Tanzu Application Platform v1.2

VMware Tanzu Application Platform 1.2

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

https://docs.vmware.com/

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

Contents

Tanzu Application Platform v1.2

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.

Tanzu Application Platform overview

Tanzu Application Platform:

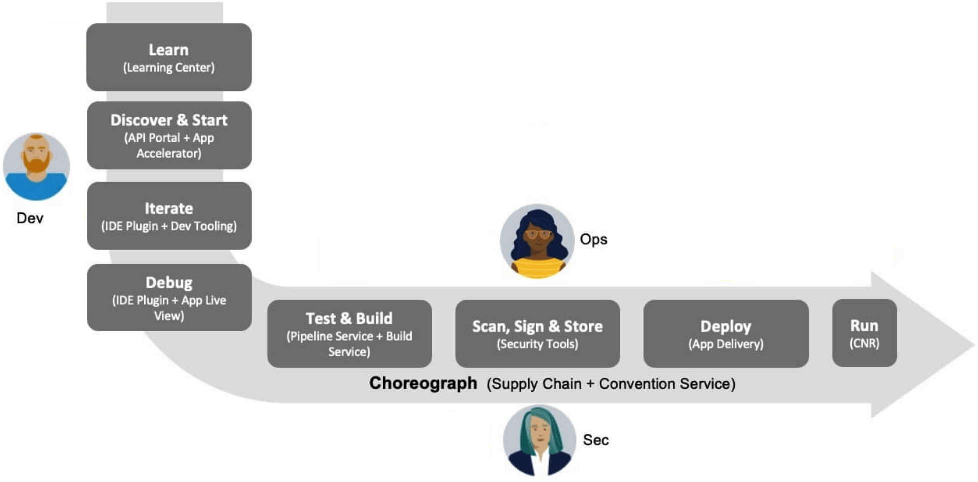
- Delivers a superior developer experience for enterprises building and deploying cloud-native applications on Kubernetes.
- Allows developers to quickly build and test applications regardless of their familiarity with Kubernetes.
- Helps application teams get to production faster by automating source-to-production 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.


Learn (Learning Center)
Discover & Start (API Portal + App Accelerator)
Iterate (IDE Plugin + Dev Tooling)
Debug (IDE Plugin + App Live View)
Dev
Ops
Test & Build (Pipeline Service + Build Service)
Scan, Sign & Store (Security Tools)
Deploy (App Delivery)
Run (CNR)
Choreograph (Supply Chain + Convention Service)
Sec

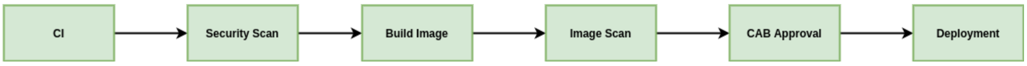
- **Inner Loop**
  - The inner loop describes a developer's development cycle of iterating on code.
  - Inner loop activities include coding, testing, and debugging before making a commit.

  - 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.

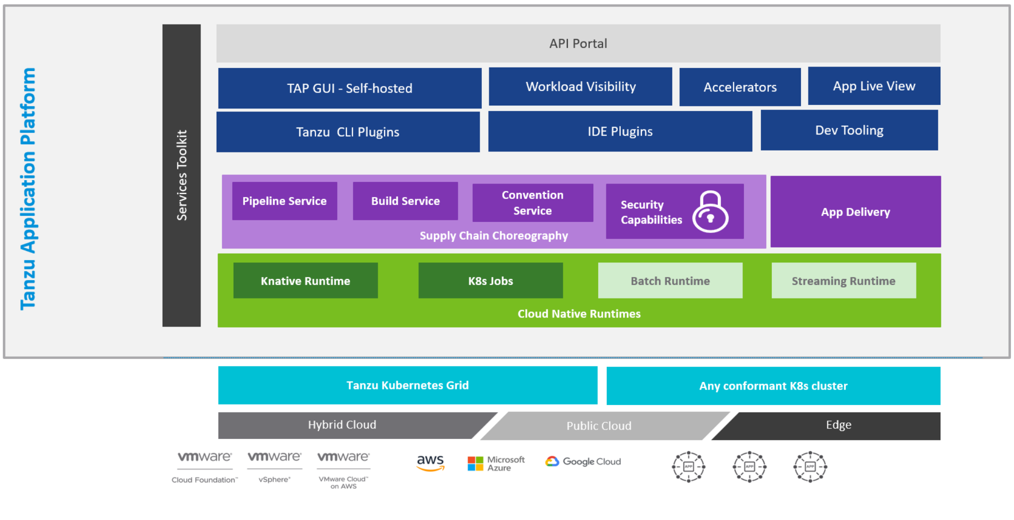


To address the developer experience gap, the path to production allows users to create a unified access point for all of the tools required for their applications to reach a customer-facing environment.

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.

Tanzu Application Platform: Layered API & Capabilities



For more information about Tanzu Application Platform components, see About Tanzu Application Platform components and 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.

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 portal for VMware Tanzu**

  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 to appear in a single instance.

- **Application Accelerator for VMware Tanzu**

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

  Accelerators are templates that codify best practices and ensure important configurations and structures are in place from the start. 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 Live View for VMware Tanzu**

  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 for VMware Tanzu**

  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.

- **Cloud Native Runtimes for VMware Tanzu**

  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.

- **Convention Service for VMware Tanzu**

  Convention Service provides a means for people in operational roles to express their hard-won knowledge and opinions about how apps run on Kubernetes as a convention. The convention service applies these opinions to fleets of developer workloads as they are deployed to the platform, saving time for operators and developers.

- **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. Tanzu Application Platform GUI support for default roles is planned for a future release.

- **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.

- **Flux CD Source Controller**

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

- **Grype**

  Grype is a vulnerability scanner for container images and file systems.

- **Services Toolkit for VMware Tanzu**

  Services Toolkit comprises a number of Kubernetes-native components that support the management, life cycle, discoverability, and connectivity of Service Resources (databases, message queues, DNS records, and so on) on Kubernetes.

- **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.

- **Supply Chain Choreographer for VMware Tanzu**

  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.

- **Supply Chain Security tools for Tanzu - Scan**

  With Supply Chain Security Tools for VMware Tanzu - 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 - Policy Controller**

  Supply Chain Security Tools - 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 - Store**

  Supply Chain Security Tools - 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 Application Platform GUI**

  Tanzu Application Platform GUI 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 Application Platform GUI is built from the Cloud Native Computing Foundation's project Backstage.

- **Tanzu Build Service**

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

  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 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 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 Learning Center**

  Learning Center provides a platform for creating and self-hosting workshops. With Learning Center, content creators can create workshops from markdown files that learners can view in a terminal shell environment with an instructional wizard UI. The UI can embed slide content, an integrated development environment (IDE), a web console for accessing the Kubernetes cluster, and other custom web applications.

  Although Learning Center requires Kubernetes to run, and it teaches users about Kubernetes, you can use it to host training for other purposes as well. For example, you can use it to train users on web-based applications, use of databases, or programming languages.

- **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.

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

Installation profiles in Tanzu Application Platform v1.3

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 all of the Tanzu Application Platform packages.
- **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 Application Platform GUI and Metadata Store.

The following table lists the packages contained in each profile:

Capability Name	Full	Iterate	Build	Run	View
API Portal	✓				✓
Application Accelerator	✓				✓
Application Live View (Build)	✓	✓	✓		
Application Live View (Run)	✓	✓		✓	
Application Live View GUI Backend	✓				✓
Application Single Sign-On	✓	✓		✓	
Cloud Native Runtimes	✓	✓		✓	
Convention controller	✓	✓	✓		
Default Roles	✓	✓	✓	✓	
Developer Conventions	✓	✓			
Flux Source Controller	✓	✓	✓	✓	✓
Grype	✓		✓		
Learning Center	✓				✓
Out of the Box Delivery - Basic	✓	✓		✓	
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	✓	✓	✓	✓	
Policy Controller	✓	✓		✓	
Service Bindings	✓	✓		✓	
Services Toolkit	✓	✓		✓	
Source Controller	✓	✓	✓	✓	✓
Spring Boot Convention	✓	✓	✓		
Supply Chain Choreographer	✓	✓	✓	✓	
Supply Chain Security Tools - Policy Controller	✓	✓		✓	
Supply Chain Security Tools - Scan	✓		✓		
Supply Chain Security Tools - Store	✓				✓
Tanzu Build Service	✓	✓	✓		

Tanzu Application Platform GUI	✓				✓
Tekton Pipelines	✓	✓	✓		
Telemetry	✓	✓	✓	✓	✓

Note: Only one supply chain can be installed at any given time. For information on switching supply chains, see Add testing and security 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 for VMware Tanzu	Functions (beta)	Extended Scanning Coverage using Anchore Grype	Application Accelerators for VMware Tanzu
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: SCST - Scan and Store using Anchore Grype. Out of the Box Tanzu Application Platform scanning leverages a tool by Anchore called Grype. Grype provides standard CVE scanning support for a wide variety of languages. However, if you use Tanzu Build Service to build application images using a buildpack that produces a Bill of Materials in the Syft format, Tanzu Application Platform scanning can provide a more comprehensive scan of the application image.

Note

Users can leverage the open source Paketo Ruby buildpack for building Ruby applications with Tanzu Build Service.

Installing Tanzu Application Platform

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

Tanzu Application Platform release notes

This topic describes the changes in Tanzu Application Platform (commonly known as TAP) v1.2.

v1.2.2

Release Date: September 13, 2022

New features

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

Supply Chain Security Tools - Store

- Modified the vulnerability response in the `tanzu insight` CLI plug-in to only return the highest severity rating for a CVE.

Resolved issues

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

Supply Chain Security Tools - Scan

- Resolved blob source scan reporting wrong source URL to the metadata store when the .git file does not exist.

Supply Chain Security Tools - Store

- Resolved an issue where Store could not handle new method types.
- Resolved an issue where Store could not handle blob URLs in component names.

Tanzu Application Platform GUI

- Updated supply-chain package version to v0.1.26 to fix an issue in the Image Scanner Stage.

Known issues

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

Upgrading Tanzu Application Platform

- **Adding the v1.2.2 repository bundle in addition to another repository might cause a failure:**
  - While upgrading to Tanzu Application Platform v1.2.2 from any previous version, adding the v1.2.2 repo bundle in addition to the existing repo bundle can fail. For the workaround, see Troubleshoot installing Tanzu Application Platform.
  - You might observe an error with package installs `ReconcileFailed True Expected to find at least one version` until Tanzu Application Platform is upgraded to v1.2.2, but this does not affect the functionality of any components.

Grype scanner

- **Scanning Java source code that uses Gradle package manager may not reveal vulnerabilities:**
    - For most languages, source code scanning only scans files present in the source code repository. Except for support added for Java projects using Maven, no network calls are made to fetch dependencies. For languages using dependency lock files, such as Golang and Node.js, Grype uses the lock files to check the dependencies for vulnerabilities.
    - For Java using Gradle, dependency lock files are not guaranteed, so Grype uses the dependencies present in the built binaries (`.jar` or `.war` files) instead.
    - Because VMware does not encourage committing binaries to source code repositories, Grype fails to find vulnerabilities during a source scan. The vulnerabilities are still found during the image scan after the binaries are built and packaged as images.

Supply Chain Security Tools - Policy Controller

This issue is also present in previous releases of Supply Chain Security Tools - Policy Controller.

- `kubectl run` **pods fail to validate in non-default namespaces:**
    - When policy verification occurs on an image deployed through `kubectl run` on a non-default namespace, the verification will fail to create the keychain required if the image requires credentials.

Learning Center

- **session.objects, environment.objects, and session.patches are not deployed:**
    - Due to a security improvement in Learning Center, session.objects, environment.objects, and session.patches are not working properly. VMware resolved this issue by Learning Center `v0.2.4` in TAP 1.3.2.

v1.2.1

Release Date: August 9, 2022

New features

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

Tanzu Build Service

- Improved error messaging.
- Removed noisy logging from AWS credential helper.

