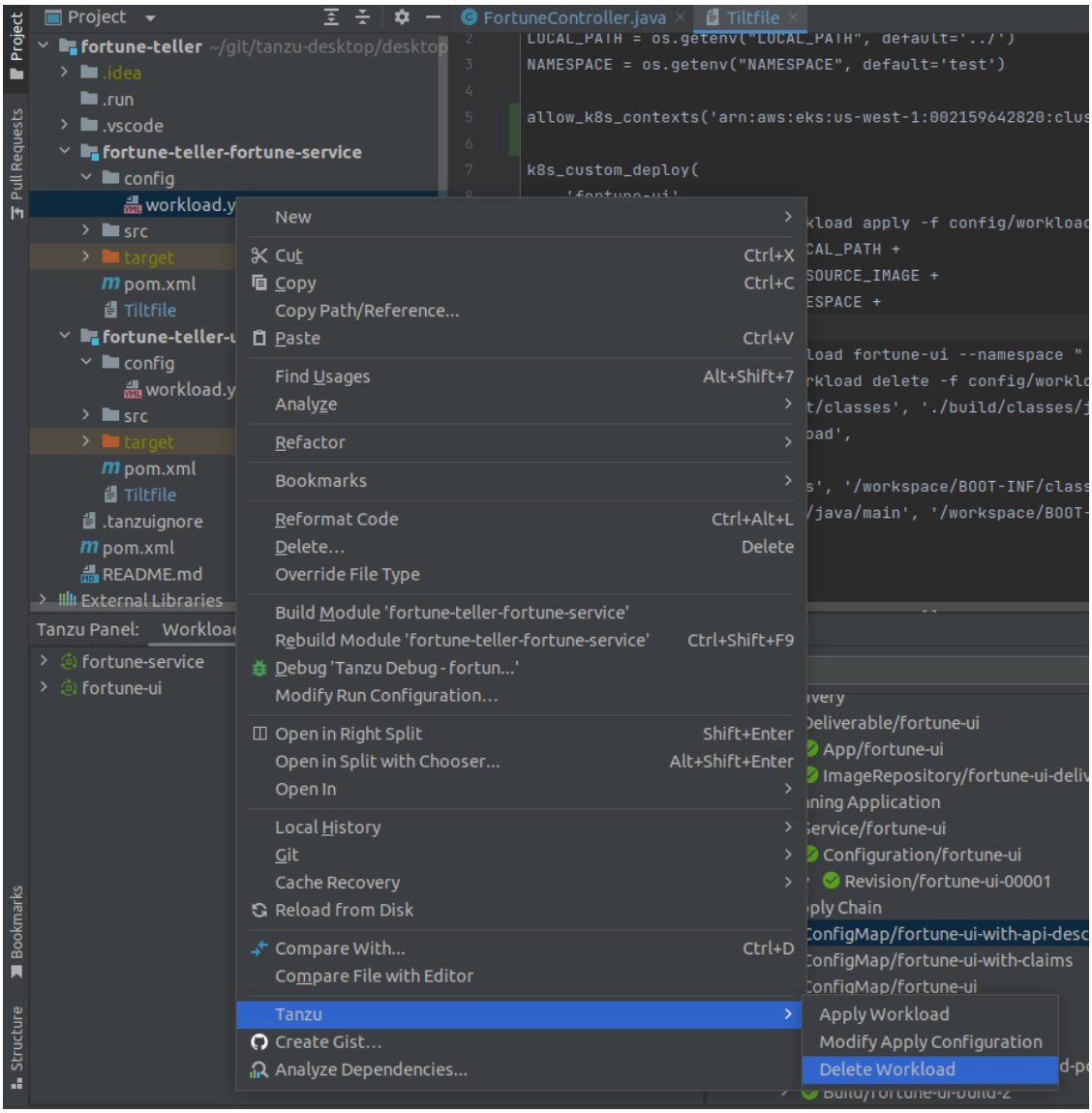
1. Right-click anywhere in the IntelliJ project explorer and click **Tanzu** > **Apply Workload** or right-click on an associated workload in the Workloads panel and click **Apply Workload**.
2. Click **Tanzu > Modify Apply Configuration**.

The `Tanzu workload apply` command is triggered in the terminal and the workload is applied. A new workload appears on the Tanzu panel.

Delete a workload

The extension enables you to delete workloads on your Kubernetes cluster that has Tanzu Application Platform.

To delete a workload right-click anywhere in the IntelliJ project explorer and click **Tanzu** > **Delete Workload** or right-click on an associated workload in the Workloads panel and click **Delete Workload**.



A message appears that prompts you to delete the workload and not warn again, delete the workload, or cancel. A notification appears showing that the workload was deleted.

Debugging on the cluster

The extension enables you to debug your application on a Kubernetes cluster that has Tanzu Application Platform.

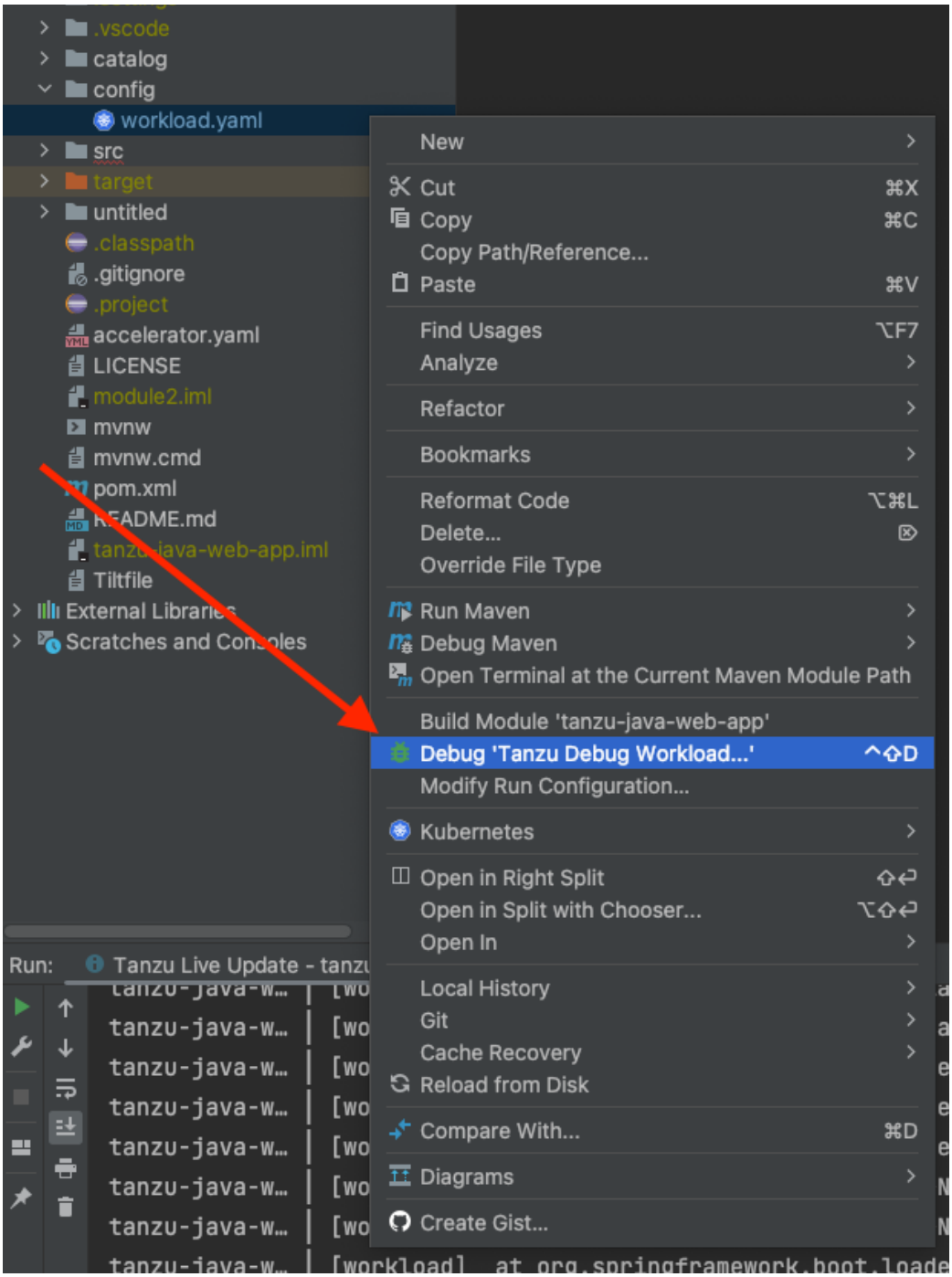
Debugging requires a single-document `workload.yaml` file in your project. For how to create `workload.yaml`, see Set up Tanzu Developer Tools.

The developer sandbox experience enables developers to Live Update their code, and simultaneously debug the updated code, without having to deactivate Live Update when debugging.

Start debugging on the cluster

To start debugging on the cluster:

1. Add a breakpoint in your code.
2. Right-click the `workload.yaml` file in your project and click **Debug 'Tanzu Debug Workload...'** in the pop-up menu or right-click on an associated workload in the Workloads panel and click **Debug Workload**.



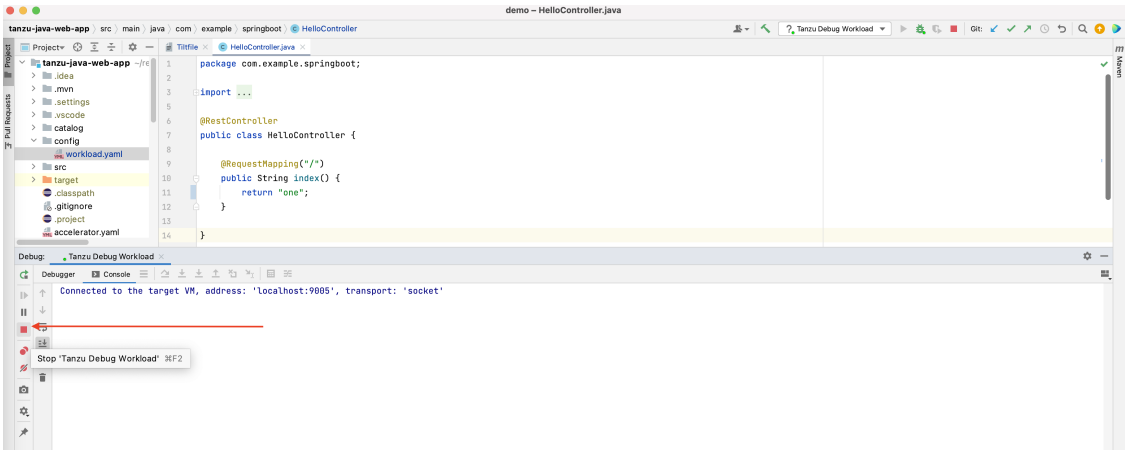
3. Ensure that the configuration parameters are set:

    - **Source Image:** This is the registry location for publishing local source code. For example, `registry.io/yourapp-source`. The source image parameter is optional if you have configured Local Source Proxy.
    - **Local Path:** This is the path on the local file system to a directory of source code to build.
    - **Namespace:** This is the namespace that workloads are deployed into.

    You can also manually create Tanzu Debug configurations by using the **Edit Configurations** IntelliJ UI.

Stop Debugging on the Cluster

Click the stop button in the **Debug** overlay to stop debugging on the cluster.



Live Update

See the following sections for how to use Live Update.

Start Live Update

Before using Live Update, verify that your auto-save setting is either off or on with a delay. The delay must be long enough for the application to restart between auto saves to allow enough time for your app to Live Update when files change. This auto-save setting is in **Preferences** > **Appearance & Behavior** > **System Settings** > **Autosave**.

To start Live Update:

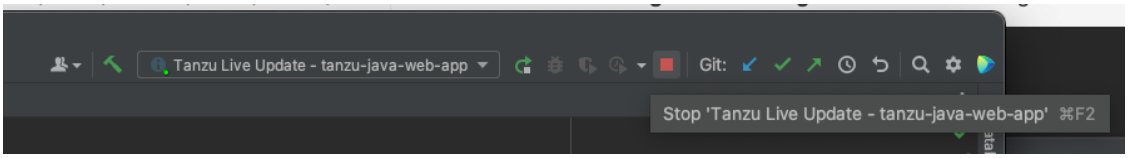
1. Right-click your project's Tiltfile and then click **Run 'Tanzu Live Update - ...'** or right-click on an associated workload in the Workloads panel and then click **Live Update Workload**.
2. Ensure that the configuration parameters are set:
    - **Source Image:** This is the registry location for publishing local source code. For example, `registry.io/yourapp-source`. It must include both a registry and a project name. The source image parameter is optional if you have configured Local Source Proxy.
    - **Local Path:** This is the path on the local file system to a directory of source code to build.
    - **Namespace:** This is the namespace that workloads are deployed into.

Note

You must compile your code before the changes are synchronized to the container. For example, `Build Project`: `⌘+F9`.

Stop Live Update

To stop Live Update, use the native controls to stop the Tanzu Live Update Run Configuration that is running.



Tanzu Workloads panel

The current state of the workloads is visible in the Tanzu Workloads view. This view is a separate section in the bottom of the Explorer view in the Side Bar. The view shows the current status of each workload, namespace, and cluster. It also shows whether Live Update and Debug is running, stopped, or deactivated.

The Tanzu Activity tab in the Panels view enables developers to visualize the supply chain, delivery, and running application pods. The tab enables a developer to view and describe logs on each resource associated with a workload from within their IDE. The tab displays detailed error messages for each resource in an error state.

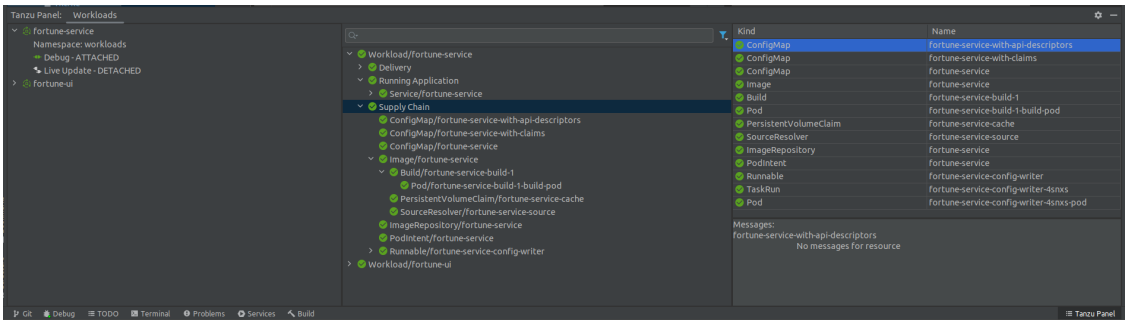
Workload commands are available from the Tanzu Workloads panel on workloads that have an associated module in the current project.

This association is based on a module name and a workload name matching. For example, a project with a module named `my-app` is associated with a deployed workload named `my-app`.

When taking an action from the Tanzu Workloads panel, the action uses the namespace of the deployed workload regardless of the configuration in the module.

For example, you might have a Live Update configuration with a namespace argument of `my-apps-1`, but running the action from a deployed workload in namespace `my-apps-2` starts a Live Update session with a namespace argument of `my-apps-2`.

The Tanzu Workloads panel uses the cluster and defaults to the namespace specified in the current kubectl context.



To add a namespace:

1. View the current context and namespace by running:

   ```
   kubectl config get-contexts
   ```

2. Set a namespace for the current context by running:

   ```
   kubectl config set-context --current --namespace=YOUR-NAMESPACE
   ```

3. If you are using the `KUBECONFIG` environment variable to organize access to different clusters on macOS, use the CLI to run the IDE instead of Dock or Spotlight. For why, see Troubleshooting.

To add additional namespaces to your Workloads panel:

1. Click on the gear icon in the upper right corner of the Workloads panel.
2. Click on **Select Namespaces...**
3. Select the checkboxes of the namespaces that you want to add to your panel.

Working with microservices in a monorepo

A monorepo is single Git repository that contains multiple workloads. Each individual workload is placed in a subfolder of the main repository.

You can find an example of this in Application Accelerator.

The relevant accelerator is called Spring SMTP Gateway, and you can obtain its source code as an accelerator or directly from the application-accelerator-samples GitHub repository.

This project is an example of a typical layout:

- `MONO-REPO-ROOT/`
  - `pom.xml` (parent pom)
  - `microservice-app-1/`
  - `pom.xml`
  - `mvnw` (and other mvn-related files for building the workload)
  - `Tiltfile` (supports Live Update)
  - `config`
    - `workload.yaml` (supports deploying and debugging from IntelliJ)
  - `src/` (contains source code for this microservice)
  - `microservice-app-2/`
  - …similar layout

Recommended structure: Microservices that can be built independently

In this example, each of the microservices can be built independently of one another. Each subfolder contains everything needed to build that workload.

This is reflected in the `source` section of `workload.yaml` by using the `subPath` attribute:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: microservice-app-1
  ...
spec:
  source:
    git:
      ref:
        branch: main
      url: https://github.com/kdvolder/sample-mono-repo.git
    subPath: microservice-app-1 # build only this
  ...
```

For setting up your own repositories, it's best practice to set up a monorepo so that each microservice can be built completely independently.

To work with these monorepos:

1. Import the monorepo as a project into IntelliJ.
2. Interact with each of the subfolders as you would interact with a project containing a single workload.

Alternative structure: Services with build-time interdependencies

Some monorepos do not have submodules that can be independently built. Instead the `pom.xml` files of the submodules are set up to have some build-time interdependencies. For example:

- A submodule `pom.xml` can reference the parent `pom.xml` as a common place for centralized dependency management.
- A microservice submodule can reference another, as a maven dependency.
- Several microservice submodules can reference one or more shared library modules.

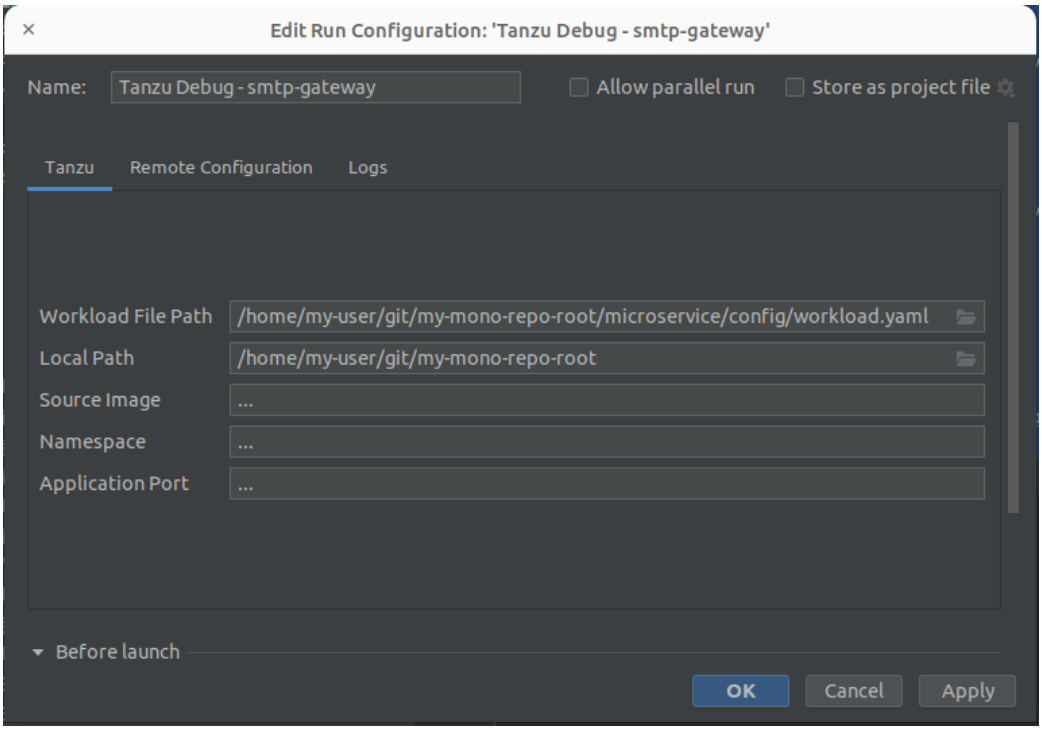
For these projects, make these adjustments:

1. Make `workload.yaml` point to the repository root, not a subfolder. Because submodules have dependencies on code outside of their own subfolder, all source code from the repository must be supplied to the workload builder.
2. Make `workload.yaml` specify additional buildpack arguments through environment variables. They differentiate the submodule that the build is targeting.

   Both of these `workload.yaml` changes are in the following example:

   ```
   apiVersion: carto.run/v1alpha1
   kind: Workload
   metadata:
   name: fortune-ui
   labels:
     apps.tanzu.vmware.com/workload-type: web
     app.kubernetes.io/part-of: fortune-ui
   spec:
   build:
     env:
        - name: BP_MAVEN_BUILD_ARGUMENTS
        value: package -pl fortune-teller-ui -am # indicate which module to build.
        - name: BP_MAVEN_BUILT_MODULE
        value: fortune-teller-ui # indicate where to find the built artefact to de
   ploy.
   source:
     git:
        url: https://github.com/my-user/fortune-teller # repository root
        ref:
        branch: main
   ```

   For more information about these and other `BP_xxx` buildpack parameters, see the Buildpack documentation.
3. Make the local path attribute in the launch configuration for each workload point to the path of the repository root. Because submodules have dependencies on code outside of their own subfolder, all source code from the repository must be supplied to the workload builder.



Change logging verbosity

The Tanzu Language Server saves logs to `~/tanzu-langserver.log`. You can change the log verbosity in **Preferences** > **Tools** > **Tanzu Developer Tools**.

Work with Native Image for Java

Native Image is technology for compiling Java code ahead of time to a binary, which is a native executable file. For more information about Native Image, see the GraalVM documentation.

Native Image requires some changes to your `workload.yaml` files, such as adding new environment variables to the build section of the workload specifications:

```
spec:
  build:
    env:
      - name: BP_NATIVE_IMAGE
        value: "true"
      - name: BP_MAVEN_BUILD_ARGUMENTS
        value: -Dmaven.test.skip=true --no-transfer-progress package -Pnative
      - name: BP_JVM_VERSION
        value: 17 ## only JVM 17 and later versions support native images. Depending o
n your configuration, this might already be the default value.
```

Use native images with Maven

If you are using Maven, you must also add a native profile that includes `native-maven-plugin` for the build phase in `pom.xml`:

```
<profiles>
        <profile>
        <id>native</id>
        <build>
           <plugins>
              <plugin>
                 <groupId>org.graalvm.buildtools</groupId>
```

```
                    <artifactId>native-maven-plugin</artifactId>
                </plugin>
            </plugins>
         </build>
        </profile>
</profiles>
```

Supported Features

There are some differences on supported features when working with Native images:

- You can deploy workloads with native images by running the `Tanzu: Apply Workload` command.
- You can delete workloads with native images by running the `Tanzu: Delete Workload` command.
- Debug and Live Update are not supported when using native images. However you can add an additional `workload.yaml` file that doesn't use a native image to iterate on your development.

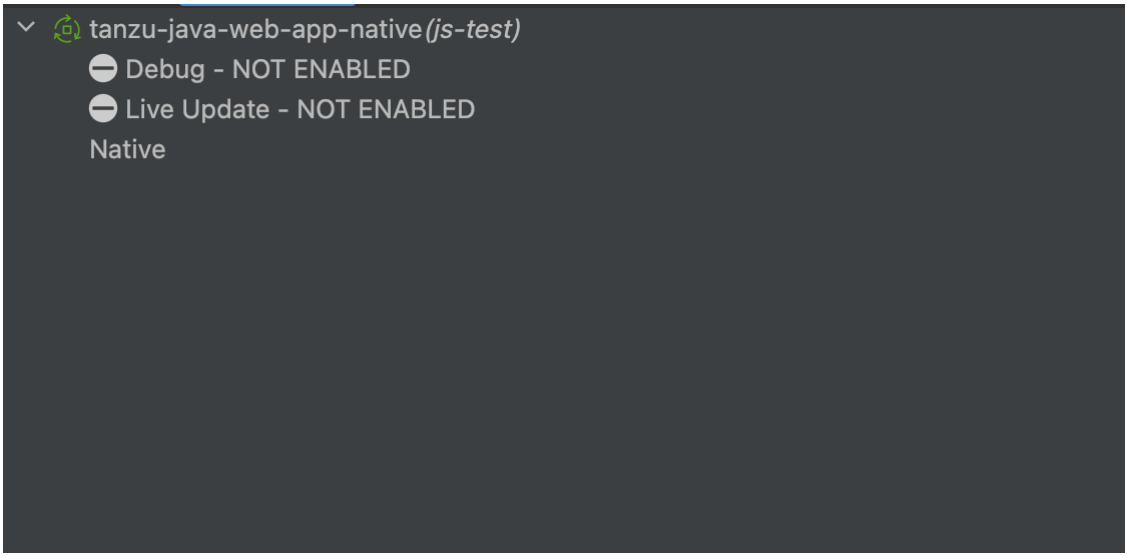
This example `workload.yaml` specification has a native image flag:

```
...
spec:
  build:
    env:
      - name: BP_NATIVE_IMAGE
        value: "true"
...
```

This example `workload.yaml` specification doesn't have a native image flag:

```
...
spec:
  build:
    env:
      #- name: BP_NATIVE_IMAGE
      #  value: "true"
...
```

The Tanzu Workloads panel adds the `Native` label to any workloads that contain native images.



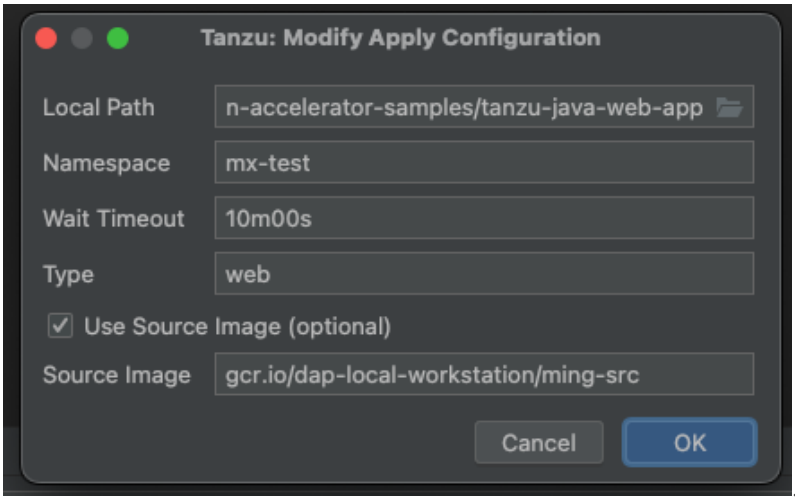
.

Update Workload Apply configurations

The `--wait-timeout` and the `--type` flags can be configured.

For more information, see the Tanzu CLI Command Reference documentation.

To use Telepresence, use `--type server`.



Use a `portforward` to access an application locally

You can create a `portforward` by clicking **Port Forward** in the pop-up menu in the Tanzu Workloads panel.

A `portforward` enables you to easily access the application, when iterating locally, in the Tanzu Workloads panel from a local URL (via the pop-up menu action) or a Knative URL (for the web type of workloads).

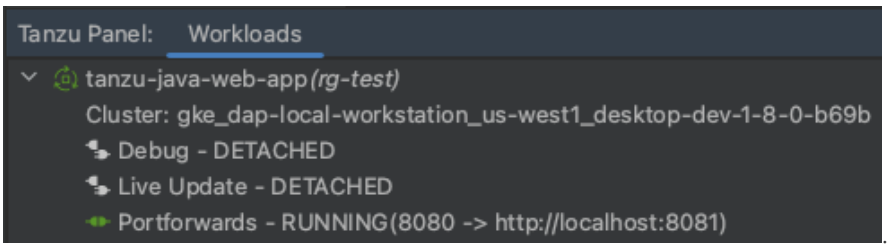
The option to use a `portforward` is only available if containers in your workload have either:

- A `PORT` environment variable
- An entry in the `ports` array that specifies `TCP` as the `protocol`

For example:

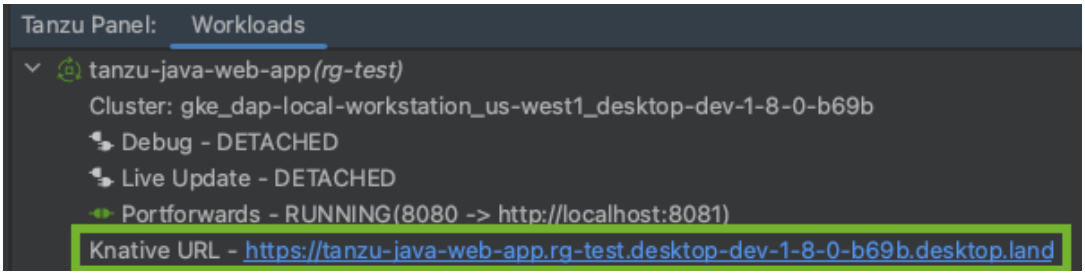
```
ports:
- containerPort: 8080
  name: user-port
  protocol: TCP
```

Existing `portforwards` are shown in the Tanzu Workloads panel. You can stop a `portforward` by clicking **Stop Port Forward** in the pop-up menu in the Tanzu Workloads panel. The option to stop a `portforward` is only available if there is an existing `portforward`.



View the Knative URL

If your workload deploys a Knative service, you can view the Knative URL through the Tanzu Workloads panel.



Glossary of terms

This topic gives you explanations of common terms used throughout the Tanzu Developer Tools for IntelliJ documentation, and within the extension itself. Some of these terms are unique to Tanzu Application Platform, while others might have a different meaning outside of Tanzu Application Platform and are included here for clarification.

Live Update

Live Update, facilitated by Tilt, enables you to deploy your workload once, save changes to the code, and see those changes reflected in the workload running on the cluster within seconds.

Tiltfile

The Tiltfile is a file with no extension that is required for Tilt to enable the Live Update feature. For more information about the Tiltfile, see the Tilt documentation.

Debugging on the cluster

The Tanzu Developer Tools on IntelliJ extension enables you to debug your application in an environment similar to production by debugging on your Tanzu Application Platform enabled Kubernetes cluster.

Note

An environment's similarity to production relies on keeping dependencies updated, among other variables.

YAML file format

YAML is a human-readable data-serialization language. It is commonly used for configuration files. For more information, see the YAML Wikipedia entry.

workload.yaml file

The workload YAML file is a required configuration file used by the Tanzu Application Platform to specify the details of an application including its name, type, and source code URL.

catalog-info.yaml file

The catalog-info YAML file enables the workloads created with the Tanzu Developer Tools for IntelliJ extension to be visible in Tanzu Developer Portal.

Code snippet

Code snippets enable you to quickly add project files that are necessary to develop using Tanzu Application Platform by creating a template in an empty file that you fill out with the required information.

Source image

The source image is the registry location to publish local source code, for example, `registry.io/yourapp-source`. This must include both a registry and a project name.

Local path

The local path value tells the Tanzu Developer Tools for IntelliJ extension which directory on your local file system to bring into the source image container image. The default local path value is the current directory where you saved the files for your open IntelliJ project.

Kubernetes context

A Kubernetes context is a set of access parameters that contains a Kubernetes cluster, a user, and a namespace. A Kubernetes context acts like a set of coordinates that describe the target of the Kubernetes commands that you run. For more information, see the Kubernetes documentation.

Kubernetes namespace

As defined by the Kubernetes documentation, in Kubernetes, namespaces provide a mechanism for isolating groups of resources within a single cluster. Names of resources need to be unique within a namespace, but not across namespaces.

Troubleshoot Tanzu Developer Tools for IntelliJ

This topic helps you troubleshoot issues with Tanzu Developer Tools for IntelliJ.

Unable to view workloads on the panel when connected to GKE cluster

Symptom

When connecting to Google's GKE clusters, an error appears with the text `WARNING: the gcp auth plugin is deprecated in v1.22+, unavailable in v1.25+; use gcloud instead.`

Cause

GKE authentication was extracted into a separate plug-in and is no longer inside the Kubernetes client or libraries.

Solution

Download and configure the GKE authentication plug-in. For instructions, see the Google documentation.

Deactivated launch controls after running a launch configuration

Symptom

When you run or debug a launch configuration, IntelliJ deactivates the launch controls.

Cause

IntelliJ deactivates the launch controls to prevent other launch configurations from being launched at the same time. These controls are reactivated when the launch configuration is started. As such, starting multiple Tanzu debug and live update sessions is a synchronous activity.

Starting a Tanzu Debug session fails with `Unable to open debugger port`

Symptom

You try to start a Tanzu Debug session and it immediately fails with an error message similar to:

```
Error running 'Tanzu Debug - fortune-teller-fortune-service': Unable to open debugger
port (localhost:5005): java.net.ConnectException "Connection refused"
```

Cause

Old Tanzu Debug launch configurations sometimes appear to be corrupted after installing a later version of the plug-in. You can see whether this is the problem you are experiencing by opening the launch configuration:

1. Right-click `workload.yaml`.
2. Click **Modify Run Configuration...** in the menu.
3. Scroll down and expand the **Before Launch** section of the dialog.
4. Verify that it contains the two Unknown Task entries `com.vmware.tanzu.tanzuBeforeRunPortForward` and `com.vmware.tanzu.tanzuBeforeRunWorkloadApply`.

Because these two tasks are unknown causes, these steps of the debug launch are not run. This in turn means that the target application is not deployed and accessible on the expected port, which causes an error when the debugger tries to connect to it.

It might be that although the launch configuration appears corrupt when seen in the launch config editor, in fact there is no corruption. It's suspected that this problem only occurs when you install a new version of the plug-in and start using it before first restarting IntelliJ.

There is possibly an issue in the IntelliJ platform that prevents completely or correctly initializing the plug-in when the plug-in is hot-swapped into an active session instead of loaded on startup.

Solution

Closing and restarting IntelliJ typically fixes this problem. If that doesn't work for you, delete the old corrupted launch configuration and recreate it.

Timeout error when Live Updating

Symptom

When you attempt to Live Update your workload, the following error message appears in the log:

```
ERROR: Build Failed: apply command timed out after 30s - see }}{{https://docs.tilt.dev/api.html#api.update_settings{{ for how to increase}}
```

Cause

Kubernetes times out on upserts over 30 seconds.

Solution

Add `update_settings (k8s_upsert_timeout_secs = 300)` to the Tiltfile. For more information, see the Tiltfile documentation.

Tanzu Panel empty when using a GKE cluster on macOS

Symptom

On macOS, the Tanzu Panel doesn't display workloads or any other resources when using a GKE cluster. Other tools, such as the Tanzu CLI Apps plug-in, display resources correctly.

Cause

`gke-cloud-auth-plugin` is required to properly authenticate to a GKE cluster. However, when starting IntelliJ from Dock or Spotlight, environment variables set by using `.profile`, `.bash_profile`, or `.zshrc` are not available. For more information, see this YouTrack issue.

This might cause `gke-cloud-auth-plugin` to be missing from `PATH` when launching IntelliJ and prevent the Tanzu Panel from reaching the cluster.

Solution

Open IntelliJ from the CLI. Example command:

```
open /Applications/IntelliJ\ IDEA.app
```

Tanzu Panel is empty when the context is set by using the `KUBECONFIG` environment variable

Symptom

On macOS, the Tanzu Panel doesn't display workloads or any other resources when setting the Kubernetes context by using a `KUBECONFIG` environment variable. Other tools, such as the Tanzu CLI Apps plug-in, display resources correctly.

Cause

When starting IntelliJ from Dock or Spotlight, environment variables set by using `.profile`, `.bash_profile`, or `.zshrc` are not available. This causes the panels to be empty because the extension can't find the right Kubernetes context. For more information, see this YouTrack issue.

Solution

Open IntelliJ from the CLI. For example:

```
open /Applications/IntelliJ\ IDEA.app
```

Tanzu panel shows workloads but doesn’t show Kubernetes resources

Symptom

The Tanzu panel shows workloads but doesn’t show Kubernetes resources in the center panel of the activity pane.

Cause

When switching the Kubernetes context, the activity pane doesn’t automatically update the namespace, but the workload pane detects the new namespace. Therefore, the Tanzu panel shows workloads but doesn’t show Kubernetes resources in the center panel of the activity pane.

Solution

Restart IntelliJ to properly detect the context change.

Tanzu Workloads panel workloads only have describe and delete action

Symptom

Some or all workloads in the Tanzu Workloads panel only have describe and delete actions.

Cause

By design, only associated workloads have apply, debug, and Live Update workload actions available.

Solution

Open a project that contains a module that can be associated with your deployed workloads.

Workload actions do not work when in a project with spaces in the name

Symptom

Workload actions and Live Update do not work. The console displays an error message similar to:

```
Error: unknown command "projects/my-app" for "apps workload apply". Process finished w
ith exit code 1
```

Cause

Improper handling of paths that contain spaces causes the shell to misinterpret some commands, such as `tanzu workload apply ...`. This causes anything executing these commands to fail when the name of a project, or any parts of its path on disk, contain spaces.

Solution

1. Close the code editor.
2. Move or rename your project folder on the disk, ensuring that no part of its path contains any spaces.

3. Delete the project settings folder from the project to start with a clean slate. The folder is `.idea` if using IntelliJ and `.vscode` if using VS Code.
4. Open the code editor and then open the project in its new location.

A lock wrongly prevents Live Update from starting again

Symptom

A lock incorrectly shows that Live Update is still running and prevents Live Update from starting again.

Cause

Restarting your computer while running Live Update without terminating the Tilt process beforehand.

Solution

Delete the Tilt lock file. The default location for the file is `~/.tilt-dev/config.lock`.

UI liveness check causes an EDT Thread Exception error

Symptom

An **EDT Thread Exception** error is logged or reported as a notification with a message similar to `"com.intellij.diagnostic.PluginException: 2007 ms to call on EDT TanzuApplyAction#update@ProjectViewPopup"`.

Cause

A UI liveness check detected something taking more time than planned on the UI thread and freezing the UI. You might experience a brief UI freeze at the same time. This happens more commonly when starting IntelliJ because some initialization processes are still running. This issue is also commonly reported by users with large projects.

If slow startup processes cause the issue, the UI freeze is likely to be brief. If a large number of files causes the issue, the UI freeze is likely to be more severe.

Solution

There is no workaround currently other than trying to reduce the number of files in your project, though that might not be practical in some cases. A fix is planned for the next release.

Frequent application restarts

Symptom

When an application is applied from IntelliJ it restarts frequently.

Cause

An application or environment behavior is triggering the application to restart.

Observed trigger behaviors include:

- The application itself writing logs to the file system in the application directory that Live Update is watching

- Autosave being set to a very high frequency in the IDE configuration

Solution

Prevent the trigger behavior. Example solutions include:

- Prevent 12-factor applications from writing to the file system.
- Reduce the autosave frequency to once every few minutes.

Overview of Tanzu Developer Tools for Visual Studio

VMware Tanzu Developer Tools for Visual Studio is the official VMware Tanzu IDE extension for Visual Studio 2022. The extension helps you develop with Tanzu Application Platform and enables you to rapidly iterate on your workloads on supported Kubernetes clusters that have Tanzu Application Platform installed.

Tanzu Developer Tools for Visual Studio currently supports .NET C# applications.

This extension is for Microsoft Visual Studio 2022 only. It is incompatible with Visual Studio Code and Visual Studio for Mac.

Note

This extension is in the beta stage of development.

Extension features

The extension has the following features:

- **Deploy applications directly from Visual Studio:**

  Rapidly iterate on your applications on Tanzu Application Platform and deploy them as workloads directly from within Visual Studio.

- **See code updates running on-cluster in seconds:**

  With the use of Live Update facilitated by Tilt, deploy your workload once, save changes to the code and then, seconds later, see those changes reflected in the workload running on the cluster.

- **Debug workloads directly on the cluster:**

  Debug your application in a production-like environment by debugging on your Kubernetes cluster that has Tanzu Application Platform. An environment's similarity to production relies on keeping dependencies updated, among other variables.

- **See workloads running on the cluster:**

  From the Tanzu Panel, you can see any workload found within the cluster and namespace specified in the current kubectl context.

- **Work with microservices in a Visual Studio solution:**

  Work with multiple solution projects that represent discrete microservices. This makes it possible to deploy, debug, and Live Update multiple workloads simultaneously from the same solution.

Note

The new variation of Out of the Box (OOTB) Basic supply chains, which outputs Carvel packages to enable configuring multiple runtime environments, is not currently supported. For more information about the variation, see Carvel Package Supply Chains.

Next steps

Install Tanzu Developer Tools for Visual Studio.

Overview of Tanzu Developer Tools for Visual Studio

VMware Tanzu Developer Tools for Visual Studio is the official VMware Tanzu IDE extension for Visual Studio 2022. The extension helps you develop with Tanzu Application Platform and enables you to rapidly iterate on your workloads on supported Kubernetes clusters that have Tanzu Application Platform installed.

Tanzu Developer Tools for Visual Studio currently supports .NET C# applications.

This extension is for Microsoft Visual Studio 2022 only. It is incompatible with Visual Studio Code and Visual Studio for Mac.

Note

This extension is in the beta stage of development.

Extension features

The extension has the following features:

- **Deploy applications directly from Visual Studio:**

  Rapidly iterate on your applications on Tanzu Application Platform and deploy them as workloads directly from within Visual Studio.

- **See code updates running on-cluster in seconds:**

  With the use of Live Update facilitated by Tilt, deploy your workload once, save changes to the code and then, seconds later, see those changes reflected in the workload running on the cluster.

- **Debug workloads directly on the cluster:**

  Debug your application in a production-like environment by debugging on your Kubernetes cluster that has Tanzu Application Platform. An environment's similarity to production relies on keeping dependencies updated, among other variables.

- **See workloads running on the cluster:**

  From the Tanzu Panel, you can see any workload found within the cluster and namespace specified in the current kubectl context.

- **Work with microservices in a Visual Studio solution:**

  Work with multiple solution projects that represent discrete microservices. This makes it possible to deploy, debug, and Live Update multiple workloads simultaneously from the same solution.

Note

The new variation of Out of the Box (OOTB) Basic supply chains, which outputs Carvel packages to enable configuring multiple runtime environments, is not currently supported. For more information about the variation, see Carvel Package Supply Chains.

Next steps

Install Tanzu Developer Tools for Visual Studio.

Install Tanzu Developer Tools for Visual Studio

This topic tells you how to install VMware Tanzu Developer Tools for Visual Studio.

Prerequisites

Before installing the extension, you must have:

- Visual Studio 2022 v17.7 or later
- kubectl
- Tilt v0.30.12 or later
- Tanzu CLI and plug-ins
- A cluster with the Tanzu Application Platform Full profile or Iterate profile

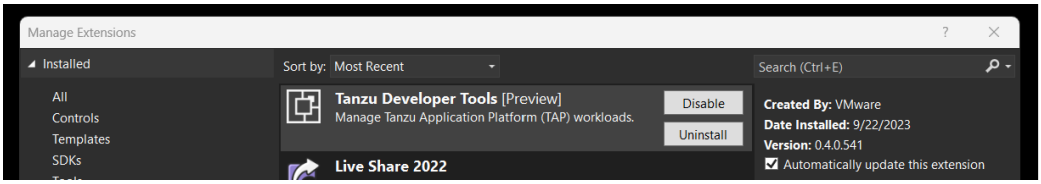
Note

If you are an app developer, someone else in your organization might have already set up the Tanzu Application Platform environment.

Install

To install Tanzu Developer Tools for Visual Studio:

1. Download VMware Tanzu Developer Tools for Visual Studio from VMware Tanzu Network.
2. Double-click the `.vsix` install file and click through the prompts.
3. Open Visual Studio and, from top menu, click **Extensions** > **Manage Extensions**.
4. Verify that the extension is installed and that it is the version you want.



Update

To update to a later version, repeat the steps in the Install section. You do not need to uninstall the current version.

Uninstall

To uninstall:

1. From the top menu, click the **Extensions** tab and then click **Manage Extensions**.
2. Select the **Installed** section and then click the **Uninstall** button for this extension.

Next steps

Getting Started with Tanzu Developer Tools for Visual Studio.

Get Started with Tanzu Developer Tools for Visual Studio

This topic guides you through getting started with VMware Tanzu Developer Tools for Visual Studio.

Prerequisite

Install Tanzu Developer Tools for Visual Studio.

Configure Local Source Proxy

Configure Local Source Proxy if you're able to do so. For more information, see the Local Source Proxy documentation.

If you cannot use Local Source Proxy, use a source image registry. Before deploying a workload, you must authenticate with an image registry to store your source code. You can use the Docker CLI to authenticate or you can set environment variables that the Tanzu CLI can use to authenticate.

Docker CLI
To authenticate by using the Docker CLI, run:

```
docker login $REGISTRY_HOSTNAME -u $REGISTRY_USERNAME -p $REGISTRY_PASSWORD
```

Tanzu CLI
To authenticate by using the Tanzu CLI, export environment variables by running:

```
export TANZU_APPS_REGISTRY_CA_CERT=PATH-TO-CA-CERT.nip.io.crt
export TANZU_APPS_REGISTRY_PASSWORD=USERNAME
export TANZU_APPS_REGISTRY_USERNAME=PASSWORD
```

`CA_CERT` is only needed for a custom or private registry.

For more information, see Workload creation fails due to authentication failure in Docker Registry.

Set up Tanzu Developer Tools

The extension makes use of the following files within your project:

- `workload.yaml`
- `catalog-info.yaml`
- `Tiltfile`
- `.tanzuignore`

You can create these files by using the instructions in this topic, or use the files in the View an example project section.

There are two ways to create these files:

- Using the code snippets that Tanzu Developer Tools provide, which create templates in empty files that you then fill in with the required information.
- Writing the files manually.

Create the `workload.yaml` file

Your project must contain a file named `workload.yaml`. For example, `MyApp\Config\workload.yaml`.

`workload.yaml` provides instructions to Supply Chain Choreographer for how to build and manage a workload. For more information, see Supply Chain Choreographer for Tanzu.

The Tanzu Developer Tools for Visual Studio extension requires at least one `workload.yaml` file per project. `workload.yaml` must be a single-document YAML file, not a multi-document YAML file.

To create a `workload.yaml` file by using Visual Studio:

1. Right-click the Solution Explorer project.
2. Click **Add** > **New Folder**.
3. Name the folder `Config`.
4. Right-click the new `Config` folder and then click **Add** > **New Item...**.
5. From the available list of items, click **Tanzu Workload** > **Add**.
6. Follow the instructions at the top of the created file.

See the following `workload.yaml` example:

```
apiVersion: carto.run/v1alpa1
kind: Workload
metadata:
 name: APP-NAME
 labels:
   apps.tanzu.vmware.com/workload-type: WORKLOAD-TYPE
   app.kubernetes.io/part-of: APP-NAME
spec:
 source:
   git:
     url: GIT-SOURCE-URL
     ref:
       branch: GIT-BRANCH-NAME
```

Where:

- `APP-NAME` is the name of your application. For example, `my-app`.
- `WORKLOAD-TYPE` is the type of workload for your app. For example, `web`. For more information, see Workload types.
- `GIT-SOURCE-URL` is the Git source code URL for your app. For example, `github.com/mycompany/myapp`.
- `GIT-BRANCH-NAME` is the branch of the Git source code you want to use. For example, `main`.

Alternatively, you can use the Tanzu CLI to create a `workload.yaml` file. For more information, see Create or update a workload.

Create the `catalog-info.yaml` file

Your project must contain a file named `catalog-info.yaml`. For example, `MyApp\Catalog\catalog-info.yaml`.

`catalog-info.yaml` enables the workloads created with Tanzu Developer Tools for Visual Studio to appear in Tanzu Developer Portal. For more information, see Overview of Tanzu Developer Portal.

To create a `catalog-info.yaml` file by using Visual Studio:

1. Right-click the Solution Explorer project.
2. Click **Add** > **New Folder**.
3. Name the folder `Catalog`.
4. Right-click the new `Catalog` folder and then click **Add** > **New Item...**.
5. From the available list of items, click **Tanzu Catalog Info** > **Add**.
6. Follow the instructions at the top of the created file.

See the following `catalog-info.yaml` example:

```
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
 name: APP-NAME
 description: APP-DESCRIPTION
 tags:
   - tanzu
 annotations:
   'backstage.io/kubernetes-label-selector': 'app.kubernetes.io/part-of=APP-NAME'
spec:
 type: service
 lifecycle: experimental
 owner: default-team
```

Where:

- `APP-NAME` is the name of your application.
- `APP-DESCRIPTION` is a description of your application.

Create the Tiltfile file

Your project must contain a file named `Tiltfile`. For example, `MyApp\Tiltfile`.

The `Tiltfile` provides the configuration for Tilt to enable your project to Live Update on the Tanzu Application Platform-enabled Kubernetes cluster. For more information, see the Tilt documentation.

To create a `Tiltfile` file by using Visual Studio:

1. Right-click the Solution Explorer project.
2. Click **Add** > **New Item...**
3. From the available list of items, click **Tanzu Tiltfile** > **Add**.
4. Follow the instructions at the top of the created file.

See the following `Tiltfile` example:

```
SOURCE_IMAGE = os.getenv("SOURCE_IMAGE", default='SOURCE-IMAGE-VALUE')
LOCAL_PATH = os.getenv("LOCAL_PATH", default='.')
NAMESPACE = os.getenv("NAMESPACE", default='default')
LIVE_UPDATE_PATH = os.getenv("LIVE_UPDATE_PATH", default='bin/Debug/net6.0')

k8s_custom_deploy(
```

```
   'APP-NAME',
   apply_cmd="tanzu apps workload apply -f Config/workload.yaml --live-update" +
       " --local-path " + LOCAL_PATH +
       " --build-env BP_DEBUG_ENABLED=true" +
       " --namespace " + NAMESPACE +
       " --output yaml" +
       " --yes",
   delete_cmd="tanzu apps workload delete " +  APP-NAME + " --namespace " + NAMESPACE
+ " --yes" ,
   deps=['bin'],
   container_selector='workload',
   live_update=[
       sync(LIVE_UPDATE_PATH, '/workspace')
   ]
)

k8s_resource('APP-NAME', port_forwards=["8080:8080"],
   extra_pod_selectors=[{'carto.run/workload-name': 'APP-NAME', 'app.kubernetes.io/com
ponent': 'run'}])
allow_k8s_contexts('CONTEXT-NAME')
```

Where `APP-NAME` is the name of your application

If your Tanzu Application Platform-enabled Kubernetes cluster is running on your local machine, you can remove the entire `allow_k8s_contexts` line. For more information about this line, see the Tilt documentation.

Create the `.tanzuignore` file

Your project can contain a file named `.tanzuignore`. When working with local source code, `.tanzuignore` excludes files from the source code that is uploaded within the image. It has syntax similar to the `.gitignore` file.

This file must be placed in the project root to work. For example, `MyApp\.tanzuignore`.

To create a `Tiltfile` file by using Visual Studio:

1. Right-click the Solution Explorer project.
2. Click **Add** > **New Item...**.
3. From the available list of items, click **Tanzu Ignore file** > **Add**.

For an example, see the `.tanzuignore` file in GitHub that is used for the sample Tanzu Java web app. You can use the file as it is or edit it for your needs.

View an example project

Before you begin, you need a container image registry to use the sample application. There are two ways to view a sample application that demonstrates the necessary configuration files:

Use Application Accelerator
If your company has configured Application Accelerator, you can obtain the sample application there if it was not removed.

To view the example by using Application Accelerator:

1. Open Application Accelerator. You might need to contact a separate team in your organization to learn they placed it.
2. Search for `Steeltoe Weather Forecast` in Application Accelerator.
3. Add the required configuration information and generate the application.

4. Unzip the application and open the directory in Visual Studio.

Clone from GitHub

To clone the example from GitHub:

1. Use `git clone` to clone the application-accelerator-samples repository from GitHub.
2. Go to the `weatherforecast-steeltoe` directory.
3. Open the `Tiltfile` and replace `your-registry.io/project` with your registry.

Next steps

Use Tanzu Developer Tools for Visual Studio.

Use Tanzu Developer Tools for Visual Studio

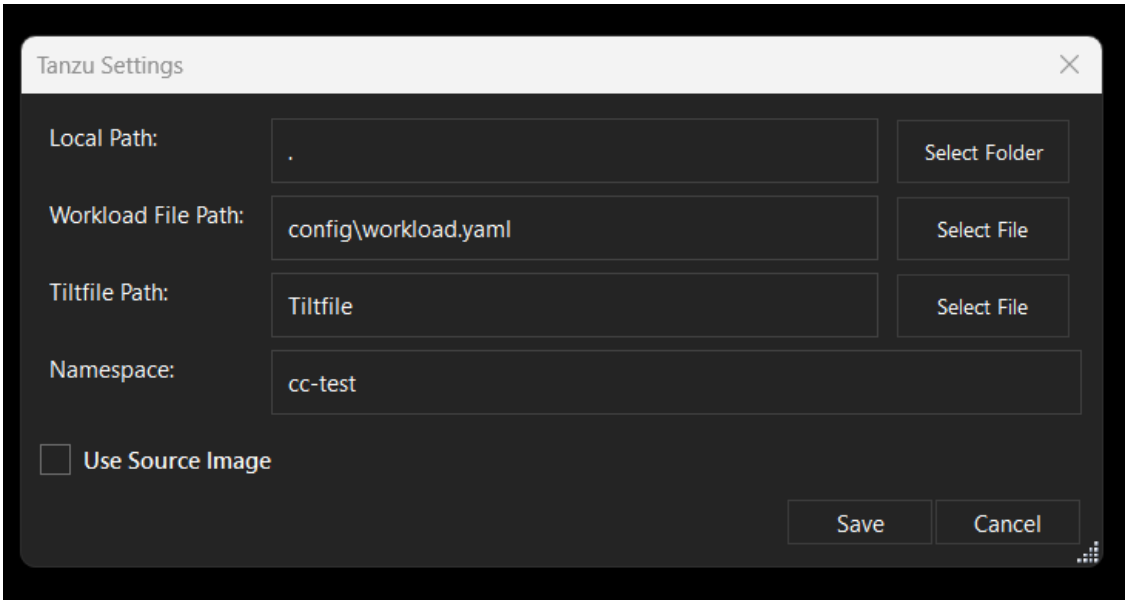
This topic tells you how to use VMware Tanzu Developer Tools for Visual Studio.

Note

This extension is in the beta stage of development.

Configure settings

To configure settings, right-click anywhere in the Solution Explorer and click **Tanzu** > **Settings...**.



Workload Actions

The extension enables you to apply, debug, and Live Update your application on a Kubernetes cluster that has Tanzu Application Platform. The developer sandbox experience enables you to Live Update your code, and simultaneously debug the updated code, without deactivating Live Update.

Apply a workload

To apply a workload, right-click anywhere in the Solution Explorer and click **Tanzu** > **Apply Workload**. Alternatively, right-click an associated workload in the Tanzu Panel and click **Apply**

Workload.

Delete a workload

To delete a workload, right-click anywhere in the Solution Explorer and click **Tanzu** > **Delete Workload**. Alternatively, right-click an associated workload in the Tanzu Panel and click **Delete Workload**.

Start debugging on the cluster

To remote debug a workload, right-click anywhere in the Solution Explorer and click **Tanzu** > **Debug Workload**. Alternatively, right-click an associated workload in the Tanzu Panel and click **Debug Workload**.

Caution

Do not use the red square Stop button to end your debugging session. Using the red square Stop button might cause the Tanzu Application Platform workload to fail. Instead, in the top menu click **Debug** > **Detach All**.

If the name of your running app process (the app DLL process), does not match the name of your .NET project as shown in the Visual Studio Solution Explorer, the remote debugging agent might fail to attach.

Live Update

See the following sections for how to use Live Update.

Start Live Update

Ensure that the following Tanzu Settings parameters are set:

- **Local Path**, which is the path on the local file system to a directory of source code to build.
- **Namespace**, which is the namespace that workloads are deployed into. Optional.
- **Source Image**, which is the registry location for publishing local source code. For example, `registry.io/yourapp-source`. It must include both a registry and a project name. The source image parameter is not needed if you configured Local Source Proxy.

To start Live Update, right-click anywhere in the Solution Explorer and click **Tanzu** > **Start Live Update**. Alternatively, right-click an associated workload in the Tanzu Panel and click **Start Live Update**.

After starting Live Update, local builds changes are synchronized with the container.

Stop Live Update

To stop Live Update, right-click anywhere in the Solution Explorer and click **Tanzu** > **Stop Live Update**. Alternatively, right-click an associated workload in the Tanzu Panel and click **Stop Live Update**.

Tanzu Workloads panel

The current state of the workloads is visible in the Tanzu Workloads view. This view is a separate section in the bottom of the Explorer view in the Side Bar. The view shows the current status of

each workload, namespace, and cluster. It also shows whether Live Update and Debug is running, stopped, or deactivated.

The Tanzu Activity tab in the Panels view enables developers to visualize the supply chain, delivery, and running application pods. The tab enables a developer to view and describe logs on each resource associated with a workload from within their IDE. The tab displays detailed error messages for each resource in an error state.

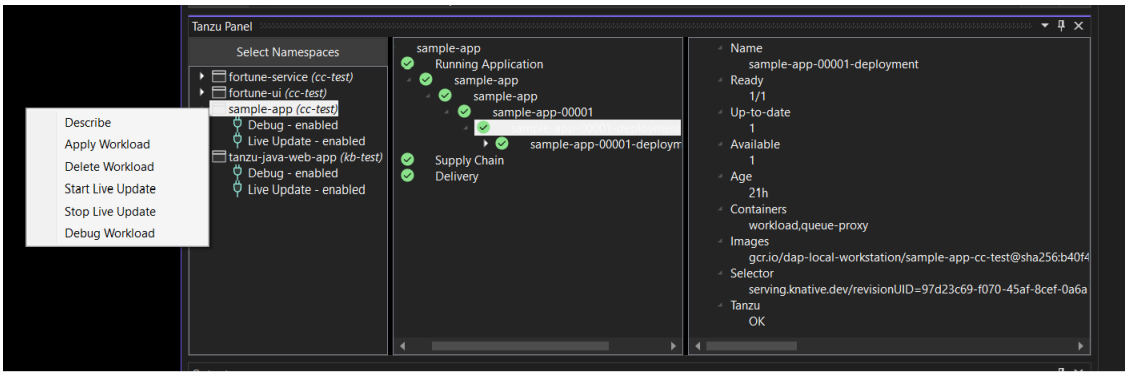
Workload commands are available from the Tanzu Workloads panel on workloads that have an associated module in the current project.

This association is based on a module name and a workload name matching. For example, a project with a module named `my-app` is associated with a deployed workload named `my-app`.

When taking an action from the Tanzu Workloads panel, the action uses the namespace of the deployed workload regardless of the configuration in the module.

For example, you might have a Live Update configuration with a namespace argument of `my-apps-1`, but running the action from a deployed workload in namespace `my-apps-2` starts a Live Update session with a namespace argument of `my-apps-2`.

The Tanzu Workloads panel uses the cluster and defaults to the namespace specified in the current kubectl context. To view the Tanzu Workloads panel, right-click anywhere in the Solution Explorer and click **Tanzu** > **View Workloads**.



Extension logs

The extension creates log entries in two files named `tanzu-dev-tools-{GUID}.log` and `tanzu-language-server-{GUID}.log`. These files are in the directory where Visual Studio Installer installed the extension.

To find the log files from PowerShell, run:

```
dir $Env:LOCALAPPDATA\Microsoft\VisualStudio\*\Extensions\*\Logs\tanzu-*.log
```

To find the log files from CMD, run:

```
dir %LOCALAPPDATA%\Microsoft\VisualStudio\*\Extensions\*\Logs\tanzu-*.log
```

You can override the log file paths by setting the environment variables `TANZU_DT_LOG` and `TANZU_LS_LOG`.

Troubleshoot Tanzu Developer Tools for Visual Studio

This topic tells you how to troubleshoot issues you encounter with VMware Tanzu Developer Tools for Visual Studio.

Stop button causes workload to fail

Symptom

Clicking the red square Stop button in the Visual Studio top toolbar causes the Tanzu Application Platform workload to fail or become unresponsive indefinitely.

Solution

To end a debugging session, in the top menu click **Debug** > **Detach All**.

Frequent application restarts

Symptom

When an application is applied from Visual Studio it restarts frequently.

Cause

An application or environment behavior is triggering the application to restart.

Observed trigger behaviors include:

- The application itself writing logs to the file system in the application directory that Live Update is watching
- Autosave being set to a very high frequency in the IDE configuration

Solution

Prevent the trigger behavior. Example solutions include:

- Prevent 12-factor applications from writing to the file system.
- Reduce the autosave frequency to once every few minutes.

Overview of Tanzu Developer Tools for VS Code

VMware Tanzu Developer Tools for Visual Studio Code (VS Code) is the official VMware Tanzu IDE extension for VS Code. The extension helps you develop with Tanzu Application Platform and enables you to rapidly iterate on your workloads on supported Kubernetes clusters that have Tanzu Application Platform installed.

Tanzu Developer Tools for VS Code currently supports VS Code on macOS and Windows OS for Java applications.

Extension features

The extension has the following features:

- **Deploy applications directly from VS Code:**

  Rapidly iterate on your applications on Tanzu Application Platform by deploying them as workloads directly from within VS Code.

- **See code updates running on-cluster in seconds:**

  With Live Update (facilitated by Tilt), you can deploy your workload once, save changes to the code and then see those changes reflected within seconds in the workload running on the cluster.

- **Debug workloads directly on the cluster:**

  Debug your application in a production-like environment by debugging on your Kubernetes cluster that has Tanzu Application Platform. An environment's similarity to production relies on keeping dependencies and other variables updated.

- **See workloads running on the cluster:**

  From the Tanzu Workloads panel you can see any workload found within the cluster and namespace specified in the current kubectl context.

- **Make a ready-to-code development environment by using development containers (alpha):**

  Use developer containers to make a ready-to-code development environment. This enables developers to connect to a pre-configured development container that includes all Tanzu tools and IDE extensions required for development on Tanzu pre-installed.

- **Connect to a sandbox cluster:**

  You can connect to a Tanzu Application Platform sandbox instance from Tanzu Developer Tools for VS Code to try Tanzu Application Platform and see its benefits. To use Developer Sandbox and connect to it from your IDE, see the Tanzu Academy website.

Note

The new variation of Out of the Box (OOTB) Basic supply chains, which outputs Carvel packages to enable configuring multiple runtime environments, is not currently supported. For more information about the variation, see Carvel Package Supply Chains.

Overview of Tanzu Developer Tools for VS Code

VMware Tanzu Developer Tools for Visual Studio Code (VS Code) is the official VMware Tanzu IDE extension for VS Code. The extension helps you develop with Tanzu Application Platform and enables you to rapidly iterate on your workloads on supported Kubernetes clusters that have Tanzu Application Platform installed.

Tanzu Developer Tools for VS Code currently supports VS Code on macOS and Windows OS for Java applications.

Extension features

The extension has the following features:

- **Deploy applications directly from VS Code:**

  Rapidly iterate on your applications on Tanzu Application Platform by deploying them as workloads directly from within VS Code.

- **See code updates running on-cluster in seconds:**

  With Live Update (facilitated by Tilt), you can deploy your workload once, save changes to the code and then see those changes reflected within seconds in the workload running on the cluster.

- **Debug workloads directly on the cluster:**

  Debug your application in a production-like environment by debugging on your Kubernetes cluster that has Tanzu Application Platform. An environment's similarity to production relies

on keeping dependencies and other variables updated.

- **See workloads running on the cluster:**

  From the Tanzu Workloads panel you can see any workload found within the cluster and namespace specified in the current kubectl context.

- **Make a ready-to-code development environment by using development containers (alpha):**

  Use developer containers to make a ready-to-code development environment. This enables developers to connect to a pre-configured development container that includes all Tanzu tools and IDE extensions required for development on Tanzu pre-installed.

- **Connect to a sandbox cluster:**

  You can connect to a Tanzu Application Platform sandbox instance from Tanzu Developer Tools for VS Code to try Tanzu Application Platform and see its benefits. To use Developer Sandbox and connect to it from your IDE, see the Tanzu Academy website.

Note

The new variation of Out of the Box (OOTB) Basic supply chains, which outputs Carvel packages to enable configuring multiple runtime environments, is not currently supported. For more information about the variation, see Carvel Package Supply Chains.

Install Tanzu Developer Tools for VS Code

This topic tells you how to install VMware Tanzu Developer Tools for Visual Studio Code (VS Code).

Prerequisites

Before installing the extension, you must have:

- VS Code
- kubectl
- Tilt v0.30.12 or later
- Tanzu CLI and plug-ins
- A cluster with the Tanzu Application Platform Full profile or Iterate profile

If you are an app developer, someone else in your organization might have already set up the Tanzu Application Platform environment.

Docker Desktop and local Kubernetes are not prerequisites for using Tanzu Developer Tools for VS Code.

Install

To install VMware Tanzu Developer Tools for VS Code:

1. Open Visual Studio Code.
2. Open the command palette.
3. In the search box enter `Extension`.
4. Click **Extensions: Install Extensions**.

5. The **Extensions** view opens on the left side of your screen. In the search box enter `Tanzu`.
6. Click **Tanzu Developer Tools** and then click **Install**.

Configure

To configure VMware Tanzu Developer Tools for VS Code:

1. Ensure that you are targeting the correct cluster. For more information, see the Kubernetes documentation.
2. Go to **Code** > **Preferences** > **Settings** > **Extensions** > **Tanzu Developer Tools** and set the following:
    - **Confirm Apply Config**: This controls whether the extension asks to confirm user input when applying a workload.
    - **Confirm Debug Port**: This controls whether the extension asks for the debug port when running `Tanzu: Start Java Debug`.
    - **Confirm Local Port**: This controls whether the extension asks for the local port when running `Tanzu: Start Java Debug`.
    - **Confirm Delete**: This controls whether the extension asks for confirmation when deleting a workload.
    - **Enable Live Hover**: This enables Live Hover. For more information, see Integrating Live Hover by using Spring Boot Tools. Restart VS Code for this change to take effect.
    - **Source Image**: The registry location for publishing local source code. For example, `registry.io/yourapp-source`. This must include both a registry and a project name. A source image registry location is optional when Local Source Proxy is configured.
    - **Local Path**: (Optional) The path on the local file system to a directory of source code to build. This is the current directory by default.
    - **Namespace**: (Optional) This is the namespace that workloads are deployed into. The namespace set in kubeconfig is the default.
    - **Tracked Namespaces**: (Optional) Comma-separated list of namespaces. Resources in these namespaces appear in the Tanzu Workloads panel and the Activity panel. If empty, the namespace in the current context is the default.
    - **Wait Timeout**: (Optional) This sets how long to wait for a workload to become ready.
    - **Workload Type**: (Optional) This distinguishes the workload type. Examples include web and server.

Uninstall

To uninstall VMware Tanzu Developer Tools for VS Code:

1. Go to **Code** > **Preferences** > **Settings** > **Extensions**.
2. Right-click the extension and then click **Uninstall**.

Next steps

Proceed to Getting started with Tanzu Developer Tools for Visual Studio Code.

Get started with Tanzu Developer Tools for VS Code

This topic guides you through getting started with VMware Tanzu Developer Tools for Visual Studio Code (VS Code).

Prerequisite

Install VMware Tanzu Developer Tools for Visual Studio Code.

Configure Local Source Proxy or use a source image registry

Configure Local Source Proxy if you're able to do so. For more information, see the Local Source Proxy documentation.

If you cannot use Local Source Proxy, use a source image registry. Before deploying a workload, you must authenticate with an image registry to store your source code. You can use the Docker CLI to authenticate or you can set environment variables that the Tanzu CLI can use to authenticate.

Docker CLI
To authenticate by using the Docker CLI, run:

```
docker login $REGISTRY_HOSTNAME -u $REGISTRY_USERNAME -p $REGISTRY_PASSWORD
```

Tanzu CLI
To authenticate by using the Tanzu CLI, export environment variables by running:

```
export TANZU_APPS_REGISTRY_CA_CERT=PATH-TO-CA-CERT.nip.io.crt
export TANZU_APPS_REGISTRY_PASSWORD=USERNAME
export TANZU_APPS_REGISTRY_USERNAME=PASSWORD
```

`CA_CERT` is only needed for a custom or private registry.

For more information, see Workload creation fails due to authentication failure in Docker Registry.

Set up Tanzu Developer Tools

The extension makes use of the following files within your project:

- `workload.yaml`
- `catalog-info.yaml`
- `Tiltfile`
- `.tanzuignore`

You can create these files by using the instructions in this topic, or use the files in the View an example project section.

There are two ways to create these files:

- Using the code snippets that Tanzu Developer Tools provide, which create templates in empty files that you then fill in with the required information.
- Writing the files manually.

Create the `workload.yaml` file

`workload.yaml` provides instructions to the Supply Chain Choreographer about how to build and manage a workload.

The extension requires only one `workload.yaml` file per project. `workload.yaml` must be a single-document YAML file, not a multidocument YAML file.

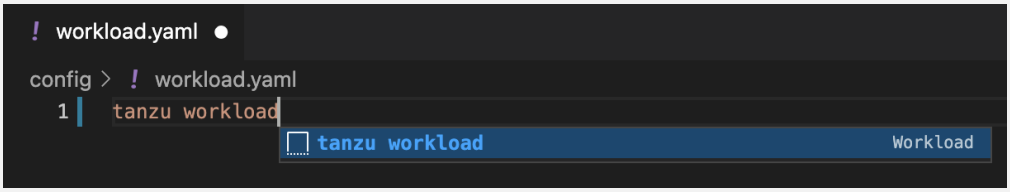
Before beginning to write your `workload.yaml` file, ensure that you know:

- The name of your application. For example, `my app`.
- The workload type of your application. For example, `web`.
- The GitHub source code URL. For example, `github.com/mycompany/myapp`.
- The Git branch of the source code that you intend to use. For example, `main`.

Code snippets

To create a `workload.yaml` file by using code snippets:

1. (Optional) Create a directory named `config` in the root directory of your project. For example, `my project/config`.
2. Create a file named `workload.yaml` in the new config directory. For example, `my project/config/workload.yaml`.
3. Open the new `workload.yaml` file in VS Code, enter `tanzu workload` in the file to trigger the code snippets, and either press Enter or left-click the `tanzu workload` text in the drop-down menu.



4. Fill in the template by pressing the Tab key.

Manual

To create your `workload.yaml` file manually, follow this example:

```
apiVersion: carto.run/v1alpa1
kind: Workload
metadata:
 name: APP-NAME
 labels:
   apps.tanzu.vmware.com/workload-type: WORKLOAD-TYPE
   app.kubernetes.io/part-of: APP-NAME
spec:
 source:
   git:
     url: GIT-SOURCE-URL
     ref:
       branch: GIT-BRANCH-NAME
```

Where:

- `APP-NAME` is the name of your application.
- `WORKLOAD-TYPE` is the type of this workload. For example, `web`.
- `GIT-SOURCE-URL` is your GitHub source code URL.
- `GIT-BRANCH-NAME` is the Git branch of your source code.

Alternatively, you can use the Tanzu CLI to create a `workload.yaml` file. For more information about the Tanzu CLI command, see Create or update a workload in the Tanzu CLI documentation.

Create the `catalog-info.yaml` file

`catalog-info.yaml` enables the workloads of this project to appear in Tanzu Developer Portal.

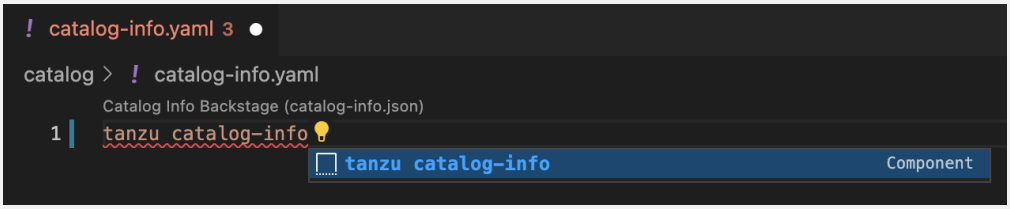
Before beginning to write your `catalog-info.yaml` file, ensure that you:

- Know the name of your application. For example, `my app`.
- Have a description of your application ready.

Code snippets

To create a `catalog-info.yaml` file by using the code snippets:

1. (Optional) Create a directory named `catalog` in the root directory of your project. For example, `my project/catalog`.
2. Create a file named `catalog-info.yaml` in the new config directory. For example, `my project/catalog/catalog-info.yaml`.
3. Open the new `catalog-info.yaml` file in VS Code, enter `tanzu catalog-info` in the file to trigger the code snippets, and then either press Enter or left-click the `tanzu catalog-info` text in the drop-down menu.



4. Fill in the template by pressing the Tab key.

Manual

To create your `catalog-info.yaml` file manually, follow this example:

```
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
 name: APP-NAME
 description: APP-DESCRIPTION
 tags:
   - tanzu
 annotations:
   'backstage.io/kubernetes-label-selector': 'app.kubernetes.io/part-of=APP-NAME'
spec:
 type: service
 lifecycle: experimental
 owner: default-team
```

Where:

- `APP-NAME` is the name of your application
- `APP-DESCRIPTION` is the description of your application

Create the Tiltfile file

The Tiltfile file provides the Tilt configuration to enable your project to Live Update on your Kubernetes cluster that has Tanzu Application Platform. The Tanzu Developer Tools extension requires only one **Tiltfile** per project.

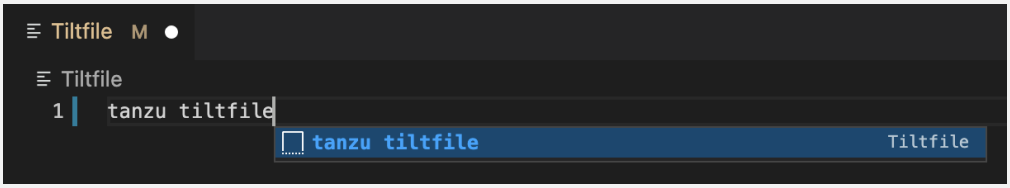
Before beginning to write your Tiltfile file, ensure that you know:

- The name of your application. For example, `my app`.
- Whether you want to compile the source image from a local directory other than the project directory or otherwise leave the `local path` value unchanged. For more information, see local path in the glossary.
- The path to your `workload.yaml` file. For example, `config/workload.yaml`.
- The name of your current Kubernetes context, if the targeting Kubernetes cluster enabled by Tanzu Application Platform is not running on your local machine.

Code Snippets

To create a Tiltfile file by using the code snippets:

1. Create a file named `Tiltfile` with no file extension in the root directory of your project. For example, `my project/Tiltfile`.
2. Open the new Tiltfile file in VS Code and enter `tanzu tiltfile` in the file to trigger the code snippets, and then either press Enter or left-click the `tanzu tiltfile` text in the drop-down menu.



3. Fill in the template by pressing the Tab key.
4. If the targeting Kubernetes cluster enabled by Tanzu Application Platform is not running on your local machine, add a new line to the end of the **Tiltfile** template and enter:

```
allow_k8s_contexts('CONTEXT-NAME')
```

Where `CONTEXT-NAME` is the name of your current Kubernetes context.

Manual

To create a Tiltfile file manually, follow this example:

```
LOCAL_PATH = os.getenv("LOCAL_PATH", default='.')
NAMESPACE = os.getenv("NAMESPACE", default='default')

k8s_custom_deploy(
   'APP-NAME',
   apply_cmd="tanzu apps workload apply -f PATH-TO-WORKLOAD-YAML --live-update" +
       " --local-path " + LOCAL_PATH +
       " --namespace " + NAMESPACE +
       " --yes >/dev/null" +
       " && kubectl get workload APP-NAME --namespace " + NAMESPACE + " -o yaml",
   delete_cmd="tanzu apps workload delete -f PATH-TO-WORKLOAD-YAML --namespace " + N
AMESPACE + " --yes" ,
   deps=['pom.xml', './target/classes'],
   container_selector='workload',
   live_update=[
       sync('./target/classes', '/workspace/BOOT-INF/classes')
   ]
)
```