Resolved issues

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

Tanzu Application Platform GUI

- Supply Chain plug-in:
    - ConfigMap has no conditions and as a result its status is `Unknown`.
    - ConfigWriter shows an error but no error details are displayed.
    - Kaniko-based image builds cannot show data in the UI.
    - Need to refresh browser to show successful or error messages.
- Runtime Resource Visibility plug-in: The Maven artifacts access error is fixed.

Known issues

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

Tanzu Application Platform

- **Unable to add Tanzu Application Platform repo into clusters attached to Tanzu Mission Control with pre-installed Cluster Essentials v1.2:** For the solution, see Troubleshoot installing Tanzu Application Platform.

Supply Chain Security Tools - Scan

- **Blob source scan is reporting wrong source URL:** When running a source scan of a blob compressed file, it looks for a `.git` directory present in the files to extract information that is useful for the report sent to the Supply Chain Security Tools - Store deployment. This problem happens when you use Grype Scanner ScanTemplates earlier than version `v1.2.0` because the Scan Controller has a deprecated path to support previous ScanTemplates. VMware plans to resolve this issue by Supply Chain Security Tools - Scan `v1.3.0`. For the solution, see Observability and troubleshooting.

Grype scanner

- **Scanning Java source code that uses Gradle package manager may not reveal vulnerabilities:**
  - For most languages, source code scanning only scans files present in the source code repository. Except for support added for Java projects using Maven, no network calls are made to fetch dependencies. For languages using dependency lock files, such as Golang and Node.js, Grype uses the lock files to check the dependencies for vulnerabilities.
  - For Java using Gradle, dependency lock files are not guaranteed, so Grype uses the dependencies present in the built binaries (`.jar` or `.war` files) instead.
  - Because VMware does not encourage committing binaries to source code repositories, Grype fails to find vulnerabilities during a source scan. The vulnerabilities are still found during the image scan after the binaries are built and packaged as images.

v1.2.0

Release Date: July 12, 2022

New features

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

Application Accelerator

- Accelerator fragments are now available.
  - Allows for re-usable accelerator fragments to be imported into other accelerators.
- Tanzu Application Accelerator for VS Code Extension is now available.
  - Allows for developers to quickly generate projects from their organization's Accelerator Catalog right from within VS Code.
- Added the `subPath` field for Accelerators and Fragments.
  - Provides the option for altering the location of the root of an accelerator or fragment.

Application Live View

- Live Hover Integration with Spring Tools Suite:

  - Users can hover over Spring Actuator endpoints to see live data. You can enable this feature from Preferences.

Application Single Sign-On

- Application Single Sign-On package comes installed with iterate, run, and full profiles.
- Secure a workload with AppSSO. For more information, see AppSSO documentation.
- AppSSO Starter Java Accelerator shows how to enable SSO on a Spring Boot application.
- OpenID Connect Identity Providers are supported.
- Grant types supported: authorization code, client credentials, refresh token.
- Audit logs for troubleshooting.
- Secure tokens - Token signature keys are created and applied to AuthServer so that tokens can be signed and verified.
- TLS secured.

Tanzu CLI - Apps plug-in

- Added support for `--sub-path` flag where users can specify a relative path inside the repository or image to treat as application root for source.
- Added `--service-account` flag to specify ServiceAccount name used by the workload to create resources submitted by the supply chain.
- Added shorthand `-s` for `--source-image` flag.
- Added support for `--output` flag to `workloads list` command.
- Added support for JSON or YAML params using new flag `--param-yaml`.
- Added support for creating workloads from JAR, WAR, and ZIP files through the `--local-path` flag.
- Added source information from workload in the `workload get` command output.
- Added new command `tanzu apps cluster-supply-chain get`.
- Added support for excluding files on local path using `.tanzuignore` file.
- Added supply chain step information in `workload get` command output.
- Added support for short names for Cartographer workload (wld) and cluster-supply-chain commands (csc).
- Added support for providing ServiceAccount name in workload commands through file input.

Source Controller

- Added support for pulling Artifacts from a Maven repository using the `MavenArtifact` CR.

  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.

Snyk Scanner (beta)

- Snyk scanner image scanning integration (Beta) is available for Supply Chain Security Tools - Scan.
  - See Snyk Installation and Configuration Guide for instructions on how to use Snyk with Tanzu Application Platform Supply Chains.

Supply Chain Choreographer

- View resource status on a workload:
  - Added ability to indicate how Cartographer can read the state of the resource and reflect it on the owner status.
  - Surfaces information about the health of resources directly on the owner status.
  - Adds a field in the spec `healthRule` where authors can specify how to determine the health of the underlying resource for that template. The resource can be in one of the following states: A stamped resource can be in one of three states: `Healthy` (status True), `Unhealthy` (status False), or `Unknown` (status Unknown). If no healthRule is defined, Cartographer defaults to listing the resource as `Healthy` once it is successfully applied to the cluster and any outputs are read off the resource.
- Cartographer Conventions v0.1.0 is now bundled with Supply Chain Choreographer.
  - As of v0.07.0 release of Convention Controller, its APIs are deprecated in favor of continuing development on Cartographer Conventions. Cartographer Conventions is now bundled with Supply Chain Choreographer.

Supply Chain Security Tools - Scan

- Scan-Link's controller abstraction from the scanners' output format allows more flexibility when you integrate new scanners.
- Supply Chain Security Tools - Scan is decoupled from the Supply Chain Security Tools - Store to ease future integration with different storage methods.
- Beta scanner support released in the Snyk Scanner package.
- Documentation is available on how to use Grype in offline and air-gapped environments.

Note: The Grype Scanner `ScanTemplate`s shipped with versions before Supply Chain Security Tools - Scan `v1.2.0` are now deprecated and are no longer supported in future releases. See Upgrading Supply Chain Security Tools - Scan for step-by-step instructions.

Supply Chain Security Tools - Sign

- Updated cosign to v1.9.0.
- Fixed resources without namespace defined causing errors.

Important: Supply Chain Security Tools - Sign is being deprecated and is being replaced by Supply Chain Security Tools - Policy Controller. Supply Chain Security Tools - Sign is no longer supported after Tanzu Application Platform v1.4.0. See Migration From Supply Chain Security Tools - Sign for migration instructions.

Supply Chain Security Tools - Policy Controller

- Initial release of Policy Controller, which uses Sigstore Policy Controller.

Supply Chain Security Tools - Store

- Added more accepted vulnerability method types (CVSSv31, OWASP).
- Updated logging format to follow the Logging RFC recommendations.
- Bumped PostgreSQL and paketo images to fix CVE-2022-1292.
- Added support for insight plug-in to consume vulnerabilities through VEX in CycloneDX 1.4 reports.
- Added support for insight plug-in to consume SPDX 2.2/3.0 reports and introduced the new `--spdxtype` option to the `tanzu insight image add` and `tanzu insight source add` commands.
- Changed insight plug-in text response to return only highest CVE.
- Added aliases for insight plug-in vulnerabilities command.

Tanzu Application Platform GUI

Plug-in improvements and additions include:

- Runtime Resources Visibility plug-in:
  - Added support for pod logs and the ability to change log levels (where application live view is supported).
  - Added memory and CPU limit configuration.
  - Added quick links to access app memory and threads usage.
  - Added additional current status information when viewing runtime resources.
  - Added Tanzu Workload integration with a workload detail page for all runtime resouces.
  - Added support for Supply Chain resources.
  - UX updates to the Runtime Resource landing page.
- Supply Chain plug-in:
  - Added ability to visualize CVE scan results in the Details pane for both Source and Image Scan stages, as well as scan policy information without using the CLI.
  - Added ability to visualize the deployment of a workload as a deliverable in a multicluster environment in the supply chain graph.
  - Added a deeplink to view approvals for PRs in a GitOps repository so that PRs can be reviewed and approved, resulting in the deployment of a workload to any cluster configured to accept a deployment.
  - Added Reason column to the Workloads table to indicate causes for errors encountered during supply chain execution.
  - Added links to a downloadable log output for each execution of the Test and Build stages of the out of the box supply chains to enable more enhanced troubleshooting methods for workloads.

Tanzu Developer Tools for VS Code

- **View workload statuses:** You can see the status of your workloads from the Workloads panel. For more information, see Using the Tanzu Dev Tools Extension.
- **Apply and Delete workload commands:** You can run **Tanzu: Apply Workload** and **Tanzu: Delete Workload** from the Command Palette. For more information, see Using the Tanzu Dev Tools Extension.
- **Live Hover Integration with Spring Tools Suite:** You can point to Spring annotations and see the live data from a running Spring Boot application by way of Spring Boot Actuator endpoints. For more information, see Live Hover integration with Spring Boot Tools (Experimental).

Tanzu Developer Tools for IntelliJ

- **Live Update and Debug your workloads:** The new IntelliJ extension enables you to Live Update and Debug your workloads. For more information, see VMware Tanzu Developer Tools for IntelliJ.

Tanzu Build Service

- Updates to dependencies are now provided as part of Tanzu Application Platform patches.
- The automatic dependency update feature is deprecated. VMware discourages configuring Tanzu Application Platform with automatic dependency updates due to compatibility risks. This feature is still supported until stated otherwise.

Services Toolkit

- Services Toolkit now integrates with Amazon RDS using the ACK Operator or Crossplane. For more information, see the Services Toolkit documentation.
- New `ClusterInstanceClass` supports service instance abstraction. It is available using `tanzu service classes list` in v0.3.0 of the Services plug-in for Tanzu CLI.
- Claimable resources are now discoverable through the `InstanceQuery` API. It is available using `tanzu service claimable list --class CLASS-NAME` in `v0.3.0` of the Services plug-in for Tanzu CLI.
- ResourceClaims now aggregate on ClusterRoles for service resources with the standard `servicebinding.io/controller: "true"` label from the Service Binding specification for Kubernetes.
- Deprecation warning: `tanzu service types list` and `tanzu service instances list` commands are now deprecated. These commands are hidden from help text but remain functional if invoked. VMware intends to continue to support these commands for either two additional minor releases (v0.6.0 of the CLI plug-in) or after one year (2023-07-12), whichever comes later. VMware recommends using `tanzu service class` and `tanzu service claimable` commands in place of `tanzu service type` and `tanzu service instance` from now on.

Breaking changes

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

Application Accelerator

- App Accelerator now ships with Open Rewrite 7.24.0 (up from 7.21.x in TAP 1.1). As a consequence, some configuration properties of the OpenRewriteRecipe transform may need to be revised. For example, when using the `ChangePackage` recipe.

Supply Chain Security Tools - Scan

- You must configure integration with Supply Chain Security Tools - Store for the Grype Scanner and Snyk Scanner packages to enable this feature. The configuration for Supply Chain Security Tools - Store in Supply Chain Security Tools - Scan is only for the deprecated Grype Scanner `ScanTemplate`s.
- For the profile configuration of Supply Chain Security Tools - Scan, the scanning component no longer takes the metadata store configurations as of v1.2.0.
  - For information about configuring metadata store by using the Grype component instead of the scanning component as of v1.2.0, see Install multicluster Tanzu Application Platform profiles .
  - See Build profile for the deprecated way of writing the metadata store configuration through the scanning component.

  Note: This doesn't apply if you are using the deprecated Grype Scanner `ScanTemplate`s prior to Grype Scanner `v1.2.0`.

  - The package name changed from `package policies` to `package main`.
  - The deny rule changed from the boolean `isCompliant` to the array of strings `deny[msg]`.
  - The sample `ScanPolicy` is different if you're using Grype Scanner with a CycloneDX structure or Snyk Scanner with a SPDX JSON structure. See Install Snyk Scanner for an example of a Scan Policy.
  - See Enforce compliance policy using Open Policy Agent for an example of the current ScanPolicy format for v1.2.0 and later.

Tanzu Build Service

Note: If your Tanzu Application Platform v1.1 installation is configured with `enable_automatic_updates: false`, you can ignore this breaking change.

- When upgrading Tanzu Application Platform to v1.2, Tanzu Build Service image resources automatically run a build that fails due to a missing dependency. This error does not persist and subsequent builds automatically resolve this error. Users can safely wait for the next build of their workloads, which is triggered by source code changes. To manually re-run builds, follow the instructions in the troubleshooting item Builds fail after upgrading to Tanzu Application Platform v1.2.