```
k8s_resource('APP-NAME', port_forwards=["8080:8080"],
   extra_pod_selectors=[{'carto.run/workload-name': 'APP-NAME', 'app.kubernetes.io/component': 'run'}])
allow_k8s_contexts('CONTEXT-NAME')
```

Where:

- `APP-NAME` is the name of your application.
- `PATH-TO-WORKLOAD-YAML` is the local file system path to `workload.yaml`. For example, `config/workload.yaml`.
- `CONTEXT-NAME` is the name of your current Kubernetes context. If your Kubernetes cluster enabled by Tanzu Application Platform is running locally on your local machine, you can remove the entire `allow_k8s_contexts` line. For more information, see the Tilt documentation.

Create a `.tanzuignore` file

The `.tanzuignore` file specifies the file paths to exclude from the source code image. When working with local source code, you can exclude files from the source code to be uploaded within the image. Directories must not end with the system path separator (`/` or `\`). See this example. in GitHub.

View an example project

Before you begin, you need a container registry for the sample application.

You can view a sample application that demonstrates the necessary configuration files. There are two ways to obtain the sample application:

Application Accelerator
If your company has configured Application Accelerator, you can obtain the sample application there if it was not removed. To do so:

1. Open Application Accelerator.
2. Search for `Tanzu Java Web App` in Application Accelerator.
3. Add the required configuration information and generate the application.
4. Unzip the file and open the project in a VS Code workspace.

Clone from GitHub
To clone the sample application from GitHub:

1. Run `git clone` to clone the tanzu-java-web-app repository from GitHub.
2. Change into the `tanzu-java-web-app` directory.
3. Open the Tiltfile and replace `your-registry.io/project` with your container registry.

Next steps

Use Tanzu Developer Tools for VS Code.

Use Tanzu Developer Tools for VS Code

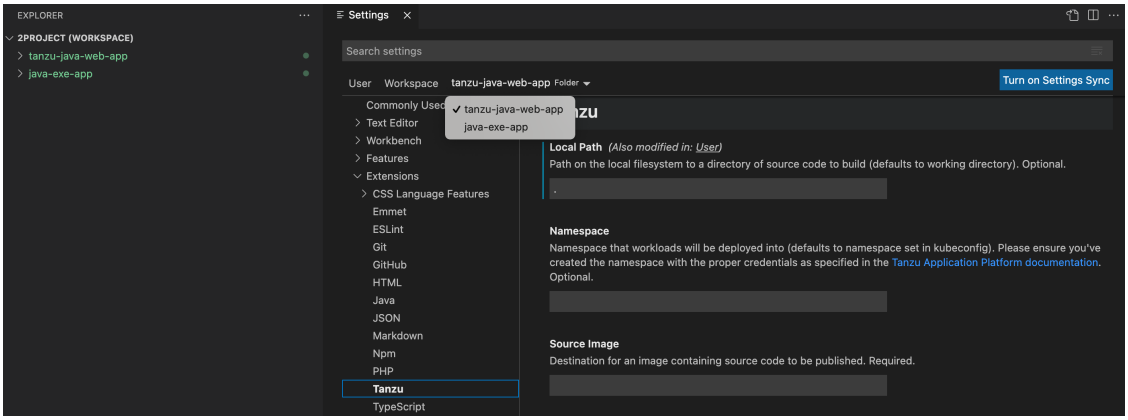
This topic tells you how to use VMware Tanzu Developer Tools for Visual Studio Code (VS Code).

Ensure that the project you want to use the extension with has the required files specified in Get started with Tanzu Developer Tools for VS Code.

The extension requires only one Tiltfile and one `workload.yaml` per project. The `workload.yaml` must be a single-document YAML file, not a multidocument YAML file.

Configure for multiple projects in the workspace

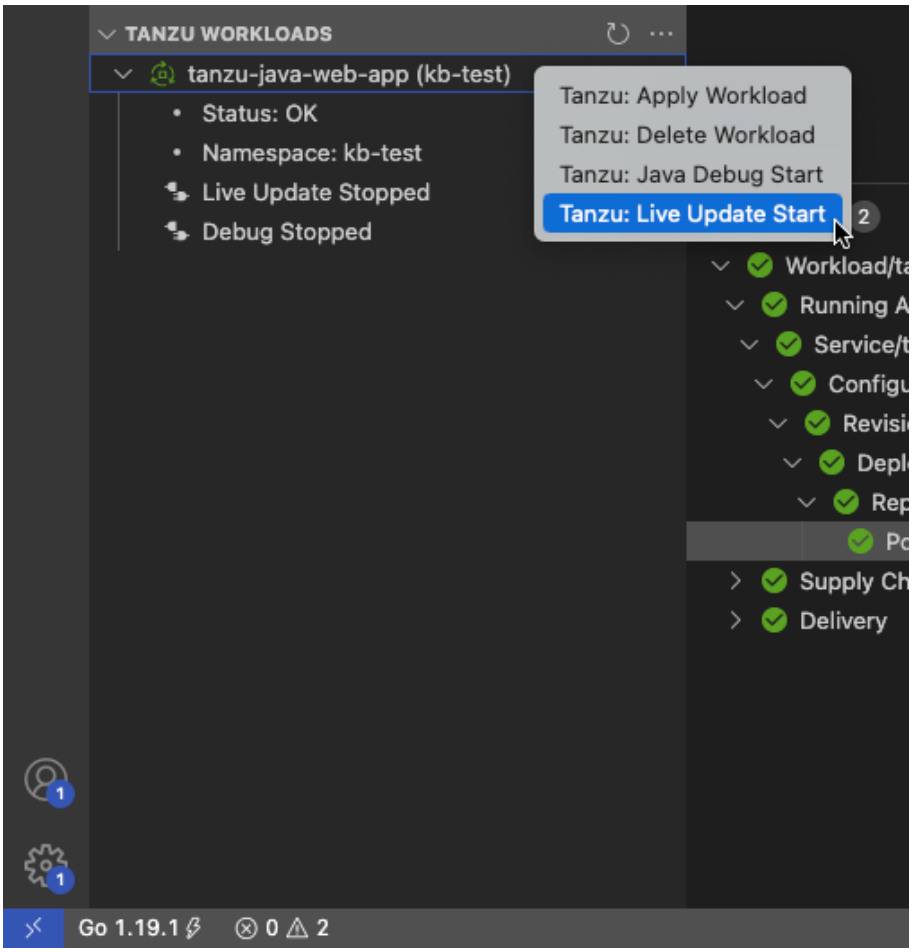
When working with multiple projects in a single workspace, you can configure the extension settings on a per-project basis by using the drop-down menu in **Settings**.



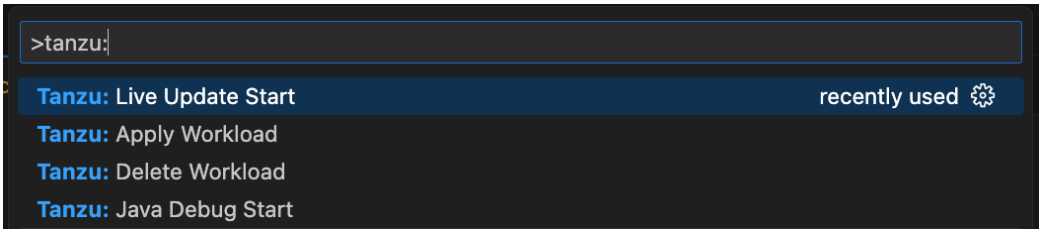
Workload Commands

All commands are available by right-clicking anywhere in the VS Code project explorer, on an associated workload in the Tanzu Workloads panel, or in the Command Palette (⇧⌘P on Mac and Ctrl+Shift+P on Windows).

- Screenshot of pop-up menu opened from the workload panel:



- Screenshot of the command palette

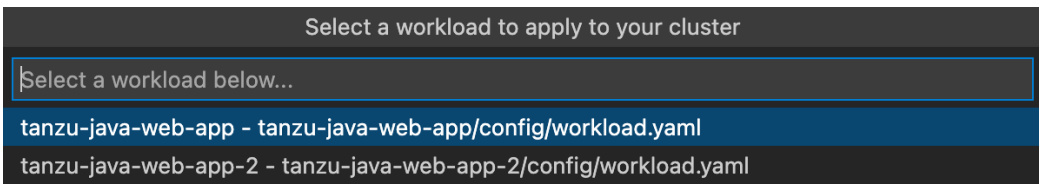


Apply a workload

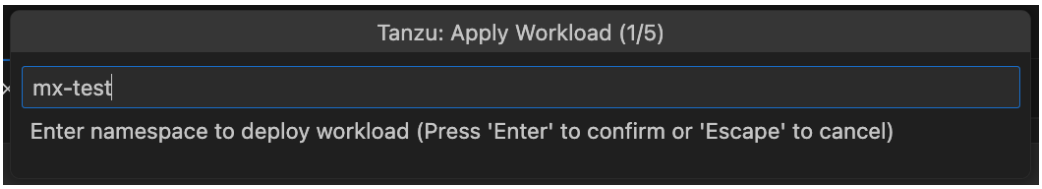
The extension enables you to apply workloads on your Kubernetes cluster that has Tanzu Application Platform.

To apply a workload:

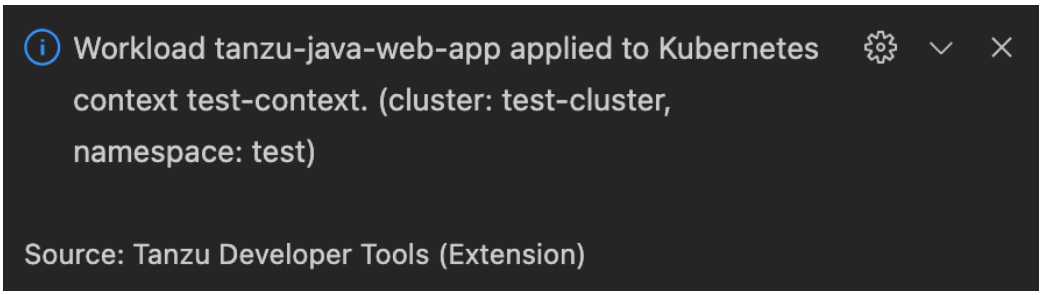
1. Right-click anywhere in the VS Code project explorer, on an associated workload in the Tanzu Workloads panel, or open the Command Palette (⇧⌘P on Mac and Ctrl+Shift+P on Windows).
2. Select the `Tanzu: Apply Workload` command.
3. If applicable, select the workload to apply.



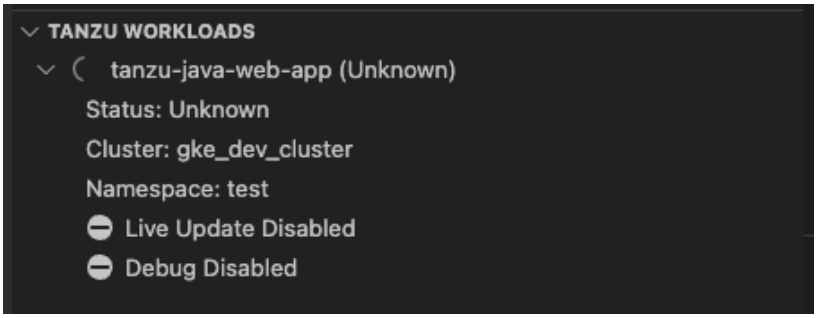
A series of dialog boxes appear that ask for information.



A notification appears showing that the workload was applied.



A new workload appears on the Tanzu Workloads panel.



After the workload is deployed, the status on the Tanzu Workloads panel changes to `Ready`.

Debugging on the cluster

The extension enables you to debug your application on your Kubernetes cluster that has Tanzu Application Platform.

Debugging requires a `workload.yaml` file in your project. For information about creating a `workload.yaml` file, see Get Started with Tanzu Developer Tools for VS Code.

The developer sandbox experience enables developers to Live Update their code, and simultaneously debug the updated code, without having to deactivate Live Update when debugging.

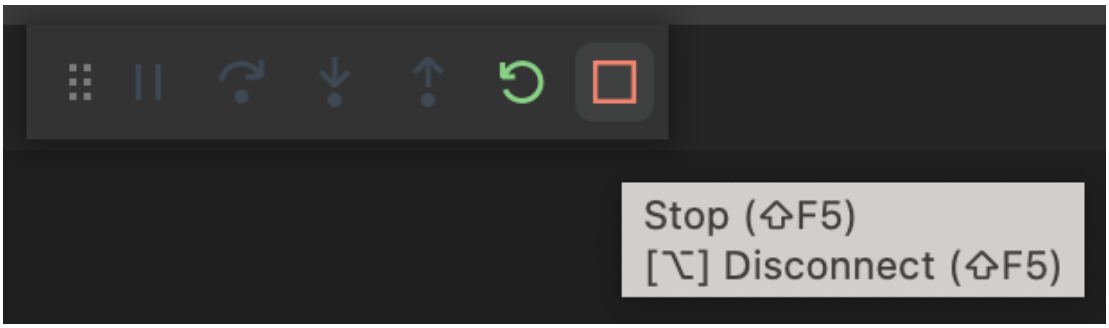
Start debugging on the cluster

To start debugging on the cluster:

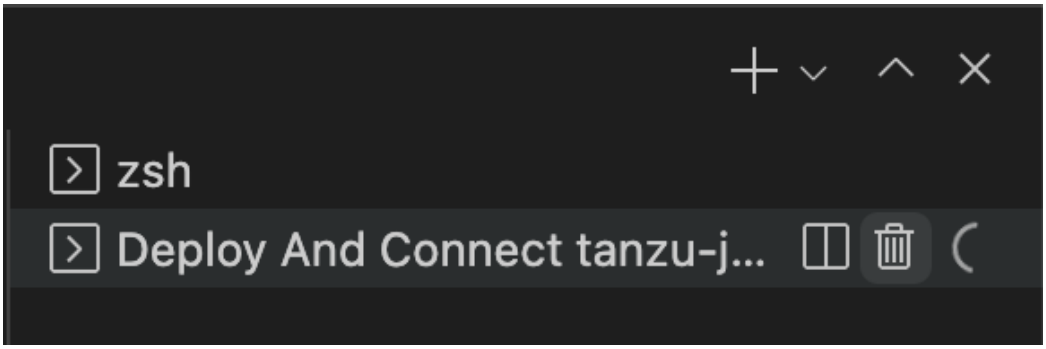
1. Add a breakpoint in your code.
2. Right-click anywhere in the VS Code project explorer, on an associated workload in the Tanzu Workloads panel, or open the Command Palette (⇧⌘P on Mac and Ctrl+Shift+P on Windows).
3. Select the `Tanzu: Java Debug Start` command..

Stop Debugging on the cluster

To stop debugging on the cluster, you can click the stop button in the Debug overlay.



Alternatively, you can press ⌘+J (Ctrl+J on Windows) to open the panel and then click the trash can button for the debug task running in the panel.



Debug apps in a microservice repository

To debug multiple apps in a microservice repository:

1. Add each app folder as a workspace folder. For instructions, see the Visual Studio Code documentation.
2. Update the `tanzu.debugPort` setting so that it does not conflict with other debugging sessions. For how to update individual workspace folder settings, see the Visual Studio Code documentation.

Live Update

With the use of Live Update facilitated by Tilt, the extension enables you to deploy your workload once, save changes to the code, and see those changes reflected in the workload running on the cluster within seconds.

Live Update requires a `workload.yaml` file and a Tiltfile in your project. For information about how to create a `workload.yaml` and a Tiltfile, see Get Started with Tanzu Developer Tools for VS Code.

The developer sandbox experience enables developers to Live Update their code, and simultaneously debug the updated code, without having to deactivate Live Update when debugging.

Start Live Update

Before using Live Update, verify that your auto-save setting is either off or on with a delay. The delay must be long enough for the application to restart between auto saves to allow enough time for your app to Live Update when files change. The auto-save setting is in **Preferences** > **Text Editor** > **Files** > **Auto Save** > **Auto Save Delay**.

To start Live Update:

1. Right-click anywhere in the VS Code project explorer, on an associated workload in the Tanzu Workloads panel, or open the Command Palette (⇧⌘P on Mac and Ctrl+Shift+P on

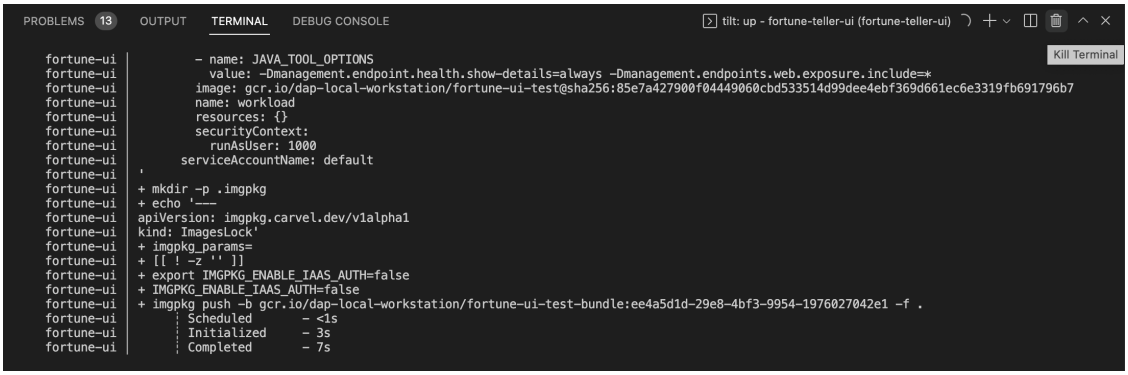
Windows).

2. Select the `Tanzu: Live Update Start` command.

Stop Live Update

When Live Update stops, your application continues to run on the cluster, but the changes you made and saved in your editor are not present in your running application unless you redeploy your application to the cluster.

To stop Live Update, click the trash can button in the terminal pane to stop the Live Update process.



Live Update apps in a microservices repository

To Live Update multiple apps in a microservice repository:

1. Add each app folder as a workspace folder. For instructions, see the Visual Studio Code documentation.
2. Ensure that a port is available to port-forward the Knative service. For example, you might have this in your Tiltfile:

   ```
   k8s_resource('tanzu-java-web-app', port_forwards=["NUMBER:8080"],
                extra_pod_selectors=[{'carto.run/workload-name': 'tanzu-java-web-ap
   p', 'app.kubernetes.io/component': 'run'}])
   ```

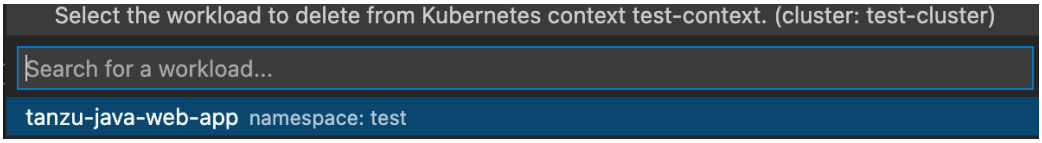
   Where `NUMBER` is the port you choose. For example, `port_forwards=["9999:8080"]`.

Delete a workload

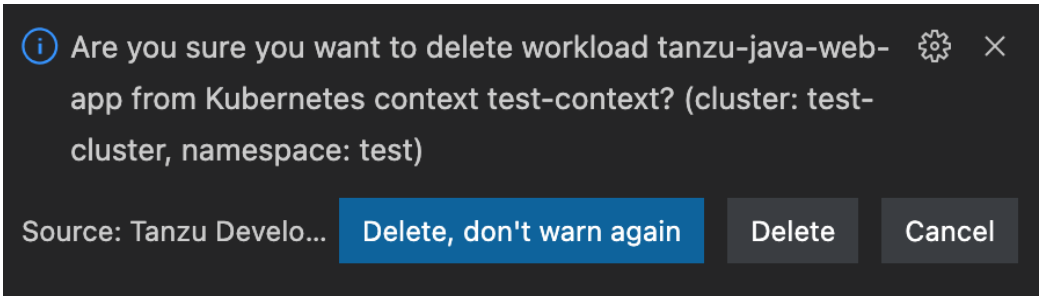
The extension enables you to delete workloads on your Kubernetes cluster that has Tanzu Application Platform.

To delete a workload:

1. Right-click anywhere in the VS Code project explorer, on an associated workload in the Tanzu Workloads panel, or open the Command Palette (⇧⌘P on Mac and Ctrl+Shift+P on Windows).
2. Select the `Tanzu: Delete Workload` command.
3. If applicable, select the workload to delete.



If the **Tanzu: Confirm Delete** setting is enabled, a message appears that prompts you to delete the workload and not warn again, delete the workload, or cancel.

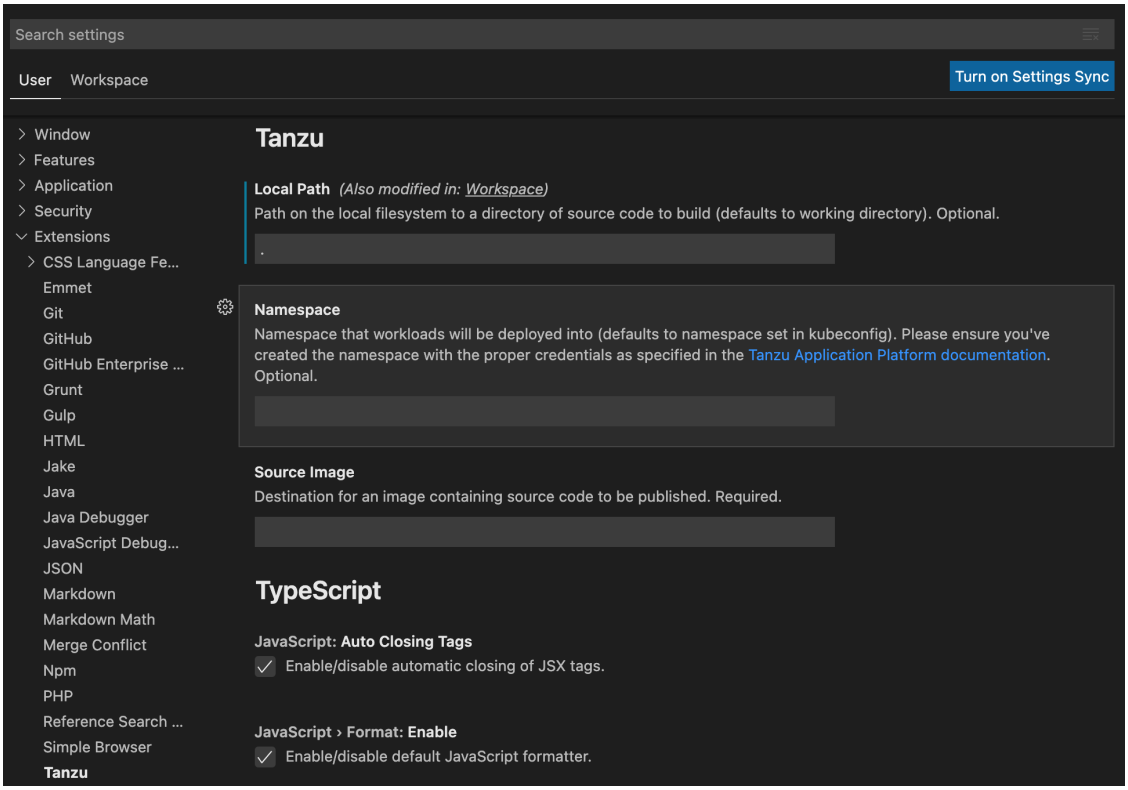


A notification appears showing that the workload was deleted.

Deploy to different namespaces

To deploy to a different namespace from where you created the workload:

1. Go to **Code** > **Preferences** > **Settings**.
2. Expand the **Extensions** section of the settings and click **Tanzu**.
3. In the **Namespace** option, add the namespace you want to deploy to. This is the `default` namespace by default.



Tanzu Workloads panel

The current state of the workloads is visible in the Tanzu Workloads view. This view is a separate section in the bottom of the Explorer view in the Side Bar. The view shows the current status of each workload, namespace, and cluster. It also shows whether Live Update and Debug is running, stopped, or deactivated.

The Tanzu Activity tab in the Panels view enables developers to visualize the supply chain, delivery, and running application pods. The tab enables a developer to view and describe logs on each

resource associated with a workload from within their IDE. The tab displays detailed error messages for each resource in an error state.

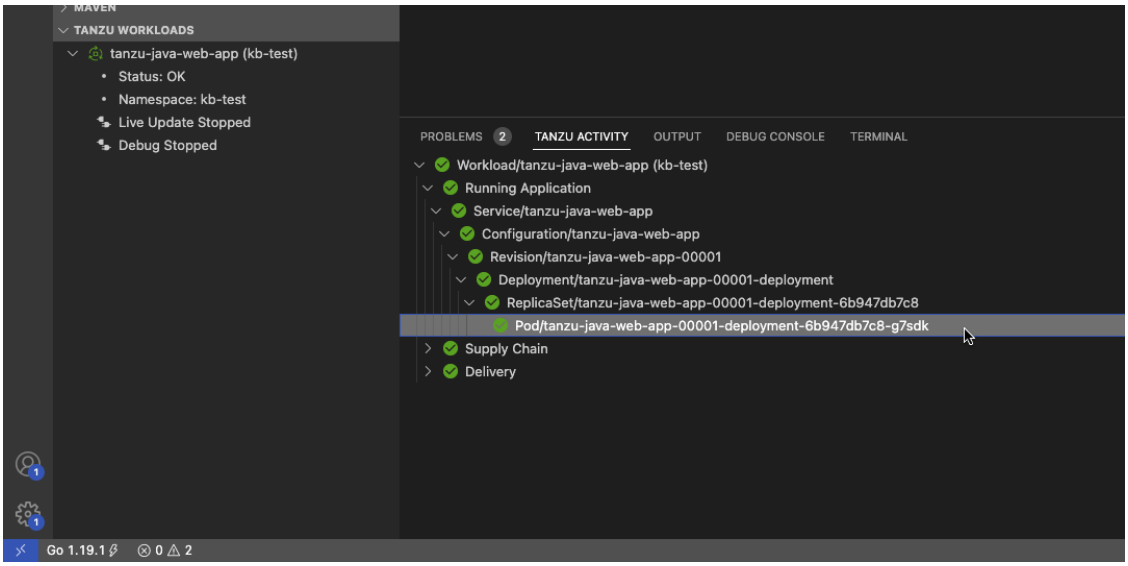
Workload commands are available from the Tanzu Workloads panel on workloads that have an associated module in the current project.

This association is based on a module name and a workload name matching. For example, a project with a module named `my-app` is associated with a deployed workload named `my-app`.

When taking an action from the Tanzu Workloads panel, the action uses the namespace of the deployed workload regardless of the configuration in the module.

For example, you might have a Live Update configuration with a namespace argument of `my-apps-1`, but running the action from a deployed workload in namespace `my-apps-2` starts a Live Update session with a namespace argument of `my-apps-2`.

The Tanzu Workloads panel uses the cluster and defaults to the namespace specified in the current kubectl context.



To add a namespace:

1. View the current context and namespace by running:

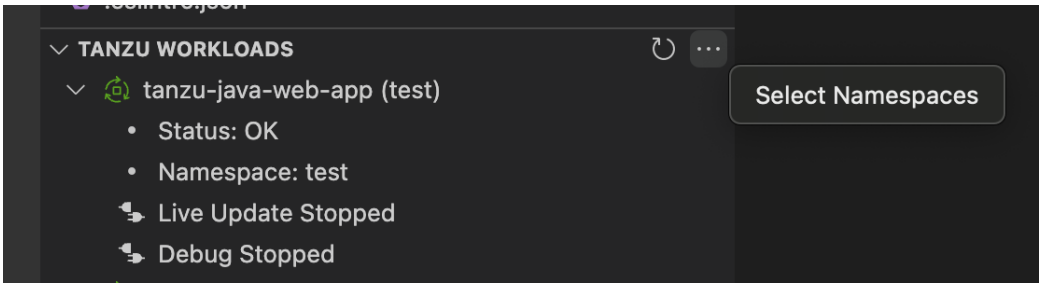
   ```
   kubectl config get-contexts
   ```

2. Set a namespace for the current context by running:

   ```
   kubectl config set-context --current --namespace=YOUR-NAMESPACE
   ```

3. Use one of these methods to add additional namespaces to your Tanzu Workloads panel:

   - Go to **Preferences** > **Extensions** > **Tanzu Developer Tools** > **Tracked Namespaces** and then select the namespaces that you want.
   - Go to **Workload Panel** > **Additional Options** > **Select Namespaces** and then select the namespaces that you want.



Working with Microservices in a Monorepo

A monorepo is single Git repository that contains multiple workloads. Each individual workload is placed in a subfolder of the main repository.

You can find an example of this in Application Accelerator. The relevant accelerator is called Spring SMTP Gateway, and you can obtain its source code as an accelerator or directly from the application-accelerator-samples GitHub repository.

This project exemplifies a typical layout:

- `MONO-REPO-ROOT/`
  - `pom.xml` (parent pom)
  - `microservice-app-1/`
  - `pom.xml`
  - `mvnw` (and other mvn-related files for building the workload)
  - `Tiltfile` (supports Live Update)
  - `config`
    - `workload.yaml` (supports deploying and debugging from IntelliJ)
  - `src/` (contains source code for this microservice)
  - `microservice-app-2/`
  - …similar layout

Recommended structure: Microservices that can be built independently

In this example, each of the microservices can be built independently of one another. Each subfolder contains everything needed to build that workload.

This is reflected in the `source` section of `workload.yaml` by using the `subPath` attribute:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: microservice-app-1
  ...
spec:
  source:
    git:
      ref:
        branch: main
      url: https://github.com/kdvolder/sample-mono-repo.git
    subPath: microservice-app-1 # build only this
  ...
```

For setting up your own repositories, it's best practice to set up a monorepo so that each microservice can be built completely independently.

To work with these monorepos:

- Import the monorepo as a project into VS Code.
- Interact with each of the subfolders in the same way you would interact with a project containing a single workload.

Alternative structure: Services with build-time interdependencies

Some monorepos do not have submodules that can be independently built. Instead the `pom.xml` files of the submodules are set up to have some build-time interdependencies. For example:

- A submodule `pom.xml` can reference the parent `pom.xml` as a common place for centralized dependency management.
- A microservice submodule can reference another, as a maven dependency.
- Several microservice submodules can reference one or more shared library modules.

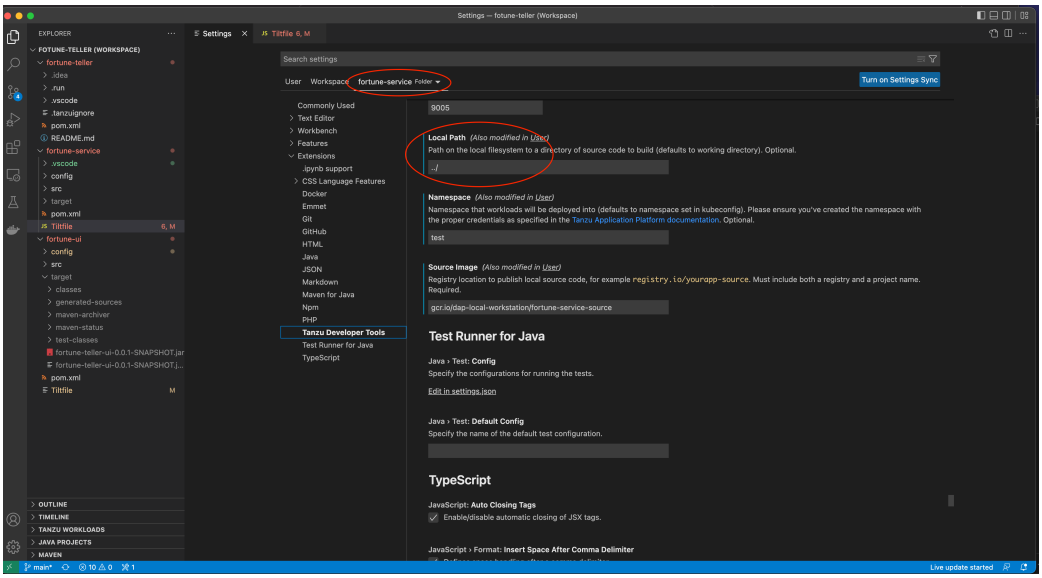
For these projects, make these adjustments:

1. Make `workload.yaml` point to the repository root, not a subfolder. Because submodules have dependencies on code outside of their own subfolder, all source code from the repository must be supplied to the workload builder.
2. Make `workload.yaml` specify additional buildpack arguments through environment variables. They differentiate the submodule that the build is targeting.

   Both of these `workload.yaml` changes are in the following example:

   ```
   apiVersion: carto.run/v1alpha1
   kind: Workload
   metadata:
   name: fortune-ui
   labels:
     apps.tanzu.vmware.com/workload-type: web
     app.kubernetes.io/part-of: fortune-ui
   spec:
   build:
     env:
       - name: BP_MAVEN_BUILD_ARGUMENTS
       value: package -pl fortune-teller-ui -am # indicate which module to build.
       - name: BP_MAVEN_BUILT_MODULE
       value: fortune-teller-ui # indicate where to find the built artefact to de
   ploy.
   source:
     git:
       url: https://github.com/my-user/fortune-teller # repo root
       ref:
       branch: main
   ```

   For more information about these and other `BP_xxx` buildpack parameters, see the Buildpack Documentation.
3. Make the local path preference for each subfolder point to the path of the repository root Because submodules have dependencies on code outside of their own subfolder, all source code from the repository must be supplied to the workload builder.



Changing logging verbosity

The Tanzu Language Server saves logs to `~/tanzu-langserver.log`. You can change the log verbosity in **Preferences** > **Settings** > **Extensions** > **Tanzu Developer Tools** > **Language Server: Log Verbosity**.

Working with Java Native images

Native Image is technology for compiling Java code ahead of time to a binary, which is a native executable file. For more information about Native Image, see the GraalVM documentation.

Native Image requires some changes to your `workload.yaml` files, such as adding new environment variables to the build section of the workload specifications:

```
spec:
  build:
    env:
      - name: BP_NATIVE_IMAGE
        value: "true"
      - name: BP_MAVEN_BUILD_ARGUMENTS
        value: -Dmaven.test.skip=true --no-transfer-progress package -Pnative
      - name: BP_JVM_VERSION
        value: 17 ## only JVM 17 and later versions support native images. Depending o
n your configuration, this might already be the default value.
```

Use native images with Maven

If you are using Maven, you must also add a native profile that includes `native-maven-plugin` for the build phase in `pom.xml`:

```
<profiles>
        <profile>
        <id>native</id>
        <build>
           <plugins>
              <plugin>
                 <groupId>org.graalvm.buildtools</groupId>
                   <artifactId>native-maven-plugin</artifactId>
                </plugin>
            </plugins>
         </build>
```

```
        </profile>
</profiles>
```

Supported Features

There are some differences on supported features when working with Native images:

- You can deploy workloads with native images by running the `Tanzu: Apply Workload` command.
- You can delete workloads with native images by running the `Tanzu: Delete Workload` command.
- Debug and Live Update are not supported when using native images. However you can add an additional `workload.yaml` file that doesn't use a native image to iterate on your development.

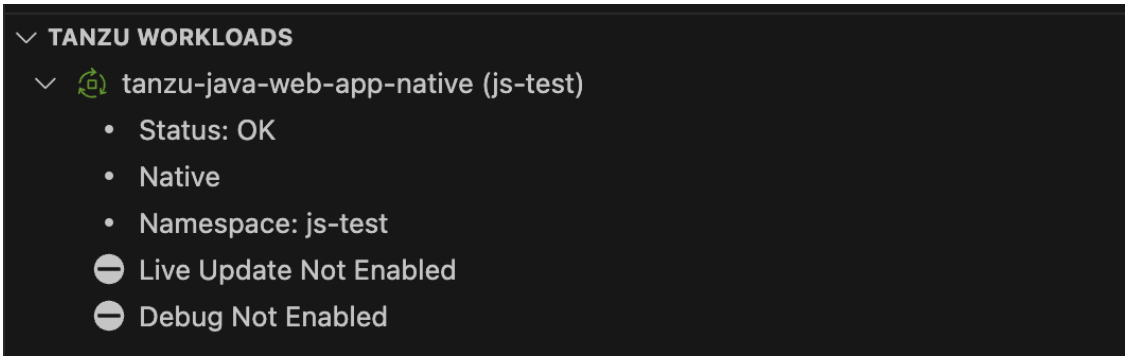
This example `workload.yaml` specification has a native image flag:

```
...
spec:
  build:
    env:
      - name: BP_NATIVE_IMAGE
        value: "true"
...
```

This example `workload.yaml` specification doesn't have a native image flag:

```
...
spec:
  build:
    env:
      #- name: BP_NATIVE_IMAGE
      #  value: "true"
...
```

The Tanzu Workloads panel adds the `Native` label to any workloads that contain native images.



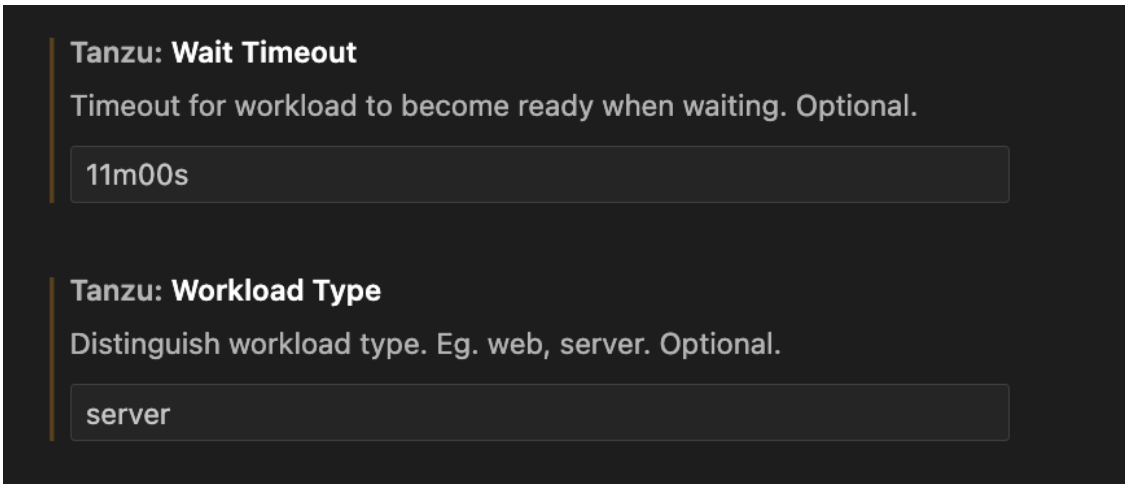
.

Update Workload Apply configurations

The `--wait-timeout` and the `--type` flags can be configured.

For more information, see the Tanzu CLI Command Reference documentation.

To use Telepresence, use `--type server`.



.

Use a `portforward` to access an application locally

You can create a `portforward` by clicking **Tanzu: Portforward** in the pop-up menu in the Tanzu Workloads panel.

A `portforward` enables you to easily access the application, when iterating locally, in the Tanzu Workloads panel from a local URL (via the pop-up menu action) or a Knative URL (for the web type of workloads).

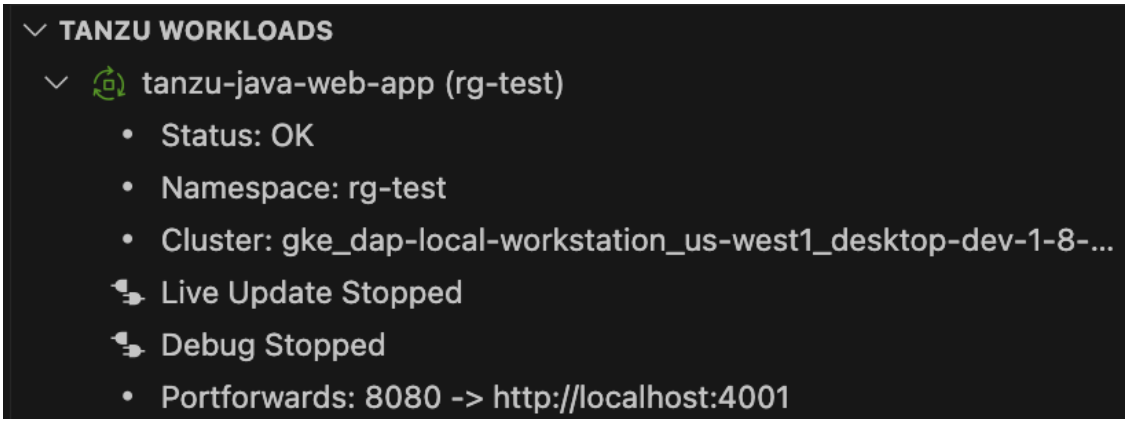
The option to use a `portforward` is only available if containers in your workload have either:

- A `PORT` environment variable
- An entry in the `ports` array that specifies `TCP` as the `protocol`

For example:

```
ports:
- containerPort: 8080
  name: user-port
  protocol: TCP
```

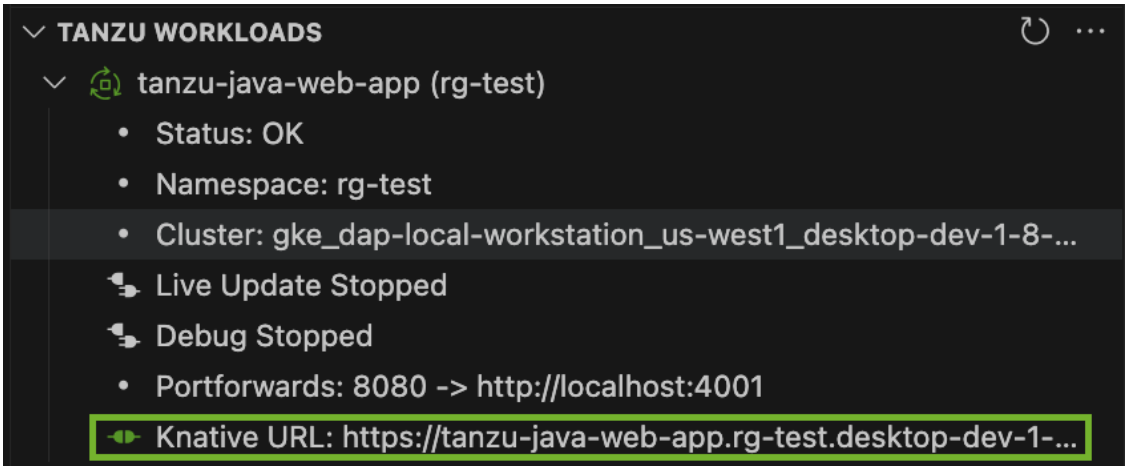
Existing `portforwards` are shown in the Tanzu Workloads panel. You can stop a `portforward` by clicking **Tanzu: Stop Portforward** in the pop-up menu in the Tanzu Workloads panel. The option to stop a `portforward` is only available if there is an existing `portforward`.



.

Viewing the Knative URL

If your workload deploys a Knative service, you can view the Knative URL through the Tanzu Workloads panel.



Use development containers to make a development environment (alpha)

This topic tells you about using development containers (sometimes shortened to dev containers) to make a ready-to-code development environment. This enables you to connect to a pre-configured development container that includes all Tanzu tools and IDE extensions required for development on Tanzu pre-installed.

Caution

Support for development containers is in alpha testing. It is intended for evaluation and test purposes only.

Overview of development containers

A development container enables you to use a container as a full-featured development environment. You can use it to:

- Run an application
- Separate tools, libraries, and runtime environments needed for working with a codebase
- Aid in continuous integration and testing

For more information, see the Development Containers documentation.

Prerequisites

Obtain the following:

- Docker Desktop v2.0 or later or Rancher Desktop on Windows or macOS. If using Docker Desktop, ensure that you meet the system requirements for the VS Code Dev Containers extension.
- The VS Code Dev Containers extension.
- AMD architecture, because ARM, such as Apple M1 Silicon, is not currently supported.

Create a new project or open an existing one

You can start a new project with support for development containers or you can add support for development containers to an existing project:

New project

To set up a new project by using accelerators and development containers:

1. From the Accelerator page in Tanzu Developer Portal, select the Tanzu Java Web App accelerator.
2. Select the options you want.
3. Select the check box for **Include .devcontainer.json (amd64 support only)**.
4. Open a project in VS Code.
5. Follow the VS Code prompts to restart the IDE in `devcontainer` to connect to the development container.

Existing project

To add a development container to an existing project:

1. Open the project with VS Code and ensure that the Tanzu Developer Tools for VS Code plug-in is v1.1.0 or later.
2. Create an empty new file named `devcontainer`.
3. Type `tanzu devcontainer` and press Enter.
4. Rename the file as `.devcontainer.json`.
5. Follow the Visual Studio Code prompts to restart the IDE in `devcontainer` to connect to the development container.

Connect to your cluster

When you start working in your development container for the first time, you must log in and connect to your Tanzu Kubernetes cluster. The method for doing so depends on your cloud provider. For example, for Google Cloud you might run a command similar to:

```
gcloud container clusters get-credentials ${your_cluster_name} --region ${your_region}
\
--project ${your_google_cloud_project_name}
```

For how to connect to a cluster, see the documentation for your specific cloud provider. Examples:

- Azure Kubernetes Service documentation
- Amazon Elastic Kubernetes Service documentation
- Google Kubernetes Engine documentation

Restart the IDE

After you log in to the cluster, restart the IDE for it to detect the new cluster connection. Press CTRL+SHIFT+P (CMD+SHIFT+P on macOS) and click **Developer: Reload Window** in the command palette.

You are now ready to start working on your code, deploy it to your cluster, and monitor your workloads in the Tanzu Panel. To continue your development with Tanzu Developer Tools for VS Code, see Use Tanzu Developer Tools for VS Code.

(Optional) Use local file mounts

Use the `mounts` property in the `.devcontainer.json` file to add a volume bound to any local directory. This is useful for sharing your Kubernetes cluster credentials with the development container, such as a macOS or Linux host.

For example:

```
"mounts": [
        "source=${localEnv:HOME}/.kube/,target=/home/vscode/.kube/,type=bind,consistency=cached"
    ],
```

For more information about using mounts with development containers, see the Visual Studio Code documentation.

VMware General Terms and other legal requirements connected to Tanzu CLI

When the `devcontainer` is created for the first time you are prompted to review and agree with the VMware General Terms and to participate in the Customer Experience Improvement Program (CEIP).

The selections you make are stored and you will not prompted again when you next start `devcontainer`.

Pinniped compatibility

This topic tells you the compatibility details of Pinniped in GitHub.

OAuth

OAuth login is compatible only when both `--skip-browser` and `--skip-listen` flags are not set.

LDAP

LDAP authentication is not compatible with VMware Tanzu Developer Tools for Visual Studio Code.

Integrate Live Hover by using Spring Boot Tools

For more information about this feature, see the **Live application information hovers** section of the Spring Boot Tools Marketplace page.

Prerequisites

To integrate Live Hover by using Spring Boot Tools you need:

- A Tanzu Spring Boot application, such as tanzu-java-web-app
- Spring Boot Extension Pack (includes Spring Boot Dashboard) extension

Activate the Live Hover feature

Activate the Live Hover feature by enabling it in **Code** > **Preferences** > **Settings** > **Extensions** > **Tanzu Developer Tools**.

Deploy a Workload to the Cluster

Follow these steps to deploy the workload for an app to a cluster, making live hovers appear. The examples in some steps reference the sample tanzu-java-web-app.

1. Clone the repository by running:

   ```
   git clone REPOSITORY-ADDRESS
   ```

   Where `REPOSITORY-ADDRESS` is your repository address. For example, `https://github.com/vmware-tanzu/application-accelerator-samples`.

2. Open the project in VS Code, with the Live Hover feature enabled, by running:

   ```
   TAP_LIVE_HOVER=true code ./PROJECT-DIRECTORY
   ```

   Where `PROJECT-DIRECTORY` is your project directory. For example, `./application-accelerator-samples/tanzu-java-web-app`.

3. Verify that you are targeting the cluster on which you want to run the workload by running:

   ```
   kubectl cluster-info
   ```

   For example:

   ```
   $ kubectl cluster-info
   Kubernetes control plane is running at https://...
   CoreDNS is running at https://...

   To further debug and diagnose cluster problems, use 'kubectl cluster-info dum
   p'.
   ```

   Tanzu Developer Tools for VS Code periodically connects to your cluster to search for pods from which live data can be extracted and shown. Tanzu Developer Tools for VS Code uses your current context from `~/.kube/config` to choose which cluster to connect with.

4. If you don't have the workload running yet, run `Tanzu: Apply Workload` from the Command Palette. Tanzu Developer Tools for VS Code periodically searches for pods in your cluster that correspond to the workload configurations it finds in your workspace.