Grype Scanner

- Provide information to integrate with the Supply Chain Security Tools - Store in the `tap-values.yaml` file for the Grype Scanner v1.2 and later.

Resolved issues

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

Application Accelerator

- Limit server logging to startup and generate zip requests.
- Update engine to use Spring Boot v2.7.0.

Supply Chain Security Tools - Scan

- `Go` updated to v1.18.2.
- `Open Policy Agent` updated to v0.40.0.

Grype Scanner

- `ncurses` updated to v6.1-5.ph3.

Tanzu CLI - Apps plug-in

- Updated output for list when there are no workloads. It now shows a more user-friendly message `No workloads found.`
- Fixed error messaging for empty kubeconfig and invalid kube context.
- Fixed incorrect error message for `workload create` when the user did not have enough permissions to create a workload.
- Removing namespace from `--service-ref` is not ignored.
- Issue for Windows error x509: certificate signed by unknown authority by upgrading imgpkg v0.29.0. The new version supports loading Windows root CA certificates.

Services Toolkit

- ResourceClaims no longer mutate service resources with an annotation to mark a claimed resource.
- ResourceClaims no longer require the `update` permission when adding new service resources to Tanzu Application Platform.

Service Bindings

- Added a new ClusterRole `service-binding-provisioned-services` with label selector `servicebinding.io/controller: "true"` for get, list, and watch, which fixes the issue where

Service Binding controller was aggregating non-provisioned service RBAC to the controller manager.

Spring Boot Conventions

- No environment variables are added if conventions are not applied. Fixes the issue where `JAVA_TOOL_OPTS` was added to non-JAVA apps.
- Controller does not error out if no image metadata is present. Fixes the edge case when the image metadata is missing.

Tanzu Application Platform GUI

- Supply Chain plug-in:
  - **Details for ConfigMap CRD now appear as expected:** The error `Unable to retrieve conditions for ConfigMap...` no longer appears in the details section after clicking on the ConfigMap stage in the graph view of a supply chain.
  - **Scan results now appear as expected:** Current CVEs found during Image or Source scanning now appear as expected.

Known issues

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

Tanzu Application Platform

- **Failure to connect to AWS EKS clusters:** When connecting to AWS EKS clusters, an error might appear with the text `Error: Unable to connect: connection refused. Confirm kubeconfig details and try again` or `invalid apiVersion "client.authentication.k8s.io/v1alpha1"`. To prevent this, see Failure to connect to AWS EKS clusters.
- **Failure to add Tanzu Application Platform repo:** Unable to add Tanzu Application Platform repo into clusters attached to Tanzu Mission Control with pre-installed Cluster Essentials v1.2. For the solution, see Troubleshoot installing Tanzu Application Platform.

Application Live View

- Application Live View with custom CA does not support air-gapped installation.

Application Single Sign-On

- Application Single Sign-On with custom CA does not support air-gapped installation.

Convention Service

- **Issue:** If the self-signed certificate authority (CA) for a registry is provided through `convention-controller.ca_cert_data`, it is not successfully propagated to the convention service. For the solution, see Troubleshoot Convention Service.

Functions (beta)

- When using Live Update, hot reload of your function on your cluster might not display changes made to your function. To manually push changes to the cluster, run the `tilt up` command.

Supply Chain Security Tools - Scan

- **Blob Source Scan is reporting wrong source URL:** When running a Source Scan of a blob compressed file, it looks for a `.git` directory present in the files to extract information that is usefull for the report sent to the Supply Chain Security Tools - Store deployment. This problem happens when you use Grype Scanner ScanTemplates earlier than version `v1.2.0` because the Scan Controller has a deprecated path to support previous ScanTemplates. This will be removed by Supply Chain Security Tools - Scan `v1.3.0`. For the solution, see Observability and troubleshooting.

Grype scanner

- **Scanning Java source code that uses Gradle package manager may not reveal vulnerabilities:**
    - For most languages, Source Code Scanning only scans files present in the source code repository. Except for support added for Java projects using Maven, no network calls are made to fetch dependencies. For languages using dependency lock files, such as Golang and Node.js, Grype uses the lock files to check the dependencies for vulnerabilities.
    - For Java using Gradle, dependency lock files are not guaranteed, so Grype uses the dependencies present in the built binaries (`.jar` or `.war` files) instead.
    - Because VMware does not encourage committing binaries to source code repositories, Grype fails to find vulnerabilities during a Source Scan. The vulnerabilities are still found during the Image Scan after the binaries are built and packaged as images.

Tanzu Application Platform GUI

- **Tanzu Application Platform GUI doesn't work in Safari:** Tanzu Application Platform GUI does not work in the Safari web browser.
- **Supply Chain plug-in:**
    - The delivery section of the supply chain graph might show deliverables that do not pertain to the selected workload. This occurs if there is more than one `Build` cluster per namespace.
    - For `Deliverables` to show up for a `Workload`, they must have the following labels in both resources: `carto.run/workload-name`, `app.kubernetes.io/part-of`, and `carto.run/supply-chain-name`.
    - ConfigMap has no conditions and as a result its status is `Unknown`.
    - ConfigWriter shows an error but no error details are displayed.
    - You might receive the error `TypeError: Cannot read properties of undefined (reading 'data')` when viewing a workload in a supply chain. Use the CLI tools instead to view the status of the workload in the supply chain.
- **Back-end Kubernetes plug-in reporting 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 Application Platform GUI. 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.
- **Runtime Resource Visibility plug-in:** When accessing the **Runtime Resources** tab from the **Component** view, the following warning appears: `Access error when querying cluster 'host' for resource '/apis/source.apps.tanzu.vmware.com/v1alpha1/mavenartifacts' (status: 403). Contact your administrator.` This issue is resolved in v1.2.1. In v1.2.0, the user can fix this issue by troubleshooting the Maven artifacts access error.

VS Code Extension

- **Debugging ending prematurely:** When debugging an application with service bindings, debugging sessions might prematurely end on the first run only. This is because of services being late-bound.

- **Workloads panel only supports `kubeconfig`:** The Workloads panel only supports the default `kubeconfig` file, which is usually in `~/.kube/config`.
- **Unable to view workloads on the panel when connected to GKE cluster:** When connecting to Google's GKE clusters, an error might appear with the text `WARNING: the gcp auth plugin is deprecated in v1.22+, unavailable in v1.25+; use gcloud instead`. To fix this, see Troubleshooting.
- **Warning notification when canceling an action:** A warning notification can appear when running `Tanzu: Debug Start`, `Tanzu: Live Update Start`, or `Tanzu: Apply`, which says that no workloads or Tiltfiles were found. For more information, see Troubleshooting.
- **Live update might not work when using server or worker Workload types:** When using `server` or `worker` as workload type, live update might not work. For more information, see Troubleshooting

Intellij Extension

- **Debugging ending prematurely:** When debugging an application with service bindings, debugging sessions might prematurely end on the first run only. This is because of services being late-bound.
- **Unable to view workloads on the panel when connected to GKE cluster:** When connecting to Google's GKE clusters, an error might appear with the text `WARNING: the gcp auth plugin is deprecated in v1.22+, unavailable in v1.25+; use gcloud instead`. To fix this, see Troubleshooting.
- **Live update might not work when using server or worker Workload types:** When using `server` or `worker` as workload type, live update might not work. For more information, see Troubleshooting

Supply Chain Security Tools - Store

- Querying by insight source returns zero CVEs even though there are CVEs in the source scan: When attempting to look up CVE and affected packages, querying `insight source get` (or other `insight source` commands) may return zero results due to supply chain configuration and repo URL. See Troubleshoot Supply Chain Security Tools - Store

Installing Tanzu Application Platform

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

- Installing Tanzu Application Platform online. For Tanzu Application Platform on a Kubernetes cluster with internet access.
- Installing Tanzu Application Platform in an air-gapped environment. For Tanzu Application Platform on a Kubernetes cluster air-gapped from external traffic.

Installing Tanzu Application Platform

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

- Installing Tanzu Application Platform online. For Tanzu Application Platform on a Kubernetes cluster with internet access.
- Installing Tanzu Application Platform in an air-gapped environment. For Tanzu Application Platform on a Kubernetes cluster air-gapped from external traffic.

Prerequisites for installing Tanzu Application Platform

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

VMware Tanzu Network and container image registry requirements

Installation requires:

- 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 Learning Center: Similar to Cloud Native Runtimes, allocate a wildcard subdomain for your workshops and content. This is also specified by 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.
- Tanzu Application Platform GUI: If you decide to implement the shared ingress and include Tanzu Application Platform GUI, 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 Application Platform GUI, 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`.
- 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`.

Tanzu Application Platform GUI

For Tanzu Application Platform GUI, you must have:

- Latest version of Chrome, Firefox, or Edge. Tanzu Application Platform GUI currently does not support Safari browser.
- Git repository for Tanzu Application Platform GUI'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 Application Platform GUI Blank Catalog from the Tanzu Application section of VMware Tanzu Network.
  - To install, navigate to Tanzu Network. Under the list of available files to download, there is a folder titled `tap-gui-catalogs-latest`. Inside that folder is a compressed archive titled `Tanzu Application Platform GUI 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 Application Platform GUI.
- The Tanzu Application Platform GUI 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 Configuring the Tanzu Application Platform GUI database

Kubernetes cluster requirements

Installation requires Kubernetes cluster v1.22 or v1.23 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 require the Amazon EBS CSI Driver due to CSIMigrationAWS is enabled by default in Kubernetes 1.23.
    - Users currently on EKS Kubernetes version 1.22 must install the Amazon EBS CSI Driver before upgrading to Kubernetes version 1.23. 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.
- Minikube.
  - Reference the resource requirements in the following section.
  - Hyperkit driver is supported on macOS only. Docker driver is not supported.
- Tanzu Kubernetes Grid multicloud.
- vSphere with Tanzu v7.0 U3a.
  For vSphere with Tanzu, pod security policies must be configured so that Tanzu Application Platform controller pods can run as root. For more information, see the Kubernetes documentation.

  To set the pod security policies, run:

  ```
  kubectl create clusterrolebinding default-tkg-admin-privileged-binding --cluste
  rrole=psp:vmware-system-privileged --group=system:authenticated
  ```

  For more information about pod security policies on Tanzu for vSphere, see Using Pod Security Policies with Tanzu Kubernetes Clusters in VMware vSphere Product Documentation.

Resource requirements

- To deploy all Tanzu Application Platform packages, your cluster must have at least:
  - 8 CPUs for i9 (or equivalent) available to Tanzu Application Platform components.
  - 12 CPUs for i7 (or equivalent) available to Tanzu Application Platform components.
  - 8 GB of RAM across all nodes available to Tanzu Application Platform.
  - 12 GB of RAM is available to build and deploy applications, including Minikube. VMware recommends 16 GB of RAM for an optimal experience.
  - 70 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. See Kubernetes documentation for more information.

Tools and CLI requirements

Installation requires:

- The Kubernetes CLI, kubectl, v1.22 or v1.23, installed and authenticated with admin rights for your target cluster. See Install Tools in the Kubernetes documentation.