5. The workload takes time to build and then start a running pod. To see if a pod has started running, run:

   ```
   kubectl get pods
   ```

   For example:

   ```
   $ kubectl get pods
   NAME                                                READY   STATUS      REST
   ARTS   AGE
   tanzu-java-web-app-00001-deployment-8596bfd9b4-5vgx2   2/2     Running     0
   20s
   tanzu-java-web-app-build-1-build-pod                0/1     Completed   0
   2m26s
   tanzu-java-web-app-config-writer-fpnzb-pod          0/1     Completed   0
   67s
   ```

   In this example, live data can be extracted from the `...-0001-deployment-...` pod.

6. Open a Java file, such as `HelloController.java`. After a delay of up to 30 seconds, because of a 30-second polling loop, green highlights appear in your code.

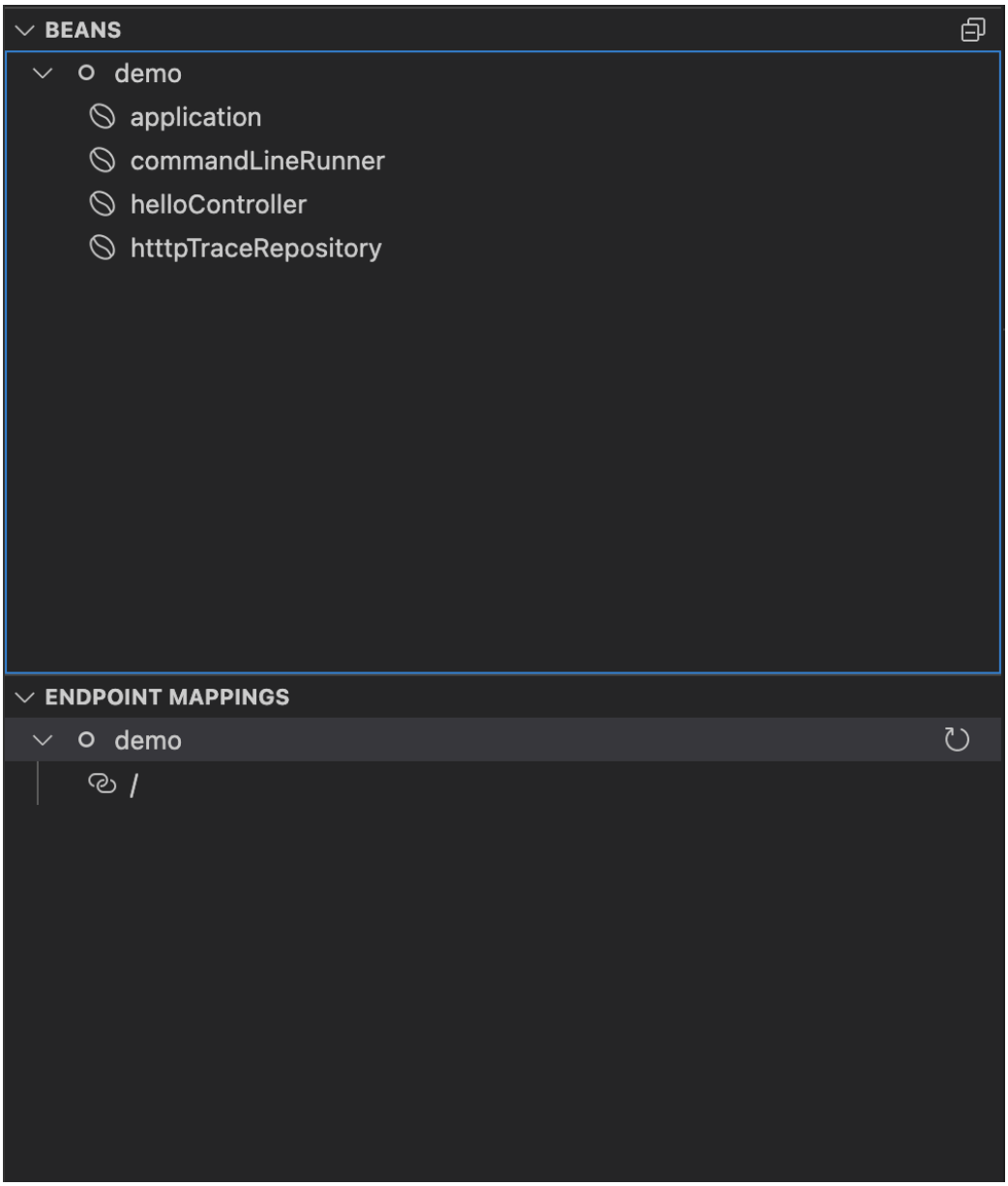
```
@RestController
public class HelloController {

    http://tanzu-java-web-app.my-apps.vipins2appsso.cloudfocused.in/ (Count=1 Total=0.06s Max=0.06s)
    @RequestMapping("/")
    public String index() {

        return "Greetings from Spring Boot + Tanzu + TAP!";
    }

}
```

7. Hover over any of the bubbles to see live information about the corresponding element.
8. The `Live Beans` and `Live Endpoint Mapping` information are displayed in Spring Boot Dashboard. To view the Spring Boot Dashboard, run `View: Show Spring Boot Dashboard` from the Command Palette.



Use Memory View in Spring Boot Dashboard

This topic tells you how to use Spring Boot Dashboard to view memory use.

For more information about Spring Boot Dashboard, see Spring Boot Dashboard.

Prerequisites

To see the Memory View in Spring Boot Dashboard you need:

- A Tanzu Spring Boot application, such as tanzu-java-web-app
- The Spring Boot Extension Pack, which includes Spring Boot Dashboard

Deploy a workload

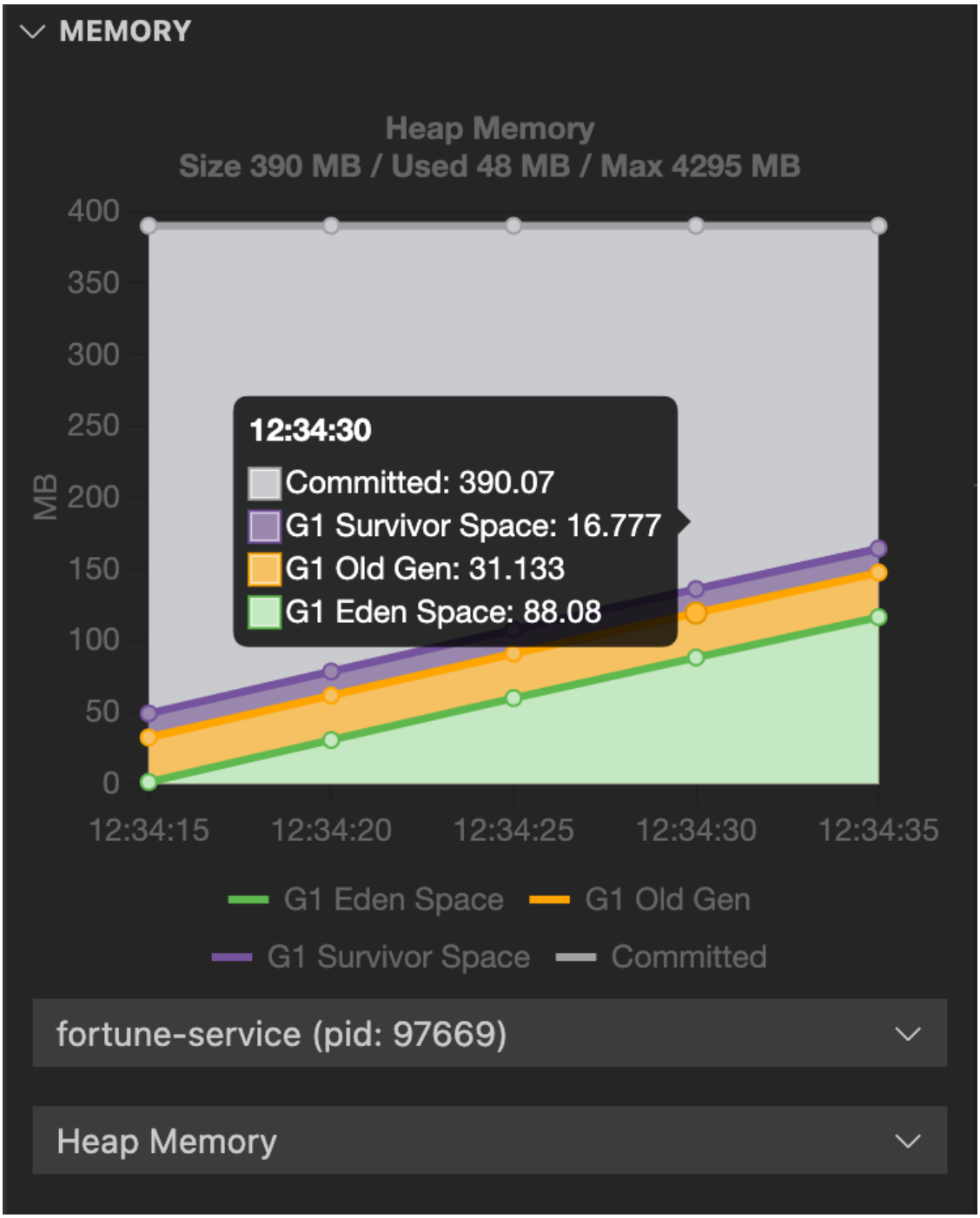
Deploy the workload for an app to a cluster by following the steps in Deploy a Workload to the Cluster.

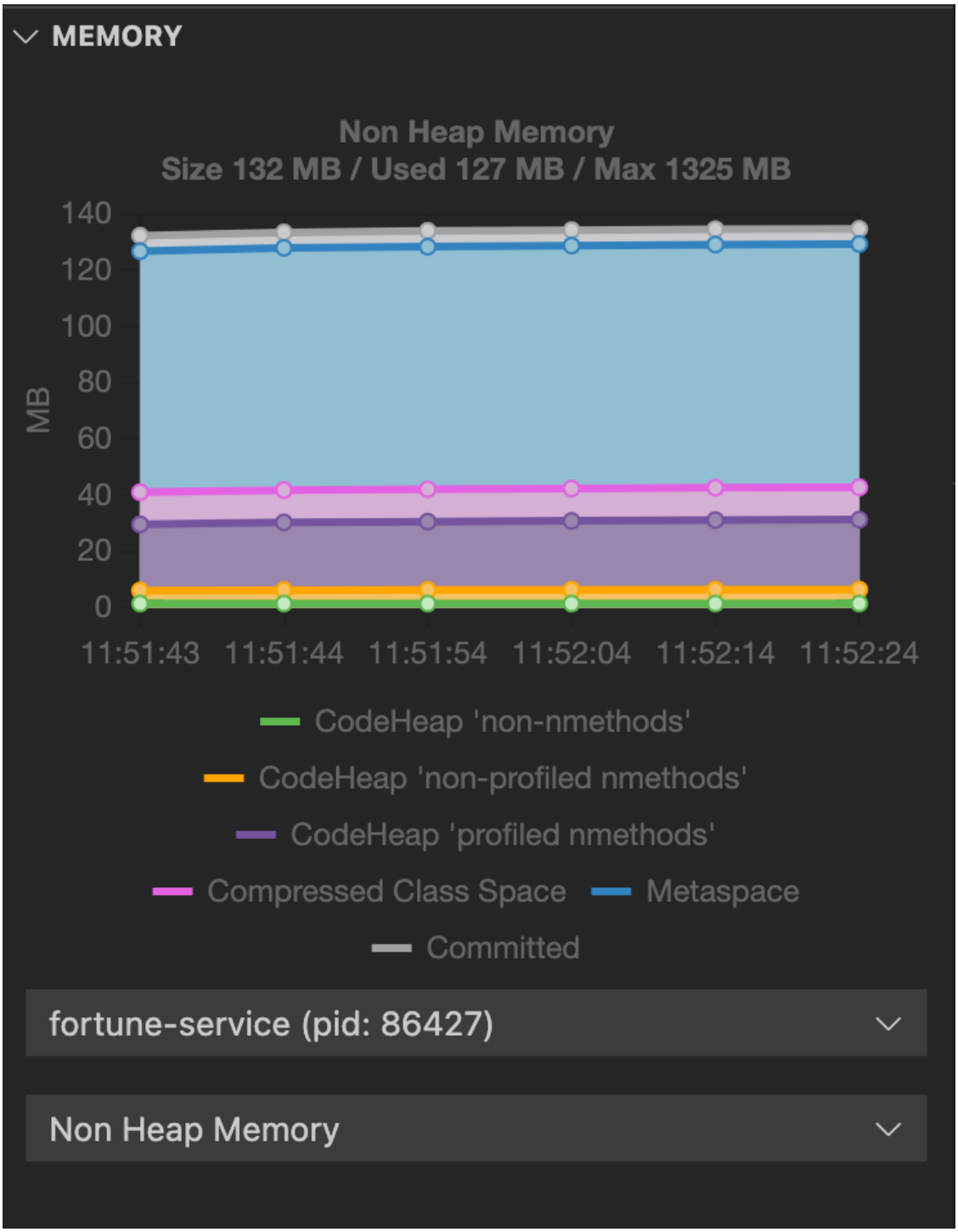
View memory use in Spring Boot Dashboard

To view the Spring Boot Dashboard, run `View: Show Spring Boot Dashboard` from the Command Palette.

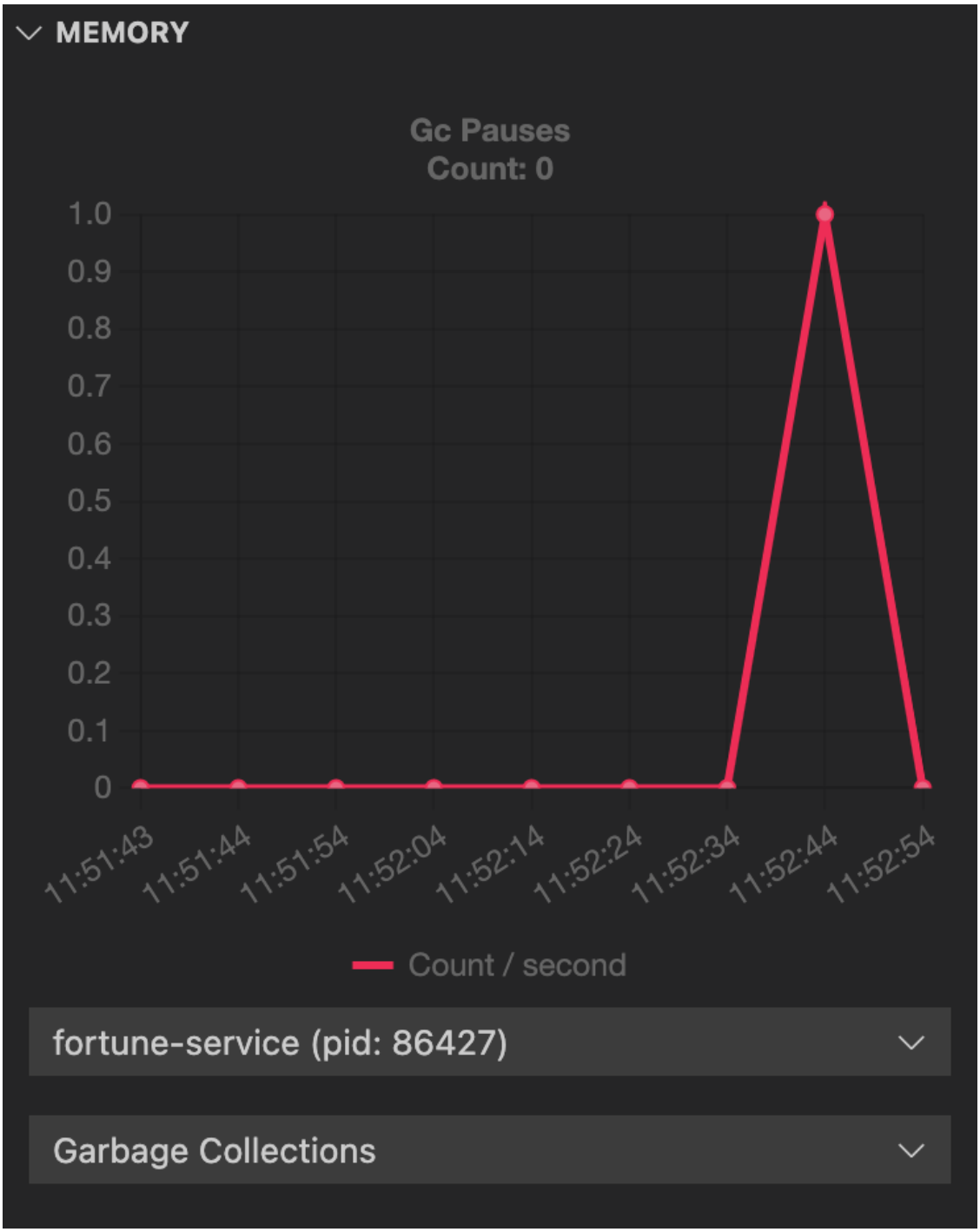
When the app is running, the Memory View section is displayed in Spring Boot Dashboard. The graphical representation in the memory view highlights the memory use inside the Java virtual machine (JVM). The drop-down menus beneath the graph enable you to switch between different running processes and graphical views.

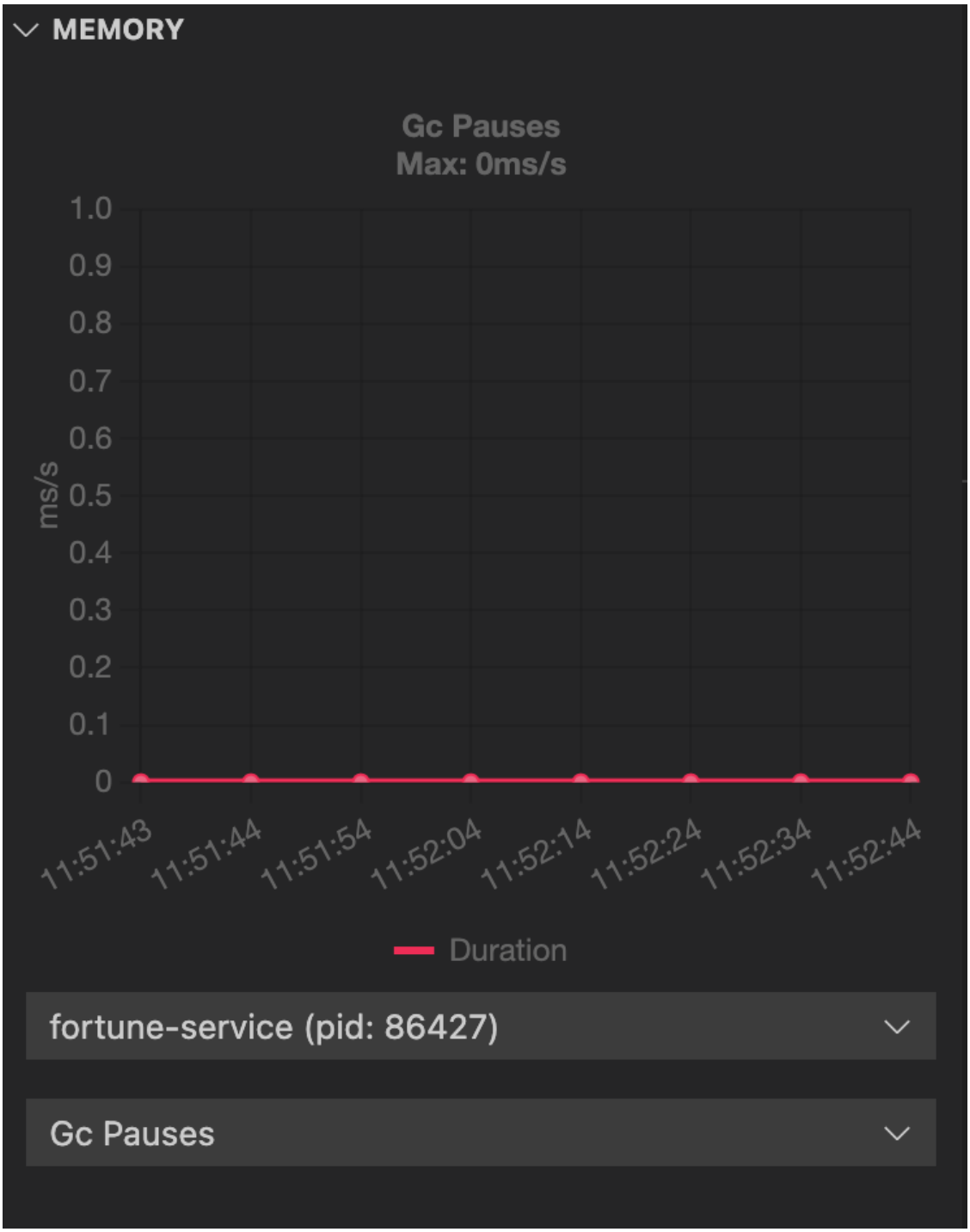
The heap and non-heap memory regions provide memory insights into the application. The real-time graphs display a stacked overview of the different spaces in memory relative to the total memory used and total memory size.


MEMORY
Heap Memory
Size 390 MB / Used 48 MB / Max 4295 MB
400
350
300
250
200
150
100
50
0
MB
12:34:30
Committed: 390.07
G1 Survivor Space: 16.777
G1 Old Gen: 31.133
G1 Eden Space: 88.08
12:34:15
12:34:20
12:34:25
12:34:30
12:34:35
G1 Eden Space
G1 Old Gen
G1 Survivor Space
Committed
fortune-service (pid: 97669)
Heap Memory

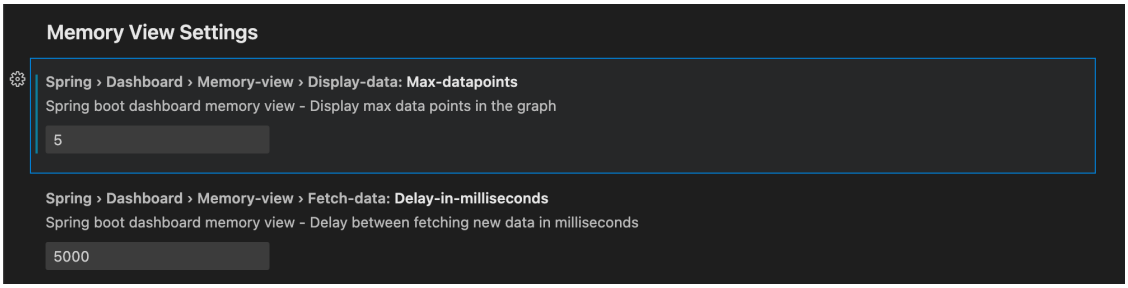


The memory view also contains graphs to display the garbage-collection pauses and garbage-collection events. Long and frequent garbage-collection pauses indicate that the app is having a memory problem that requires further investigation.


MEMORY
Gc Pauses
Count: 0
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
11:51:43
11:51:44
11:51:54
11:52:04
11:52:14
11:52:24
11:52:34
11:52:44
11:52:54
Count / second
fortune-service (pid: 86427)
Garbage Collections



The graphs show only real-time data. You can configure the number of data points to view and the interval by changing the settings. To access the settings on macOS, go to **Code** > **Preferences** > **Settings** > **Extensions** > **Spring Boot Dashboard** > **Memory View Settings**. The navigation path might differ on other operating systems, such as Windows and Linux.



Troubleshoot Tanzu Developer Tools for VS Code

This topic tells you what to do when you encounter issues with VMware Tanzu Developer Tools for Visual Studio Code (VS Code).

Unable to view workloads on the panel when connected to GKE cluster

Symptom

When connecting to Google's GKE clusters, an error appears with the text `WARNING: the gcp auth plugin is deprecated in v1.22+, unavailable in v1.25+; use gcloud instead.`

Cause

GKE authentication was extracted into a separate plug-in and is no longer inside the Kubernetes client or libraries.

Solution

Download and configure the GKE authentication plug-in. For instructions, see the Google documentation.

Live Update fails with `UnsupportedClassVersionError`

Symptom

After live-update has synchronized changes you made locally to the running workload, the workload pods start failing with an error message similar to the following:

```
Caused by: org.springframework.beans.factory.CannotLoadBeanClassException: Error loadi
ng class
[com.example.springboot.HelloController] for bean with name 'helloController' defined
in file
[/workspace/BOOT-INF/classes/com/example/springboot/HelloController.class]: problem wi
th class file
or dependent class; nested exception is
java.lang.UnsupportedClassVersionError: com/example/springboot/HelloController has bee
n compiled by
a more recent version of the Java Runtime (class file version 61.0), this version of t
he
Java Runtime only recognizes class file versions up to 55.0
```

Cause

The classes produced locally on your machine are compiled to target a later Java virtual machine (JVM). The error message mentions `class file version 61.0`, which corresponds to Java 17. The buildpack, however, is set up to run the application with an earlier JVM. The error message mentions `class file versions up to 55.0`, which corresponds to Java 11.

The root cause of this is a misconfiguration of the Java compiler that VS Code uses. The cause might be a suspected issue with the VS Code Java tooling, which sometimes fails to properly configure the compiler source and target compatibility-level from information in the Maven POM.

For example, in the `tanzu-java-web-app` sample application the POM contains the following:

```
<properties>
        <java.version>11</java.version>
```

```
        ...
</properties>
```

This correctly specifies that the app must be compiled for Java 11 compatibility. However, the VS Code Java tooling sometimes fails to take this information into account.

Solution

Force the VS Code Java tooling to re-read and synchronize information from the POM:

1. Right-click the `pom.xml` file.
2. Click **Reload Projects**.

This causes the internal compiler level to be set correctly based on the information from `pom.xml`. For example, Java 11 in `tanzu-java-web-app`.

Timeout error when Live Updating

Symptom

When you attempt to Live Update your workload, the following error message appears in the log:

```
ERROR: Build Failed: apply command timed out after 30s - see }}{{https://docs.tilt.dev/api.html#api.update_settings{{ for how to increase}}
```

Cause

Kubernetes times out on upserts over 30 seconds.

Solution

Add `update_settings (k8s_upsert_timeout_secs = 300)` to the Tiltfile. For more information, see the Tiltfile documentation.

Task-related error when running a Tanzu Debug launch configuration

Symptom

When you attempt to run a Tanzu Debug launch configuration, you see a task-related error message similar to the following:

```
Could not find the task 'tanzuManagement: Kill Port Forward my-app
```

Cause

The task you're trying to run is no longer supported.

Solution

Delete the launch configuration from your `launch.json` file in your `.vscode` directory.

Tanzu Workloads panel workloads only show delete command

Symptom

Some or all workloads in the Tanzu Workloads panel only have describe and delete actions.

Cause

By design, only associated workloads have apply, debug, and Live Update workload actions available.

Solution

Open a project that contains a module that can be associated with your deployed workloads.

Workload actions do not work when in a project with spaces in the name

Symptom

Workload actions and Live Update do not work. The console displays an error message similar to:

```
Error: unknown command "projects/my-app" for "apps workload apply". Process finished w
ith exit code 1
```

Cause

Improper handling of paths that contain spaces causes the shell to misinterpret some commands, such as `tanzu workload apply ...`. This causes anything executing these commands to fail when the name of a project, or any parts of its path on disk, contain spaces.

Solution

1. Close the code editor.
2. Move or rename your project folder on the disk, ensuring that no part of its path contains any spaces.
3. Delete the project settings folder from the project to start with a clean slate. The folder is `.idea` if using IntelliJ and `.vscode` if using VS Code.
4. Open the code editor and then open the project in its new location.

Cannot apply workload because of a malformed kubeconfig file

Symptom

You cannot apply a workload. You see an error message when you attempt to do so.

Cause

Your kubeconfig file (`~/.kube/config`) is malformed.

Solution

Fix your kubeconfig file.

Live Update changes aren't visible when using development containers on Windows

Symptom

Live Update deploys your workload but changes are not visible when using development containers on Windows.

Cause

Monitoring file system changes on Windows directories, while mounted in a Docker container, is very slow and might not work in your specific setup.

Solution

1. Clone the repository inside of a container volume instead of mounting a local directory in the container. This improves I/O performance.
2. Run the `Dev Containers: Clone Repository in Container Volume...` command while the project is open to get started. For more information about this command, see the Visual Studio Code documentation.

You can only store a project in a development container

Symptom

When using a development container, and creating a new project using the Application Accelerator extension in VS Code, the only available location to store the project is inside the development container.

Cause

VS Code is running inside a development container. Therefore the development container file system is the file system made available when creating a new project.

Solution

Use local file mounts to expose additional directories in the host file system to the development container. For more information, see Use Mounts.

Frequent application restarts

Symptom

When an application is applied from VS Code it restarts frequently.

Cause

An application or environment behavior is triggering the application to restart.

Observed trigger behaviors include:

- The application itself writing logs to the file system in the application directory that Live Update is watching
- Autosave being set to a very high frequency in the IDE configuration

Solution

Prevent the trigger behavior. Example solutions include:

- Prevent 12-factor applications from writing to the file system.

- Reduce the autosave frequency to once every few minutes.

Overview of Tanzu Supply Chain

Tanzu Supply Chain is a tool that provides a golden path to production for your teams. Tanzu Supply Chain replaces the supply-chain solution Supply Chain Choreographer, which is based on Cartographer.

> **Caution**
>
> Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.

Documentation structure

The Tanzu Supply Chain documentation is organized into the following sections:

- Platform Engineering with Supply Chain: For platform engineers that want to author supply chains that simplify developer cognitive load, and ensure compliant, secure delivery of source to production.
- Developing software with Supply Chain: For developers that want to deliver their software following team and organization standards.
- Reference

Overview of Tanzu Supply Chain

Tanzu Supply Chain is a tool that provides a golden path to production for your teams. Tanzu Supply Chain replaces the supply-chain solution Supply Chain Choreographer, which is based on Cartographer.

> **Caution**
>
> Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.

Documentation structure

The Tanzu Supply Chain documentation is organized into the following sections:

- Platform Engineering with Supply Chain: For platform engineers that want to author supply chains that simplify developer cognitive load, and ensure compliant, secure delivery of source to production.
- Developing software with Supply Chain: For developers that want to deliver their software following team and organization standards.
- Reference

Tanzu Supply Chain for platform engineers

This topic tells you about the Tanzu Supply Chain features that are focused on platform engineering.

Caution

Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.

Prepare

Install these CLI tools:

- Install the Tanzu CLI.
- Install the Tanzu Supply Chain CLI plug-ins.

Install and configure Tanzu Supply Chain

- Install Tanzu Supply Chain
- Configure Tanzu Supply Chain

Tutorials

- Build your first SupplyChain
- Install an authored SupplyChain
- Build your first Component
- Add Stages to your SupplyChain

How-to topics: `SupplyChain` Authoring

- Construct a Supply Chain using the Tanzu CLI
- Configure a Supply Chain using the Tanzu CLI

How-to topics: Deploying and Managing Supply Chains

- Manage SupplyChains with GitOps

How-to topics: Starter Supply Chains

- Create Starter SupplyChains

Out of the Box catalog of components

- Catalog of Tanzu Supply Chain Components
- Output types for catalog components

Reference topics

- Overview of SupplyChains

- Overview of Components
- Overview of Resumptions
- Overview of Workloads
- Overview of WorkloadRuns

Tanzu Supply Chain tutorials for Platform Engineering

This section contains tutorials for Tanzu Supply Chain platform engineers.

> **Caution**
>
> Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.

In this section:

- Build your first SupplyChain
- Install an authored SupplyChain
- Build your first Component
- Add Stages to your SupplyChain

Build your first Supply Chain

This topic tells you how to use the Tanzu Supply Chain CLI plug-in to create a SupplyChain for developers to use.

> **Caution**
>
> Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.

This SupplyChain retrieves the source code from a Git repository, and proceeds with building and packaging it as a Carvel package. Subsequently, it initiates a pull request (PR) to transfer the Carvel package to a GitOps repository, facilitating the installation of the built package on the Run clusters. In this tutorial, you will construct a supply chain to build, test, and package Java Maven applications.

Prerequisites

1. Ensure that the Tanzu CLI and Tanzu Supply Chain CLI plug-ins are installed on your local machine.
2. Ensure that Tanzu Supply Chain is installed on the Tanzu Application Platform cluster that you are using to author your first supply chain.

Browse the component catalog

When you have completed the prerequisites, you have the Tanzu Supply Chain controller, Managed Resource Controller, and Component packages installed on the cluster and you are ready to build your first SupplyChain.

1. As a Platform Engineer, you want to know which components are available to use in your SupplyChain. Run:

   ```
   tanzu supplychain component list
   ```

   Example output:

   ```
   Listing components from the catalog
   NAMESPACE                   NAME                                AGE  DESCRIPTION
   alm-catalog                 app-config-server-1.0.0             18h  Generates con
   figuration for a Server application from a Conventions PodIntent.
   alm-catalog                 app-config-web-1.0.0                18h  Generates con
   figuration for a Web application from a Conventions PodIntent.
   alm-catalog                 app-config-worker-1.0.0             18h  Generates con
   figuration for a Worker application from a Conventions PodIntent.
   alm-catalog                 carvel-package-1.0.0                18h  Generates a c
   arvel package from OCI images containing raw YAML files and YTT
                                                                        files.
   alm-catalog                 deployer-1.0.0                      18h  Deploys K8s r
   esources to the cluster.
   alm-catalog                 source-package-translator-1.0.0     18h  Takes the typ
   e source and immediately outputs it as type package.
   conventions-component       conventions-1.0.0                   18h  The Conventio
   ns component analyzes the `image` input as described in the
   git-writer-catalog          git-writer-1.0.0                    18h  Writes carvel
   package config directly to a gitops repository
   git-writer-catalog          git-writer-pr-1.0.0                 18h  Writes carvel
   package config to a gitops repository and opens a PR
   kaniko-catalog              kaniko-build-1.0.0                  18h  Builds an app
   with kaniko
   source-provider             source-git-provider-1.0.0           18h  Source git pr
   ovider retrieves source code and monitors a git repository.
   tbs-catalog                 buildpack-build-1.0.0               18h  Builds an app
   with buildpacks using kpack
   trivy-app-scanning-catalog  trivy-image-scan-1.0.0              18h  Performs a tr
   ivy image scan using the scan 2.0 components

     To view the details of a component, use 'tanzu supplychain component get'
   ```

   Use the `-w/--wide` flag to see a more detailed output including a inputs/outputs of each component.

2. To get more information about each component on the cluster, use the `tanzu supplychain component get` command. For example, to get the information about the `source-git-provider` component, run:

   ```
   tanzu supplychain component get source-git-provider-1.0.0 --show-details
   ```

   Example output:

   ```
     Overview
     name:          source-git-provider-1.0.0
     namespace:     source-provider
     age:           19h
     status:        True
     reason:        Ready
     description:   Source git provider retrieves source code and monitors a git re
   pository.

     Configuration
     source:
       #! Use this object to retrieve source from a git repository.
       #! The tag, commit and branch fields are mutually exclusive, use only one.
   ```

```
    #! Required
    git:
      #! A git branch ref to watch for new source
      #! Example: "main"
      branch: ""
      #! A git commit sha to use
      commit: ""
      #! A git tag ref to watch for new source
      #! Example: "v1.0.0"
      tag: ""
      #! The url to the git source repository
      #! Example: "https://github.com/acme/my-workload.git"
      #! Required
      url: ""
    #! The sub path in the bundle to locate source code
    #! Example: "sub-dir"
    subPath: ""

  Outputs
  git
    ├─ digest: $(resumptions.check-source.results.sha)
    ├─ type: git
    └─ url: $(resumptions.check-source.results.url)
  source
    ├─ digest: $(pipeline.results.digest)
    ├─ type: source
    └─ url: $(pipeline.results.url)

  Pipeline
    ├─ name: source-git-provider
    └─   parameters
      ├─ git-url: $(workload.spec.source.git.url)
      ├─ sha: $(resumptions.check-source.results.sha)
      └─ workload-name: $(workload.metadata.name)

  Resumptions
  - check-source runs source-git-check task every 60s
      Parameters
      ├─ git-branch: $(workload.spec.source.git.branch)
      ├─ git-commit: $(workload.spec.source.git.commit)
      ├─ git-tag: $(workload.spec.source.git.tag)
      └─ git-url: $(workload.spec.source.git.url)

  To generate a supplychain using the available components, use 'tanzu supplych
ain generate'
```

Generate a SupplyChain

1. Now that you know what components are available to create your SupplyChain, start the authoring process. Use the `tanzu supplychain init` command to scaffold the current directory. Run:

   ```
   mkdir myfirstsupplychaingroup
   cd myfirstsupplychaingroup
   tanzu supplychain init --group supplychains.tanzu.vmware.com --description "Thi
   s is my first Supplychain group"
   ```

   Example output:

   ```
   Initializing group supplychains.tanzu.vmware.com
   Creating directory structure
   ├─ supplychains/
   ├─ components/
   ```

```
├─ pipelines/
├─ tasks/
├─ Makefile
├─ README.md
└─ config.yaml

Writing group configuration to config.yaml
```

The `tanzu supplychain init` command creates the following:

- `config.yaml` file that contains the information about the group name, and the description of the SupplyChain group.
- `supplychains`, `components`, `pipelines`, and `tasks` directories which are auto-populated by the authoring wizard later in this tutorial.
- `Makefile` which has the targets to install and uninstall the SupplyChain and related dependencies on any build or full profile clusters.
- `README.md` file which has instructions on how to use the targets in the `Makefile`.

2. Your current directory is now initialized, use the SupplyChain authoring wizard to generate your first SupplyChain. Start the wizard:

```
tanzu supplychain generate --allow-defaults --allow-overrides
```

3. In the wizard prompts that follow, add the following values:

Prompt	Value
What Kind would you like to use as the developer interface?	`MavenAppBuildv1`
Give Supply chain a description?	`Supply chain that pulls the Maven app source code from Git repository, builds it using buildpacks and packages the output as Carvel package.`
Select a component as the first stage of the supply chain?	`source-git-provider-1.0.0`
Select a component as the next stage of the supply chain?	`buildpack-build-1.0.0`
Select a component as the next stage of the supply chain?	`conventions-1.0.0`
Select a component as the next stage of the supply chain?	`app-config-server-1.0.0`
Select a component as the next stage of the supply chain?	`carvel-package-1.0.0`
Select a component as the next stage of the supply chain?	`git-writer-pr-1.0.0`
Select a component as the next stage of the supply chain?	`Done`

After you have selected the components, the wizard creates the required files to deploy your SupplyChain in the current directory. Example output:

```
✓ Successfully fetched all component dependencies
Created file supplychains/mavenappbuildv1.yaml
Created file components/app-config-server-1.0.0.yaml
Created file components/buildpack-build-1.0.0.yaml
Created file components/carvel-package-1.0.0.yaml
Created file components/conventions-1.0.0.yaml
```

```
Created file components/git-writer-pr-1.0.0.yaml
Created file components/source-git-provider-1.0.0.yaml
Created file pipelines/app-config-server.yaml
Created file pipelines/buildpack-build.yaml
Created file pipelines/carvel-package.yaml
Created file pipelines/conventions.yaml
Created file pipelines/git-writer.yaml
Created file pipelines/source-git-provider.yaml
Created file tasks/calculate-digest.yaml
Created file tasks/carvel-package-git-clone.yaml
Created file tasks/carvel-package.yaml
Created file tasks/check-builders.yaml
Created file tasks/fetch-tgz-content-oci.yaml
Created file tasks/git-writer.yaml
Created file tasks/gitops-git-clone.yaml
Created file tasks/prepare-build.yaml
Created file tasks/source-git-check.yaml
Created file tasks/source-git-clone.yaml
Created file tasks/store-content-oci.yaml
```

4. You have now authored your first SupplyChain. View the SupplyChain definition created by the wizard by viewing the manifest created in the `supplychains/` directory. Run:

```
cat supplychains/mavenappbuildv1.yaml
```

Example output

```
apiVersion: supply-chain.apps.tanzu.vmware.com/v1alpha1
kind: SupplyChain
metadata:
    name: mavenappbuildv1
spec:
    defines:
        group: supplychains.tanzu.vmware.com
        kind: MavenAppBuildv1
        plural: mavenappbuildv1s
        version: v1alpha1
    description: Supply chain that pulls the maven app source code from Git rep
ository, builds it using buildpacks and packages the output as Carvel package.
    stages:
        - componentRef:
            name: source-git-provider-1.0.0
          name: source-git-provider
        - componentRef:
            name: buildpack-build-1.0.0
          name: buildpack-build
        - componentRef:
            name: conventions-1.0.0
          name: conventions
        - componentRef:
            name: app-config-server-1.0.0
          name: app-config-server
        - componentRef:
            name: carvel-package-1.0.0
          name: carvel-package
        - componentRef:
            name: git-writer-pr-1.0.0
          name: git-writer-pr
    config:
    #   overrides:
          ...
    #   defaults:
          ...
```

The CLI generated `SupplyChain` manifest has the following information:

- Name of the SupplyChain.
- The Kind definition of the Developer API (Workload) it will create.
- Stages in the SupplyChain. The order of the stages are important as the stages are executed sequentially. Each stage is represented by a component.
- `overrides`: The configuration supplied by a platform engineer for a stage (component) in the SupplyChain that cannot be overridden by developers using the `Workload` (Developer API) created by this SupplyChain.
- `defaults`: The configuration supplied by a platform engineer for a stage (component) in the SupplyChain that can be overridden by developers using the `Workload` (Developer API) created by this SupplyChain. If the developer does not override the configuration, the platform engineer provides the default value.

Configure a SupplyChain to run securely

To configure a SupplyChain to run securely, decide where the SupplyChain stages run, and refine the Developer API (Workload) using overrides and defaults.

Decide where the SupplyChain stages run

Tanzu Supply Chain execute workloads created by developers in accordance with the continuous delivery guidelines established by platform engineers. To ensure compliance and security, platform engineers must supply configurations and sensitive secrets that are not accessible to developers. While supply chains can operate within hardened namespaces to address these concerns, certain elements of a supply chain workload must remain configurable by developers.