Next steps

- Accepting Tanzu Application Platform EULAs and installing the Tanzu CLI

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

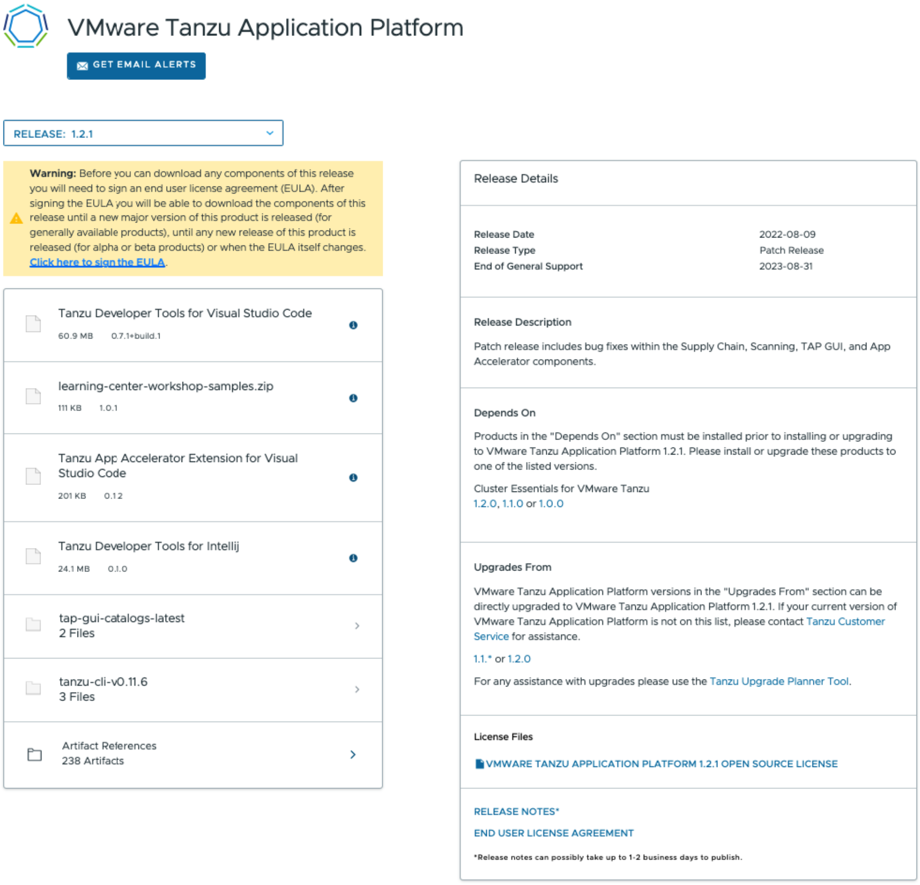
Before downloading and installing Tanzu Application Platform packages, you must accept the End User License Agreements (EULAs) as follows:

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.



VMware Tanzu Application Platform

RELEASE: 1.2.1

Warning: Before you can download any components of this release you will need to sign an end user license agreement (EULA). After signing the EULA you will be able to download the components of this release until a new major version of this product is released (for generally available products), until any new release of this product is released (for alpha or beta products) or when the EULA itself changes. Click here to sign the EULA.

Tanzu Developer Tools for Visual Studio Code
60.9 MB 0.7.1+build.1

learning-center-workshop-samples.zip
111 KB 1.0.1

Tanzu App Accelerator Extension for Visual Studio Code
201 KB 0.1.2

Tanzu Developer Tools for Intellij
24.1 MB 0.1.0

tap-gui-catalogs-latest
2 Files

tanzu-cli-v0.11.6
3 Files

Artifact References
238 Artifacts

Release Details

Release Date	2022-08-09
Release Type	Patch Release
End of General Support	2023-08-31

Release Description

Patch release includes bug fixes within the Supply Chain, Scanning, TAP GUI, and App Accelerator components.

Depends On

Products in the "Depends On" section must be installed prior to installing or upgrading to VMware Tanzu Application Platform 1.2.1. Please install or upgrade these products to one of the listed versions.

Cluster Essentials for VMware Tanzu
1.2.0, 1.1.0 or 1.0.0

Upgrades From

VMware Tanzu Application Platform versions in the "Upgrades From" section can be directly upgraded to VMware Tanzu Application Platform 1.2.1. If your current version of VMware Tanzu Application Platform is not on this list, please contact Tanzu Customer Service for assistance.

1.1.* or 1.2.0

For any assistance with upgrades please use the Tanzu Upgrade Planner Tool.

License Files

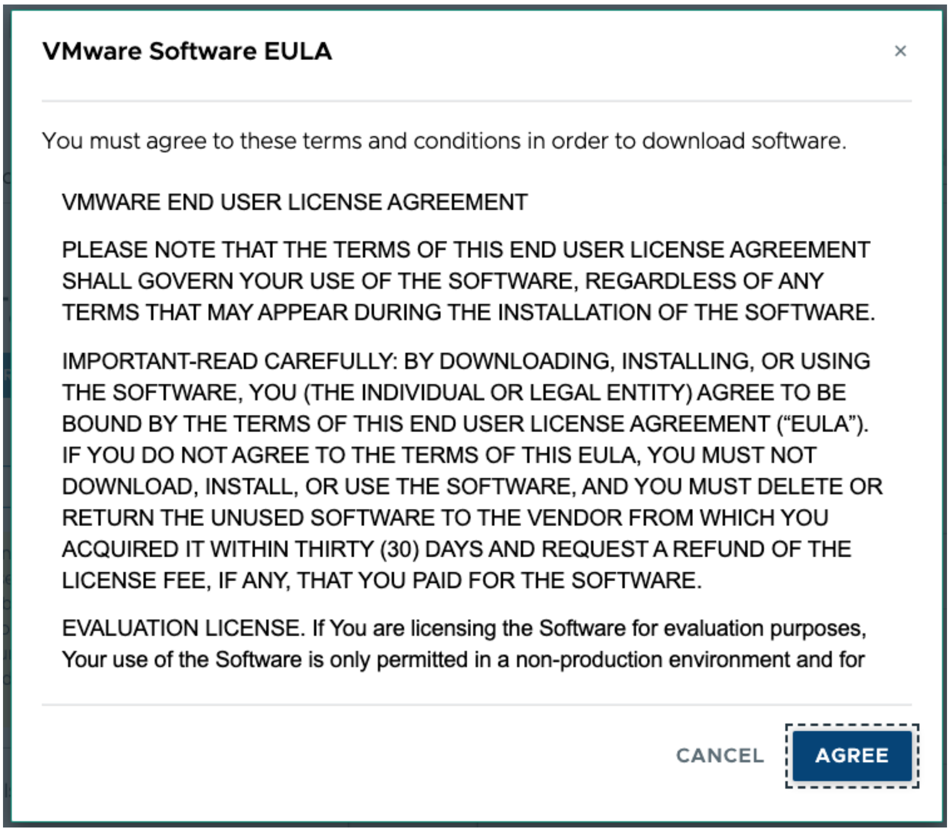
VMWARE TANZU APPLICATION PLATFORM 1.2.1 OPEN SOURCE LICENSE

RELEASE NOTES*

END USER LICENSE AGREEMENT

*Release notes can possibly take up to 1-2 business days to publish.

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



VMware Software EULA

You must agree to these terms and conditions in order to download software.

VMWARE END USER LICENSE AGREEMENT

PLEASE NOTE THAT THE TERMS OF THIS END USER LICENSE AGREEMENT SHALL GOVERN YOUR USE OF THE SOFTWARE, REGARDLESS OF ANY TERMS THAT MAY APPEAR DURING THE INSTALLATION OF THE SOFTWARE.

IMPORTANT-READ CAREFULLY: BY DOWNLOADING, INSTALLING, OR USING THE SOFTWARE, YOU (THE INDIVIDUAL OR LEGAL ENTITY) AGREE TO BE BOUND BY THE TERMS OF THIS END USER LICENSE AGREEMENT ("EULA"). IF YOU DO NOT AGREE TO THE TERMS OF THIS EULA, YOU MUST NOT DOWNLOAD, INSTALL, OR USE THE SOFTWARE, AND YOU MUST DELETE OR RETURN THE UNUSED SOFTWARE TO THE VENDOR FROM WHICH YOU ACQUIRED IT WITHIN THIRTY (30) DAYS AND REQUEST A REFUND OF THE LICENSE FEE, IF ANY, THAT YOU PAID FOR THE SOFTWARE.

EVALUATION LICENSE. If You are licensing the Software for evaluation purposes, Your use of the Software is only permitted in a non-production environment and for

Set the Kubernetes cluster context

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. 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

You use the Tanzu CLI and plug-ins to install and use the Tanzu Application Platform functions and features.

> **Note**
>
> Follow the steps in this topic if you do not want to use a profile to install the Tanzu CLI and plug-ins. For more information about profiles, see About Tanzu Application Platform components and profiles.

To install the Tanzu CLI and plug-ins:

1. Sign in to VMware Tanzu Network.
2. Go to the Tanzu Application Platform product page.
3. Select **Release 1.2.2** from the release drop-down menu.
4. Click **tanzu-cli-tap-1.2.2** to list the Tanzu framework bundles.
5. Click and download the Tanzu framework bundle for your operating system.
6. (Optional) If an earlier upgrade attempt failed, you can uninstall the previous version of the Tanzu CLI and associated plug-ins and files. See Remove Tanzu CLI, plug-ins, and associated files for more information.

For Windows installation instructions, see Install Tanzu CLI: Windows.

Install Tanzu CLI: Linux or macOS

1. Create a `$HOME/tanzu` directory on your local machine.
2. Unpack the downloaded TAR file into the `$HOME/tanzu` directory by running:

   - **For Linux:**

     ```
     tar -xvf tanzu-framework-linux-amd64.tar -C $HOME/tanzu
     ```

- **For macOS:**

  ```
  tar -xvf tanzu-framework-darwin-amd64.tar -C $HOME/tanzu
  ```

3. Set the environment variable `TANZU_CLI_NO_INIT` to `true` to ensure the local downloaded versions of the CLI core and plug-ins are installed by running:

   ```
   export TANZU_CLI_NO_INIT=true
   ```

4. Install or update the CLI core by running:

   Note: Replace v0.11.6 with the version you've downloaded.

   - **For Linux:**

     ```
     cd $HOME/tanzu
     export VERSION=v0.11.6
     sudo install cli/core/$VERSION/tanzu-core-linux_amd64 /usr/local/bin/tanzu
     ```

   - **For macOS:**

     ```
     cd $HOME/tanzu
     export VERSION=v0.11.6
     install cli/core/$VERSION/tanzu-core-darwin_amd64 /usr/local/bin/tanzu
     ```

5. Confirm the installation by running:

   ```
   tanzu version
   ```

   Expected outcome:

   ```
   version: v0.11.6
   ...
   ```

6. Proceed to Install/Update Tanzu CLI plug-ins

Install Tanzu CLI: Windows

1. Open the Windows file browser.

2. Create a `Program Files\tanzu` directory on your local machine.

3. From the `Downloads` directory, right-click the `tanzu-framework-windows.amd64.zip` file, select the **Extract All...** menu option, enter `C:\Program files\tanzu` in the **Files are extracted to this directory:** text box, and click the **Extract**.

4. From the `Program Files\tanzu` directory, move and rename; the executable file from

   ```
   Program Files\tanzu\cli\core\v0.11.6\tanzu-core-windows_amd64.exe
   ```

   to

   ```
   Program Files\tanzu\tanzu.exe
   ```

5. From the `Program Files` directory, right-click the `tanzu` directory and select **Properties > Security**.

6. Ensure that your user account has the **Full Control** permission.

7. Use Windows Search to search for `env`, select **Edit the system environment variables**, click **Environment Variables** on the bottom right of the dialogue box.

8. Find and select the **Path** row under **System variables**, click **Edit**.

9. Click **New**, enter the path value, click **OK**.

   Note: The path value must not include **tanzu.exe**. For example, `C:\Program Files\tanzu`.