Important

Each stage within a supply chain can be configured to operate within either the `supplychain` namespace, where the `Supplychain` will be deployed, or the `developer` namespace, where the `Workload` will be generated by the developer. By default, every stage in a supply chain operates within the `supplychain` namespace. This namespace is secure, restricting developers access to create or view sensitive resources such as secrets. Nevertheless, there are scenarios where developers must provide secrets required by specific stages within the supply chain.

For our generated supply chain, the `source-git-provider` stage operates within the `developer` namespace. Developers need the capability to create a git-secret for their private Git repository where their source code is stored, without requiring involvement from platform engineers. Some stages, such as `buildpack-build` and `git-writer-pr`, should not operate within the developer namespace. The `buildpack-build` stage requires registry credentials for storing the built images, and the `git-writer-pr` stage requires git-secret with write permissions to generate a PR to the GitOps repository. Developers do not have access to provide or view these secrets due to compliance reasons, as they might contain sensitive corporate data.

To do this, update the `source-git-provider` stage of the generated `supplychains/mavenappbuildv1.yaml` file as follows:

```
...
      - componentRef:
          name: source-git-provider-1.0.0
        name: source-git-provider
```

```
        securityContext:
          runAs: workload
...
```

This tells the SupplyChain to run the `source-git-provider` stage in the `developer` namespace where the developer source Git repository secret needed by `source-git-provider` is provided by developers. All other steps run in the `supplychain` namespace where a platform engineer can provide the necessary secrets for the `buildpack-build` stage and the `git-writer-pr` stage during supply chain installation.

Refine the Developer API (Workload) using overrides and defaults

When you generate a supply chain using the `tanzu supplychain generate` command, use the `--allow-defaults` and `--allow-overrides` to generate all the fields that are accessible in the Developer API (Workload) for developers to configure. These fields represent the configurations exposed by the components. Run the `tanzu supplychain component get` command to view the configurations exposed by the components. Inspect the `config.overrides` section of the `supplychains/mavenappbuildv1.yaml` file generated by the CLI to see all the values that developers can specify in a `Workload`.

Some fields, such as `spec.registry`, can only be defined by platform engineers, who can provide the necessary secrets to access those locations in the secure `supplychain` namespace. Developers cannot override these values. This is where the overrides feature becomes relevant. Platform engineers uncomment the fields they want to override and provide the corresponding values at the supply chain level. When these overrides are implemented, these fields are no longer accessible within the `Workload`.

To do this, update the `config.overrides` section of the generated `supplychains/mavenappbuildv1.yaml` file as follows:

```
...
  config:
    overrides:
        # Platform Engineer provided registry overrides
        - path: spec.registry.repository
          value: "YOUR-REGISTRY-REPO"
        - path: spec.registry.server
          value: "YOUR-REGISTRY-SERVER"

        # Platform Engineer provided build overrides
        - path: spec.build.builder.kind
          value: clusterbuilder
        - path: spec.build.builder.name
          value: default
        - path: spec.build.cache.enabled
          value: false
        - path: spec.build.cache.image
          value: ""
        - path: spec.build.serviceAccountName
          value: default

        # Platform Engineer provided carvel package component overrides
        - path: spec.carvel.caCertData
          value: ""
        - path: spec.carvel.iaasAuthEnabled
          value: false
        - path: spec.carvel.packageDomain
          value: "default.tap"
        - path: spec.carvel.serviceAccountName
          value: "default"
        - path: spec.carvel.valuesSecretName
          value: ""
```

```
        # Platform Engineer provided GitOps repo overrides
        - path: spec.gitOps.baseBranch
          value: main
        - path: spec.gitOps.branch
          value: main
        - path: spec.gitOps.subPath
          value: "YOUR-GITOPS-REPO-SUBPATH"
        - path: spec.gitOps.url
          value: "YOUR-GITOPS-REPO-URL"
...
```

Review our SupplyChain

Here is an example of what a `SupplyChain` looks like when the configuration is complete:

```
apiVersion: supply-chain.apps.tanzu.vmware.com/v1alpha1
kind: SupplyChain
metadata:
    name: mavenappbuildv1
spec:
    defines:
        group: supplychains.tanzu.vmware.com
        kind: MavenAppBuildv1
        plural: mavenappbuildv1s
        version: v1alpha1
    description: Supply chain that pulls the maven app source code from Git repositor
y, builds it using buildpacks and packages the output as Carvel package.
    stages:
        - componentRef:
            name: source-git-provider-1.0.0
          name: source-git-provider
          securityContext:
            runAs: workload
        - componentRef:
            name: buildpack-build-1.0.0
          name: buildpack-build
        - componentRef:
            name: conventions-1.0.0
          name: conventions
        - componentRef:
            name: app-config-server-1.0.0
          name: app-config-server
        - componentRef:
            name: carvel-package-1.0.0
          name: carvel-package
        - componentRef:
            name: git-writer-pr-1.0.0
          name: git-writer-pr

    config:
      overrides:
        # Platform Engineer provided registry overrides
        - path: spec.registry.repository
          value: "YOUR-REGISTRY-REPO"
        - path: spec.registry.server
          value: "YOUR-REGISTRY-SERVER"

        # Platform Engineer provided build overrides
        - path: spec.build.builder.kind
          value: clusterbuilder
        - path: spec.build.builder.name
          value: default
        - path: spec.build.cache.enabled
```

```
      value: false
    - path: spec.build.cache.image
      value: ""
    - path: spec.build.serviceAccountName
      value: default

    # Platform Engineer provided carvel package component overrides
    - path: spec.carvel.caCertData
      value: ""
    - path: spec.carvel.iaasAuthEnabled
      value: false
    - path: spec.carvel.packageDomain
      value: "default.tap"
    - path: spec.carvel.serviceAccountName
      value: "default"
    - path: spec.carvel.valuesSecretName
      value: ""

    # Platform Engineer provided GitOps repo overrides
    - path: spec.gitOps.baseBranch
      value: main
    - path: spec.gitOps.subPath
      value: "YOUR-GITOPS-REPO-SUBPATH"
    - path: spec.gitOps.url
      value: "YOUR-GITOPS-REPO-URL"
```

Next Steps

Deploy the `SupplyChain` to your Tanzu Application Platform cluster. For instructions, see Install an authored Supply Chain.

Useful links

- Supply Chain API Reference
- Component Catalog

Install an authored Supply Chain

This topic tells you how to install a `SupplyChain` authored using the Tanzu Supply Chain CLI plug-in.

Caution

Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.

Prepare

- Ensure that you created a `SupplyChain` by following the tutorial Build your first SupplyChain.
- The `make install` command requires the `kapp` CLI to be installed on your computer. Install kapp CLI.

Install

To install an authored Supply Chain:

1. Create a namespace where you want to install the `SupplyChain`:

```
kubectl create namespace mysupplychains
```

2. The `Makefile` was generated in the Build your first Supply Chain tutorial. Use the `Makefile` to install the `SupplyChain` with required `Components` and Tekton resources, such as `Pipelines` and `Tasks` to execute the logic of the components. Run:

```
NAMESPACE=mysupplychains make install

...

Changes

Namespace       Name                        Kind       Age  Op      Op st.  Wait
to    Rs  Ri
mysupplychains  app-config-server           Pipeline   -    create  -       reco
ncile  -   -
^               app-config-server-1.0.0     Component  -    create  -       reco
ncile  -   -
^               buildpack-build             Pipeline   -    create  -       reco
ncile  -   -
^               buildpack-build-1.0.0       Component  -    create  -       reco
ncile  -   -
^               calculate-digest            Task       -    create  -       reco
ncile  -   -
^               carvel-package              Pipeline   -    create  -       reco
ncile  -   -
^               carvel-package              Task       -    create  -       reco
ncile  -   -
^               carvel-package-1.0.0        Component  -    create  -       reco
ncile  -   -
^               carvel-package-git-clone    Task       -    create  -       reco
ncile  -   -
^               check-builders              Task       -    create  -       reco
ncile  -   -
^               conventions                 Pipeline   -    create  -       reco
ncile  -   -
^               conventions-1.0.0           Component  -    create  -       reco
ncile  -   -
^               fetch-tgz-content-oci       Task       -    create  -       reco
ncile  -   -
^               git-writer                  Pipeline   -    create  -       reco
ncile  -   -
^               git-writer                  Task       -    create  -       reco
ncile  -   -
^               git-writer-pr-1.0.0         Component  -    create  -       reco
ncile  -   -
^               gitops-git-clone            Task       -    create  -       reco
ncile  -   -
^               prepare-build               Task       -    create  -       reco
ncile  -   -
^               source-git-check            Task       -    create  -       reco
ncile  -   -
^               source-git-clone            Task       -    create  -       reco
ncile  -   -
^               source-git-provider         Pipeline   -    create  -       reco
ncile  -   -
^               source-git-provider-1.0.0   Component  -    create  -       reco
ncile  -   -
^               store-content-oci           Task       -    create  -       reco
ncile  -   -

Op:      23 create, 0 delete, 0 update, 0 noop, 0 exists
Wait to: 23 reconcile, 0 delete, 0 noop
```

```
1:47:20PM: ---- applying 23 changes [0/23 done] ----
1:47:20PM: create task/store-content-oci (tekton.dev/v1) namespace: mysupplycha
ins
1:47:20PM: create pipeline/git-writer (tekton.dev/v1) namespace: mysupplychains
1:47:20PM: create pipeline/app-config-server (tekton.dev/v1) namespace: mysuppl
ychains
1:47:20PM: create task/carvel-package (tekton.dev/v1) namespace: mysupplychains
1:47:20PM: create task/gitops-git-clone (tekton.dev/v1) namespace: mysupplychai
ns
1:47:20PM: create task/calculate-digest (tekton.dev/v1) namespace: mysupplychai
ns
1:47:20PM: create pipeline/source-git-provider (tekton.dev/v1) namespace: mysup
plychains
1:47:20PM: create pipeline/carvel-package (tekton.dev/v1) namespace: mysupplych
ains
1:47:20PM: create pipeline/buildpack-build (tekton.dev/v1) namespace: mysupplyc
hains
1:47:20PM: create pipeline/conventions (tekton.dev/v1) namespace: mysupplychain
s
1:47:21PM: create component/app-config-server-1.0.0 (supply-chain.apps.tanzu.vm
ware.com/v1alpha1) namespace: mysupplychains
1:47:21PM: create component/carvel-package-1.0.0 (supply-chain.apps.tanzu.vmwar
e.com/v1alpha1) namespace: mysupplychains
1:47:21PM: create component/buildpack-build-1.0.0 (supply-chain.apps.tanzu.vmwa
re.com/v1alpha1) namespace: mysupplychains
1:47:21PM: create component/git-writer-pr-1.0.0 (supply-chain.apps.tanzu.vmwar
e.com/v1alpha1) namespace: mysupplychains
1:47:21PM: create task/fetch-tgz-content-oci (tekton.dev/v1) namespace: mysuppl
ychains
1:47:21PM: create task/check-builders (tekton.dev/v1) namespace: mysupplychains
1:47:21PM: create task/carvel-package-git-clone (tekton.dev/v1) namespace: mysu
pplychains
1:47:21PM: create component/conventions-1.0.0 (supply-chain.apps.tanzu.vmware.c
om/v1alpha1) namespace: mysupplychains
1:47:22PM: create task/source-git-check (tekton.dev/v1) namespace: mysupplychai
ns
1:47:22PM: create task/prepare-build (tekton.dev/v1) namespace: mysupplychains
1:47:22PM: create task/git-writer (tekton.dev/v1) namespace: mysupplychains
1:47:22PM: create task/source-git-clone (tekton.dev/v1) namespace: mysupplychai
ns
1:47:22PM: create component/source-git-provider-1.0.0 (supply-chain.apps.tanzu.
vmware.com/v1alpha1) namespace: mysupplychains
1:47:22PM: ---- waiting on 23 changes [0/23 done] ----
1:47:22PM: ok: reconcile component/source-git-provider-1.0.0 (supply-chain.app
s.tanzu.vmware.com/v1alpha1) namespace: mysupplychains
1:47:22PM: ok: reconcile component/app-config-server-1.0.0 (supply-chain.apps.t
anzu.vmware.com/v1alpha1) namespace: mysupplychains
1:47:22PM: ok: reconcile task/prepare-build (tekton.dev/v1) namespace: mysupply
chains
1:47:22PM: ok: reconcile task/gitops-git-clone (tekton.dev/v1) namespace: mysup
plychains
1:47:22PM: ok: reconcile task/carvel-package-git-clone (tekton.dev/v1) namespac
e: mysupplychains
1:47:23PM: ok: reconcile task/store-content-oci (tekton.dev/v1) namespace: mysu
pplychains
1:47:23PM: ok: reconcile pipeline/git-writer (tekton.dev/v1) namespace: mysuppl
ychains
1:47:23PM: ok: reconcile pipeline/carvel-package (tekton.dev/v1) namespace: mys
upplychains
1:47:23PM: ok: reconcile task/git-writer (tekton.dev/v1) namespace: mysupplycha
ins
1:47:23PM: ok: reconcile pipeline/app-config-server (tekton.dev/v1) namespace:
mysupplychains
1:47:23PM: ok: reconcile task/calculate-digest (tekton.dev/v1) namespace: mysup
plychains
```

```
1:47:23PM: ok: reconcile pipeline/source-git-provider (tekton.dev/v1) namespac
e: mysupplychains
1:47:23PM: ok: reconcile pipeline/buildpack-build (tekton.dev/v1) namespace: my
supplychains
1:47:23PM: ok: reconcile pipeline/conventions (tekton.dev/v1) namespace: mysupp
lychains
1:47:23PM: ok: reconcile component/git-writer-pr-1.0.0 (supply-chain.apps.tanz
u.vmware.com/v1alpha1) namespace: mysupplychains
1:47:23PM: ok: reconcile component/carvel-package-1.0.0 (supply-chain.apps.tanz
u.vmware.com/v1alpha1) namespace: mysupplychains
1:47:23PM: ok: reconcile component/buildpack-build-1.0.0 (supply-chain.apps.tan
zu.vmware.com/v1alpha1) namespace: mysupplychains
1:47:23PM: ok: reconcile task/source-git-check (tekton.dev/v1) namespace: mysup
plychains
1:47:23PM: ok: reconcile component/conventions-1.0.0 (supply-chain.apps.tanzu.v
mware.com/v1alpha1) namespace: mysupplychains
1:47:23PM: ok: reconcile task/source-git-clone (tekton.dev/v1) namespace: mysup
plychains
1:47:23PM: ok: reconcile task/fetch-tgz-content-oci (tekton.dev/v1) namespace:
mysupplychains
1:47:23PM: ok: reconcile task/carvel-package (tekton.dev/v1) namespace: mysuppl
ychains
1:47:23PM: ok: reconcile task/check-builders (tekton.dev/v1) namespace: mysuppl
ychains
1:47:23PM: ---- applying complete [23/23 done] ----
1:47:23PM: ---- waiting complete [23/23 done] ----

Succeeded

...

Changes

Namespace       Name        Kind          Age  Op      Op st.  Wait to    Rs  Ri
mysupplychains  appbuildv1  SupplyChain   -    create  -       reconcile  -   -

Op:      1 create, 0 delete, 0 update, 0 noop, 0 exists
Wait to: 1 reconcile, 0 delete, 0 noop

1:47:24PM: ---- applying 1 changes [0/1 done] ----
1:47:25PM: create supplychain/appbuildv1 (supply-chain.apps.tanzu.vmware.com/v1
alpha1) namespace: mysupplychains
1:47:25PM: ---- waiting on 1 changes [0/1 done] ----
1:47:25PM: ok: reconcile supplychain/appbuildv1 (supply-chain.apps.tanzu.vmwar
e.com/v1alpha1) namespace: mysupplychains
1:47:25PM: ---- applying complete [1/1 done] ----
1:47:25PM: ---- waiting complete [1/1 done] ----

Succeeded
```

Note

By default, the `make install` command installs in the default namespace in your `kubeconfig` file.

Useful links

- GitOps Managed Supplychains
- Supply Chain API Reference
- Component Catalog

Next Steps

Create your first `Component` to add to your `SupplyChain`. For instructions, see Build your first component.

Build your first component

This topic tells you how to use Tanzu Supply Chain to build a component.

Caution

Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.

Prepare

Build a `SupplyChain` resource if you do not already have one. For how to build one, see Build your first SupplyChain.

Get started

In Build your first Supply Chain, you built a `SupplyChain` resource that retrieves the source code from a Git repository, and builds and packages it as a Carvel package. The `SupplyChain` resource then initiates a pull request to transfer the Carvel package to a GitOps repository, which enables the installation of the built package on the Run clusters.

This topic tells you how to create a component that you can use in your `SupplyChain` resource for unit-testing your Maven apps. You will create a component called `maven-unit-tester`, which will run unit tests on the source pulled by the `source-git-provider` stage.

```
tanzu supplychain component get source-git-provider-1.0.0 --show-details
```

Example:

```
...
  Outputs
   git
    ├─ digest: $(resumptions.check-source.results.sha)
    ├─ type: git
    └─ url: $(resumptions.check-source.results.url)
   source
    ├─ digest: $(pipeline.results.digest)
    ├─ type: source
    └─ url: $(pipeline.results.url)
...
```

Create a Tekton `Pipeline`

The first step for creating a `Component` resource is to make a Tekton `Pipeline` resource that includes the logic the component uses. In this case, `maven-unit-tester`:

- Obtains the source code extracted by the `source-git-provider` component.
- Uses the source code as input.
- Runs the unit test command on the source repository.

Tekton `Pipeline`:

- Defines `parameters` that a developer can provide by using the workload.
- Creates a shared `workspace` to store intermediate files, such as the source code from the `source-git-provider` stage.
- Provides tasks that run the unit-testing logic.

To create a Tekton `Pipeline`:

1. Define `parameters` as shown in the following YAML:

```
---
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
    name: maven-unit-tester-1.0.0
spec:
    description: Test maven application using mvn test command
    params:
      - description: url of source registry image
        name: source-url
        type: string
      - description: sub directory in source to become root dir for build
        name: sub-path
        default: "."
        type: string
      - description: The test command to run in my task
        name: testcmd
        type: string
      - description: The test command to run in my task
        name: workload-name
        type: string
```

2. Define a Tekton workspace as shown in the following YAML:

```
    workspaces:
      - name: shared-data
```

3. Add two tasks to the `Pipeline` resource. Task 1 pulls the output from `source-git-provider` and puts it in the workspace `shared-data` that was defined in the `Pipeline` resource earlier. Task 2 runs the unit tests and records the output as a Tekton Result.

   See the following example YAML:

```
    tasks:
      # task 1 pulls the output from source-git-provider and puts it in a works
pace called shared-data
      - name: fetch-source
        params:
          - name: url
            value: $(params.source-url)
        taskRef:
          kind: Task
          name: fetch-tgz-content-oci
        workspaces:
          - name: store
            workspace: shared-data

      # task 2 runs the unit tests
      - name: test
        runAfter:
          - fetch-source
        params:
```

```
          - name: sub-path
            value: $(params.sub-path)
          - name: testcmd
            value: $(params.testcmd)
          - name: workload-name
            value: $(params.workload-name)
        taskSpec:
          params:
            - name: sub-path
            - name: testcmd
            - name: workload-name
          results:
            - description: Error message if things failed
              name: message
              type: string
          steps:
            - name: test
              image: maven
              env:
                - name: HOME
                  value: /tekton/home/
              script: |-
                #!/usr/bin/env sh
                set -eu
                ls -al $(workspaces.store.path)
                cd $(workspaces.store.path)/$(params.sub-path)

                set +e
                $(params.testcmd)
                testval=$?
                set -e

                if [ $testval -ne 0 ]; then
                  echo "Unit tests stage failed. Please run 'tanzu workload log
s $(params.workload-name)' to see the run logs for more details." | tee "$(resu
lts.message.path)"
                  exit $testval
                fi
          stepTemplate:
              computeResources: {}
              securityContext:
                  allowPrivilegeEscalation: false
                  capabilities:
                      drop:
                          - ALL
                  runAsNonRoot: true
                  runAsUser: 1001
                  seccompProfile:
                      type: RuntimeDefault
        workspaces:
          - name: store
            workspace: shared-data
```

4. Add the version of the pipeline in the name of the YAML file and the name of the pipeline. This enables you to create new versions of the pipelines without interfering with the old one.
5. Store the `Pipeline` resource in the `pipelines` directory created by the CLI as `maven-unit-tester-1.0.0.yaml`, as shown in the following YAML:

```
---
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
    name: maven-unit-tester-1.0.0
```

```
spec:
    description: Test maven application using mvn test command
    params:
      - description: url of source registry image
        name: source-url
        type: string
      - description: sub directory in source to become root dir for build
        name: sub-path
        default: "."
        type: string
      - description: The test command to run in my task
        name: testcmd
        type: string
      - description: The test command to run in my task
        name: workload-name
        type: string
    workspaces:
      - name: shared-data
    # My tasks starts here
    tasks:
      # task 1 pulls the output from source-git-provider and puts it in a works
pace called shared-data
      - name: fetch-source
        params:
          - name: url
            value: $(params.source-url)
        taskRef:
          kind: Task
          name: fetch-tgz-content-oci
        workspaces:
          - name: store
            workspace: shared-data

      # task 2 runs the unit tests
      - name: test
        runAfter:
          - fetch-source
        params:
          - name: sub-path
            value: $(params.sub-path)
          - name: testcmd
            value: $(params.testcmd)
          - name: workload-name
            value: $(params.workload-name)
        taskSpec:
          params:
            - name: sub-path
            - name: testcmd
            - name: workload-name
          results:
            - description: Error message if things failed
              name: message
              type: string
          steps:
            - name: test
              image: maven
              env:
                - name: HOME
                  value: /tekton/home/
              script: |-
                #!/usr/bin/env sh
                set -eu
                ls -al $(workspaces.store.path)
                cd $(workspaces.store.path)/$(params.sub-path)

                set +e
```

```
                $(params.testcmd)
                testval=$?
                set -e

                if [ $testval -ne 0 ]; then
                  echo "Unit tests stage failed. Please run 'tanzu workload log
s $(params.workload-name)' to see the run logs for more details." | tee "$(resu
lts.message.path)"
                  exit $testval
                fi
          stepTemplate:
              computeResources: {}
              securityContext:
                  allowPrivilegeEscalation: false
                  capabilities:
                      drop:
                          - ALL
                  runAsNonRoot: true
                  runAsUser: 1001
                  seccompProfile:
                      type: RuntimeDefault
        workspaces:
          - name: store
            workspace: shared-data
```

Create a `SupplyChain` `Component`

The section describes how to write a `Component` manifest that defines the following:

- Configuration for a `Component` that is exposed as part of the Developer API. This is the configuration that a platform engineer can override at a `SupplyChain` level or that a developer can provide by using the `Workload` if a platform engineer does not override it at the `SupplyChain` level.
- `Inputs`, which is a list of named `Outputs` from a previous stage in a `SupplyChain` resource that the current `Component` resource depends on.
- `Outputs`, which are named outputs from this component another `Component` resource can use as `Inputs`.
- The Pipeline Run Definition section, which defines the work performed by this component. `spec.pipelineRun` is used to create a Tekton `PipelineRun` and has many similarities.
- Define the `Component` resource configuration as shown in the following YAML:

```
---
apiVersion: supply-chain.apps.tanzu.vmware.com/v1alpha1
kind: Component
metadata:
    name: maven-unit-tester-1.0.0
spec:
    description: Execute a maven unit test command using the mvn CLI
    config:
        - path: spec.test.cmd
          schema:
            description: |
                Unit testing mvn command to run
            required:
                - cmd
            type: string
            default: "mvn test"
            example: "mvn test"
```

- Define `Component` inputs and outputs. In this case, the output of the `source-git-provider` component is used as an input for the `maven-unit-tester` component.

```
    inputs:
      - name: source
        type: source
```

- Define `Component PipelineRun` details. The `Pipeline` resource created earlier is referenced here. The values are passed from the `Workload` and `Inputs` to satisfy the parameters for the `Pipeline` resource. See the following example YAML:

```
    pipelineRun:
        params:
            - name: source-url
              value: $(inputs.source.url)
            - name: sub-path
              value: $(workload.spec.source.subPath)
            - name: testcmd
              value: $(workload.spec.test.cmd)
            - name: workload-name
              value: $(workload.metadata.name)
        pipelineRef:
            name: maven-unit-tester-1.0.0
        taskRunTemplate:
            podTemplate:
                securityContext:
                    fsGroup: 1000
                    runAsGroup: 1000
                    runAsUser: 1001
        workspaces:
            - name: shared-data
              volumeClaimTemplate:
                spec:
                    accessModes:
                        - ReadWriteOnce
                    resources:
                        requests:
                            storage: 1Gi
```

- Store the `Component` resource in the `components` directory created by the CLI as `maven-unit-tester-1.0.0.yaml`. See the following example YAML:

```
---
apiVersion: supply-chain.apps.tanzu.vmware.com/v1alpha1
kind: Component
metadata:
    name: maven-unit-tester-1.0.0
spec:
    description: Execute a maven unit test command using the mvn CLI
    inputs:
      - name: source
        type: source
    config:
        - path: spec.test.cmd
          schema:
            description: |
                Unit testing mvn command to run
            required:
                - cmd
            type: string
            default: "mvn test"
            example: "mvn test"
    pipelineRun:
        params:
```

```
            - name: source-url
              value: $(inputs.source.url)
            - name: sub-path
              value: $(workload.spec.source.subPath)
            - name: testcmd
              value: $(workload.spec.test.cmd)
            - name: workload-name
              value: $(workload.metadata.name)
        pipelineRef:
            name: maven-unit-tester-1.0.0
        taskRunTemplate:
            podTemplate:
                securityContext:
                    fsGroup: 1000
                    runAsGroup: 1000
                    runAsUser: 1001
        workspaces:
            - name: shared-data
              volumeClaimTemplate:
                spec:
                    accessModes:
                        - ReadWriteOnce
                    resources:
                        requests:
                            storage: 1Gi
```

Install the component

You can now use the `make install` target created by the `tanzu supplychain` CLI to install the new `Component` and `Pipeline` resources to see if they reconcile.

Install the component by running:

```
NAMESPACE=mysupplychains make install
```

This command also displays any errors that Tekton throws.

Next Steps

Add your new `Component` to a `SupplyChain`. For instructions, see Add stages to your Supply Chain.

Useful links

- Component API Reference
- Component Catalog

Add stages to your Supply Chain

This topic tells you how to add a stage to a `SupplyChain` resource.

Caution

Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.

Preparation

You must have:

- A `SupplyChain` resource. For how to build one, see Build your first SupplyChain.
- A component. For how to build one, see Build your first component.

Add a stage to an existing `SupplyChain`

This section describes how to add a stage to an existing `SupplyChain` resource if the `SupplyChain` resource has not been deployed to clusters used by developers.

Important

If the `SupplyChain` resource is in use by developers with existing running workloads, do not add a stage. Instead, create a new `SupplyChain` resource with an updated version in the name, such as `MavenAppBuildv2`.

Creating a new `SupplyChain` resource with every breaking Workload API change provides developers with a migration path and a chance to test their app on the new `SupplyChain` resource.

To add a stage:

1. Add a component to the `SupplyChain` resource by adding the following YAML snippet between `source-git-provider` and the `buildpack-build` stage in your `SupplyChain` resource:

   ```
       - componentRef:
           name: COMPONENT-NAME
         name: unit-testing
   ```

   Where `COMPONENT-NAME` is the component name. For example, `maven-unit-tester-1.0.0`.

2. Update the `SupplyChain` resource by running:

   ```
   NAMESPACE=mysupplychains make install
   ```

Useful links

- Component API Reference
- Component Catalog

How-to guides for platform engineers

This section contains Tanzu Supply Chain how-to guides for platform engineers.

Caution

Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.

In this section:

How-to guides: Installing Tanzu Supply Chain

- Installing Tanzu Supply Chain

How-to guides: `SupplyChain` authoring

- Construct a Supply Chain using the CLI

How-to guides: Deploying and managing `SupplyChain` resources

- GitOps Managed SupplyChains

How-to guides: Starter `SupplyChain` resources

- Create starter SupplyChains

Install the Tanzu Supply Chain CLI plug-ins

This topic tells you how to install the Tanzu Supply Chain, and Tanzu Workload CLI plug-ins.

Caution

Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.

To start working with Tanzu Supply Chain, there are two Tanzu CLI plug-ins that you must install:

1. The Tanzu Supply Chain CLI plug-in is used to author and manage SupplyChains and Components.
2. The Tanzu Workload CLI plug-in is used by developers to discover and use the Workloads that you authored with the Tanzu Supplychain CLI plug-in.

Important The Tanzu Supply Chain beta release does not include the installation of the Tanzu Workload and Supply Chain` CLI plug-ins as part of the plug-in group.

Prerequisites

Ensure that you installed or updated the core Tanzu CLI. For more information, see Install Tanzu CLI.

1. Install the Tanzu CLI plug-ins by running:

   ```
   tanzu plugin install supplychain
   tanzu plugin install workload
   ```

2. Verify that the plug-ins are installed correctly by running:

   ```
   tanzu supplychain version
   tanzu workload version
   ```

Uninstall Tanzu Supply Chain CLI plug-ins

Run:

```
tanzu plugin delete supplychain
tanzu plugin delete workload
```

Install Tanzu Supply Chain

This section tells you how to install Tanzu Supply Chain.

Caution

Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.

There are two ways to install Tanzu Supply Chain:

- Install with the 'authoring' profile (Recommended)
- Installing manually on a Full or Build profile

After you have installed Tanzu Supply Chain, you must configure your system.

- Post-install configuration

Install Tanzu Supply Chain with the authoring profile (recommended)

This topic describes the recommended method for installing Tanzu Supply Chain using the Authoring profile (beta). This profile installs all the required packages.

Caution

Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.

The Authoring profile has the following additional packages that the Iterate profile does not have:

Packages

- Tanzu Supply Chain packages
    - `supply-chain.apps.tanzu.vmware.com`
    - `supply-chain-catalog.apps.tanzu.vmware.com`
    - `managed-resource-controller.apps.tanzu.vmware.com`
- Catalog Component packages
    - `alm-catalog.component.apps.tanzu.vmware.com`
    - `buildpack-build.component.apps.tanzu.vmware.com`
    - `conventions.component.apps.tanzu.vmware.com`
    - `git-writer.component.apps.tanzu.vmware.com`
    - `source.component.apps.tanzu.vmware.com`

    - `trivy.app-scanning.component.apps.tanzu.vmware.com`
- App Scanning package
    - `app-scanning.apps.tanzu.vmware.com`

Install Tanzu Supply Chain

1. Update your `tap-values.yaml` file to contain:

    ```
    profile: authoring
    ```

2. Confirm that the required packages are installed and reconciled successfully by running:

    ```
    kubectl get pkgi -A
    ```

    Example output

    ```
    NAMESPACE     NAME                                PACKAGE NAME
    PACKAGE VERSION                        DESCRIPTION           AGE
    tap-install   alm-catalog-component               alm-catalog.component.apps.tan
    zu.vmware.com           0.1.4                                Reconcile succeed
    ed   15d
    ...
    tap-install   buildpack-build-component           buildpack-build.component.app
    s.tanzu.vmware.com       0.0.2                                Reconcile succee
    ded   15d
    ...
    tap-install   conventions-component               conventions.component.apps.tan
    zu.vmware.com           0.0.3                                Reconcile succeed
    ed   15d
    ...
    tap-install   git-writer-component                git-writer.component.apps.tanz
    u.vmware.com            0.1.3                                Reconcile succeed
    ed   15d
    ...
    tap-install   managed-resource-controller         managed-resource-controller.ap
    ps.tanzu.vmware.com     0.1.2                                Reconcile succeed
    ed   15d
    ...
    tap-install   namespace-provisioner               namespace-provisioner.apps.tan
    zu.vmware.com           0.6.2                                Reconcile succeed
    ed   15d
    ...
    tap-install   source-component                    source.component.apps.tanzu.vm
    ware.com                0.0.1                                Reconcile succeed
    ed   15d
    ...
    tap-install   supply-chain                        supply-chain.apps.tanzu.vmwar
    e.com                    0.1.16                               Reconcile succee
    ded   15d
    tap-install   supply-chain-catalog                supply-chain-catalog.apps.tanz
    u.vmware.com            0.1.1                                Reconcile succeed
    ed   15d
    ...
    tap-install   trivy-app-scanning-component        trivy.app-scanning.component.a
    pps.tanzu.vmware.com    0.0.1-alpha.build.40376886+b5f4e614   Reconcile succeed
    ed   15d
    ...
    ```

Note

> As the Authoring profile adds additional packages in addition to what is already installed with the Iterate profile, the `tap-values.yaml` file for both profiles can look the same except for the `profile` value.

Next step

Configure Tanzu Supply Chain.

Install Tanzu Supply Chain manually (not recommended)

This topic tells you how to install Tanzu Supply Chain manually. This installation method is not recommended.

> **Caution**
>
> Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.

To install Tanzu Supply Chain manually, you must install additional packages that are not bundled with existing installation profiles.

The following Supply Chain packages are required.

- `supply-chain.apps.tanzu.vmware.com`
- `supply-chain-catalog.apps.tanzu.vmware.com`
- `managed-resource-controller.apps.tanzu.vmware.com`

The following Component packages are required if you're authoring a supply chain:

- `source.component.apps.tanzu.vmware.com`
- `conventions.component.apps.tanzu.vmware.com`
- `buildpack-build.component.apps.tanzu.vmware.com`
- `alm-catalog.component.apps.tanzu.vmware.com`
- `git-writer.component.apps.tanzu.vmware.com`
- `trivy-scanning.component.apps.tanzu.vmware.com`

1. To install these packages, run the following script:

```
export SUPPLY_CHAIN_VERSION=$(kubectl get package -n tap-install -o=jsonpath
='{.items[?(@.spec.refName=="supply-chain.apps.tanzu.vmware.com")].spec.versio
n}')
echo $SUPPLY_CHAIN_VERSION

tanzu package install supply-chain \
-p supply-chain.apps.tanzu.vmware.com \
-v $SUPPLY_CHAIN_VERSION \
-n tap-install

export SUPPLY_CHAIN_CATALOG_VERSION=$(kubectl get package -n tap-install -o=jso
npath='{.items[?(@.spec.refName=="supply-chain-catalog.apps.tanzu.vmware.co
m")].spec.version}')
echo $SUPPLY_CHAIN_CATALOG_VERSION
```

```
tanzu package install supply-chain-catalog \
-p supply-chain-catalog.apps.tanzu.vmware.com \
-v $SUPPLY_CHAIN_CATALOG_VERSION \
-n tap-install

export MANAGED_RESOURCE_VERSION=$(kubectl get package -n tap-install -o=jsonpat
h='{.items[?(@.spec.refName=="managed-resource-controller.apps.tanzu.vmware.co
m")].spec.version}')
echo $MANAGED_RESOURCE_VERSION

tanzu package install managed-resource-controller \
-p managed-resource-controller.apps.tanzu.vmware.com \
-v $MANAGED_RESOURCE_VERSION \
-n tap-install

export SOURCE_COMPONENT_VERSION=$(kubectl get package -n tap-install -o=jsonpat
h='{.items[?(@.spec.refName=="source.component.apps.tanzu.vmware.com")].spec.ve
rsion}')
echo $SOURCE_COMPONENT_VERSION

tanzu package install source-component \
-p source.component.apps.tanzu.vmware.com \
-v $SOURCE_COMPONENT_VERSION \
-n tap-install

export CONVENTIONS_COMPONENT_VERSION=$(kubectl get package -n tap-install -o=js
onpath='{.items[?(@.spec.refName=="conventions.component.apps.tanzu.vmware.co
m")].spec.version}')
echo $CONVENTIONS_COMPONENT_VERSION

tanzu package install conventions-component \
-p conventions.component.apps.tanzu.vmware.com \
-v $CONVENTIONS_COMPONENT_VERSION \
-n tap-install

export BUILDPACK_COMPONENT_VERSION=$(kubectl get package -n tap-install -o=json
path='{.items[?(@.spec.refName=="buildpack-build.component.apps.tanzu.vmware.co
m")].spec.version}')
echo $BUILDPACK_COMPONENT_VERSION

tanzu package install buildpack-build-component \
-p buildpack-build.component.apps.tanzu.vmware.com \
-v $BUILDPACK_COMPONENT_VERSION \
-n tap-install

export ALM_CATALOG_COMPONENT_VERSION=$(kubectl get package -n tap-install -o=js
onpath='{.items[?(@.spec.refName=="alm-catalog.component.apps.tanzu.vmware.co
m")].spec.version}')
echo $ALM_CATALOG_COMPONENT_VERSION

tanzu package install alm-catalog-component \
-p alm-catalog.component.apps.tanzu.vmware.com \
-v $ALM_CATALOG_COMPONENT_VERSION \
-n tap-install

export GIT_WRITER_COMPONENT_VERSION=$(kubectl get package -n tap-install -o=jso
npath='{.items[?(@.spec.refName=="git-writer.component.apps.tanzu.vmware.co
m")].spec.version}')
echo $GIT_WRITER_COMPONENT_VERSION

tanzu package install git-writer-component \
-p git-writer.component.apps.tanzu.vmware.com \
-v $GIT_WRITER_COMPONENT_VERSION \
-n tap-install
```

```
export TRIVY_SCANNING_COMPONENT_VERSION=$(kubectl get package -n tap-install -o
=jsonpath='{.items[?(@.spec.refName=="trivy-scanning.component.apps.tanzu.vmwar
e.com")].spec.version}')
echo $TRIVY_SCANNING_COMPONENT_VERSION

tanzu package install trivy-scanning-component \
-p trivy-scanning.component.apps.tanzu.vmware.com \
-v $TRIVY_SCANNING_COMPONENT_VERSION \
-n tap-install
```

2. Confirm that the required packages are installed and reconciled successfully by running:

```
kubectl get pkgi -A
```

Example output:

```
NAMESPACE     NAME                                PACKAGE NAME
PACKAGE VERSION                        DESCRIPTION           AGE
tap-install   alm-catalog-component               alm-catalog.component.apps.tan
zu.vmware.com           0.1.4                                Reconcile succeed
ed   15d
...
tap-install   buildpack-build-component           buildpack-build.component.app
s.tanzu.vmware.com       0.0.2                                Reconcile succee
ded   15d
...
tap-install   conventions-component               conventions.component.apps.tan
zu.vmware.com           0.0.3                                Reconcile succeed
ed   15d
...
tap-install   git-writer-component                git-writer.component.apps.tanz
u.vmware.com            0.1.3                                Reconcile succeed
ed   15d
...
tap-install   managed-resource-controller         managed-resource-controller.ap
ps.tanzu.vmware.com     0.1.2                                Reconcile succeed
ed   15d
...
tap-install   namespace-provisioner               namespace-provisioner.apps.tan
zu.vmware.com           0.6.2                                Reconcile succeed
ed   15d
...
tap-install   source-component                    source.component.apps.tanzu.vm
ware.com                0.0.1                                Reconcile succeed
ed   15d
...
tap-install   supply-chain                        supply-chain.apps.tanzu.vmwar
e.com                    0.1.16                               Reconcile succee
ded   15d
tap-install   supply-chain-catalog                supply-chain-catalog.apps.tanz
u.vmware.com            0.1.1                                Reconcile succeed
ed   15d
...
tap-install   trivy-app-scanning-component        trivy.app-scanning.component.a
pps.tanzu.vmware.com    0.0.1-alpha.build.40376886+b5f4e614   Reconcile succeed
ed   15d
...
```

Configure Tanzu Supply Chain

This topic tells you what to configure to complete your Tanzu Supply Chain installation.

Caution

> Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.

After you install Tanzu Supply Chain, use Namespace Provisioner to configure service accounts and permissions.

VMware recommended that you use Namespace Provisioner to configure the following:

OCI Store configuration: Supply Chains persist data between stages by reading and writing to an OCI repository. The location of the OCI repository is configured by a Kubernetes Secret named `oci-store` that exists within the developer namespace. Access to this repository is controlled by a Tekton annotated secret that can have any name with the `tekton.dev/docker-0` annotation pointing to the OCI repository.

Permissions for the `buildpack-build` component and Cluster Builders: You must add some additional permissions to use the `buildpack-build` component to create images with Tanzu Build Service configured with `ClusterBuilders.

Configure Tanzu Supply Chain using Namespace Provisioner

1. Create a `Secret` in the `tap-install` namespace that has the location and credentials for the `oci-store` as follows:

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: supply-chain-oci-store-credentials
  namespace: tap-install
type: Opaque
stringData:
  ocistore.yaml: |
    tanzusupplychain:
      ocistore:
        username: REGISTRY-USERNAME
        password: REGISTRY-PASSWORD
        server: REGISTRY-SERVER
        repository: REGISTRY-REPO
EOF
```

2. Configure Namespace Provisioner to use the accelerator sample. This creates the required resources for configuring `oci-store` and `buildpack-build` permissions. Update the `namespace_provisioner` section of your `tap-values.yaml` file as follows:

```
namespace_provisioner:
  additional_sources:
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/tanzu-supply-chain
      url: https://github.com/vmware-tanzu/application-accelerator-samples.git
  import_data_values_secrets:
  - name: supply-chain-oci-store-credentials
    namespace: tap-install
    create_export: true
  default_parameters:
    supply_chain_service_account:
```

```
      secrets:
      - oci-store-credentials
```

Namespace Provisioner creates the required secrets and role bindings in your developer namespace.

Create Developer Namespaces

```
kubectl create namespace dev
kubectl label namespaces dev apps.tanzu.vmware.com/tap-ns=""
```

Configure Namespace Provisioner to support custom Supply Chains

If there is a requirement for injecting additional secrets to the Service Account such as `git-secret`, for all Developer namespaces managed by Namespace Provisioner, update the `namespace_provisioner` section of `tap-values.yaml` as follows:

```
namespace_provisioner:
  ...
  default_parameters:
    supply_chain_service_account:
      secrets:
      # Add secrets here
      - git-secret
```

- Single Namespace using annotation:

```
kubectl annotate ns DEVELOPER-NAMESPACE param.nsp.tap/delivery_service_account.secrets
='["git-secret"]'
```

How to Author a Supply Chain

This section contains guides for platform engineers about authoring SupplyChain resources.

Caution

Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.

In this section:

- Construct a Supply Chain using the CLI
- Configure a Supply Chain using the Tanzu CLI

Construct a Supply Chain using the Tanzu CLI

This topic tells you how to construct a SupplyChain resource by using the Tanzu Supply Chain CLI plug-in.

Caution

Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.

Prerequisites

1. Ensure that the Tanzu CLI, and Tanzu Supply Chain CLI plug-in are installed on your local machine.
2. Ensure that Tanzu Supply Chain packages and Catalog Component packages are installed on the Tanzu Application Platform cluster that you are using to author your first supply chain.
3. If you Install Tanzu Supply Chain with the authoring profile (recommended), these packages are automatically installed.
4. If you Install Tanzu Supply Chain manually (not recommended), you must install the packages individually.

SupplyChain authoring

Deploy SupplyChain authoring resources such as `SupplyChain`, `Component`, and the Tekton `Pipeline`/`Task` to clusters through a Git repository by using the GitOps source promotion methodology.

For a Platform Engineer, the recommended approach is:

- Author the `SupplyChain` by using a set of YAML files within a Git-backed file system.
- Test and debug by pushing all files to a single namespace on the `authoring` profile cluster.
- When you are satisfied with the new or modified `SupplyChain`, use a pull request to commit to Git.

Managing a potentially large number of YAML manifests manually can be error-prone. Platform Engineers can use the Tanzu Supplychain CLI plug-in to streamline the authoring process for `SupplyChains` tailored to their developers.

Initialize the local directory

First use the `tanzu supplychain init` command to initialize the local directory for the `tanzu supplychain generate` command. Run:

```
tanzu supplychain init --group supplychains.tanzu.vmware.com --description "MY-SUPPLYC
HAIN-GROUP"
```

Where

- `--group`: (Optional) Group of the supplychains. The default is `supplychains.tanzu.vmware.com`. Used for auto-populating `spec.defines.group` of the SupplyChain API.
- `--description`: (Optional) Description of the Group. The default is "".

The `tanzu supplychain init` command creates:

- `config.yaml` file that contains the information about the group name, and the description of the Supplychain group.
- `supplychains`, `components`, `pipelines`, and `tasks` directories which are auto-populated by the authoring wizard later in this tutorial.

- `Makefile` which has the targets to install or uninstall the SupplyChain and related dependencies on any Build or Full profile clusters.
- `README.md` file which has instructions on how to use the targets in the `Makefile`.

Important

After being set up with the designated `group`, the local directory becomes a hub for shipping one or more `SupplyChains`. Within this local directory, every `SupplyChain` shares the same `group`, and this group information is stored in the `config.yaml` file. Conversely, in your GitOps repository, multiple directories can exist, each initialized with distinct groups such as `hr.supplychains.company.biz`, `finance.supplychains.company.biz`, and so on. Each of these directories is capable of accommodating multiple `SupplyChains` tailored to their respective groups.

Example output from `tanzu supplychain init` command:

```
Initializing group supplychains.tanzu.vmware.com
Creating directory structure
 ├─ supplychains/
 ├─ components/
 ├─ pipelines/
 ├─ tasks/
 ├─ Makefile
 ├─ README.md
 └─ config.yaml

Writing group configuration to config.yaml
```

Inspecting Components available to author Supply Chains

1. As a Platform Engineer, you want to know which components are available to use in your SupplyChain. Run:

   ```
   tanzu supplychain component list
   ```

   Example output

   ```
   Listing components from the catalog
     NAME                              INPUTS
   OUTPUTS                                                    AGE
     app-config-server-1.0.0           conventions[conventions]
   oci-yaml-files[oci-yaml-files], oci-ytt-files[oci-ytt-files]  14d
     app-config-web-1.0.0              conventions[conventions]
   oci-yaml-files[oci-yaml-files], oci-ytt-files[oci-ytt-files]  14d
     app-config-worker-1.0.0           conventions[conventions]
   oci-yaml-files[oci-yaml-files], oci-ytt-files[oci-ytt-files]  14d
     carvel-package-1.0.0              oci-yaml-files[oci-yaml-files], oci-ytt-file
   s[oci-ytt-files]  package[package]
   14d
     deployer-1.0.0                    package[package]
   <none>                                                     14d
     source-package-translator-1.0.0   source[source]
   package[package]                                           14d
     conventions-1.0.0                 image[image]
   conventions[conventions]                                   14d
     app-config-web-1.0.0              conventions[conventions]
   oci-yaml-files[oci-yaml-files], oci-ytt-files[oci-ytt-files]  14d
     git-writer-1.0.0                  package[package]
   <none>                                                     14d
   ```

```
  git-writer-pr-1.0.0              package[package]
git-pr[git-pr]                                          14d
  source-git-provider-1.0.0        <none>
source[source], git[git]                                14d
  buildpack-build-1.0.0            source[source], git[git]
image[image]                                            14d
  trivy-image-scan-1.0.0           image[image], git[git]
<none>                                                  14d

  To view the details of a component, use 'tanzu supplychain component get'
```

Use the `-w/--wide` flag to see a more detailed output including a description of each component.

Important

The `tanzu supplychain component list` command scans for `Component` custom resources labeled with `supply-chain.apps.tanzu.vmware.com/catalog`. Those `Component` custom resources possessing this label are the ones taken into account for authoring `SupplyChains` with the Tanzu CLI. Notably, the `Components` installed during the SupplyChain installation lack this label. This labeling distinction serves as the basis for differentiating between “Cataloged” and “Installed” `Components` in the CLI.

2. To get more information about each component on the cluster, run the `tanzu supplychain component get` command. For example, to get information about the `source-git-provider` component, run:

```
tanzu supplychain component get source-git-provider-1.0.0 -n source-provider --
show-details
```

Example output

```
  Overview
  name:         source-git-provider-1.0.0
  namespace:    source-provider
  age:          14d
  status:       True
  reason:       Ready
  description:  Monitors a git repository

  Configuration
  source:
    #! Use this object to retrieve source from a git repository.
    #! The tag, commit, and branch fields are mutually exclusive, use only one.
    #! Required
    git:
      #! A git branch ref to watch for new source
      branch: "main"
      #! A git commit sha to use
      commit: ""
      #! A git tag ref to watch for new source
      tag: "v1.0.0"
      #! The url to the git source repository
      #! Required
      url: "https://github.com/acme/my-workload.git"
    #! The sub path in the bundle to locate source code
    subPath: ""

  Outputs
```

```
  git
    ├─ digest: $(resumptions.check-source.results.sha)
    ├─ type: git
    └─ url: $(resumptions.check-source.results.url)
  source
    ├─ digest: $(pipeline.results.digest)
    ├─ type: source
    └─ url: $(pipeline.results.url)

  Pipeline
    ├─ name: source-git-provider
    └─   parameters
      ├─ git-url: $(workload.spec.source.git.url)
      ├─ sha: $(resumptions.check-source.results.sha)
      └─ workload-name: $(workload.metadata.name)

  Resumptions
  - check-source runs source-git-check task every 300s
      Parameters
      ├─ git-branch: $(workload.spec.source.git.branch)
      ├─ git-commit: $(workload.spec.source.git.commit)
      ├─ git-tag: $(workload.spec.source.git.tag)
      └─ git-url: $(workload.spec.source.git.url)

  To generate a supplychain using the available components, use 'tanzu supplych
ain generate'
```

Generate the SupplyChain

The Tanzu Supply Chain CLI plug-in supports two modes of operation for generating SupplyChains.

- Interactive: Use a guided wizard.
- Non-Interactive: Use flags.

Interactive

Start the wizard:

```
tanzu supplychain generate
```

The wizard prompts for the following:

Prompt	Example value
What Kind would you like to use as the developer interface?	`AppBuildV1` (This value is used for auto-populating the `spec.defines` section of the SupplyChain API)
Give Supply chain a description?	`Supply chain that pulls the source code from Git repository, builds it using buildpacks and package the output as Carvel package.`
Select a component as the first stage of the supply chain?	`source-git-provider-1.0.0`
Select a component as the first stage of the supply chain?	`buildpack-build-1.0.0`
Select a component as the first stage of the supply chain?	`conventions-1.0.0`
Select a component as the first stage of the supply chain?	`app-config-server-1.0.0`
Select a component as the first stage of the supply chain?	`carvel-package-1.0.0`

Prompt	Example value
Select a component as the first stage of the supply chain?	`git-writer-pr-1.0.0`
Select a component as the first stage of the supply chain?	`Done`

The Tanzu Supply Chain CLI knows what stages are already part of the SupplyChain and removes them from the list of stages to add.

Non-interactive

Generate the Supply chain by using flags:

```
tanzu supplychain generate \
--kind AppBuildV1 \
--description "Supply chain that pulls the source code from git repo, builds it usin
g buildpacks and package the output as Carvel package." \
--component "source-git-provider-1.0.0" \
--component "buildpack-build-1.0.0" \
--component "conventions-1.0.0" \
--component "app-config-server-1.0.0" \
--component "carvel-package-1.0.0" \
--component "git-writer-pr-1.0.0"
```

When you have selected the components, the Tanzu Supply Chain CLI plug-in creates the required files to deploy your SupplyChain in the current directory

For example:

```
✓ Successfully fetched all component dependencies
Created file supplychains/appbuildv1.yaml
Created file components/app-config-server-1.0.0.yaml
Created file components/buildpack-build-1.0.0.yaml
Created file components/carvel-package-1.0.0.yaml
Created file components/conventions-1.0.0.yaml
Created file components/git-writer-pr-1.0.0.yaml
Created file components/source-git-provider-1.0.0.yaml
Created file pipelines/app-config-server.yaml
Created file pipelines/buildpack-build.yaml
Created file pipelines/carvel-package.yaml
Created file pipelines/conventions.yaml
Created file pipelines/git-writer.yaml
Created file pipelines/source-git-provider.yaml
Created file tasks/calculate-digest.yaml
Created file tasks/carvel-package-git-clone.yaml
Created file tasks/carvel-package.yaml
Created file tasks/check-builders.yaml
Created file tasks/fetch-tgz-content-oci.yaml
Created file tasks/git-writer.yaml
Created file tasks/gitops-git-clone.yaml
Created file tasks/prepare-build.yaml
Created file tasks/source-git-check.yaml
Created file tasks/source-git-clone.yaml
Created file tasks/store-content-oci.yaml
```

Enforce proper ordering of Components in the SupplyChain

Components have zero or more inputs and outputs. The inputs for a component must be fulfilled by a preceding component in the SupplyChain. If not, there will be a component at a stage in a

SupplyChain that will not run. Proper ordering is handled differently depending on whether you are authoring using the interactive or non-interactive method.

Interactive
The entries that get populated for stage selection already take the ordering logic into account. The CLI only shows components for selection if the inputs for that component are already satisfied by another component in the SupplyChain.

Non-interactive
The SupplyChain returns an error if a component expects an input that has not been output by a previous stage. For example:

```
$ tanzu supplychain generate \
--kind AppBuildV1 \
--description "Supply chain that pulls the source code from Git repository, builds i
t using buildpacks and packages the output as Carvel package." \
--component "buildpack-build-1.0.0" \
--component "conventions-1.0.0" \
--component "app-config-server-1.0.0" \
--component "carvel-package-1.0.0" \
--component "git-writer-pr-1.0.0"
```

Example error output:

```
Error: unable to find the component buildpack-build-1.0.0 or the component buildpack
-build-1.0.0 does not match expected input value
```

For detailed information about the API specification for SupplyChain, see the SupplyChain API reference documentation.

Ensure that your Components and Supply Chains adhere to version constraints

For information about versioning SupplyChains and Components to avoid delivery failures of your SupplyChain resources to your Build clusters, see Supply Chains enforce immutability.

Reference Guides

Out of the Box Catalog of Components

- Catalog of Tanzu Supply Chain Components
- Output Types for Catalog Components

Reference Guides

- Overview of SupplyChains
- Overview of Components
- Overview of Resumptions
- Overview of Workloads
- Overview of WorkloadRuns

Configure a Supply Chain using the Tanzu CLI

This topic tells you how to construct a SupplyChain configuration.

Caution

Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.

Prerequisites

1. Ensure that the Tanzu CLI, and Tanzu Supply Chain CLI plug-in are installed on your local machine.
2. Ensure that Tanzu Supply Chain packages and Catalog Component packages are installed on the Tanzu Application Platform cluster that you are using to author your supply chain.

- If you Install Tanzu Supply Chain with the authoring profile (recommended), these packages are automatically installed.
- If you Install Tanzu Supply Chain manually (not recommended), you must install the packages individually.

SupplyChain configuration

SupplyChains can be configured to supply default and override values for each component. This allows a platform engineer to either pre-populate common default values for a component or override values to always be some value that the developer cannot modify.

Generate SupplyChain with overrides

Platform engineers generate SupplyChains with overrides to allow them to define values that cannot be changed by developers using the `Workload` (Developer API). By configuring overrides for each component in the SupplyChain, the generated `Workload` will not contain values that have been overriden.

Overrides consist of:

- `path`: The path to the configuration value, formatted as either:
    1. The full path to the field you want to set.
    2. The path to any structure where all desired child fields must be set.
- `value`: A string or YAML structured value.

Overrides use case

As a platform engineer, I want all built images to be accessible only through my organizations QA registry.

1. Generate the SupplyChain by supplying the `--allow-overrides` flag:

```
tanzu supplychain generate \
    --kind AppBuildV1 \
    --description "Supply chain that pulls the source code from git repo, build
s it using buildpacks and package the output as Carvel package." \
    --component "source-git-provider-1.0.0" \
    --component "buildpack-build-1.0.0" \
```

```
    --component "conventions-1.0.0" \
    --component "app-config-server-1.0.0" \
    --component "carvel-package-1.0.0" \
    --component "git-writer-pr-1.0.0" \
    --allow-overrides
```

The Tanzu Supply Chain CLI plug-in creates the required files to deploy your SupplyChain in the current directory:

```
✓ Successfully fetched all component dependencies
Created file supplychains/appbuildv1.yaml
...
```

2. To configure overrides, open `supplychains/appbuildv1.yaml` in your editor and navigate to the following section:

```
...
config:
  overrides:
      # Platform Engineer provided registry overrides
      - path: spec.registry.repository
        value: "YOUR-REGISTRY-REPO"
      - path: spec.registry.server
        value: "YOUR-REGISTRY-SERVER"

      # Platform Engineer provided build overrides
      - path: spec.build.builder.kind
        value: clusterbuilder
      - path: spec.build.builder.name
        value: default
      - path: spec.build.cache.enabled
        value: false
      - path: spec.build.cache.image
        value: ""
      - path: spec.build.serviceAccountName
        value: default

      # Platform Engineer provided carvel package component overrides
      - path: spec.carvel.caCertData
        value: ""
      - path: spec.carvel.iaasAuthEnabled
        value: false
      - path: spec.carvel.packageDomain
        value: "default.tap"
      - path: spec.carvel.serviceAccountName
        value: "default"
      - path: spec.carvel.valuesSecretName
        value: ""

      # Platform Engineer provided GitOps repo overrides
      - path: spec.gitOps.baseBranch
        value: main
      - path: spec.gitOps.branch
        value: main
      - path: spec.gitOps.subPath
        value: "YOUR-GITOPS-REPO-SUBPATH"
      - path: spec.gitOps.url
        value: "YOUR-GITOPS-REPO-URL"
```

3. Configure overrides using either a full path to the field you want to set or a path to any structure where all desired child fields must be set. For example:

Full path

Example path `spec.registry.repository`. As this example does not provide a value for `spec.registry.server`, it will not be available to modify in the `Workload`.

```
config:
  overrides:
    - path: spec.registry.repository
      value: "https://my-registry.url.com"
```

Path to any key representing a YAML object
Examples:

1. Path `spec.registry`:

   ```
   config:
     overrides:
       - path: spec.registry
         value:
           repository: "https://my-registry.url.com"
   ```

2. Path `spec`. In this example, there is no value for `spec.registry.server`, it will not be available to modify in the `Workload`.

   ```
   config:
     overrides:
       - path: spec
         value:
           registry:
             repository: "https://my-registry.url.com"
   ```

3. Path `spec` with empty value. This example results in a `Workload` without a spec.

   ```
   config:
     defaults:
       - path: spec
         value: {}
   ```

Generate SupplyChain with defaults

Platform engineers generate SupplyChains with defaults to allow them to define default values that can be changed by developers using the `Workload` (Developer API). By configuring defaults for each component in the SupplyChain, the generated `Workload` will contain default values.

Defaults consist of:

- `path`: path to the configuration value, formatted as either:
  1. The full path to the field you want to set.
  2. The path to any structure where all desired child fields must be set.
- `value`: String or YAML structured value.

Defaults use case

1. Generate the SupplyChain by supplying the `--allow-defaults` flag:

   ```
   tanzu supplychain generate \
       --kind AppBuildV1 \
       --description "Supply chain that pulls the source code from git repo, build
   s it using buildpacks and package the output as Carvel package." --component "s
   ource-git-provider-1.0.0" --component "buildpack-build-1.0.0" --component "conv
   entions-1.0.0" \
   ```

```
    --component "app-config-server-1.0.0" \
    --component "carvel-package-1.0.0" \
    --component "git-writer-pr-1.0.0" \
    --allow-defaults
```

The Tanzu Supply Chain CLI plug-in creates the required files to deploy your SupplyChain in the current directory:

```
✓ Successfully fetched all component dependencies
Created file supplychains/appbuildv1.yaml
...
```

2. To configure defaults, open the `supplychains/appbuildv1.yaml` file in your editor and navigate to the following section:

```
  ...
  config:
    defaults:
        # Platform Engineer provided registry defaults
        - path: spec.registry.repository
        value: "YOUR-REGISTRY-REPO"
        - path: spec.registry.server
        value: "YOUR-REGISTRY-SERVER"

        # Platform Engineer provided build defaults
        - path: spec.build.builder.kind
          value: clusterbuilder
        - path: spec.build.builder.name
          value: default
        - path: spec.build.cache.enabled
          value: false
        - path: spec.build.cache.image
          value: ""
        - path: spec.build.serviceAccountName
          value: default

        # Platform Engineer provided carvel package component defaults
        - path: spec.carvel.caCertData
          value: ""
        - path: spec.carvel.iaasAuthEnabled
          value: false
        - path: spec.carvel.packageDomain
          value: "default.tap"
        - path: spec.carvel.serviceAccountName
          value: "default"
        - path: spec.carvel.valuesSecretName
          value: ""

        # Platform Engineer provided GitOps repo defaults
        - path: spec.gitOps.baseBranch
          value: main
        - path: spec.gitOps.branch
          value: main
        - path: spec.gitOps.subPath
          value: "YOUR-GITOPS-REPO-SUBPATH"
        - path: spec.gitOps.url
          value: "YOUR-GITOPS-REPO-URL"
```

3. Configure defaults using either a full path to the field you want to set or a path to any structure where all desired child fields must be set.

Full path
Example path `spec.registry.repository`:

```
config:
  defaults:
    - path: spec.registry.repository
      value: "https://my-default-registry.url.com"
```

Path to any key representing a YAML object

Examples

1. Path `spec.registry`:

   ```
   config:
     defaults:
       - path: spec.registry
         value:
           repository: "https://my-default-registry.url.com"
   ```

2. Path `spec`:

   ```
   config:
     defaults:
       - path: spec
         value:
           registry:
             repository: "https://my-default-registry.url.com"
   ```

Reference Guides

SupplyChain API

- SupplyChain API

Reference Guides

- Overview of SupplyChains
- Overview of Components
- Overview of Resumptions
- Overview of Workloads
- Overview of WorkloadRuns

Deploy and Manage Supply Chains

This section contains guides for platform engineers about deploying and managing SupplyChain resources.

Caution

Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.

In this section:

- GitOps managed SupplyChains

Manage Supply Chains with GitOps

This topic tells you how to manage SupplyChains using GitOps.

Caution

Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.

SupplyChains, especially the authoring resources SupplyChain, Component, and Tekton Pipeline/Tekton Task, are delivered to clusters using a Git repository and GitOps source promotion style.

The expected flow is as follows:

1. Author the SupplyChain as a collection of YAML files in a file system backed by Git.
2. Test and debug by pushing all the files to a single namespace.
3. When you're happy with your new or modified SupplyChain, commit it to Git and create a pull request.
4. Using continuous integration, test, and approve the pull request.
5. Using continuous deployment, deliver your edits to build clusters.

Note

Both the integration and deployment of your SupplyChains should be managed by SupplyChains. VMware will release examples of integration and delivery SupplyChains for SupplyChains in a future release.

Create starter Supply Chains

This topic gives you recipes for authoring useful, minimal Supply Chains to get started with.

Build and deploy an application recipe

This Supply Chain builds and deploys an application from source.

It performs the following actions:

- Pulls the application source from Git.
- Builds a container image using Buildpacks.
- Generates a runtime definition with the image.
- Generates a Kubernetes Deployment and Service.
- Generates a Carvel package to make the application deployable.
- Deploys the application.

Complete the following steps:

1. Ensure that you have first initialized a working directory by running tanzu supplychain init.
2. Generate the Supply Chain by running:

```
tanzu supplychain generate \
  --kind WebApp \
  --description "Build and deploy an application from Git" \
  --component source-git-provider-1.0.0 \
  --component buildpack-build-1.0.0 \
  --component conventions-1.0.0 \
  --component app-config-web-1.0.0 \
  --component carvel-package-1.0.0 \
  --component deployer-1.0.0
```

Note

To deploy other workload types, replace the `app-config-web-1.0.0` component with another option such as `app-config-server-1.0.0`, or `app-config-worker-1.0.0`.

Build an application and store the artifact in a Git recipe

This Supply Chain builds a Carvel package from the application source and stores it in a Git repository for deployment to a runtime environment.

It performs the following actions:

- Pulls the application source from Git.
- Builds a container image using Buildpacks.
- Generates a runtime definition with the image.
- Generates a Knative service.
- Generates a Carvel package to make the application deployable.
- Creates a pull request against a Git repository with the Carvel package contents.

Complete the following steps:

1. Ensure that you have initialized a working directory by running tanzu supplychain init.
2. Generate the Supply Chain by running:

```
tanzu supplychain generate \
  --kind CarvelPackage \
  --description "Build an application from source and store the Carvel package
in Git" \
  --component source-git-provider-1.0.0 \
  --component buildpack-build-1.0.0 \
  --component conventions-1.0.0 \
  --component app-config-web-1.0.0 \
  --component carvel-package-1.0.0 \
  --component git-writer-pr-1.0.0
```

Note

To write directly to a Git repository without creating a pull request, replace the `git-writer-pr-1.0.0` component with `git-writer-1.0.0`.

Deploy an application package from a Git recipe

This Supply Chain deploys a Carvel package from a Git repository.

It performs the following actions:

- Pulls the application package from Git.
- Translates the Carvel package to a deployable package.
- Deploys the application.

Complete the following steps:

1. Ensure that you have initialized a working directory by running tanzu supplychain init.
2. Generate the Supply Chain by running:

```
tanzu supplychain generate \
  --kind PackageDeploy \
  --description "Deploy a Carvel package from Git" \
  --component source-git-provider-1.0.0 \
  --component source-package-translator-1.0.0 \
  --component deployer-1.0.0
```

Supply Chain Choreographer

The recipes in this topic are analogous to out of the box supply chains, and profile experiences in Supply Chain Choreographer. Use this mapping to help decide which recipe to start with. These recipes do not provide exact parity with out of the box supply chains.

- Iterate Profile: Use Build and deploy an application recipe.
- Supply Chain Basic: Use Build an application and store the artifact in a Git recipe.
- Delivery Basic: Use Deploy an application package from a Git recipe.

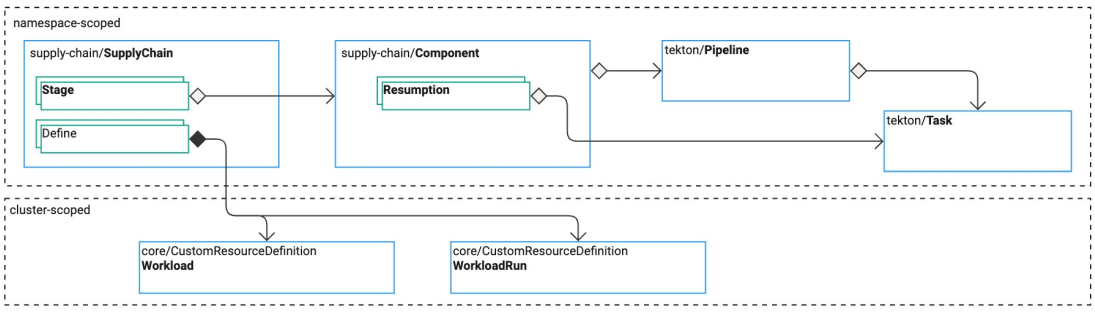
Explanations for platform engineers

This topic tells you about the core architectural concepts of Tanzu Supply Chain. This topic focuses on authoring a platform for application development.

Caution

Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.

Tanzu Supply Chain enables platform engineers to author a seamless experience for development work without expertise in Kubernetes.



About the primitives of Tanzu Supply Chain

Tanzu Supply Chain has the following primitives:

- `SupplyChain` defines the `Workload` kind and the components to use.
- `Workload` provides a developer API.
- `WorkloadRun` provides a record of each progression through a supply chain.
- `Component` provides an abstraction for a piece of work and reasons to trigger new runs.
- `Resumptions` provide a process for defining the triggering of new runs.

Overview of the Tanzu Supply Chain system

Each `SupplyChain` resource combines multiple `Component` resources in `stages`. `SupplyChain` resources define a process for converting configuration workloads into final artifacts. They are similar to pipelines in other CI/CD systems, but with some key differences:

- The `SupplyChain` defines a `Workload` custom resource definition (CRD), and the `Workload` becomes the interface that users, typically developers, consume to have work performed.
- Users are typically unaware of the inner workings of the `SupplyChain` and `Component` resources.
- `Component` resources encapsulate the work of generating a final or intermediate artifact and, importantly, the work of discovering new work to be performed.

The flow of operations is as follows:

1. Apply a valid `SupplyChain`, `Components`, and accompanying Tekton `Tasks` and `Pipelines`.
2. Tanzu Supply Chain generates a `Workload` CRD as defined by the `SupplyChain`.
3. A developer uses the Tanzu CLI to see the `Workload` kinds available to them.
4. The developer generates a `Workload` and fills in the required configuration.
5. Tanzu Supply Chain applies the `Workload`.
6. Tanzu Supply Chain starts the first `WorkloadRun`.
7. Tanzu Supply Chain captures the progress and artifacts of the first `WorkloadRun`.
8. Tanzu Supply Chain monitors `Component` resources that have `Resumptions` to detect if it must generate new runs. Tanzu Supply Chain generates new runs if:
    - There is new configuration in the `Workload`
    - There is new source code, base images, or other triggers from `Resumptions`
9. The developer observes the progress of runs using the Tanzu CLI.

Managing SupplyChains with GitOps

`SupplyChain` resources, such as the authoring resources `SupplyChain`, `Component`, Tekton `Pipeline`, and Tekton `Task`, are delivered to clusters through a Git repository and GitOps source promotion style.

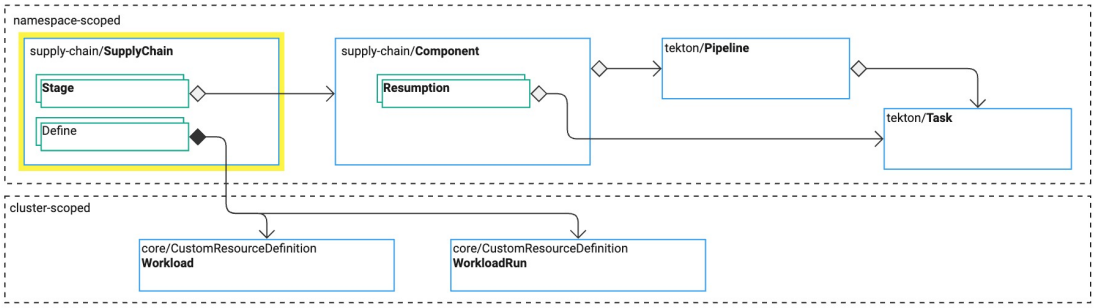
For more information, see Manage SupplyChains with GitOps.

Overview of SupplyChain

This topic tells you about the `SupplyChain` primitive in Tanzu Supply Chain. The `SupplyChain` primitive unifies the Tanzu Supply Chain operation. For reference information, see SupplyChain.

Caution

Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.



`SupplyChain` describes a process with stages

In physical manufacturing a supply chain is the process that delivers an end product to customers, starting with the raw materials. In software, a supply chain delivers an operational end product to customers, starting with source code. VMware refers to this as the golden path to production.

Tanzu Supply Chain provides a primitive called `SupplyChain`, which is a Kubernetes custom resource that you use to define all, or portions of, your software supply chain.

This section describes typical uses of the `SupplyChain` primitive for Tanzu Supply Chain.

`SupplyChain` describes a build process

A `SupplyChain` primitive can describe the process of converting source code into a runnable or deployable package.

Typical stages in this process are:

- Build:
  - Compile a binary from source.
  - Create an OCI image from the binary.
- Configure:
  - Create deployment artifacts, such as Kubernetes Pod definitions.
- Package:
  - Create packaging artifacts, such as a Carvel package or a Helm Chart.

`SupplyChain` defines a configuration resource

A `SupplyChain` primitive brings together the API for a user to apply to the cluster by:

- Defining a group and kind for a resource called a `Workload`. For reference information, see spec.defines.
- Specifying components used in the stages of the `SupplyChain`. For reference information, see spec.stages[].

By selecting components, `SupplyChain` aggregates each configuration for each component as a single API specification for the `Workload`.

Note

`Workload` might be renamed in a later Tanzu Application Platform version.

`SupplyChain` enforces immutability

The version of your `SupplyChain` primitive that is embedded in the name must adhere to the rules described in this section.

A patch update is required to update `SupplyChain` without an API change. The controller ensures taht this rule cannot be broken when comparing `SupplyChain` primitives on the cluster.

For example, you can apply to a cluster:

- A `SupplyChain` primitive with the name `serverappv1s.example.com-1.0.0` and the kind `ServerAppV1s`
- A `SupplyChain` primitive with the name `serverappv1s.example.com-1.0.1` and the kind `ServerAppV1s`

If the generated API for the kind is unchanged, then the later version is accepted. If there is a change, the `SupplyChain` primitive that was applied first succeeds, and the others reflect the error in their statuses. This rule ensures that you cannot accidentally break the kind API that is running.

These rules ensure that potentially thousands of `Workload` and `Run` resources on the cluster do not break.

Recommended version practices:

- If the API and general behavior are unchanged by a change to the `spec.stages`:
  - Use a patch update, such as `1.2.5` to `1.2.6`
  - Keep the same kind, such as `ServerAppV1`
- If the API is unchanged, but something significantly different occurs because of changes to the `spec.stages`, consider doing:
  - An update to the minor or major version, such as `1.2.5` to `1.3.0`
  - An update to the kind, such as `ServerAppV2`
  - A change of kind, such as `ServerAppWithApprovalV1`
- If the API changes, consider doing:
  - An update to the minor or major version, such as `1.2.5` to `1.3.0`
  - An update to the kind, such as `ServerAppV2`

This ensures effective communication to your users. New kind versions typically indicate that the user must migrate their resources to the new API.

Integrity validation

A `SupplyChain` primitive is not valid if:

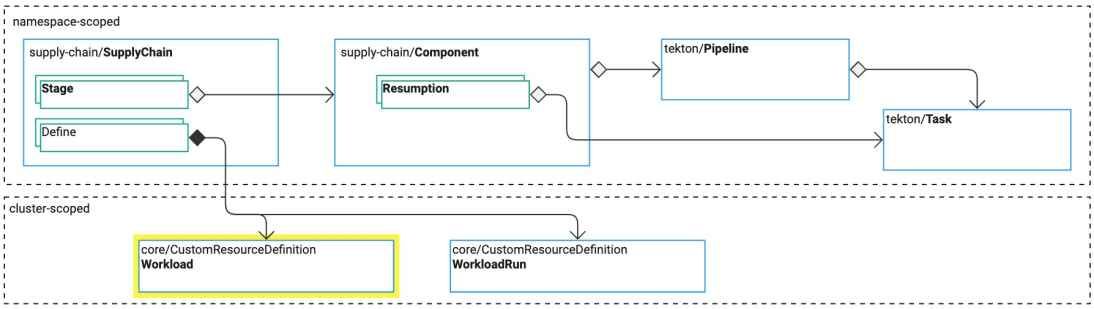
- A required field is missing.
- The `Component` resources referenced are not in the same namespace.
- The `Component` resources referenced contain values that are not satisfied by their position in `spec.stages`.
- The name does not match the `spec.defines` section.
- `SupplyChain` breaks the versioning rules.

For more information, see status.conditions[].

Overview of Workloads

This topic tells you about the `Workload` resource in Tanzu Supply Chain. For reference information, see Workload CRD.

> **Caution**
>
> Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.

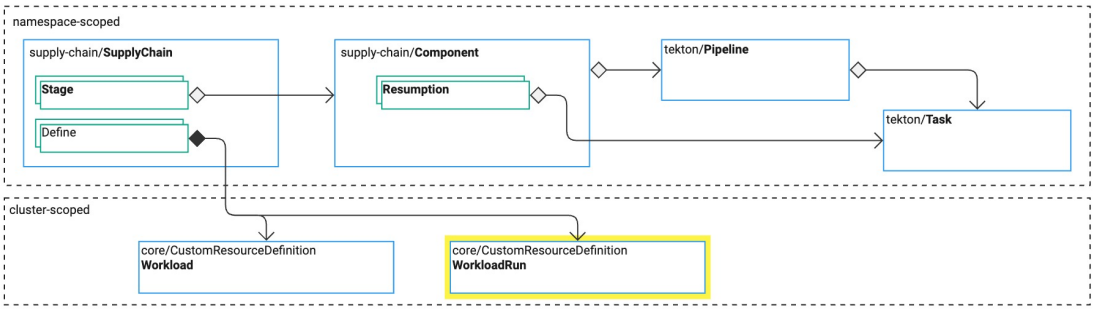


The `SupplyChain` resource defines the `Workload` custom resource definition (CRD).

Overview of WorkloadRun

This topic tells you about the `WorkloadRun` resource in Tanzu Supply Chain. For reference information, see WorkloadRun CRD.

> **Caution**
>
> Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.



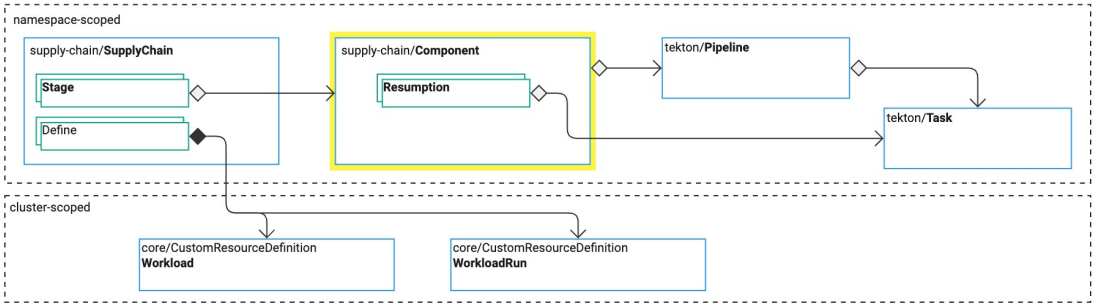
The `SupplyChain` resource defines the `WorkloadRun` custom resource definition (CRD).

Overview of Components

This topic tells you about the `Component` resource in Tanzu Supply Chain. For reference information, see Component API.

Caution

Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.



Components encapsulate the work to be performed in composable and reusable pieces. Components are analogous to steps, stages, jobs, and tasks in other CI/CD offerings. Components are different to other CI/CD offerings in three distinct ways:

- Component configuration requirements are declared, static, and enforced. The configuration is used to build a `Workload` resource that is strongly-typed and well-documented.
- Components are observed by users through a strict error-reporting API.
- Components can exhibit reentrant behavior. If you define a resumption for a component, the resumption can trigger new workload runs. This keeps the stored information about the component in transient dependencies within the component. For more information, see Resumptions.

These design constraints exist to:

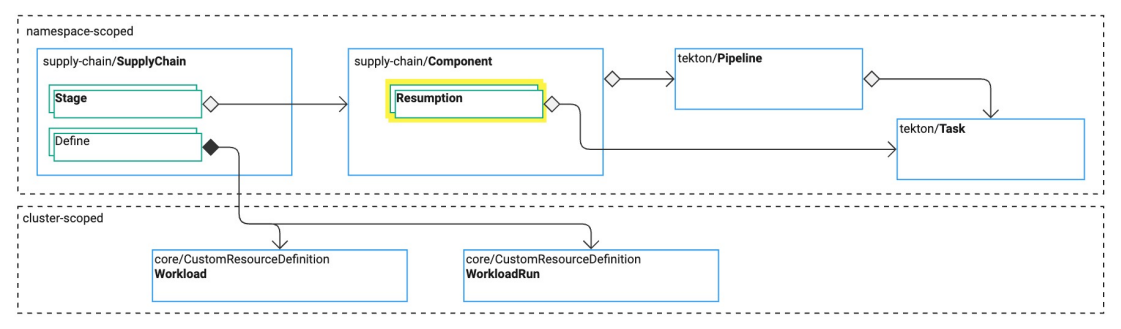
- Simplify the end-user experience by providing a single, well-defined API and a way of mitigating errors if they arise.
- Simplify the authoring experience by requiring only minimal Kubernetes experience to construct supply chains that meet your organization's needs.

Overview of Resumptions

This topic tells you about the concept of resumptions in Tanzu Supply Chain. For reference information about resumptions, see Component API.

Caution

Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.



Resumptions are an important part of the Tanzu Supply Chain `Component` resource.

The `Component` resource deals with the configuration to pass to resumptions and pipelines, or the pipeline to run when the `Component` stage is reached in the supply chain. Resumptions are focused on the reasons to run again.

Resumptions are executed on a timer that is specified in the `resumptions` array of the component. When resumptions trigger they execute a Tekton `TaskRun` to detect new values. These are common for detecting changes to source repositories, image repositories, and new versions of binaries.

If a run reaches a stage with a resumption, the system generates a `resumptionKey` based on the value of the parameters that are passed to the resumption. If a resumption has already been executed with the same parameters, the result of that resumption is used. This allows hundreds or thousands of workloads to reuse the same common inputs from resumptions without needing to run another resumption.

After the last completion of a resumption `TaskRun`, resumptions wait for the period specified in `resumptions[].trigger.runAfter` before running it again.

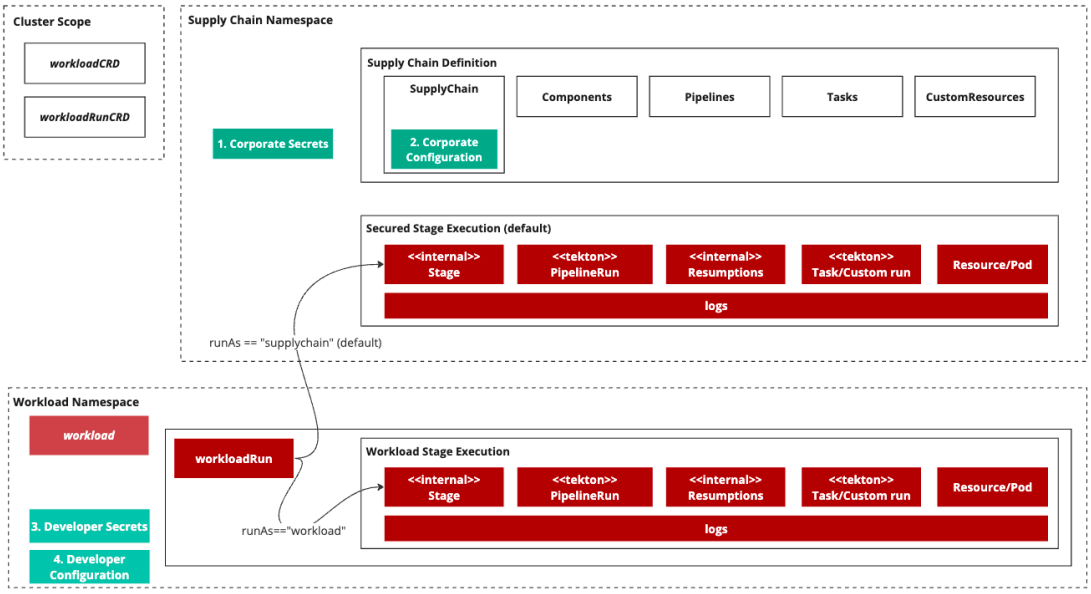
When a result for a resumption changes, all the `WorkloadRun` resources with the same `resumptionKey` are cloned, truncated back to the stage of the resumption, and progress starts from there. This is the mechanism for resumptions triggering a new `WorkloadRun`.

Security Model

This topic tells you about the security model for Tanzu Supply Chain. This security model is for executing stages of a Supply Chain where workloads exist in a separate namespace to the Supply Chain.

Caution

Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.



Runs for associated workloads are created in the same namespace as the workload.

`Stages`, `Resumptions`, `TaskRuns`, and `PipelineRuns` are created by default in the Supply Chain namespace. This gives the components in these stages visibility over platform secrets in the Supply Chain namespace.

If you want a stage of a pipeline to execute in the workload namespace, use the `securityContext.runAs` setting. For example, you can allow a source component to retrieve the source from a developer-controlled repository by using secrets that the developer provides in the workload namespace.

For more information, see the `securityContext` specification in the `SupplyChain` API topic.

Tanzu Supply Chain for Developers

This section contains the developer-focused features of Tanzu Supply Chain.

> **Caution**
>
> Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.

Prerequisites

Start using Tanzu Supply Chain by installing these CLI tools:

- Install the Tanzu CLI
- Install the Tanzu Workload CLI plug-in

Getting Started with Tutorials

- Deploy your first Workload

How to Guides

- Work with Workloads

Tutorials for Developers

This section contains tutorials for developers about how to use Tanzu Supply Chain.

Caution

Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.

In this section:

- Deploy your first workload

Tutorial: Deploy your first workload

This topic tells you how to use the Tanzu Workload CLI plug-in to create your first `Workload`.

Caution

Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.

In this tutorial, the platform engineer has already created some Supply Chains for you to use. The Supply Chains can pull the source code from the source repository and build it. The built artifact is then pushed to a GitOps repository that the platform engineer selected.

Prepare

Install:

- Tanzu CLI
- Tanzu Workload CLI plug-in

Deploy a workload

To get started:

1. See which `SupplyChain` resources are available to you, and which kinds of `Workload` resources you can create by using those `SupplyChain` resources, by running:

   ```
   tanzu workload kind list
   ```

   Example output:

   ```
   $ tanzu workload kind list