10. Click **New** following the **System Variables** section, add a new environmental variable named `TANZU_CLI_NO_INIT` with a variable value `true`, click **OK**.
11. Use Windows Search to search for `cmd`, select **Command Prompt** to open the command line terminal.
12. Verify the Tanzu CLI installation by running:

    ```
    tanzu version
    ```

    Expected outcome:

    ```
    version: v0.11.6
    ...
    ```

13. Proceed to Install/Update Tanzu CLI plug-ins

Install/Update Tanzu CLI plug-ins

To install or update Tanzu CLI plug-ins from your terminal, follow these steps:

1. Install plug-ins from the `$HOME/tanzu` directory (if on Linux or macOS) or `Program Files\tanzu` directory (if on Windows) by running:

    ```
    tanzu plugin install --local cli all
    ```

2. Verify that you installed the plug-ins by running:

    ```
    tanzu plugin list
    ```

    Expected outcome:

    ```
    NAME                DESCRIPTION
    SCOPE       DISCOVERY             VERSION      STATUS
    login               Login to the platform
    Standalone  default               v0.11.6      not installed
    management-cluster  Kubernetes management-cluster operations
    Standalone  default               v0.11.6      not installed
    package             Tanzu package management
    Standalone  default               v0.11.6      installed
    pinniped-auth       Pinniped authentication operations (usually not directly in
    voked)             Standalone  default               v0.11.6      not installed
    secret              Tanzu secret management
    Standalone  default               v0.11.6      installed
    services            Discover Service Types, Service Instances and manage Resour
    ce Claims (ALPHA)  Standalone                        v0.3.0-rc.2  installed
    accelerator         Manage accelerators in a Kubernetes cluster
    Standalone                        v1.2.0-build.1       installed
    apps                Applications on Kubernetes
    Standalone                        v0.7.0-build.1       installed
    insight             post & query image, package, source, and vulnerability data
    Standalone                        v1.2.1       installed
    ```

Next steps

For online installation:

- Deploying Cluster Essentials*
- Installing the Tanzu Application Platform package and profiles

For air-gapped installation:

- Deploying Cluster Essentials*
- Install Tanzu Application Platform in an air-gapped environment (beta)

* *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.*

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. Install Docker if it is not already installed.
2. Log in to your image registry by running:

   ```
   docker login MY-REGISTRY
   ```

   Where `MY-REGISTRY` is your own container registry.
3. Log in to the VMware Tanzu Network registry with your VMware Tanzu Network credentials by running:

   ```
   docker login registry.tanzu.vmware.com
   ```
4. Set up environment variables for installation use by running:

   ```
   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.
   - `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.2.2`.
   - `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.
5. 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 | grep -v sha | sort -V
   ```

6. 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 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_PASSSWORD` are the credentials to run `docker login registry.tanzu.vmware.com`
- `TAP_VERSION` is your Tanzu Application Platform version. For example, `1.2.2`

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. 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
```

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:    tapmdc.azurecr.io/mdc/1.0.2/tap-packages
TAG:           1.2.2
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.

5. 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
     build.appliveview.tanzu.vmware.com                   Application Live View Co
   nventions for VMware Tanzu                        Application Live View convent
   ion server
     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.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.
     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
     eventing.tanzu.vmware.com                            Eventing
   Eventing is an event-driven architecture platform based on Knative Eventing
     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
     image-policy-webhook.signing.apps.tanzu.vmware.com   Image Policy Webhook
   Image Policy Webhook enables defining of a policy to restrict unsigned containe
   r

   images.
     learningcenter.tanzu.vmware.com                      Learning Center for Tanz
   u Application Platform                            Guided technical workshops
     metadata-store.apps.tanzu.vmware.com                 Supply Chain Security To
   ols - Store                                       Post SBoMs and query for imag
   e, package, and vulnerability metadata.
   ```

```
  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 Application Platfo
rm GUI                                            web app graphical user interf
ace for Tanzu Application Platform
  tap-telemetry.tanzu.vmware.com                       Telemetry Collector for
Tanzu Application Platform                        Tanzu Application Plaform Tel
emetry
  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.
  workshops.learningcenter.tanzu.vmware.com            Workshop Building Tutori
al                                                Workshop Building Tutorial
```

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.

   > **Important**
   >
   > Keep the values file for future configuration use.

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.

> **Important**
>
> While installing Tanzu Application Platform v1.3 and later, exclude the policy controller `policy.apps.tanzu.vmware.com`, or deploy a Sigstore Stack to use as a TUF Mirror.

```
shared:
  ingress_domain: "INGRESS-DOMAIN"
  image_registry:
    project_path: "SERVER-NAME/REPO-NAME"
    username: "KP-DEFAULT-REPO-USERNAME"
    password: "KP-DEFAULT-REPO-PASSWORD"
  kubernetes_distribution: "openshift" # To be passed only for OpenShift. Defaults to
"".
  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.

excluded_packages:
- policy.apps.tanzu.vmware.com

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: "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:
    ssh_secret: "SSH-SECRET-KEY" # Takes "" as value by default; but can be overridden
by setting a different value.
```

```
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_username: "KP-DEFAULT-REPO-USERNAME"
  kp_default_repository_password: "KP-DEFAULT-REPO-PASSWORD"

tap_gui:
  service_type: ClusterIP # If the shared.ingress_domain is set as above, this must be
set to ClusterIP.
  app_config:
    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.

scanning:
  metadataStore:
    url: "" # Configuration is moved, so set this string to empty.

grype:
  namespace: "MY-DEV-NAMESPACE"
  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-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`.
  - Alternatively, you can configure this credential as a secret reference.
- `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.
  - For Google Cloud Registry, use the contents of the service account JSON file.
  - Alternatively, you can configure this credential as a secret reference.
- `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"`.
- `SSH-SECRET-KEY` is the SSH secret key in the developer namespace for the supply chain to fetch source code from and push configuration to. This field is only required if you use a private repository, otherwise, leave it empty. 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.

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 create 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) 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: "KP-DEFAULT-REPO"
  kp_default_repository_username: "KP-DEFAULT-REPO-USERNAME"
  kp_default_repository_password: "KP-DEFAULT-REPO-PASSWORD"
  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 v1.3.0 supports building applications with the Ubuntu 22.04 (Jammy) stack. By default, workloads are built with Ubuntu 18.04 (Bionic) stack. However, if you do not need access to the Bionic stack, you can install Tanzu Application Platform without the Bionic stack and all workloads are built with the Jammy stack by default.

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`.

(Optional) Exclude Image Policy Webhook

Image Policy Webhook is deprecated. To exclude this package, update your `tap-values` file with a section listing the exclusion:

```
...
excluded_packages:
  - image-policy-webhook.signing.apps.tanzu.vmware.com
...
```

See Exclude packages from a Tanzu Application Platform profile for more information.

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.

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

To install the `full` dependencies package:

1. If you have not done so already, 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_username: "KP-DEFAULT-REPO-USERNAME"
     kp_default_repository_password: "KP-DEFAULT-REPO-PASSWORD"
     exclude_dependencies: true
   ...
   ```

2. Get the latest version of the `buildservice` package by running:

   ```
   tanzu package available list buildservice.tanzu.vmware.com --namespace tap-inst
   all
   ```

3. Relocate the Tanzu Build Service full dependencies package repository by running:

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

   Where `VERSION` is the version of the `buildservice` package you retrieved in the previous step.

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

   ```
   tanzu package repository add tbs-full-deps-repository \
     --url ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/tbs-full-deps:VERSION \
     --namespace tap-install
   ```

   Where `VERSION` is the version of the `buildservice` package you retrieved earlier.

5. Install the full dependencies package by running:

   ```
   tanzu package install full-tbs-deps -p full-tbs-deps.tanzu.vmware.com -v VERSIO
   N -n tap-install
   ```

   Where `VERSION` is the version of the `buildservice` package you retrieved earlier.

Access Tanzu Application Platform GUI

To access Tanzu Application Platform GUI, you can use the host name that you configured earlier. This host name is pointed at the shared ingress. To configure LoadBalancer for Tanzu Application Platform GUI, see Accessing Tanzu Application Platform GUI.

You're now ready to start using Tanzu Application Platform GUI. Proceed to the Getting Started topic or the Tanzu Application Platform GUI - 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

Viewing 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, use `tanzu package available get -n tap-install cnrs.tanzu.vmware.com/$TAP_VERSION --values-schema` for Cloud Native Runtimes.

```
profile: full

# ...

# For example, CNRs specific values go under its name
cnrs:
  provider: local

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

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

Package	Top-level Key
see table below	shared
API portal	api_portal
Application Accelerator	accelerator
Application Live View	appliveview
Application Live View Connector	appliveview_connector
Application Live View Conventions	appliveview-conventions
Cartographer	cartographer
Cloud Native Runtimes	cnrs
Convention Controller	convention_controller
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
Image Policy Webhook	image_policy_webhook
Build Service	buildservice
Tanzu Application Platform GUI	tap_gui
Learning Center	learningcenter

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	Used By	Description
ca_cert_data	convention_controller, source_controller	Optional: PEM Encoded certificate data to trust TLS connections with a private CA.

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

Installing individual packages

You can install Tanzu Application Platform through predefined profiles or through individual packages. This page provides links to install instructions for each of the individual packages. For more information about installing through profiles, see About Tanzu Application Platform components and 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 portal

- Install Application Accelerator
- Install Application Live View
- Install Application Single Sign-On
- Install cert-manager, Contour, and Flux CD
- Install Cloud Native Runtimes
- Install default roles for Tanzu Application Platform
- Install Developer Conventions
- Install Learning Center for Tanzu Application Platform
- 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 Application Platform GUI
- Install Tanzu Build Service
- Install Tekton

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.
7.1            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

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

Follow these steps to enable your current user to submit jobs to the Supply Chain:

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

   ```
   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. 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`.
   - `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 with the above command:

```
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:

```
kubectl create secret docker-registry registry-credentials --docker-server=REGISTRY-SERVER --docker-username=REGISTRY-USERNAME --docker-password=REGISTRY-PASSWORD -n YOUR-NAMESPACE
```

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. To add secrets, a service account to execute the supply chain, and RBAC rules to authorize the service account to the developer namespace, run:

```
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
```

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 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-viewer
     tanzu rbac binding add -g GROUP-FOR-APP-EDITOR -n YOUR-NAMESPACE -r app-editor
     ```

     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.

Additional configuration for testing and scanning

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

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 v0.27.2 or later
- Tanzu CLI and plug-ins
- A cluster with the Tanzu Application Platform Full profile or Iterate profile

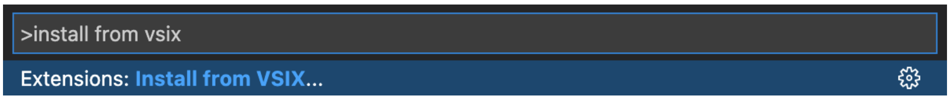
If you are an app developer, someone else in your organization might have already set up the Tanzu Application Platform environment.

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

Install

To install the extension:

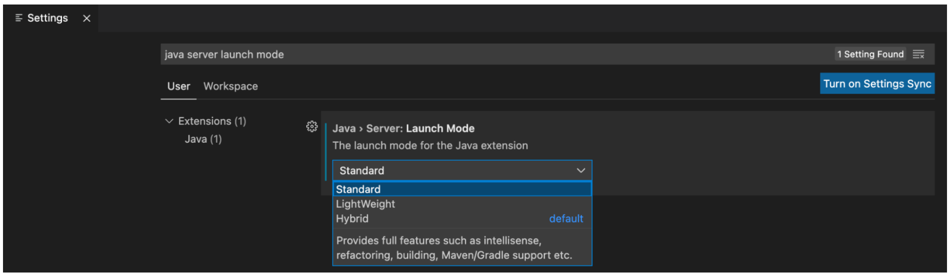
1. Sign in to VMware Tanzu Network and download Tanzu Developer Tools for Visual Studio Code.
2. Open VS Code.
3. Press cmd+shift+P to open the Command Palette and run `Extensions: Install from VSIX...`.
4. Select the extension file **tanzu-vscode-extension.vsix**.

   >install from vsix

   Extensions: Install from VSIX...

5. If you do not have the following extensions, and they do not automatically install, install them from VS Code Marketplace:
   - Debugger for Java
   - Language Support for Java(™) by Red Hat
   - YAML
6. Ensure Language Support for Java is running in Standard Mode. You can configure it in the **Settings** menu by going to **Code** > **Preferences** > **Settings** under **Java > Server: Launch Mode**.

   java server launch mode

   Java › Server: Launch Mode

   The launch mode for the Java extension

   Standard

   Standard

   LightWeight

   Hybrid default

   Provides full features such as intellisense, refactoring, building, Maven/Gradle support etc.

   When the JDK and Language Support for Java are configured correctly, you see that the integrated development environment creates a directory target where the code is compiled.

Configure

To configure VMware Tanzu Developer Tools for VS Code:

1. Ensure that you are targeting the correct cluster. For more informatiom, see the Kubernetes documentation.
2. Go to **Code** > **Preferences** > **Settings** > **Extensions** > **Tanzu Developer Tools** and set the following:
   - **Confirm Delete**: This controls whether the extension asks for confirmation when deleting a workload.
   - **Enable Live Hover**: For more information, see Integrating Live Hover by using Spring Boot Tools. Reload VS Code for this change to take effect.
   - **Source Image**: (Required) The registry location for publishing local source code. For example, `registry.io/yourapp-source`. This must include both a registry and a project name.
   - **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.

Uninstall

To uninstall VMware Tanzu Developer Tools for VS Code:

1. Go to **Code** > **Preferences** > **Settings** > **Extensions**.
2. Right-click the extension and select **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 an air-gapped environment
5.	Install Tanzu Build Service full dependencies.	Install the Tanzu Build Service dependencies
6.	Configure custom certificate authorities for Tanzu Application Platform GUI.	Configure custom certificate authorities for Tanzu Application Platform GUI
7.	Add the certificate for the private Git repository in the Accelerator system namespace.	Configure Application Accelerator
8.	Apply patch to Grype.	Use Grype in offline and air-gapped environments

Step	Task	Link
9.	Set up developer namespaces to use your installed packages.	Set up developer namespaces to use your installed packages

* *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 instructions, 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 an air-gapped environment
5.	Install Tanzu Build Service full dependencies.	Install the Tanzu Build Service dependencies
6.	Configure custom certificate authorities for Tanzu Application Platform GUI.	Configure custom certificate authorities for Tanzu Application Platform GUI
7.	Add the certificate for the private Git repository in the Accelerator system namespace.	Configure Application Accelerator
8.	Apply patch to Grype.	Use Grype 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

* *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 instructions, 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:

   ```
   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 TAP_VERSION=VERSION-NUMBER
   export REGISTRY_CA_PATH=PATH-TO-CA
   ```

   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, `1.2.2`

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
   ```

   Where:

   - `TANZUNET-REGISTRY-USERNAME` is your username of the VMware Tanzu Network.
   - `TANZUNET-REGISTRY-PASSWORD` is your password of the VMware Tanzu Network.

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

   ```
   imgpkg copy \
     --tar tap-packages-$TAP_VERSION.tar \
     --to-repo $IMGPKG_REGISTRY_HOSTNAME/tap-packages \
     --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 \
       --username $IMGPKG_REGISTRY_USERNAME \
       --password $IMGPKG_REGISTRY_PASSWORD \
       --namespace tap-install \
       --export-to-all-namespaces \
       --yes
   ```

6. Create a internal registry secret by running:

```
tanzu secret registry add registry-credentials \
    --server   $MY_REGISTRY \
    --username $MY_REGISTRY_USER \
    --password $MY_REGISTRY_PASSWORD \
    --namespace tap-install \
    --export-to-all-namespaces \
    --yes
```

Where:

- `MY_REGISTRY` is where the workload images and the Tanzu Build Service dependencies are stored.
- `MY_REGISTRY_USER` is the user with write access to `MY_REGISTRY`.
- `MY_REGISTRY_PASSWORD` is the password for `MY_REGISTRY_USER`.

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

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

Where:

- `$TAP_VERSION` is the Tanzu Application Platform version environment variable you defined earlier.
- `TARGET-REPOSITORY` is the necessary repository.

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.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.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.
  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
  eventing.tanzu.vmware.com                            Eventing
Eventing is an event-driven architecture platform based on Knative Eventing
  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
  learningcenter.tanzu.vmware.com                      Learning Center for Tanz
u Application Platform                            Guided technical workshops
  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 Application Platfo
rm GUI                                            web app graphical user interf
ace for Tanzu Application Platform
  tap-telemetry.tanzu.vmware.com                       Telemetry Collector for
Tanzu Application Platform                        Tanzu Application Plaform Tel
emetry
  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.
  workshops.learningcenter.tanzu.vmware.com            Workshop Building Tutori
al                                                Workshop Building Tutorial
```

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:

> **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`.

```
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
scanning:
  metadataStore:
    url: ""
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:
  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:
      ssh_secret: "SSH-SECRET"
  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
      BAgMAk1OMRQwEgYDVQQHDAtNaW5uZWFwb2xpczEPMA0GA1UECgwGVk13YXJlMRMw
      -----END CERTIFICATE-----

tap_gui:
  service_type: ClusterIP
  app_config:
    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.

grype:
  namespace: "MY-DEV-NAMESPACE"
  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.

- `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 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`.
  - 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"`.
- `SSH-SECRET` is the secret name for https authentication, certificate authority, and SSH authentication. 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.
- `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).

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

When installing Tanzu Build Service on an air-gapped environment, the `lite` dependencies cannot be used as they require Internet access. You must install the `full` dependencies.

To install `full` dependencies:

1. Get the latest version of the Tanzu Build Service package by running:

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

2. Relocate the Tanzu Build Service `full` dependencies package repository by running:

   ```
   imgpkg copy -b registry.tanzu.vmware.com/tanzu-application-platform/full-tbs-deps-package-repo:VERSION \
     --to-tar=tbs-full-deps.tar
   # move tbs-full-deps.tar to environment with registry access
   imgpkg copy --tar tbs-full-deps.tar \
     --to-repo=INSTALL-REGISTRY-HOSTNAME/TARGET-REPOSITORY/tbs-full-deps
   ```

   Where:

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

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

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

   Where:

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

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

   ```
   tanzu package install full-tbs-deps -p full-tbs-deps.tanzu.vmware.com -v VERSION -n tap-install
   ```

   Where `VERSION` is the version of the Tanzu Build Service package you retrieved earlier.

Next steps

- Configuring custom CAs for Tanzu Application Platform GUI

Configuring custom certificate authorities for Tanzu Application Platform GUI

This topic tells you how to configure Tanzu Application Platform GUI (commonly known as TAP GUI) to trust unusual certificate authorities (CA) when making outbound connections.

You use overlays with PackageInstalls to make this possible. There are two ways to implement this workaround: you can deactivate all SSL verification or you can add a custom CA.

Deactivate all SSL verification

To deactivate SSL verification to allow for self-signed certificates, set the Tanzu Application Platform GUI 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 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 Application Platform GUI instance is deployed. For example, `tap-gui`.

Add a custom CA

If you want to keep verification enabled, you can add a custom CA and mount it to the Tanzu Application Platform GUI pod, and then set the pod's environment variable as `NODE_EXTRA_CA_CERTS=PATH-TO-MOUNTED-FILE`. To do so:

1. Encode the extra CA certificates you want to trust in base64. You can provide many certificates in PEM format in the same file. Get the output you need for the next step by running:

   ```
   cat FILENAME | base64 -w0
   ```

   Where `FILENAME` is your filename. For example, `cert-chain.pem`.

2. Copy the output from the previous command into the example field `tap-gui-certs.crt` into the following example `secret.yaml`:

   ```
   ---
   apiVersion: v1
   kind: Secret
   metadata:
   name: tap-gui-extra-certs
   namespace: tap-gui
   type: Opaque
   data:
   tap-gui-certs.crt: "ENCODED-LIST-OF-CERTS"
   ```

   Adjust metadata and naming from this example accordingly.

3. Apply the secret to your cluster by running:

   ```
   kubectl apply -f secret.yaml
   ```

4. To set the environment variable `NODE_EXTRA_CA_CERTS`, use the `package_overlays` key in the Tanzu Application Platform values file. For instructions, see Customizing Package

Installation.

The following is an example overlay to add a custom CA. It assumes that your Tanzu Application Platform GUI instance is deployed in the namespace `tap-gui`. Adjust all names accordingly.

```
#@ load("@ytt:overlay", "overlay")

#@overlay/match by=overlay.subset({"kind": "Deployment", "metadata": {"name":
"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
      - env:
          - name: NODE_EXTRA_CA_CERTS
            value: /etc/tap-gui-certs/tap-gui-certs.crt
        volumeMounts:
          - name: tap-gui-extra-certs
            mountPath: /etc/tap-gui-certs
            readOnly: true
    volumes:
      - name: tap-gui-extra-certs
        secret:
          secretName: tap-gui-extra-certs
```

Next steps

- Configure Application Accelerator

Configure Application Accelerator

This topic describes how to configure Application Accelerator for use in air-gapped environments or other customized installations.

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, then you do not have to configure anything additional. Accelerators are created either using the Tanzu CLI or by applying a YAML manifest using `kubectl`.

Examples for creating accelerators

A minimal example for creating an accelerator

A minimal example could look like the following manifest:

hello-fun.yaml

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: hello-fun
spec:
  git:
    url: https://github.com/sample-accelerators/hello-fun
    ref:
      branch: main
```

This minimal example creates an accelerator named `hello-fun`. The `displayName`, `description`, `iconUrl`, and `tags` fields 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 https://github.com/sample-accelerators/hello-fun. For example:

accelerator.yaml

```
accelerator:
  displayName: Hello Fun
  description: A simple Spring Cloud Function serverless app
  iconUrl: https://raw.githubusercontent.com/simple-starters/icons/master/icon-cloud.p
ng
  tags:
  - java
  - spring
  - cloud
  - function
  - serverless
  - tanzu
...
```

To create this accelerator with `kubectl` run:

```
kubectl apply --namespace --accelerator-system --filename hello-fun.yaml
```

Or, you could use the Tanzu CLI and run:

```
tanzu accelerator create hello-fun --git-repo https://github.com/sample-accelerators/h
ello-fun --git-branch main
```

An example for creating an accelerator with customized properties

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

my-hello-fun.yaml

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: my-hello-fun
spec:
  displayName: My Hello Fun
  description: My own Spring Cloud Function serverless app
  iconUrl: https://raw.githubusercontent.com/sample-accelerators/icons/master/icon-tan
zu-light.png
  tags:
    - spring
    - cloud
    - function
    - serverless
  git:
    ignore: ".git/, bin/"
    url: https://github.com/sample-accelerators/hello-fun
    ref:
      branch: test
```

To create this accelerator with `kubectl` you could run:

```
kubectl apply --namespace --accelerator-system --filename my-hello-fun.yaml
```

Or, you could use the Tanzu CLI and run:

```
tanzu accelerator create my-hello-fun --git-repo https://github.com/sample-accelerator
s/hello-fun --git-branch main \
  --description "My own Spring Cloud Function serverless app" \
  --display-name "My Hello Fun" \
  --icon-url https://raw.githubusercontent.com/sample-accelerators/icons/master/icon-t
```

```
anzu-light.png \
  --tags "spring,cloud,function,serverless"
```

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

Using non-public repositories

For Git repositories that aren't accessible anonymously, you need to 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. 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. See fluxcd/source-controller HTTPS Certificate Authority
- For SSH repositories, the secret must contain identity, identity.pub and known_hosts fields. See fluxcd/source-controller SSH authentication.
- For Image repositories that aren't publicly available, an image pull secret may be provided. For more information, see 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.

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>
```

This creates a secret such as the following:

https-credentials.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: https-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 private repository
  git:
    url: <repository-URL>
    ref:
      branch: main
    secretRef:
      name: https-credentials
```

If you are using the Tanzu CLI, then 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, first create a secret with the HTTP credentials and the certificate.

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>
```

This creates a secret that looks like this:

https-ca-credentials.yaml

```
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 private repository
  git:
    url: <repository-URL>
    ref:
      branch: main
    secretRef:
      name: https-ca-credentials
```

If you are using the Tanzu CLI, then 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, first create a secret with the SSH credentials like 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
```

This example assumes that you don't have a key file already created. If you do, skip the `ssh-keygen` and `ssh-keyscan` steps and replace the values for the `kubectl create secret` command using the following:

- `--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>`

The secret that is created will look like this:

ssh-credentials.yaml

```
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 private repository
  git:
    url: <repository-URL>
    ref:
      branch: main
    secretRef:
      name: ssh-credentials
```

If you are using the Tanzu CLI, then 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. The easiest way to do that is to 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, first create a secret with the image-pull credentials:

```
create secret generic registry-credentials \
    --namespace accelerator-system \
    --from-literal=username=<user> \
    --from-literal=password=<password>
```

This creates a secret that looks like this:

https-credentials.yaml

```
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 private repository
  source:
    image: "registry.example.com/test/private-acc-src:latest"
    imagePullSecrets:
    - name: registry-credentials
```

If you are using the Tanzu CLI, then add the `--secret-ref` flag to your `tanzu accelerator create` command and provide the name of the secret for that flag.

Next steps

- Using Grype in offline and air-gapped environments

Using Grype in offline and air-gapped environments

The `grype` CLI attempts to perform two over the Internet calls: one to verify for later versions of the CLI and another to update the vulnerability database before scanning.