    KIND                                        VERSION   DESCRIPTION
    appbuildv1s.supplychains.tanzu.vmware.com   v1alpha1  Supply chain that pulls t
   he source code from git repo, builds it using
                                                         buildpacks and package th
   e output as Carvel package.
   ```

```
  To generate a workload for one of these kinds, use 'tanzu workload generate'
```

The command output shows that you have a `appbuildv1s.supplychains.tanzu.vmware.com` kind that you can use to generate the workload. The Tanzu CLI also shows a hint about the next command to use in the process.

2. For this tutorial, build your workload by using the tanzu-java-web-app sample application.

   Use the `tanzu workload generate` command to create a `Workload` of the kind `AppBuildV1`. A selector is displayed if multiple kinds are available. If a single kind is available, it is used to generate the scaffold of the `Workload`. Run:

   ```
   tanzu workload generate tanzu-java-web-app
   ```

   Example output:

   ```
   apiVersion: supplychains.tanzu.vmware.com/v1alpha1
   kind: AppBuildV1
   metadata:
    name: tanzu-java-web-app
   spec:
    #! Configuration for the generated Carvel Package
    carvel:
      ...
    gitOps:
      ...
    #! Configuration for the registry to use
    registry:
      ...
    source:
      ...
    #! Kpack build specification
    build:
      ...
   ```

3. Pipe the output into a `workload.yaml` file. If you have more than one kind available in the cluster, you must provide a `--kind` flag. The `--kind` flag supports tab auto-completion. Run:

   ```
   tanzu workload generate tanzu-java-web-app --kind appbuildv1s.supplychains.tanz
   u.vmware.com > workload.yaml
   ```

4. Add the appropriate values in the `workload.yaml` file for each required entry. For example:

   ```
   apiVersion: supplychains.tanzu.vmware.com/v1alpha1
   kind: AppBuildV1
   metadata:
     name: tanzu-java-web-app
   spec:
     gitOps:
       baseBranch: "main"
       subPath: "packages"
       url: GITOPS-REPO-PATH
     registry:
       repository: REPOSITORY-PATH
       server: REGISTRY-SERVER
     source:
       git:
         branch: "main"
         url: "https://github.com/vmware-tanzu/application-accelerator-samples.gi
   t"
       subPath: "tanzu-java-web-app"
     carvel:
   ```

```
    packageName: "tanzu-java-web-app"
    packageDomain: "tanzu.vmware.com"
```


Caution

The beta version of the Tanzu Supply Chain does not support platform engineer-level overrides and defaults. Therefore, the `Workload` generate command also shows the entries that a platform engineer is supposed to set, such as the registry details.

When the overrides feature is available, a platform engineer can set platform-level values, such as the registry details. These entries are not part of the `generate` command output because that is something a platform engineer does not want a developer to override.

This results in a much smaller `Workload` spec and one that only has values that a developer can provide for the `SupplyChain`. This helps to keep the platform engineering role and the developer role separate.

5. Use the `tanzu workload apply` command to apply your `AppBuildV1` workload to the cluster. If your workload YAML file is not named `workload.yaml`, use the `-f` flag to point to it. For this tutorial, use the `dev` namespace. Run:

```
tanzu workload apply
```

Example output:

```
  Creating workload:
     1 + |---
     2 + |apiVersion: supplychains.tanzu.vmware.com/v1alpha1
     3 + |kind: AppBuildV1
     4 + |metadata:
     5 + |  name: tanzu-java-web-app
     6 + |  namespace: dev
     7 + |spec:
     8 + |  carvel:
     9 + |    packageDomain: tanzu.vmware.com
    10 + |    packageName: tanzu-java-web-app
    11 + |  gitOps:
    12 + |    baseBranch: main
    13 + |    subPath: packages
    14 + |    url: GITOPS-REPO-PATH
    15 + |  registry:
    16 + |    repository: REPOSITORY-PATH
    17 + |    server: REGISTRY-SERVER
    18 + |  source:
    19 + |    git:
    20 + |      branch: main
    21 + |      url: https://github.com/vmware-tanzu/application-accelerator-sam
ples.git
    22 + |    subPath: tanzu-java-web-app
Create workload tanzu-java-web-app from workload.yaml? [yN]: y
✓ Successfully created workload tanzu-java-web-app
```

6. See all the workloads of each kind running in your namespace by running:

```
tanzu workload list
```

Example output:

```
Listing workloads from the dev namespace

 NAME                KIND                                          VERSION   AGE
 tanzu-java-web-app  appbuildv1s.supplychains.tanzu.vmware.com  v1alpha1  30m

  To see more details about a workload, use 'tanzu workload get workload-name -
-kind workload-kind'
```

7. See all the workloads running in all namespaces by running:

```
tanzu workload list -A
```

8. See how the `tanzu-java-web-app` workload in the workload list is progressing by running:

```
tanzu workload get tanzu-java-web-app
```

Example output:

```
  Overview
  name:       tanzu-java-web-app
  kind:       appbuildv1s.supplychains.tanzu.vmware.com/tanzu-java-web-app
  namespace:  dev
  age:        15m

  Runs:
  ID                           STATUS   DURATION  AGE
  tanzu-java-web-app-run-454m5  Running  2m        2m

  To view a run information, use 'tanzu workload run get run-id'
```

From the output, you see that a `WorkloadRun` named `tanzu-java-web-app-run-454m5` was created when you applied the `AppBuildV1` workload and it is in the `Running` state. There are multiple causes for why a new `WorkloadRun` is created for your `Workload`, but few are developer-triggered.

Updates to your `Workload`, such as changing the values in `workload.yaml` and reapplying them to the cluster, create a new `WorkloadRun`. Platform engineering activities such as updating the Buildpack builder images can also cause your `Workload` to rebuild, creating a new `WorkloadRun` with newer base images.

Other activities, such as pushing new commits to the source-code repository referenced by the `Workload`, can also cause a new run of the `Workload` to build the latest source from your Git repository.

9. Use the `tanzu workload run get` command to see more information about which stages your workload is going through. Use the optional `--show-details` flag for more detailed output. Run:

```
tanzu workload run get tanzu-java-web-app-run-454m5 --show-details
```

Example output:

```
  Overview
  name:        tanzu-java-web-app
  kind:        appbuildv1s.supplychains.tanzu.vmware.com/tanzu-java-web-app
  run id:      appbuildv1runs.supplychains.tanzu.vmware.com/tanzu-java-web-app-
run-454m5
  status:      Running
  namespace:   dev
  age:         14m
```

```
  Spec
     1 + |---
     2 + |apiVersion: supplychains.tanzu.vmware.com/v1alpha1
     3 + |kind: AppBuildV1
     4 + |metadata:
     5 + |  name: tanzu-java-web-app
     6 + |  namespace: dev
     7 + |spec:
     8 + |  carvel:
     9 + |    packageDomain: tanzu.vmware.com
    10 + |    packageName: tanzu-java-web-app
    11 + |  gitOps:
    12 + |    baseBranch: main
    13 + |    subPath: packages
    14 + |    url: GITOPS-REPO-PATH
    15 + |  registry:
    16 + |    repository: REPOSITORY-PATH
    17 + |    server: REGISTRY-SERVER
    18 + |  source:
    19 + |    git:
    20 + |      branch: main
    21 + |      url: https://github.com/vmware-tanzu/application-accelerator-sam
ples.git
    22 + |    subPath: tanzu-java-web-app

  Stages
  ├─ source-git-provider
  │  ├─   check-source - Success
  │  │  ├─ Duration: 31s
  │  │  └─ Results
  │  │     ├─ message: using git-branch: main
  │  │     ├─ sha: e4e23867bcffcbf7a165e2fefe3c48dc28b076d6
  │  │     └─ url: https://github.com/vmware-tanzu/application-accelerator-samp
les.git
  │  └─   pipeline - Success
  │     ├─ Duration: 1m28s
  │     └─ Results
  │        ├─ url: IMAGE-URL
  │        └─ digest: IMAGE-SHA
  ├─ buildpack-build
  │  ├─   check-builders - Success
  │  │  ├─ Duration: 26s
  │  │  └─ Results
  │  │     ├─ builder-image: BUILDER-IMAGE-USED
  │  │     ├─ message: Builders resolved
  │  │     └─ run-image: RUN-IMAGE-USED
  │  └─   pipeline - Success
  │     ├─ Duration: 2m59s
  │     └─ Results
  │        ├─ url: IMAGE-URL
  │        └─ digest: IMAGE-SHA
  ├─ conventions
  │  └─   pipeline - Success
  │     ├─ Duration: 33s
  │     └─ Results
  │        ├─ url: IMAGE-URL
  │        └─ digest: IMAGE-SHA
  ├─ app-config-server
  │  └─   pipeline - Running
  │     └─ Duration: 22.37089s
  ├─ carvel-package
  │  └─   pipeline - Not Started
  └─ git-writer-pr
     └─   pipeline - Not Started
```

The output shows you:

- An overview of your workload, such as `name`, `kind`, and `namespace`
- The last 2 successful `WorkloadRuns`
- The last failed `WorkloadRun`
- All running `WorkloadRuns`
- The error section from the last failed `WorkloadRun`

When your `WorkloadRun` has gone through the Supply Chain, the output of the `Workload` and `WorkloadRun` is as follows:

Workload Run Output
tanzu workload run get tanzu-java-web-app-run-454m5 –show-details

```
  Overview
  name:         tanzu-java-web-app
  kind:         appbuildv1s.supplychains.tanzu.vmware.com/tanzu-java-web-app
  run id:       appbuildv1runs.supplychains.tanzu.vmware.com/tanzu-java-web-app-run-4
54m5
  status:       Succeeded
  namespace:    dev
  age:          68m

  Spec
  1 + |---
  2 + |apiVersion: supplychains.tanzu.vmware.com/v1alpha1
  3 + |kind: AppBuildV1
  4 + |metadata:
  5 + |  name: tanzu-java-web-app
  6 + |  namespace: dev
  7 + |spec:
  8 + |  carvel:
  9 + |    packageDomain: tanzu.vmware.com
  10 + |    packageName: tanzu-java-web-app
  11 + |  gitOps:
  12 + |    baseBranch: main
  13 + |    subPath: packages
  14 + |    url: GITOPS-REPO-PATH
  15 + |  registry:
  16 + |    repository: REPOSITORY-PATH
  17 + |    server: REGISTRY-SERVER
  18 + |  source:
  19 + |    git:
  20 + |      branch: main
  21 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples.g
it
  22 + |    subPath: tanzu-java-web-app

  Stages
  ├─ source-git-provider
  │  ├─   check-source - Success
  │  │  ├─ Duration: 31s
  │  │  └─ Results
  │  │     ├─ message: using git-branch: main
  │  │     ├─ sha: e4e23867bcffcbf7a165e2fefe3c48dc28b076d6
  │  │     └─ url: https://github.com/vmware-tanzu/application-accelerator-samples.g
it
  │  └─   pipeline - Success
  │     ├─ Duration: 1m28s
  │     └─ Results
  │        ├─ url: IMAGE-URL
  │        └─ digest: IMAGE-SHA
  ├─ buildpack-build
  │  ├─   check-builders - Success
```

```
  │  │  ├─ Duration: 26s
  │  │  └─ Results
  │  │     ├─ builder-image: BUILDER-IMAGE-USED
  │  │     ├─ message: Builders resolved
  │  │     └─ run-image: RUN-IMAGE-USED
  │  └─  pipeline - Success
  │     ├─ Duration: 2m59s
  │     └─ Results
  │        ├─ url: IMAGE-URL
  │        └─ digest: IMAGE-SHA
  ├─ conventions
  │  └─  pipeline - Success
  │     ├─ Duration: 33s
  │     └─ Results
  │        ├─ url: IMAGE-URL
  │        └─ digest: IMAGE-SHA
  ├─ app-config-server
  │  └─  pipeline - Success
  │     ├─ Duration: 1m12s
  │     └─ Results
  │        ├─ url: IMAGE-URL
  │        ├─ digest: IMAGE-SHA
  │        ├─ url-overlay: IMAGE-URL
  │        └─ digest-overlay: IMAGE-SHA
  ├─ carvel-package
  │  └─  pipeline - Success
  │     ├─ Duration: 49s
  │     └─ Results
  │        ├─ url: IMAGE-URL
  │        └─ digest: IMAGE-SHA
  └─ git-writer-pr
  └─  pipeline - Success
      ├─ Duration: 34s
      └─ Results
         ├─ url: PULL-REQUEST-URL-TO-GITOPS-REPO
         └─ digest: IMAGE-SHA
```

Workload Get Output
tanzu workload get tanzu-java-web-app

```
  Overview
name:       tanzu-java-web-app
kind:       appbuildv1s.supplychains.tanzu.vmware.com/tanzu-java-web-app
namespace:  dev
age:        74m

  Runs:
ID                           STATUS     DURATION  AGE
tanzu-java-web-app-run-454m5  Succeeded  16m       72m

  To view a run information, use 'tanzu workload run get run-id'
```

Based on the description of the `AppBuildV1` kind from the `tanzu workload kind list` command, the Supply Chain pulls the source code from the Git repository, builds it by using buildpacks, and then packages the output as a Carvel package. That output is then shipped to the GitOps repository that the platform engineer configures.

In your Supply Chain, when the `WorkloadRun` succeeds you can see the URL for the pull request to the GitOps repository in the `tanzu workload run get --show-details` output in the `gitops-pr` stage results.

You have now used Tanzu Supply Chain to deploy your first workload.

Next Steps

See these How-to guides for developers to learn more about Tanzu Supply Chain.

How-to topics for developers

This section contains how-to topics for developers using Tanzu Supply Chain.

Caution

Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.

In this section:

- Install the workload CLI plug-in
- Work with workloads

Install the Tanzu Workload CLI plug-in

This topic tells you how to install the Tanzu Workload CLI plug-in. Use this CLI plug-in to work with the workloads provided by platform engineering.

Caution

Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.

This plug-in enables you to see workload kinds, deploy workloads, and monitor the workloads. The Tanzu Supply Chain beta release does not include the installation of the Tanzu Workload CLI plug-in as part of the plug-in group.

The Tanzu Workload CLI plug-in provides commands that enable a developer to generate a `Workload` manifest, apply it to a cluster, and delete it from a cluster.

Prepare

Install the latest version of the Tanzu CLI. For more information, see Install Tanzu CLI.

Install Tanzu Workload CLI plug-in

To install the Tanzu Workload CLI plug-in:

1. Install the plug-in by running:

   ```
   tanzu plugin install workload
   ```

2. Verify that you installed the plug-in successfully by running:

   ```
   tanzu workload version
   ```

Note

If you need to uninstall the Tanzu Workload CLI plug-in, run:

```
tanzu plugin delete workload
```

Work with workloads

This topic tells you how to work with workloads when using Tanzu Supply Chain.

Caution

Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.

This topic explains how to:

- Find the kinds of workloads you can use
- Create and delete a workload
- Observe runs of your workloads

Find the kinds of workloads you can use

Use the Tanzu Workload CLI plug-in to see available `Workload` kinds by running:

```
tanzu workload kinds list
```

Example:

```
KIND                          VERSION   DESCRIPTION
buildserverapps.vmware.com  v1        Builds a Kubernetes deployment app exposed using
a Service using Buildpacks.
buildwebapps.vmware.com     v1        Pulls the source code from a Git repository and
builds a Knative Service app using Buildpacks.
buildworkerapps.vmware.com  v1        Creates a background worker app deployed as Kube
rnetes deployment using Buildpacks.

  To generate a workload for one of these kinds, use 'tanzu workload generate'
```

Create and delete a workload

In this section you:

- Generate a workload manifest
- Create a workload
- Apply a workload
- Delete a workload

Generate a workload manifest

To generate a workload manifest:

1. Generate a `Workload` manifest with default configuration by running:

   ```
   tanzu workload generate APP-NAME --kind buildwebapps.vmware.com
   ```

   Where `APP-NAME` is the name of the app. For example, `my-web-app`.

   Example:

   ```
   apiVersion: vmware.com/v1
   kind: BuildWebApp
   metadata:
    name: my-web-app
   spec:
    source:
      #! Use this object to retrieve source from a git repository.
      #! The tag, commit, and, branch fields are mutually exclusive, use only one.
      #! Required
      git:
        #! A git branch ref to watch for new source
        branch: ""
        #! A git commit sha to use
        commit: ""
        #! A git tag ref to watch for new source
        tag: ""
        #! The url to the git source repository
        #! Required
        url: ""
      #! path inside the source to build from (build has no access to paths above
   the subPath)
      subPath: ""
   ...
   #! other configuration
   ```

2. Store the `Workload` manifest in a file for use by the `tanzu workload create`, `tanzu workload apply`, and `tanzu workload get` commands, and pipe the output into a `workload.yaml` file, by running:

   ```
   tanzu workload generate APP-NAME --kind buildwebapps.vmware.com > workload.yaml
   ```

   Where `APP-NAME` is the name of the app. For example, `my-web-app`.

Create a workload

To create a workload:

1. Create a `Workload` on the cluster from a manifest by running:

   ```
   tanzu workload create --file workload.yaml --namespace build
   ```

   Example:

   ```
   Creating workload:
        1 + |---
        2 + |apiVersion: vmware.com/v1
        3 + |kind: BuildWebApp
        4 + |metadata:
        5 + |  name: my-web-app
        6 + |  namespace: build
        7 + |spec:
        8 + |  source:
        9 + |    git:
       10 + |      branch: ""
       11 + |      commit: ""
   ```

```
   12 + |      tag: ""
   13 + |      url: ""
   14 + |    subPath: ""
   15 + |  ...
Create workload my-web-app from workload.yaml? [yN]: y
Successfully created workload my-web-app
```

2. Override the `Workload` name provided in the manifest by running:

   ```
   tanzu workload create NAME --file workload.yaml --namespace build
   ```

   Where `NAME` is a name to serve as an argument for the manifest. For example, `my-web-app-2`.

Apply a workload

To apply a workload:

1. The `tanzu workload create` command is only used to create a `Workload` that does not already exist. To update an existing `Workload`, use `tanzu workload apply`. Apply a `Workload` manifest to the cluster by running:

   ```
   tanzu workload apply --file workload.yaml --namespace build
   ```

   Example:

   ```
   Creating workload:
        1 + |---
        2 + |apiVersion: vmware.com/v1
        3 + |kind: BuildWebApp
        4 + |metadata:
        5 + |  name: my-web-app
        6 + |  namespace: build
        7 + |spec:
        8 + |  source:
        9 + |    git:
       10 + |      branch: ""
       11 + |      commit: ""
       12 + |      tag: ""
       13 + |      url: ""
       14 + |    subPath: ""
       15 + |  ...
   Create workload my-web-app from workload.yaml? [yN]: y
   Successfully created workload my-web-app
   ```

2. Override the `Workload` name provided in the manifest by running:

   ```
   tanzu workload apply NAME --file workload.yaml --namespace build
   ```

   Where `NAME` is a name to serve as an argument for the manifest. For example, `my-web-app-2`.

Delete a workload

Delete a `Workload` by name within a namespace by running:

```
tanzu workload delete --file /tmp/workload.yaml --namespace build
```

Example:

```
Really delete the workload my-web-app of kind buildwebapps.vmware.com from the build n
amespace? [yN]: y
Successfully deleted workload my-web-app
```

Note

Deleting a `Workload` prevents new builds while preserving built images in the registry.

Observe the runs of your workload

Use the Tanzu Workload CLI plug-in to observe `Workloads` and their `WorkloadRuns`:

1. List all workloads on the cluster by running:

   ```
   tanzu workload list --namespace build
   ```

   Example:

   ```
   Listing workloads from the build namespace

    NAME        KIND                      VERSION  AGE
    my-web-app  buildwebapps.vmware.com  v1       6m54s
   ```

2. Get the details of the specified `Workload` within a namespace by running:

   ```
   tanzu workload get APP-NAME --namespace build
   ```

   Where `APP-NAME` is the application name. For example, `my-web-app`.

   Example:

   ```
   $ tanzu workload get my-web-app --namespace build
   Overview
    name:       my-web-app
    kind:       buildwebapps.vmware.com/my-web-app
    namespace:  build
    age:        17s

   Runs:
    ID                    STATUS   DURATION  AGE
    my-web-app-run-lxwrm  Running  0s        17s
   ```

3. Get the workload output as YAML for programmatic use by running:

   ```
   tanzu workload get APP-NAME --namespace build -o yaml
   ```

   Where `APP-NAME` is the application name. For example, `my-web-app`.

   Example:

   ```
   $ tanzu workload get my-web-app --namespace build -o yaml
   ---
   apiVersion: vmware.com/v1
   kind: BuildWebApp
   metadata:
    name: my-web-app
    namespace: build
   ```

```
spec:
 ...
```

4. Get the details of the specified workload run within a namespace by running:

```
tanzu workload run get APP-NAME-run-lxwrm -n build --show-details
```

Where `APP-NAME` is the application name. For example, `my-web-app`.

Example:

```
Overview
 name:         my-web-app
 kind:         buildwebapps.vmware.com/my-web-app
 run id:       buildwebappruns.vmware.com/my-web-app-run-lxwrm
 status:       Running
 namespace:    build
 age:          39s

Spec
     1 + |---
     2 + |apiVersion: vmware.com/v1
     3 + |kind: BuildWebApp
     4 + |metadata:
     5 + |  name: my-web-app
     6 + |  namespace: build
     7 + |spec:
     8 + |...

Stages
   ├─ source-git-provider
   │  ├─ check-source - Success
   │  │  ├─ Duration: 6s
   │  │  └─ Results
   │  │     ├─ message: using git-branch: main
   │  │     ├─ sha: <image SHA>
   │  │     └─ url: <image URL>
   │  └─ pipeline - Success
   │     ├─ Duration: 1m38s
   │     └─ Results
   │        ├─ url: <image URL>
   │        └─ digest: <image SHA>
   ├─ buildpack-build
   │  ├─ check-builders - Success
   │  │  ├─ Duration: 5s
   │  │  └─ Results
   │  │     ├─ builder-image: <image URL>
   │  │     ├─ message: Builders resolved
   │  │     └─ run-image: <image URL>
   │  └─ pipeline - Success
   │     ├─ Duration: 50s
   │     └─ Results
   │        ├─ url: <image URL>
   │        └─ digest: <image SHA>
   ├─ conventions
   │  └─ pipeline - Running
   │     └─ Duration: 53.693499s
   ├─ app-config-web
   │  └─ pipeline - Not Started
   ├─ carvel-package
   │  └─ pipeline - Not Started
   └─ git-writer-pr
      └─ pipeline - Not Started
```

Tanzu Supply Chain Reference

This section provides reference information for Tanzu Supply Chain.

Caution

Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.

In this section:

- Component Catalog
- Supply Chain CLI Plug-in Reference
- Workload CLI Plug-in Reference
- Supply Chain API

Catalog of Tanzu Supply Chain Components

Caution

Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.

This section introduces the catalog of components shipped with TAP. You will find all of these components in the “authoring” profile.

app-config-server

Version: 1.0.0

Description:

Generates configuration for a Server application from a Conventions PodIntent. Server applications contain a K8s Deployment and Service and can be configured with Ingress.

Inputs

Name	Type
conventions	conventions

Outputs

Name	Type
oci-yaml-files	oci-yaml-files
oci-ytt-files	oci-ytt-files

Config

```
spec:
  # Configuration for the registry to use
  registry:
    # The name of the registry server, e.g. docker.io
    # +required
    server:
    # The name of the repository
    # +required
    repository:
```

---

app-config-web

Version: 1.0.0

Description:

Generates configuration for a Web application from a Conventions PodIntent. Web applications contain a Knative Service.

Inputs

Name	Type
conventions	conventions

Outputs

Name	Type
oci-yaml-files	oci-yaml-files
oci-ytt-files	oci-ytt-files

Config

```
spec:
  # Configuration for the registry to use
  registry:
    # The name of the repository
    # +required
    repository:
    # The name of the registry server, e.g. docker.io
    # +required
    server:
```

---

app-config-worker

Version: 1.0.0

Description:

Generates configuration for a Worker application from a Conventions PodIntent. Worker applications contain a K8s Deployment.

Inputs

Name	Type
conventions	conventions

Outputs

Name	Type
oci-yaml-files	oci-yaml-files
oci-ytt-files	oci-ytt-files

Config

```
spec:
  # Configuration for the registry to use
  registry:
    # The name of the repository
    # +required
    repository:
    # The name of the registry server, e.g. docker.io
    # +required
    server:
```

---

buildpack-build

Version: 1.0.0

Description:

Builds an app with buildpacks using kpack

Inputs

Name	Type
source	source
git	git

Outputs

Name	Type
image	image

Config

```
spec:
  # Registry to use
  registry:
    # The registry address
    # +required
    server:
    # The repository to use
    # +required
    repository:
  # Kpack build specification
  build:
```

```
    # Service account to use
    serviceAccountName:
    env:
    # Configure workload to use a non-default builder or clusterbuilder
    builder:
      # builder kind
      kind:
      # builder name
      name:
    # cache options
    cache:
      # whether to use a cache image
      enabled:
      # cache image to use
      image:
  source:
    # path inside the source to build from (build has no access to paths above the sub
Path)
    subPath:
```

---

carvel-package

Version: 1.0.0

Description:

Generates a carvel package from OCI images containing raw YAML files and YTT files.

Inputs

Name	Type
oci-yaml-files	oci-yaml-files
oci-ytt-files	oci-ytt-files

Outputs

Name	Type
package	package

Config

```
spec:
  # Configuration for the generated Carvel Package
  carvel:
    # The name of the Carvel Package. Combines with spec.carvel.packageDomain to creat
e the Package refName. If set to "", will use the workload name.
    packageName:
    # Service account that gives kapp-controller privileges to create resources in the
namespace.
    serviceAccountName:
    # Name of the values Secret that provides customized values to the package install
ation's templating steps.
    valuesSecretName:
    # PEM encoded certificate data for the image registry where the files will be push
ed to.
    caCertData:
    # Enable the use of IAAS based authentication for imgpkg.
    iaasAuthEnabled:
```

```
    # The domain of the Carvel Package. Combines with spec.carvel.packageName to creat
e the Package refName. If set to "", will use "default.tap".
    packageDomain:
  gitOps:
    # the branch to commit changes to
    branch:
    # the relative path within the gitops repository to add the package configuration
to.
    subPath:
    # the repository to push the pull request to
    url:
  # Configuration for the registry to use
  registry:
    # The name of the repository
    # +required
    repository:
    # The name of the registry server, e.g. docker.io
    # +required
    server:
```

---

conventions

Version: 1.0.0

Description:

The Conventions component analyzes the `image` input as described in the Cartographer Conventions documentation and produces a `conventions` output image.

Depends on: - Managed Resource Controller. - Tanzu Carvel Package: `managed-resource-controller.apps.tanzu.vmware.com @ >=0.1.2` - Conventions Controller - Tanzu Carvel Package: `cartographer.tanzu.vmware.com @ >= 0.8.10`

Inputs

Name	Type
image	image

Outputs

Name	Type
conventions	conventions

Config

```
spec:
  # May contain an optional array of objects. Each object is a pair of keys: `name` an
d either `value` or `valueFrom`.
  # The Conventions component will translate these values into environment variables i
n the output object.
  env:
```

---

deployer

Version: 1.0.0

Description:

Deploys K8s resources to the cluster.

Inputs

Name	Type
package	package

Outputs

- *none*

Config

```
spec:
  # The path to the yaml to be applied to the cluster.
  subPath:
    # The path to the yaml to be applied to the cluster
    # +required
    path:
```

---

git-writer

Version: 1.0.0

Description:

Writes carvel package config directly to a gitops repository

Inputs

Name	Type
package	package

Outputs

- *none*

Config

```
spec:
  gitOps:
    # the repository to push the pull request to
    # +required
    url:
    # the branch to commit changes to
    branch:
    # the relative path within the gitops repository to add the package configuration
to.
    subPath:
```

---

git-writer-pr

Version: 1.0.0

Description:

Writes carvel package config to a gitops repository and opens a PR

Inputs

Name	Type
package	package

Outputs

Name	Type
git-pr	git-pr

Config

```
spec:
  gitOps:
    # the base branch to create PRs against
    baseBranch:
    # the relative path within the gitops repository to add the package configuration
to.
    subPath:
    # the repository to push the pull request to
    # +required
    url:
```

kaniko-build

Version: 1.0.0

Description

Builds an app with kaniko

Inputs

Name	Type
source	source
git	git

Outputs

Name	Type
image	image

Config

```
spec:
  # Kaniko build specification
  build:
    # path to dockerfile to build
    dockerfile:
    # extra args to pass to kaniko build
```

```
    extra-args:
  # Registry to use
  registry:
    # The repository to use
    # +required
    repository:
    # The registry address
    # +required
    server:
```

---

source-git-provider

Version: 1.0.0

Description:

Source git provider retrieves source code and monitors a git repository.

Inputs

- *none*

Outputs

Name	Type
source	source
git	git

Config

```
spec:
  source:
    # Use this object to retrieve source from a git repository.
    # The tag, commit and branch fields are mutually exclusive, use only one.
    # +required
    git:
      # A git branch ref to watch for new source
      branch:
      # A git commit sha to use
      commit:
      # A git tag ref to watch for new source
      tag:
      # The url to the git source repository
      # +required
      url:
    # The sub path in the bundle to locate source code
    subPath:
```

---

source-package-translator

Version: 1.0.0

Description:

Takes the type source and immediately outputs it as type package.

Inputs

Name	Type
source	source

Outputs

Name	Type
package	package

Config

none

---

trivy-image-scan

Version: 1.0.0

Description:

Performs a trivy image scan using the scan 2.0 components

Inputs

Name	Type
image	image
git	git

Outputs

- *none*

Config

```
spec:
  # Configuration for the registry to use
  registry:
    # The name of the repository
    # +required
    repository:
    # The name of the registry server, e.g. docker.io
    # +required
    server:
  source:
    # Fill this object in if you want your source to come from git.
    # The tag, commit and branch fields are mutually exclusive, use only one.
    # +required
    git:
      # A git branch ref to watch for new source
      branch:
      # A git commit sha to use
      commit:
      # A git tag ref to watch for new source
      tag:
      # The url to the git source repository
      # +required
      url:
    # The sub path in the bundle to locate source code
```

```
    subPath:
  # Image Scanning configuration
  scanning:
    service-account-scanner:
    workspace:
      size:
      bindings:
    active-keychains:
    service-account-publisher:
```

Output types for catalog components

This section describes the common types established in Tanzu Supply Chain Components, their purpose and their structure. Use this as a guide to consuming or emitting types from your own custom Components.

Caution

Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.

conventions

URL: OCI image containing the `app-config.yaml` file.
Digest: OCI image digest

This output image contains a single file: `app-config.yaml`. The `template` field in this file contains a Kubernetes Pod Template Spec, decorated by the Conventions controller. The Conventions component can also copy the settings in the configurations `spec.env` into the Pod template's `env` field.

git

URL: The source code Git repository URL.
Digest: The source code git commit sha.

git-pr

URL: URL of the pull request.
Digest: SHA digest of the pull request commit.

Output for a Git pull request.

image

URL: OCI image URL for the image built with kpack.
Digest: OCI image digest

An OCI image containing the app built with kpack.

oci-yaml-files

URL: An OCI image containing Kubernetes resource(s) as YAML files.
Digest: OCI image digest.

The resources in this OCI image can be applied to a cluster with no modifications.

The purpose of this OCI image type is to provide the raw Kubernetes resources required to deploy an application onto a cluster.

oci-ytt-files

URL: An OCI image containing files written in the ytt templating language.
Digest: OCI image digest

The files in this OCI image cannot be applied to a cluster, and are intended to be combined with Kubernetes resources from oci-yaml-files.

The purpose of this OCI image type is to provide the ytt Values Schema and Overlays that make a Carvel Package configurable. Separating the ytt files from the raw Kubernetes resources makes the app-config-* components more generally useful. If a Supply Chain author wanted to create a Supply Chain that simply deploys the raw Kubernetes YAML files created by the app-config-* components, instead of putting them into a Carvel Package, they could write a component that takes the input oci-yaml-files, and not the input oci-ytt-files.

package

URL: An OCI image containing a Carvel Package, PackageInstall, and Values YAML file.
Digest: OCI image digest.

The resources in this OCI image can either be deployed to a cluster, or written to a Git repository for use in a GitOps workflow.

The purpose of this OCI image type is to contain a single version of an application as a Carvel Package, as well as the PackageInstall and Values YAML necessary to deploy the Package.

source

URL: OCI image containing the Git repository source code.
Digest: OCI image digest.

An OCI image containing the source code retrieved from a Git repository.

tanzu workload

create, update, view and list Tanzu Workloads.

Options

```
-h, --help              help for workload
    --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
    --no-color          deactivate color, bold, animations, and emoji output
```

SEE ALSO

- tanzu workload apply - Apply a workload of specific kind on the cluster from the file
- tanzu workload create - Create a workload of specific kind on the cluster from the file
- tanzu workload delete - Delete workload
- tanzu workload generate - Generate a workload manifest with specified configuration
- tanzu workload get - Get details of a workload

- tanzu workload kind - View supported workload types (kinds)
- tanzu workload list - Lists all workloads
- tanzu workload logs - Watch workload related logs
- tanzu workload run - View workload runs

tanzu workload apply

Apply a workload of specific kind on the cluster from the file

```
tanzu workload apply [NAME] [flags]
```

Examples

```
tanzu workload apply --file workload.yaml
  tanzu workload apply my-workload --file workload.yaml --namespace my-namespace
```

Options

```
  -f, --file string      file that contains the workload manifest. Can also be a URL
(default "workload.yaml")
  -h, --help             help for apply
  -n, --namespace name   kubernetes namespace (defaulted from kube config)
  -o, --output string    output of the active workload run along with the status in se
lected format. Supported formats: "json", "yaml", "yml"
  -y, --yes              accept all prompts
```

Options inherited from parent commands

```
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
      --no-color          deactivate color, bold, animations, and emoji output
```

SEE ALSO

- tanzu workload - create, update, view and list Tanzu Workloads.

tanzu workload create

Create a workload of specific kind on the cluster from the file

```
tanzu workload create [NAME] [flags]
```

Examples

```
tanzu workload create --file workload.yaml
  tanzu workload create my-workload --file workload.yaml --namespace my-namespace
```

Options

```
  -f, --file string      file that contains the workload manifest. Can also be a URL
(default "workload.yaml")
```

```
  -h, --help             help for create
  -n, --namespace name   kubernetes namespace (defaulted from kube config)
  -o, --output string    output of the active workload run along with the status in se
lected format. Supported formats: "json", "yaml", "yml"
  -y, --yes              accept all prompts
```

Options inherited from parent commands

```
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
      --no-color          deactivate color, bold, animations, and emoji output
```

SEE ALSO

- tanzu workload - create, update, view and list Tanzu Workloads.

tanzu workload delete

Delete workload

Synopsis

Delete a workload by name within a namespace.

Deleting a workload prevents new builds while preserving built images in the registry.

```
tanzu workload delete [NAME] [flags]
```

Examples

```
tanzu workload delete NAME
  tanzu workload delete NAME --kind kindname
  tanzu workload delete NAME --kind kindname --namespace default
  tanzu workload delete --file workload.yaml
```

Options

```
  -f, --file file path   file path containing the description of a single workload, ot
her flags are layered on top of this resource. Use value "-" to read from stdin
  -h, --help             help for delete
  -k, --kind string      kind of the workload specification
  -n, --namespace name   kubernetes namespace (defaulted from kube config)
  -y, --yes              accept all prompts
```

Options inherited from parent commands

```
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
      --no-color          deactivate color, bold, animations, and emoji output
```

SEE ALSO

- tanzu workload - create, update, view and list Tanzu Workloads.

tanzu workload generate

Generate a workload manifest with specified configuration

Synopsis

Generate a workload manifest with specified configuration.

workload configuration options include: - kind of the workload to generate

```
tanzu workload generate [NAME] [flags]
```

Examples

```
tanzu workload generate NAME
  tanzu workload generate NAME --kind kindname
```

Options

```
  -h, --help          help for generate
  -k, --kind string   kind of the workload specification.
```

Options inherited from parent commands

```
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
      --no-color          deactivate color, bold, animations, and emoji output
```

SEE ALSO

- tanzu workload - create, update, view and list Tanzu Workloads.

tanzu workload generate

Generate a workload manifest with specified configuration

Synopsis

Generate a workload manifest with specified configuration.

workload configuration options include: - kind of the workload to generate

```
tanzu workload generate [NAME] [flags]
```

Examples

```
tanzu workload generate NAME
  tanzu workload generate NAME --kind kindname
```

Options

```
  -h, --help          help for generate
  -k, --kind string   kind of the workload specification.
```

Options inherited from parent commands

```
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
      --no-color          deactivate color, bold, animations, and emoji output
```

SEE ALSO

- tanzu workload - create, update, view and list Tanzu Workloads.

tanzu workload kind

View supported workload types (kinds)

Options

```
  -h, --help   help for kind
```

Options inherited from parent commands

```
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
      --no-color          deactivate color, bold, animations, and emoji output
```

SEE ALSO

- tanzu workload - create, update, view and list Tanzu Workloads.
- tanzu workload kind list - Lists all workload kinds

tanzu workload kind list

Lists all workload kinds

Synopsis

Lists all workload kinds

kind list configuration options include: - Output the kind list formatted

```
tanzu workload kind list [flags]
```

Examples

```
tanzu workload kind list
```

Options

```
  -h, --help   help for list
```

Options inherited from parent commands

```
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
      --no-color          deactivate color, bold, animations, and emoji output
```

SEE ALSO

- tanzu workload kind - View supported workload types (kinds)

tanzu workload list

Lists all workloads

Synopsis

Lists all workloads

list configuration options include: - Output the list for all namespaces - Output the list for a specific namespace - Output the list for a specific kind - Output the list formatted

```
tanzu workload list [flags]
```

Examples

```
tanzu workload list
  tanzu workload list --all-namespaces,
  tanzu workload list --namespace my-namepsace,
  tanzu workload list --kind my-workload-kind,
  tanzu workload list --output yaml
```

Options

```
  -A, --all-namespaces   use all kubernetes namespaces
  -h, --help             help for list
  -k, --kind string      list workloads from the specified workload kind
  -n, --namespace name   kubernetes namespace (defaulted from kube config)
  -o, --output string    output the workloads formatted. Supported formats: "json", "y
aml", "yml"
```

Options inherited from parent commands

```
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
      --no-color          deactivate color, bold, animations, and emoji output
```

SEE ALSO

- tanzu workload - create, update, view and list Tanzu Workloads.

tanzu workload logs

Watch workload-related logs

Synopsis

Stream logs for a workload until canceled. To cancel, press Ctrl+C in the shell or stop the process. As new workload pods start, the logs appear. To see historical logs, use `--since`.