You must deactivate both of these external calls. For the `grype` CLI to function in an offline or air-gapped environment, the vulnerability database must be hosted within the environment. You must configure the `grype` CLI with the internal URL.

The `grype` URL accepts environment variables to satisfy these needs.

For information about setting up an offline vulnerability database, see the Anchore Grype README in GitHub.

Overview

To enable Grype in offline air-gapped environments:

1. Create ConfigMap
2. Create Patch Secret

3. Configure tap-values.yaml to use `package_overlays`
4. Update Tanzu Application Platform

Use Grype

To use Grype in offline and air-gapped environments:

1. Create a ConfigMap that contains the public ca.crt to the file server hosting the Grype database files. Apply this ConfigMap to your developer namespace.
2. Create a secret that contains the ytt overlay to add the Grype environment variables to the ScanTemplates.

```
apiVersion: v1
kind: Secret
metadata:
  name: grype-airgap-overlay
  namespace: tap-install #! namespace where tap is installed
stringData:
  patch.yaml: |
    #@ load("@ytt:overlay", "overlay")

    #@overlay/match by=overlay.subset({"kind":"ScanTemplate","metadata":{"namespace":"<DEV-NAMESPACE>"}}),expects="1+"
    #! developer namespace you are using
    ---
    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"
              - name: GRYPE_DB_AUTO_UPDATE
                value: "true"
              - name: GRYPE_DB_UPDATE_URL
                value: <INTERNAL-VULN-DB-URL> #! url points to the internal file server
              - name: GRYPE_DB_CA_CERT
                value: "/etc/ssl/certs/custom-ca.crt"
              - name: GRYPE_DB_MAX_ALLOWED_BUILT_AGE #! see note on best practices
                value: "120h"
            volumeMounts:
              #@overlay/append
              - name: ca-cert
                mountPath: /etc/ssl/certs/custom-ca.crt
                subPath: <INSERT-KEY-IN-CONFIGMAP> #! key pointing to ca certificate
        volumes:
        #@overlay/append
        - name: ca-cert
          configMap:
            name: <CONFIGMAP-NAME> #! name of the configmap created
```

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. Stale databases weaken your security posture. VMware reccomends updating the database daily. You can use the `GRYPE_DB_MAX_ALLOWED_BUILT_AGE` parameter to override the default in accordance with your security posture.

You can also add more certificates to the ConfigMap created earlier, to handle connections to a private registry for example, and mount them in the `volumeMounts` section if needed.

For example:

```
#! ...
volumeMounts:
  #@overlay/append
  #! ...
  - name: ca-cert
    mountPath: /etc/ssl/certs/another-ca.crt
    subPath: another-ca.cert #! key pointing to ca certificate
```

If you have more than one developer namespace and you want to apply this change to all of them, change the `overlay match` on top of the patch.yaml to the following:

```
#@overlay/match by=overlay.subset({"kind":"ScanTemplate"}),expects="1+"
```

3. [Optional] If Grype was installed by using a Tanzu Application Platform profile, you can skip to the next step.

If Grype was installed manually, you must update your `PackageInstall` to include the annotation to reference the overlay `Secret`.

```
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
name: grype
namespace: tap-install
annotations:
  ext.packaging.carvel.dev/ytt-paths-from-secret-name.0: grype-airgap-overlay
...
```

For more information, see Customize package installation.

1. Configure tap-values.yaml to use `package_overlays`. Add the following to your tap-values.yaml:

   ```
   package_overlays:
    - name: "grype"
      secrets:
         - name: "grype-airgap-overlay"
   ```

2. Update Tanzu Application Platform

   ```
   tanzu package installed update tap -f tap-values.yaml -n tap-install
   ```

```
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's 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 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. First confirm that the required database schema for the installed Grype version is being used. Next, 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/releases/vulnerability-db_v5_2023-02-08T
08_17_20Z_6ef73016d160043c630f.tar.gz",
            "checksum": "sha256:aab8d369933c845878ef1b53bb5c26ee49b91ddc5cd87c9eb57ffb
203a88a72f"
    }]
  }
}
```

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.

1. The `built` parameters in the `listing.json` file are incorrectly formatted. The proper format is `yyyy-MM-ddTHH:mm:ssZ`.
2. The `url` which you modified to point at an internal endpoint is not reachable from within the cluster. For information about verifying connectivity, see Debug Grype database in a cluster.

Debug Grype database in a cluster

1. Describe the failed source or image scan to determine verify the name of the ScanTemplate being used:

```
kubectl describe sourcescan/imagescan SCAN-NAME -n DEV-NAMESPACE
```

Where `SCAN-NAME` is the name of the source/image scan that failed.

1. 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
```

2. Re-run the scan.
3. Get the name of the `scan-plugin` pod.

   ```
   kubectl get pods -n DEV-NAMESPACE
   ```

4. Get a shell to the container. See the Kubernetes documentation.

   ```
   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.

5. Inside the container, run Grype CLI commands to report database status and verify connectivity from cluster to mirror. See the Anchore Grype documentation in GitHub.

   - Report current status of Grype's database, such as location, build date, and checksum:

     ```
     grype db status
     ```

   - Download the listing file configured at `db.update-url` and show databases that are available for download:

     ```
     grype db list
     ```

Set up developer namespaces to use your installed packages

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

Follow these steps to enable your current user to submit jobs to the Supply Chain:

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

   ```
   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 give to 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`.

   - `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 with the above command:

   ```
   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:

   ```
   kubectl create secret docker-registry registry-credentials --docker-server=REGISTRY-SERVER --docker-username=REGISTRY-USERNAME --docker-password=REGISTRY-PASSWORD -n YOUR-NAMESPACE
   ```

   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. To add secrets, a service account to execute the supply chain, and RBAC rules to authorize the service account to the developer namespace, run:

   ```
   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
   ```

   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 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.

Additional configuration for testing and scanning

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

Next steps

- Deploy an air-gapped workload

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 1.2.2  --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 your 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 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.
- For information about installing or updating the Tanzu CLI and plug-ins, see Install or update the Tanzu CLI and plug-ins.
- For information on Tanzu Application Platform GUI considerations, see Tanzu Application Platform GUI 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 v1.2. See Cluster Essentials documentation for more information about the upgrade procedures.

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 4 in Relocate images to a registry.

   Note: Make sure to update the `VERSION-NUMBER` to the target version of Tanzu Application Platform you are migrating to. For example, `1.2.2`.

2. Add the target version of the Tanzu Application Platform package repository:

   - If you are using Cluster Essentials 1.2 or above, run:

     ```
     tanzu package repository add tanzu-tap-repository \
     --url ${INSTALL_REGISTRY_HOSTNAME}/TARGET-REPOSITORY/tap-packages:${TAP_V
     ERSION} \
     --namespace tap-install
     ```

   - If you are using Cluster Essentials 1.1 or 1.0, run:

     ```
     tanzu package repository update tanzu-tap-repository \
     --url ${INSTALL_REGISTRY_HOSTNAME}/TARGET-REPOSITORY/tap-packages:${TAP_V
     ERSION} \
     --namespace tap-install
     ```

Note: If you are using Cluster Essentials 1.0 or 1.1, 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 tanzu-tap-repository --namespace tap-install
```

Perform the upgrade of Tanzu Application Platform

Upgrade instructions for Profile-based installation

For Tanzu Application Platform that is installed by profile, you can perform the upgrade by running:

Note: Ensure you run the following command in the directory where the `tap-values.yaml` file resides.

```
tanzu package installed update tap -p tap.tanzu.vmware.com -v VERSION  --values-file t
ap-values.yaml -n tap-install
```

Where `VERSION` is the target revision of Tanzu Application Platform you are migrating to.

Note: When upgrading to Tanzu Application Platform v1.2, Tanzu Build Service image resources automatically run a build that fails due to a missing dependency. This error does not persist and any subsequent builds will resolve this error. You can safely wait for the next build of the workloads, which is triggered by new source code changes. If you do not want to wait for subsequent builds to run automatically, follow the instructions in the troubleshooting item Builds fail after upgrading to Tanzu Application Platform v1.2.

Upgrade instructions for component-specific installation

For information about upgrading Tanzu Application Platform GUI, see upgrading Tanzu Application Platform GUI. For information about upgrading Supply Chain Security Tools - Scan, see Upgrading Supply Chain Security Tools - Scan.

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
1.2.1            Reconcile succeeded
  api-portal                           api-portal.tanzu.vmware.com
1.0.21           Reconcile succeeded
  appliveview                          backend.appliveview.tanzu.vmware.com
1.2.0            Reconcile succeeded
  appliveview-connector                connector.appliveview.tanzu.vmware.com
1.2.0            Reconcile succeeded
  appliveview-conventions              conventions.appliveview.tanzu.vmware.com
1.2.0            Reconcile succeeded
  appsso                               sso.apps.tanzu.vmware.com
1.0.0            Reconcile succeeded
  buildservice                         buildservice.tanzu.vmware.com
1.6.0            Reconcile succeeded
  cartographer                         cartographer.tanzu.vmware.com
0.4.2            Reconcile succeeded
  cert-manager                         cert-manager.tanzu.vmware.com
1.5.3+tap.2      Reconcile succeeded
```

```
  cnrs                                cnrs.tanzu.vmware.com
1.3.0            Reconcile succeeded
  contour                             contour.tanzu.vmware.com
1.18.2+tap.2     Reconcile succeeded
  conventions-controller              controller.conventions.apps.tanzu.vmware.com
0.7.0            Reconcile succeeded
  developer-conventions               developer-conventions.tanzu.vmware.com
0.7.0            Reconcile succeeded
  fluxcd-source-controller            fluxcd.source.controller.tanzu.vmware.com
0.16.4           Reconcile succeeded
  grype                               grype.scanning.apps.tanzu.vmware.com
1.2.2            Reconcile succeeded
  image-policy-webhook                image-policy-webhook.signing.apps.tanzu.vmware.c
om   1.1.3            Reconcile succeeded
  learningcenter                      learningcenter.tanzu.vmware.com
0.2.1            Reconcile succeeded
  learningcenter-workshops            workshops.learningcenter.tanzu.vmware.com
0.2.1            Reconcile succeeded
  metadata-store                      metadata-store.apps.tanzu.vmware.com
1.2.2            Reconcile succeeded
  ootb-delivery-basic                 ootb-delivery-basic.tanzu.vmware.com
0.8.0-build.4    Reconcile succeeded
  ootb-supply-chain-testing-scanning  ootb-supply-chain-testing-scanning.tanzu.vmware.
com  0.8.0-build.4    Reconcile succeeded
  ootb-templates                      ootb-templates.tanzu.vmware.com
0.8.0-build.4    Reconcile succeeded
  policy-controller                   policy.apps.tanzu.vmware.com
1.0.1            Reconcile succeeded
  scanning                            scanning.apps.tanzu.vmware.com
1.2.2            Reconcile succeeded
  service-bindings                    service-bindings.labs.vmware.com
0.7.2            Reconcile succeeded
  services-toolkit                    services-toolkit.tanzu.vmware.com
0.7.1            Reconcile succeeded
  source-controller                   controller.source.apps.tanzu.vmware.com
0.4.1            Reconcile succeeded
  spring-boot-conventions             spring-boot-conventions.tanzu.vmware.com
0.4.1            Reconcile succeeded
  tap                                 tap.tanzu.vmware.com
1.2.0            Reconcile succeeded
  tap-auth                            tap-auth.tanzu.vmware.com
1.0.1            Reconcile succeeded
  tap-gui                             tap-gui.tanzu.vmware.com
1.2.3            Reconcile succeeded
  tap-telemetry                       tap-telemetry.tanzu.vmware.com
0.2.0            Reconcile succeeded
  tekton-pipelines                    tekton.tanzu.vmware.com
0.33.5           Reconcile succeeded
```

Opting out of telemetry collection

This topic tells you how to opt out of the VMware Customer Experience Improvement Program (CEIP). By default, when you install Tanzu Application Platform (commonly known as TAP), you are opted into telemetry collection.

Note: If you opt out of telemetry collection, VMware cannot offer you proactive support and the other benefits that accompany participation in the CEIP.