```
tanzu workload logs <NAME> [flags]
```

Examples

```
tanzu workload logs NAME
  tanzu workload logs NAME --since 1h
  tanzu workload logs NAME --since 1h --namespace default
  tanzu workload logs NAME --since 1h --namespace default --run runname
```

Options

```
  -h, --help              help for logs
  -k, --kind string       kind of the workload specification
  -n, --namespace name    kubernetes namespace (defaulted from kube config)
      --run name          workload run name
      --since duration    time duration to start reading logs from (default 1h0m0s)
  -t, --timestamp         print timestamp for each log line
```

Options inherited from parent commands

```
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
      --no-color          deactivate color, bold, animations, and emoji output
```

SEE ALSO

tanzu workload - create, update, view and list Tanzu Workloads.

tanzu workload run

View workload runs

Options

```
  -h, --help   help for run
```

Options inherited from parent commands

```
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
      --no-color          deactivate color, bold, animations, and emoji output
```

SEE ALSO

- tanzu workload - create, update, view and list Tanzu Workloads.
- tanzu workload run get - Get workload run

tanzu workload run get

Get workload run

Synopsis

Get the details of the specified workload run within a namespace.

workload run get configuration options include: - Specify the namespace of the workload run - Output the workload run formatted - Show more details of the workload run

```
tanzu workload run get <NAME> [flags]
```

Examples

```
tanzu workload run get NAME
  tanzu workload run get NAME --namespace namespace
  tanzu workload run get NAME --namespace namespace --show-details
```

Options

```
  -h, --help              help for get
  -n, --namespace name    kubernetes namespace (defaulted from kube config)
  -o, --output string     output the workload run details formatted. Supported formats:
"json", "yaml", "yml"
      --show-details      show more details of the workload run
```

Options inherited from parent commands

```
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
      --no-color          deactivate color, bold, animations, and emoji output
```

SEE ALSO

- tanzu workload run - View workload runs

tanzu supplychain

supplychain management

Options

```
  -h, --help              help for supplychain
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
      --no-color          deactivate color, bold, animations, and emoji output
```

SEE ALSO

- tanzu supplychain component - Interacting with supplychain components
- tanzu supplychain generate - Generate a supplychain scaffold
- tanzu supplychain get - Get supplychain
- tanzu supplychain init - Initialize supplychain
- tanzu supplychain list - Lists all supplychains

tanzu supplychain component

Interacting with supplychain components

Options

```
  -h, --help   help for component
```

Options inherited from parent commands

```
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
      --no-color          deactivate color, bold, animations, and emoji output
```

SEE ALSO

- tanzu supplychain - supplychain management
- tanzu supplychain component get - Get details of a component
- tanzu supplychain component list - Lists all supplychain components

tanzu supplychain component get

Get details of a component

Synopsis

Get the details of the specified component within a namespace.

component get configuration options include: - Specify the namespace of the component (optional) - Output the component formatted - Show more details of the component

```
tanzu supplychain component get <NAME> [flags]
```

Examples

```
tanzu supplychain component get NAME
  tanzu supplychain component get NAME --namespace default
  tanzu supplychain component get NAME --namespace default --show-details
```

Options

```
  -h, --help             help for get
  -n, --namespace name   kubernetes namespace
  -o, --output string    output the component details formatted. Supported formats: "j
son", "yaml", "yml"
      --show-details     show more details of the component
```

Options inherited from parent commands

```
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
      --no-color          deactivate color, bold, animations, and emoji output
```

SEE ALSO

- tanzu supplychain component - Interacting with supplychain components

tanzu supplychain component list

Lists all supplychain components

Synopsis

Lists all available supplychain components

component list configuration options include: - Output the component list formatted

```
tanzu supplychain component list [flags]
```

Examples

```
tanzu supplychain component list
  tanzu supplychain component list --output yaml
  tanzu supplychain component list --wide
```

Options

```
  -h, --help            help for list
  -o, --output string   output the supplychain component formatted. Supported formats:
"json", "yaml", "yml"
  -w, --wide            output the supplychain component list with additional informat
ion
```

Options inherited from parent commands

```
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
      --no-color          deactivate color, bold, animations, and emoji output
```

SEE ALSO

- tanzu supplychain component - Interacting with supplychain components

tanzu supplychain generate

Generate a supplychain scaffold

Synopsis

Generate a supplychain scaffold

Generate configuration options include: - Kind, component list and the description

```
tanzu supplychain generate [flags]
```

Examples

```
tanzu supplychain generate --kind ServerAppV1 --description DESCRIPTION
```

Options

```
  --allow-defaults       add a section for defaults to the supplychain scaffold
  --allow-overrides      add a section for overrides to the supplychain scaffold
  -c, --component strings    add a component to the supplychain scaffold. This flag ca
n be specified multiple times and the order on the command-line will be reflected on t
he generated scaffold output
  -d, --description string   description of the supplychain
  -h, --help                 help for generate
  -k, --kind string          kind to use as the developer interface
```

Options inherited from parent commands

```
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
      --no-color          deactivate color, bold, animations, and emoji output
```

SEE ALSO

- tanzu supplychain - supplychain management

tanzu supplychain get

Get supplychain

Synopsis

Get the details of the specified supplychain within a namespace.

supplychain get configuration options include: - Specify the namespace of the supplychain - Output the supplychain formatted

```
tanzu supplychain get <NAME> [flags]
```

Examples

```
tanzu supplychain get NAME
```

Options

```
  -h, --help             help for get
  -n, --namespace name   kubernetes namespace (defaulted from kube config)
  -o, --output string    output the supplychains formatted. Supported formats: "json",
"yaml", "yml"
```

Options inherited from parent commands

```
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
      --no-color          deactivate color, bold, animations, and emoji output
```

SEE ALSO

- tanzu supplychain - supplychain management

tanzu supplychain init

Initialize supplychain

Synopsis

Initialize supplychain

Initialize configuration options include: - group name and the description

```
tanzu supplychain init [flags]
```

Examples

```
tanzu supplychain init --group supplychains.tanzu.vmware.com --description DESCRIPTION
```

Options

```
  -d, --description string   description of the group
  -g, --group string         group of the supplychains (default "supplychains.tanzu.vm
ware.com")
  -h, --help                 help for init
```

Options inherited from parent commands

```
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
      --no-color          deactivate color, bold, animations, and emoji output
```

SEE ALSO

- tanzu supplychain - supplychain management

tanzu supplychain list

Lists all supplychains

Synopsis

Lists all supplychains

list configuration options include: - Output the list for all namespaces - Output the list for a specific namespace - Output the list formatted

```
tanzu supplychain list [flags]
```

Examples

```
tanzu supplychain list
  tanzu supplychain list --all-namespaces
```

Options

```
  -A, --all-namespaces   use all kubernetes namespaces
  -g, --group string     list all supplychains from a specific group
  -h, --help             help for list
  -n, --namespace name   kubernetes namespace (defaulted from kube config)
  -o, --output string    output the supplychains formatted. Supported formats: "json",
"yaml", "yml"
  -w, --wide             output the supplychain list with additional information
```

Options inherited from parent commands

```
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
      --no-color          deactivate color, bold, animations, and emoji output
```

SEE ALSO

- tanzu supplychain - supplychain management

Tanzu Supply Chain API

This topic gives you reference information about Tanzu Supply Chain APIs.

> **Caution**
>
> Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.

Tanzu Supply Chain introduces two concrete resources and two Duck-typed resources to your Kubernetes cluster.

The concrete resources are:

- `SupplyChain`, which describes a path to production and describes the `Workload` resource you can consume to run this supply chain.
- `Component`, which describes a reusable operation to apply to source, images, configuration, and other artifacts.

The Duck type resources are:

- `Workload`: This resource CRD is created on the cluster when a valid supply chain is applied. This is the API that your `SupplyChain` describes for users to consume.
- `WorkloadRun`: This is a single attempt to deliver the `Workload` through a `SupplyChain`. Every time there is new configuration or external inputs to a workload, a run is generated.

`Workload` and `WorkloadRun` are described as Duck type resources because their final names and CRD are described by the `SupplyChain` and `Component` resources when applied to the cluster. Many fields of each CRD are the same across the system.

SupplyChain API

This topic gives you reference information about the `SupplyChain` resource for Tanzu Supply Chain.

Caution

Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.

The supply chain defines the object kind of the `Workload`, the `Components` used, and their order.

Type and Object Metadata

```
apiVersion: supply-chain.apps.tanzu.vmware.com/v1alpha1
kind: SupplyChain
```

`metadata.name`

`metadata.name` is in the form:

`<plural-name>.<group>-<Major>.<minor>.<patch>`

- `plural-name` must match the plural form of `defines.kind`, without the version. For example: `kind: JavaServerAppV3` would have a `plural-name` of `javaserverappv3s`
- `group` must mach `defines.group` (see `spec.defines.group` below)
- `<Major>.<minor>.<patch>` is the version definition.

```
metadata:
  name: hostedapps.widget.com-0.0.1
```

Spec

`spec.config`

The `spec.stages` structure introduces `Components` where the aggregated config sections, form the SupplyChain's configuration and become the Workload's spec.

This configuration can be altered by changing defaults, or overridden and hidden from the `Workload`.

`spec.config.defaults`

Change or add defaults for the config that is presented in the Workload's spec.

`spec.config.defaults` is an array of `path:` and `value:` fields, where:

- `path`: Path to the configuration value, formatted as either:
  - The full path to the field you want to set.
  - The path to any structure where all desired child fields must be set.
- `value`: A string or structure value.

`spec.config.overrides`

Override the value of a config field, so that it no longer appears in the Workload's spec.

`spec.config.overrides` is an array of `path:` and `value:` fields, where:

- `path`: path to the configuration value, formatted as either:
  - The full path to the field you wish to set.
  - The path to any structure where all desired child fields must be set.
- `value`: string or structure value.

Example

```
apiVersion: supply-chain.apps.tanzu.vmware.com/v1alpha1
kind: SupplyChain
spec:
  config:
    defaults:
      - path: spec.source
        value:
          branch: develop
          url: https://our.git.com/our-monorepo
    overrides:
      - path: spec.registry.repository
        value: "YOUR-REGISTRY-REPO"
      - path: spec.registry.server
        value: "YOUR-REGISTRY-SERVER"
```

spec.description

The `spec.description` field is visible to an app developer when they use the CLI to discover available Workload kinds:

```
tanzu workload kind list --wide

KIND                       VERSION     AGE   DESCRIPTION
serverappv2.example.com    v1alpha1    12m   Server application supply chain
```

Recommendation: embed complete documentation in the description.

The description field supports multi-line Plain text or Markdown.

spec.defines

The `spec.defines` object defines the `Workload` custom resource definition (CRD).

spec.defines.group

`spec.defines.group` (**required**) is used to fill in the `group` field in the [CustomResourceDefinitionSpec].

`spec.defines.group` is the classic domain-formatted group of any Kubernetes object. Use your organization's top level domain, or a departmental domain.

spec.defines.kind

`spec.defines.kind` (**required**) is the name of the resource in CamelCase.

spec.defines.plural

`spec.defines.plural` (**required**) is typically the plural down-cased form of the kind. It must be all lowercase.

Recommendation: pluralize the name after the version, e.g: `WebAppV1` becomes `webappv1s`

`spec.defines.singular`

`spec.defines.singular` is optional and defaults to the lowercase of `kind`, for example `ServerAppv1` becomes `serverappv1`.

`spec.defines.shortnames`

`spec.defines.shortnames` is a list and defaults to empty. Use this to specify an array of aliases for your kind. These are great to simplify `kubectl` commands.

Example

```
kind: ServerAppV1
plural: serverappv1s
shortnames:
  - serverapp1
  - sa1
```

`spec.defines.categories`

`spec.defines.categories` is a list and defaults to empty. `spec.defines.categories` specify a collection term for a group of kinds, so that `kubectl get <category>` returns instances of all kinds in the category.

Example

Using `kubectl get apps` would include this kind in the listing

```
kind: ServerAppV1
plural: serverappv1s
categories:
  - apps
```

Complete Example

```
spec:
  defines: # Describes the workload
    kind: HostedApp
    pluralName: hostedapps
    group: example.com
    version: v1alpha1
    categories:
      - apps
    shortnames:
      - hosted1
      - ha1
```

`spec.stages[]`

`spec.stages` break the work to be done by this supply chain into a serial collection of “stages”, each with a component.

This is where you define the operations of this `SupplyChain`.

`spec.stages[].name`

Each stage has a `name`, which is shown to the user in the CLI and UI. `name` can only be composed of hyphens(`-`) and lower case alphanumeric characters (`a-z0-9`).

`spec.stages[].componentRef`

Each stage also has a `componentRef` with a single field `name`. `componentRef.Name` refers to the name of a Tanzu Supply Chain [Component] resource. Currently, [Component] must exist in the same namespace as the `SupplyChain`. For more information, see the Tanzu Supply Chain known issues.

The supply chain returns an error if a component expects an input that has not been output by a previous stage.

`spec.stages[].securityContext`

Each stage can specify a `securityContext` with a single field `runAs`. When this is set to `workload` the stage is executed in the namespace of the Workload rather than the Supply Chain. See Security Model.

Example

```
apiVersion: supply-chain.apps.tanzu.vmware.com/v1alpha1
kind: SupplyChain
spec:
  stages:
    - name: fetch-source
      componentRef:
        name: source-git-provider-1.0.0
      securityContext:
        runAs: workload
    - name: build
      componentRef:
        name: golang-builder-1.0.0
    - name: commit
      componentRef:
        name: commit-writer-1.0.0
```

Status

`status.conditions[]`

Every `status.conditions[]` in Tanzu Supply Chain resources follows a strict set of conventions.

The top-level condition type is `Ready` as SupplyChain is a “living” resource.

The sub-types are:

RBACDefined

Reason	Meaning
Ready	The RoleBindings for the kind declared in `spec.defines` was created on cluster
AlreadyExists	The RoleBinding record already exists. Most common cause of this issue is another Supply Chain with the same `spec.defines` section.
UnknownError	The RoleBinding record failed due to an exceptional error. Look at the reconciler logs and contact Tanzu Support

APIsDefined

Reason	Meaning
Ready	The CRD for the kind declared in `spec.defines` was created on cluster
Conflict	The CRD already exists and is managed by another SupplyChain. Most common cause of this issue is another Supply Chain with the same `spec.defines` section.
Invalid	The CRD is invalid Most common cause of this is an illegal OpenAPIV3Schema in the [Component].
Unknown error	The CRD could not be created due to an exceptional error. Look at the reconciler logs and contact Tanzu Support

StageMapping

Reason	Meaning
Ready	The stages pass all the validation rules.
NoSuchComponent	The referenced component cannot be found.
NoSuchInput	The input to the stage is not emitted by any stage.
InputNotSatisfiedUntilLater	The input to the stage is not emitted by a previous stage.
InputMismatch	The input matches a previous output by name, however the type does not match.
OutputRedefined	The output redefines an existing output. Shadowing of outputs is not supported.

Component API

This topic gives you reference information about the `Component` resource for Tanzu Supply Chain.

> **Caution**
>
> Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.

Components define the work to be done in one Stage of the `SupplyChain`.

Type and Object Metadata

```
apiVersion: supply-chain.apps.tanzu.vmware.com/v1alpha1
kind: Component
```

`metadata.name`

`metadata.name:` must have a `-M.m.p` suffix, representing the major, minor and patch of this version of the component. Changes to the config section should coincide with a bump to major or minor versions. Reserve patch increments for changes that do not alter the API, or the behavior significantly.

```
metadata:
  name: source-1.0.0
```

Spec

spec.description

`spec.description` describes the component's purpose. You will see this description in `tanzu workload run list`.

```
spec:
  description: Gets the latest source and stores it in an OCI Image
```

spec.config

`spec.config` defines the configuration in a workload (`spec` of the workload) that is required for the component to operate.

`spec.config` is an array with three fields:

- `path`: describes the path in the workload where this configuration is appended/merged. It must start with `spec.`
- `schema` and `required` define a property. See the (Kubernetes) [https://kubernetes.io/docs/home/] documentation.

Example

```
config:
  - path: spec.source.git
    schema:
      type: object
      description: |
        Fill this object in if you want your source to come from git.
        The tag, commit and branch fields are mutually exclusive,
        use only one.
      properties:
        tag:
          description: A git tag ref to watch for new source
          type: string
        commit:
          description: A git commit sha to use
          type: string
        branch:
          description: A git branch ref to watch for new source
          type: string
        url:
          description: The url to the git source repository
          type: string
      required:
        - url
```

spec.inputs

```
inputs:
  - name: source
    type: source
```

spec.pipelineRun

The `spec.pipelineRun` section defines the work done by this component. `spec.pipelineRun` is used to create a [Tekton PipelineRun] and has many similarities.

`spec.pipelineRun.pipelineRef`

The `spec.pipelineRun.pipelineRef` is required, and it has one field `name` that must refer to the `metadata.name` of a [Tekton Pipeline] that resides in the same namespace as the `Component` and `SupplyChain`.

`spec.pipelineRun.workspaces`

If you need to define workspaces to pass to the Tekton `PipelineRun`, use `spec.pipelineRun.workspaces`. This field is an array of workspace definitions, and is identical to the Tekton Workspaces specification.

`spec.pipelineRun.params`

`spec.pipelineRun.params` are the same as [Tekton PipelineRun Parameters] with one major difference: you can populate them using templates.

The available references for templating are:

reference	source	examples
`$(workload.spec...)`	The workload spec	`$(workload.spec.source.git.url)`
`$(workload.metadata...)`	The workload metadata	`$(workload.metadata.labels)`, `$(workload.metadata.name)`
`$(inputs.<input-name>.[url|digest])`	An input url or digest	`$(inputs.image.url)`, `$(inputs.image.digest)`
`$(resumptions.<resumption-name>.results...)`	A resumption result	`$(resumptions.check-source.results.sha)`

Example

```
spec:
  pipelineRun:
    pipelineRef:
      name: source
    params:
      - name: git-url
        value: $(workload.spec.source.git.url)
      - name: sha
        value: $(resumptions.check-source.results.sha)
    workspaces:
      - name: shared-data
        volumeClaimTemplate:
          spec:
            accessModes:
              - ReadWriteOnce
            resources:
              requests:
                storage: 1Gi
```

spec.outputs

```
  outputs:
    - name: source
      type: source
```

spec.resumptions[]

`spec.resumptions[]` define Tekton `TaskRuns`. They are optional, but useful to describe small, fast tasks that check for dependency changes, such as new source, or new base images.

For a detailed explanation of resumptions, see Core Concepts: Resumptions.

`spec.resumptions[].name`

`spec.resumptions[].name` is visible to end users, so make sure it's suitably meaningful. It's recommended that it's in the form: `check-RESOURCE-TYPE` where `RESOURCE-TYPE` is the kind of resource being polled, such as `source` or `base-image`.

`spec.resumptions[].trigger.runaAfter`

`spec.resumptions[].trigger.runaAfter` describes the rerun period for the task. The task is executed, and after it completes (successfully or otherwise), Tanzu Supply Chain waits the `runaAfter` period and then executes the task again. This continues indefinitely.

`runAfter` can be specified using the `time.ParseDuration()` specification.

`spec.resumptions[].taskRef`

The `spec.resumptions[].taskRef` has one field `name` that must refer to the `metadata.name` of a Tekton `Task` that resides in the same namespace as the `Component` and `SupplyChain`.

This is the task that run's on the resumptions `spec.resumptions[].trigger`

`spec.resumptions[].params`

`spec.resumptions[].params` are the same as Tekton `TaskRun` Parameters with one major difference: you can populate them using templates.

The available references for templating references are:

reference	source	examples
`$(workload.spec...)`	The workload spec	`$(workload.spec.source.git.url)`
`$(workload.metadata...)`	The workload metadata	`$(workload.metadata.labels)`, `$(workload.metadata.name)`
`$(inputs.<input-name>.[url|digest])`	An input url or digest	`$(inputs.image.url)`, `$(inputs.image.digest)`

Example

```
resumptions:
  - name: check-source
    trigger:
      runAfter: 60s
    taskRef:
      name: check-source
    params:
      - name: git-branch
        value: $(workload.spec.source.git.branch)
      - name: git-url
        value: $(workload.spec.source.git.url)
      - name: git-commit
        value: $(workload.spec.source.git.commit)
      - name: git-tag
        value: $(workload.spec.source.git.tag)
```

Status

`status.conditions[]`

Every `status.conditions[]` in Tanzu Supply Chain resources follows a strict set of conventions

`Component` resources are “living”, however they are resistant to changes in their spec, They’re designed to be immutable on production servers, so that accidental spec changes do not break the API delivered to end users.

If a `Component`’s top level condition `Ready` is ever something other than `status: "True"` then the `reason` field should describe the problem with the component.

`status.details`

`status.details` describe some of the observations that Tanzu Supply Chains made about this component. These fields are used to improve the output of `kubernetes get component <name> -owide`, and they summarize the component for platform engineers when they author a `SupplyChain`.

```
status:
  conditions:
  - lastTransitionTime: "2024-02-25T00:40:46Z"
    message: ""
    reason: Ready
    status: "True"
    type: Ready
  details:
    hasResumptions: "False"
    inputs: conventions[conventions[]
    outputs: oci-yaml-files[oci-yaml-files[], oci-ytt-files[oci-ytt-files[]
  observedGeneration: 1
```

`status.docs`

`status.docs` contains a human-readable explanation of the content of the component. It’s useful for CLIs and UIs, and provides a clean summary of the component.

Example

```
docs: |
  # source

  Version: 1.0.0

  ## Description

  Monitors a git repository

  ## Inputs

  * _none_

  ## Outputs

  | Name  | Type |
  | ---   | ---  |
  | source | [source](./output-types.hbs.md#source) |

  ## Config

  ``yaml
```

```
  spec:
    source:
      # Fill this object in if you want your source to come from git.
      # The tag, commit and branch fields are mutually exclusive, use only one.
      git:
        # A git branch ref to watch for new source
        branch:
        # A git commit sha to use
        commit:
        # A git tag ref to watch for new source
        tag:
        # The url to the git source repository
        # +required
        url:
  ``
```

Workload CRD

This topic gives you reference information about the `Workload` resource for Tanzu Supply Chain.

Caution

Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.

`Workload` resources are custom resource definitions (CRDs) created by `SupplyChain` resources. They are also one of the two duck type resources in Tanzu Supply Chain.

Static CustomResourceDefinitions API

Every `Workload` resource is defined as a `CustomResourceDefinition`:

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
```

`metadata.labels`

`Workload` resources always have the following labels. The `chain-name` and `chain-namespace` labels reference the location of the SupplyChain resource that created this `Workload`. The `chain-role` identifies this as a Workload. The other possible value is `workload-run`.

```
metadata:
  labels:
    supply-chain.apps.tanzu.vmware.com/chain-name: apps.example.com-1.0.0
    supply-chain.apps.tanzu.vmware.com/chain-namespace: app-sc
    supply-chain.apps.tanzu.vmware.com/chain-role: workload
```

`metadata.name`

The name of the resource is always in the form `<plural>.<group>` from the Supply Chain Defines API.

Example

```
metadata:
  name: appv1s.widget.com
```

`spec.group`, `spec.names` and `spec.versions`

The CRD `group`, `names`, and `versions` are filled with the details found in the Supply Chain Defines API.

Additionally, the `spec.names[].categories[]` array includes a category of `all-workloads`. This ensures that commands such as `kubectl get all-workloads` find all the `SupplyChain`-defined `Workload` resources a user can access.

Example

```
spec:
  conversion:
    strategy: None
  group: example.com
  names:
    categories:
      - all-workloads
    kind: AppV1
    listKind: AppV1List
    plural: appv1s
    singular: appv1
  scope: Namespaced
  versions:
      name: v1alpha1
      schema:
        openAPIV3Schema:
          ...
```

Static Workload API

`metadata.labels`

`Workload` resources always have the following labels. These labels reference the location of the `SupplyChain` resource on cluster.

```
metadata:
  labels:
    supply-chain.apps.tanzu.vmware.com/chain-name: apps.example.com-1.0.0
    supply-chain.apps.tanzu.vmware.com/chain-namespace: app-sc
```

Dynamic Workload API

`spec`

The `spec` of a `Workload` resource is dynamic, however it is immutable after it is applied. The `spec` is derived by combining the Component configurations of all the SupplyChain Stages.

WorkloadRun CRD

This topic gives you reference information about the `WorkloadRun` resource for Tanzu Supply Chain.

Caution

> Tanzu Supply Chain is currently in beta and is not intended for production use. It is intended only for evaluation purposes of the next generation Supply Chain. For the current Supply Chain solution, see the Supply Chain Choreographer documentation.

`WorkloadRun` resources are custom resource definitions (CRDs) created by `SupplyChains`. They are also one of the two duck type resources in Tanzu Supply Chain.

Static CustomResourceDefinitions API

Every `WorkloadRun` resource is defined as a `CustomResourceDefinition`:

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
```

`metadata.labels`

Workload CRDs always have the following labels. The `chain-name` and `chain-namespace` labels reference the location of the SupplyChain resource that created this WorkloadRun. The `chain-role` identifies this as a WorkloadRun. The other possible value is `workload`.

```
metadata:
  labels:
    supply-chain.apps.tanzu.vmware.com/chain-name: apps.example.com-1.0.0
    supply-chain.apps.tanzu.vmware.com/chain-namespace: app-sc
    supply-chain.apps.tanzu.vmware.com/chain-role: workload-run
```

`metadata.name`

The name of the resource is always in the form `<singular>runs.<group>` from the Supply Chain Defines API.

Example

```
metadata:
  name: appv1runs.widget.com
```

`spec.group`, `spec.names` and `spec.versions`

The CRD's `group`, `names` and `versions` is filled in with the details found in the Supply Chain Defines API. However, most names have the word `run` appended.

Additionally, the `spec.names[].categories[]` array includes a category of `all-runs`. This ensures that commands such as `kubectl get all-runs` find all the `SupplyChain` defined `WorkloadRuns` a user can access.

Example

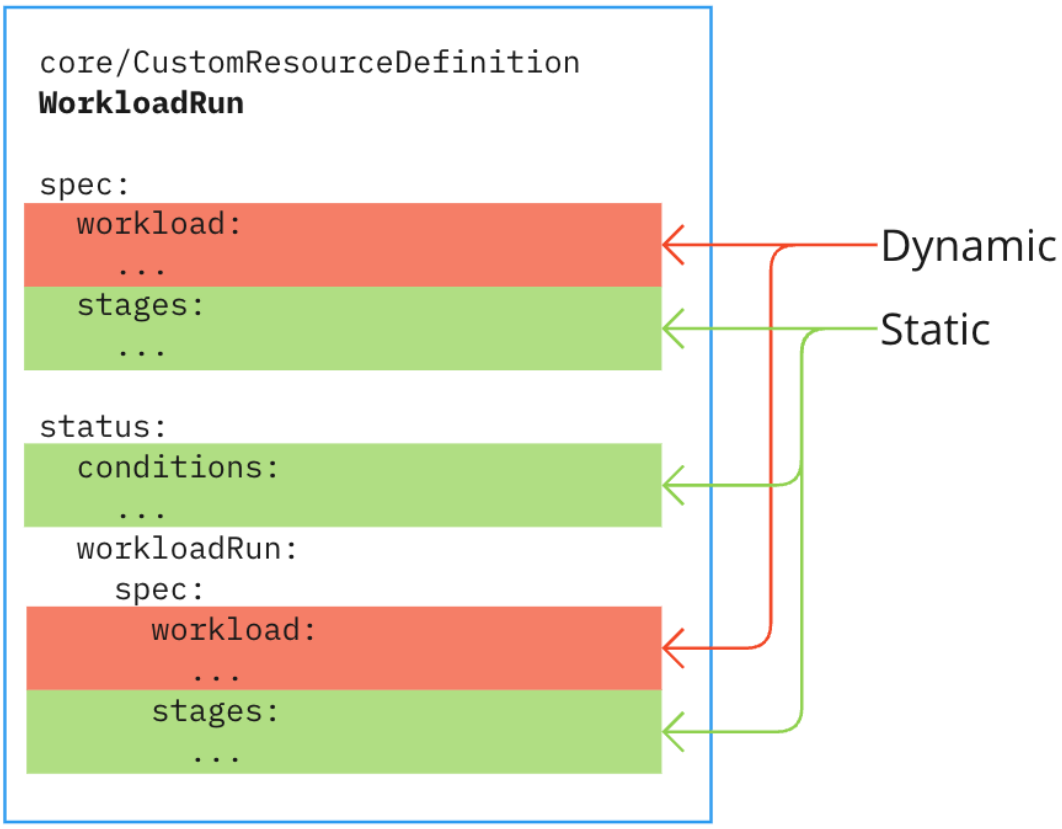
```
spec:
  conversion:
    strategy: None
  group: example.com
  names:
    categories:
      - all-runs
    kind: AppV1Run
    listKind: AppV1RunList
```

```
    plural: appv1runs
    singular: appv1run
  scope: Namespaced
  versions:
    name: v1alpha1
    schema:
      openAPIV3Schema:
        ...
```

Self-Replicating State

`WorkloadRuns` have a complex self-referencing status, that is described in detail in Core Concepts: WorkloadRuns. The majority of the `WorkloadRun` specification appears again in `status.workloadRun`.

This image shows the static and dynamic sections of a WorkloadRun.



Note

The duplication of the `WorkloadRun spec` into `spec.status.workloadRun.spec` is shown here.

The `status` is not duplicated again into the `status.workloadRun` field.

Static WorkloadRun API

`Static` means that the schema of these sections of the `WorkloadRun` is unchanging. This does not mean that these sections do not mutate during the life of the `WorkloadRun`. They do, because they track the progress of the `WorkloadRun`.

`spec.stages[]` and `status.workloadrun.spec.stages[]`

`spec.stages[]` is empty if the run was triggered by a new workload generation, or, if triggered by a Resumption, contains the stages up to but excluding the stage containing the resumption trigger.

`spec.workloadrun.spec.stages[]` initially contains a copy of `spec.stages[]` (if any exist), and as the `WorkloadRun` proceeds, contains the rest of the results for subsequent stages.

`spec.stages[].name` and `status.workloadrun.spec.stages[].name`

`name` is the name of this stage as defined in the SupplyChain.

`spec.stages[].componentRef` and `status.workloadrun.spec.stages[].componentRef`

The sub-fields, `name` and `namespace`, refer to the component that will/has run this stage.

```
name: source-1.0.0
namespace: test-basic
```

`spec.stages[].outputs[]` and `status.workloadrun.spec.stages[].outputs[]`

Each output contains the following fields:

Field	Description
`name:`	The name of the output, defined in the [Component] for this stage.
`type:`	The type of the output, defined in the [Component] for this stage.
`url:`	The url to the artifact where this output is stored. All outputs are stored as supply chain accessible resources (URLs)
`digest:`	A digest representing the state of the output.

`spec.stages[].pipeline` and `status.workloadrun.spec.stages[].pipeline`

Each stage has one `pipeline` object with the following fields:

Field	Description
`passed:`	Empty (`""`) means running, otherwise this is `false` for failed, and `true` for succeeded
`started:`	The Date/Time this pipeline started. Empty (`""`) if not started.
`completed:`	The Date/Time this pipeline finished. Empty (`""`) if still running.
`message:`	Output from the pipeline, which updates until the pipeline is completed.
`results:`	A copy of the [Tekton PipelineRun results].

Example

```
pipeline:
  passed: true
  started: "2024-02-25T19:31:48Z"
  completed: "2024-02-25T19:33:27Z"
  message: 'Tasks Completed: 3 (Failed: 0, Cancelled 0), Skipped: 0'
  results:
```

```
    - name: url
      value: registry.io/my-test/package-store@sha256:bad70a84441cfa49fea2a16a1fb0db31
48f260242cf31bb8fde9547c09ece4bf
    - name: digest
      value: bad70a84441cfa49fea2a16a1fb0db3148f260242cf31bb8fde9547c09ece4bf
```

Dynamic WorkloadRun API

`spec.workload` and `spec.status.workloadRun.spec.workload`

These are the same. Technically, `spec.workload` is the state of the `Workload` when the run is created, and `spec.status.workloadRun.spec.workload` is the state of the `Workload` after the run starts, but they never change, so are in fact identical.

Both contain the `workload.metadata` and `workload.spec` sections of the workload that were used in this run. The `WorkloadRun` "closes over" this state so that it cannot be lost. When viewing a run, you can always tell which `Workload` resource's state was used during the run.

Status

`status.conditions[]`

Every `status.conditions[]` in Tanzu Supply Chain resources follows a strict set of conventions

The top-level condition type is `Succeeded` because `Workload` is a batch resource.

The sub-types are:

PipelinesSucceeded

Reason	Meaning
Succeeded	All [Tekton PipelineRuns] are complete
Running	[Tekton PipelineRuns] are not all complete.
Failed	A [Tekton PipelineRun] failed, and it's likely that the developer can remedy any issues by following the guidance in the message.
PlatformFailed	A [Tekton PipelineRun] failed, and it's unlikely the problem can be remedied with changes to the workload or developer provided input (such as source).

`Message` contains processing information and error messages produced in the pipeline. This information must be specifically appended to the Tekton result named `message` to appear here.

ResumptionsSucceeded

Reason	Meaning
Succeeded	All [Tekton TaskRuns] for all resumptions are complete.
Running	A [Tekton TaskRun] for a resumption is incomplete.
Failed	A [Tekton TaskRun] for a resumption failed, and it's likely that the developer can remedy any issues by following the guidance in the message.
PlatformFailed	A [Tekton TaskRun] for a resumption failed, and it's unlikely the problem can be remedied with changes to the workload or developer provided input (such as source).

`Message` contains processing information and error messages produced in the taskRun. This information must be specifically appended to the Tekton result named `message` to appear here.

Resource Statuses

This topic tells you about the statuses for Tanzu Supply Chain resources.

Every status in Tanzu Supply Chain resources adheres to the same set of rules, starting with those set out by Kubernetes API Conventions.

The `spec.conditions` array is referred to as a Condition Set.

Each Condition Set has a top level condition type selected based on the behavior of the resource, Living and Batch.

Living resource

This resource is never “finished”, instead it modifies it’s managed resources or internal services to match the state of the `spec`. Examples include `services` and `pods`.

A living resource’s top-level condition type is Ready.

However, some Living resources are resistant to changes in their spec. They are “living and immutable”. For example, the Component Resource.

Batch resource

Batch resources complete. They’re expected to terminate after they have achieved the desired outcome. Typically their `spec` is immutable, or if it is mutable, the mutation has no effect.

A batch resource’s top-level condition type is `Succeeded`.

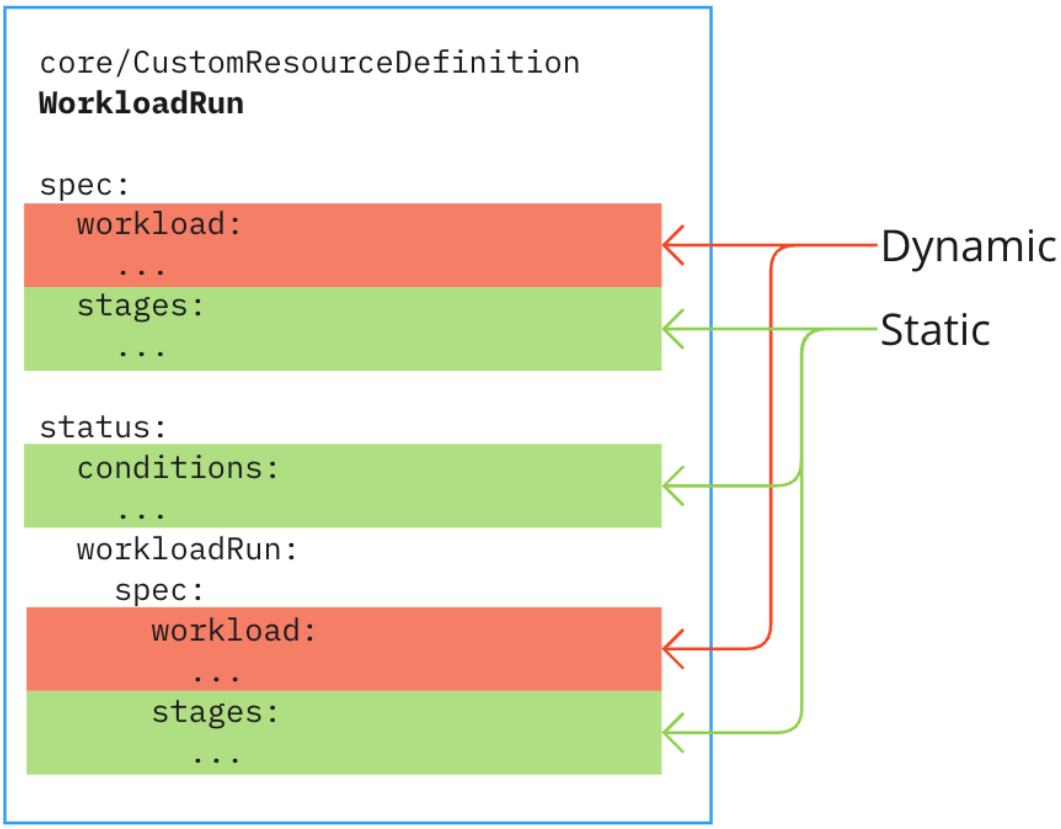
For more information, see the Kubernetes documentation.

Duck type resources

This topic tells you about the duck type resources for Tanzu Supply Chain.

The two duck type resources in Tanzu Supply Chains are Workloads and WorkloadRuns.

In brief, a duck type resource is a resource with significant commonality. A duck type resource is similar to a Class in object-oriented design, but without the inheritance.



Some sections of a duck type API are unchanging (static) and others can and do change (dynamic).

Both Tanzu Supply Chain duck types have dynamic kinds, which means there can be many Kubernetes CRDs that comply with the `WorkloadRun` duck type.

Overview of Tekton

Tekton is a cloud-native, open-source framework for creating CI/CD systems. It allows developers to build, test, and deploy across cloud providers and on-premises systems. For more information about Tekton, see the Tekton documentation.

Overview of Tekton

Tekton is a cloud-native, open-source framework for creating CI/CD systems. It allows developers to build, test, and deploy across cloud providers and on-premises systems. For more information about Tekton, see the Tekton documentation.

Install Tekton

This topic tells you how to install Tekton Pipelines from the Tanzu Application Platform package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Tekton Pipelines. For more information about profiles, see Components and installation profiles.

Prerequisites

Before installing Tekton Pipelines, complete all prerequisites to install Tanzu Application Platform.

Install Tekton Pipelines

To install Tekton Pipelines:

1. See the Tekton Pipelines package versions available to install by running:

   ```
   tanzu package available list -n tap-install tekton.tanzu.vmware.com
   ```

   For example:

   ```
   $ tanzu package available list -n tap-install tekton.tanzu.vmware.com
   \ Retrieving package versions for tekton.tanzu.vmware.com...
     NAME                     VERSION  RELEASED-AT
     tekton.tanzu.vmware.com  0.30.0   2021-11-18 17:05:37Z
   ```

2. Install Tekton Pipelines by running:

   ```
   tanzu package install tekton-pipelines -n tap-install -p tekton.tanzu.vmware.co
   m -v VERSION
   ```

   Where `VERSION` is the desired version number. For example, `0.30.0`.

   For example:

   ```
   $ tanzu package install tekton-pipelines -n tap-install -p tekton.tanzu.vmware.
   com -v 0.30.0
   - Installing package 'tekton.tanzu.vmware.com'
   \ Getting package metadata for 'tekton.tanzu.vmware.com'
   / Creating service account 'tekton-pipelines-tap-install-sa'
   / Creating cluster admin role 'tekton-pipelines-tap-install-cluster-role'
   / Creating cluster role binding 'tekton-pipelines-tap-install-cluster-rolebindi
   ng'
   / Creating package resource
   - Waiting for 'PackageInstall' reconciliation for 'tekton-pipelines'
   - 'PackageInstall' resource install status: Reconciling


    Added installed package 'tekton-pipelines'
   ```

3. Verify that you installed the package by running:

   ```
   tanzu package installed get tekton-pipelines -n tap-install
   ```

   For example:

   ```
   $ tanzu package installed get tekton-pipelines -n tap-install
   \ Retrieving installation details for tekton...
   NAME:                    tekton-pipelines
   PACKAGE-NAME:            tekton.tanzu.vmware.com
   PACKAGE-VERSION:         0.30.0
   STATUS:                  Reconcile succeeded
   CONDITIONS:              [{ReconcileSucceeded True  }]
   USEFUL-ERROR-MESSAGE:
   ```

   Verify that `STATUS` is `Reconcile succeeded`.

Configure a namespace to use Tekton Pipelines

This section covers configuring a namespace to run Tekton Pipelines. If you rely on a SupplyChain to create Tekton PipelinesRuns in your cluster, skip this step because namespace configuration is covered in Set up developer namespaces to use your installed packages. Otherwise, perform the steps in this section for each namespace where you create Tekton Pipelines.

Service accounts that run Tekton workloads need access to the image pull secrets for the Tanzu package. This includes the `default` service account in a namespace, which is created automatically but is not associated with any image pull secrets. Without these credentials, PipelineRuns fail with a timeout and the pods report that they cannot pull images.

Important

If you are using a registry with a custom CA certificate then you must provide this certificate to Tekton directly by including the CA in the service account used by the supply chain.

The Tanzu Application Platform distribution of Tekton does not support the Tanzu Application Platform field `shared.ca_cert_data`. For more information about setting the CA in the service account, see Use Git authentication with Supply Chain Choreographer.

To configure a namespace to use Tekton Pipelines:

1. Create an image pull secret in the current namespace and fill it from the `tap-registry` secret. For more information, see Relocate images to a registry.
2. Create an empty secret, and annotate it as a target of the secretgen controller, by running:

   ```
   kubectl create secret generic pull-secret --from-literal=.dockerconfigjson={} --type=kubernetes.io/dockerconfigjson
   kubectl annotate secret pull-secret secretgen.carvel.dev/image-pull-secret=""
   ```

3. After you create a `pull-secret` secret in the same namespace as the service account, add the secret to the service account by running:

   ```
   kubectl patch serviceaccount default -p '{"imagePullSecrets": [{"name": "pull-secret"}]}'
   ```

4. Verify that a service account is correctly configured by running:

   ```
   kubectl describe serviceaccount default
   ```

   For example:

   ```
   kubectl describe sa default
   Name:                default
   Namespace:           default
   Labels:              <none>
   Annotations:         <none>
   Image pull secrets:  pull-secret
   Mountable secrets:   default-token-xh6p4
   Tokens:              default-token-xh6p4
   Events:              <none>
   ```

Note

The service account has access to the `pull-secret` image pull secret.

For more details about Tekton Pipelines, see the Tekton documentation and the GitHub repository.

For information about getting started with Tekton, see the Tekton tutorial in GitHub and the getting started guide in the Tekton documentation.

Note

Windows workloads are deactivated and cause an error if any Tasks try to use Windows scripts.