Turn off telemetry collection

To turn off telemetry collection on your Tanzu Application Platform installation:

1. Ensure your Kubernetes context is pointing to the cluster where Tanzu Application Platform is installed.
2. Run the following `kubectl` command:

   ```
   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 pick up the change:

   ```
   kubectl delete pods --namespace tap-telemetry --all
   ```

Your Tanzu Application Platform deployment no longer emits telemetry, and you are opted out of the CEIP.

Getting started with the Tanzu Application Platform

Welcome to Tanzu Application Platform. The guides in this section provide hands-on instructions for developers and operators to help you get started on Tanzu Application Platform.

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 Application Platform GUI**
  See Install Tanzu Application Platform GUI. If you used the Full or View profiles provided in Installing the Tanzu Application Platform Package and Profiles, you have already installed Tanzu Application Platform GUI.
- **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:

- Deploy your first application on Tanzu Application Platform

For operators:

- Create an application accelerator

Getting started with the Tanzu Application Platform

Welcome to Tanzu Application Platform. The guides in this section provide hands-on instructions for developers and operators to help you get started on Tanzu Application Platform.

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 Application Platform GUI**
  See Install Tanzu Application Platform GUI. If you used the Full or View profiles provided in Installing the Tanzu Application Platform Package and Profiles, you have already installed Tanzu Application Platform GUI.
- **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:

- Deploy your first application on Tanzu Application Platform

For operators:

- Create an application accelerator

Create your application accelerator

This is the first in a series of Getting started how-to guides for operators. It walks you through creating an application accelerator by using Tanzu Application Platform GUI and CLI.

For background information about application accelerators, see Application Accelerator.

What you will do

- Select a Git repository to create your accelerator and clone the repository.
- Create an `accelerator.yaml` file that defines your accelerator and add it to your Git repository.
- Publish your application accelerator and view it in Tanzu Application Platform GUI.
- Begin to work with your accelerator.

Create an application accelerator

You can use any Git repository to create an accelerator provided it:

- Is public
- Contains a README.md file

You can configure these options when you create a repository on GitHub. You need the repository URL to create an accelerator.

To create a new application accelerator by using your Git repository, follow these steps:

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
   ```

   You can use any icon that has a reachable URL.
4. Add the new `accelerator.yaml` file, commit this change, and push it to your Git repository.

Publish the new accelerator

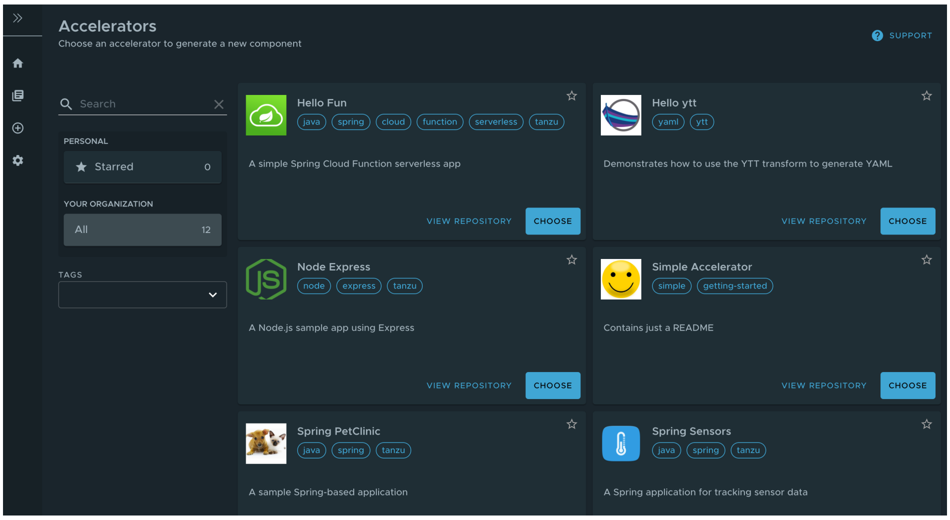
To publish the new application accelerator that is created in your Git repository, follow these steps:

1. To publish the new application accelerator, run:

   ```
   tanzu accelerator create simple --git-repository YOUR-GIT-REPOSITORY-URL --git-
   branch YOUR-GIT-BRANCH
   ```

   Where:

   - `YOUR-GIT-REPOSITORY-URL` is the URL of your Git repository.
   - `YOUR-GIT-BRANCH` is the name of the branch where you pushed the new `accelerator.yaml` file.
2. Refresh Tanzu Application Platform GUI to reveal the newly published accelerator.



Accelerators

Choose an accelerator to generate a new component

Hello Fun — A simple Spring Cloud Function serverless app

Hello ytt — Demonstrates how to use the YTT transform to generate YAML

Node Express — A Node.js sample app using Express

Simple Accelerator — Contains just a README

Spring PetClinic — A sample Spring-based application

Spring Sensors — A Spring application for tracking sensor data

It might take a few seconds for Tanzu Application Platform GUI to refresh the catalog and add an entry for your new accelerator.

Working with accelerators

Updating an accelerator

After you push any changes to your Git repository, the accelerator is refreshed based on the `git.interval` setting for the Accelerator resource. The default value is 10 minutes. To force an immediate reconciliation, run:

```
tanzu accelerator update ACCELERATOR-NAME --reconcile
```

Where `ACCELERATOR-NAME` is the name of your accelerator.

Deleting an accelerator

When you no longer need your accelerator, you can delete it by using the Tanzu CLI:

```
tanzu accelerator delete ACCELERATOR-NAME
```

Using an accelerator manifest

You can also create a separate manifest file and apply it to the cluster by using the Tanzu CLI:

1. Create a `simple-manifest.yaml` file and add the following content:

   ```
   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
   ```

   Where:

   - `YOUR-GIT-REPOSITORY-URL` is the URL of your Git repository.
   - `YOUR-GIT-BRANCH` is the name of the branch.

2. Apply the `simple-manifest.yaml` by running the following command in the directory where you created this file:

   ```
   kubectl apply -f simple-manifest.yaml
   ```

Next steps

- Add testing and security scanning to your application

Create your application accelerator

This is the first in a series of Getting started how-to guides for operators. It walks you through creating an application accelerator by using Tanzu Application Platform GUI and CLI.

For background information about application accelerators, see Application Accelerator.

What you will do

- Select a Git repository to create your accelerator and clone the repository.
- Create an `accelerator.yaml` file that defines your accelerator and add it to your Git repository.
- Publish your application accelerator and view it in Tanzu Application Platform GUI.
- Begin to work with your accelerator.

Create an application accelerator

You can use any Git repository to create an accelerator provided it:

- Is public
- Contains a README.md file

You can configure these options when you create a repository on GitHub. You need the repository URL to create an accelerator.

To create a new application accelerator by using your Git repository, follow these steps:

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
   ```

   You can use any icon that has a reachable URL.
4. Add the new `accelerator.yaml` file, commit this change, and push it to your Git repository.

Publish the new accelerator

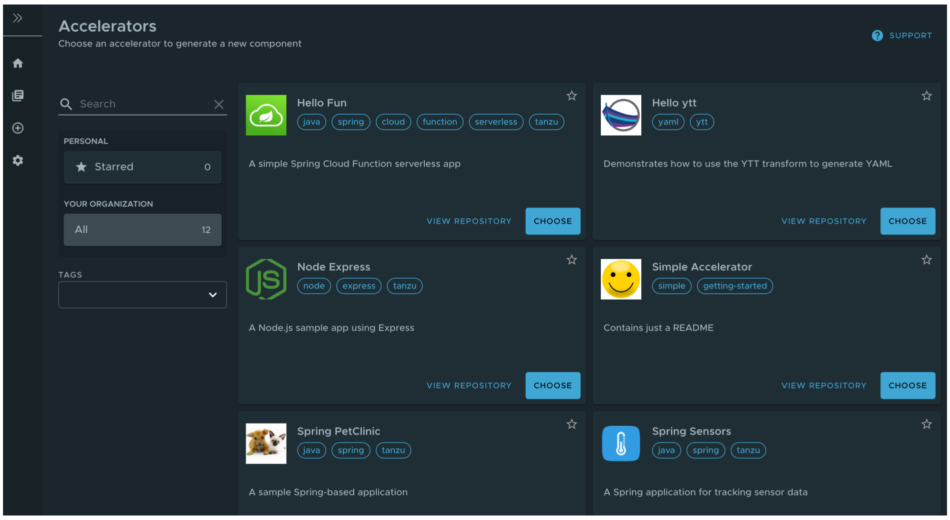
To publish the new application accelerator that is created in your Git repository, follow these steps:

1. To publish the new application accelerator, run:

   ```
   tanzu accelerator create simple --git-repository YOUR-GIT-REPOSITORY-URL --git-
   branch YOUR-GIT-BRANCH
   ```

   Where:

   - `YOUR-GIT-REPOSITORY-URL` is the URL of your Git repository.
   - `YOUR-GIT-BRANCH` is the name of the branch where you pushed the new `accelerator.yaml` file.
2. Refresh Tanzu Application Platform GUI to reveal the newly published accelerator.



A simple Spring Cloud Function serverless app

Demonstrates how to use the YTT transform to generate YAML

A Node.js sample app using Express

Contains just a README

A sample Spring-based application

A Spring application for tracking sensor data

It might take a few seconds for Tanzu Application Platform GUI to refresh the catalog and add an entry for your new accelerator.

Working with accelerators

Updating an accelerator

After you push any changes to your Git repository, the accelerator is refreshed based on the `git.interval` setting for the Accelerator resource. The default value is 10 minutes. To force an immediate reconciliation, run:

```
tanzu accelerator update ACCELERATOR-NAME --reconcile
```

Where `ACCELERATOR-NAME` is the name of your accelerator.

Deleting an accelerator

When you no longer need your accelerator, you can delete it by using the Tanzu CLI:

```
tanzu accelerator delete ACCELERATOR-NAME
```

Using an accelerator manifest

You can also create a separate manifest file and apply it to the cluster by using the Tanzu CLI:

1. Create a `simple-manifest.yaml` file and add the following content:

   ```
   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
   ```

   Where:

   - `YOUR-GIT-REPOSITORY-URL` is the URL of your Git repository.
   - `YOUR-GIT-BRANCH` is the name of the branch.

2. Apply the `simple-manifest.yaml` by running the following command in the directory where you created this file:

   ```
   kubectl apply -f simple-manifest.yaml
   ```

Next steps

- Add testing and security scanning to your application

Add testing and security scanning to your application

This how-to guide walks 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.

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.

Important: You can only have one Tekton pipeline per namespace.

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, `1.2.2`.

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 supply chain to a cluster before the developer gets access.

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: gradle
            script: |-
              cd `mktemp -d`

              wget -qO- $(params.source-url) | tar xvz -m
              ./mvnw test
```

The preceding YAML defines a 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/sample-accelerators/tanzu-java-web-app \
     --git-branch main \
     --type web \
     --label apps.tanzu.vmware.com/has-tests=true \
     --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/
   sample-accelerators/tanzu-java-web-app   True    Fetched revision: main/872ff44
   c8866b7805fb2425130edb69a9853bfdf   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

Prerequisites

- Both the Scan Controller and the default Grype scanner must be installed for scanning. Refer to the verify installation steps later in the topic.

  Note: When leveraging both Tanzu Build Service and Grype in your Tanzu Application Platform supply chain, you can receive enhanced scanning coverage for Java and Node.js workloads that includes application runtime layer dependencies.

- Add the necessary configuration to enable CVE scan results in the Tanzu Application Platform GUI. This configuration allows the Supply Chain Choreographer TAP GUI plug-in to retrieve metadata about project packages and their vulnerabilities.

To install OOTB Supply Chain with Testing and Scanning:

1. Supply Chain Security Tools (SCST) - Scan is installed as part of the Tanzu Application Platform profiles. Verify that both Scan Controller and Grype Scanner are installed by running:

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

   If the packages are not already installed, follow the steps in Supply Chain Security Tools - Scan to install the required scanning components.

   During installation of the Grype Scanner, sample ScanTemplates are installed into the `default` namespace. If the workload is deployed into another namespace, these sample ScanTemplates must also be present in the other namespace. One way to accomplish this is to install Grype Scanner again and provide the namespace in the values file.

   A ScanPolicy is required and must be in the required namespace. A sample ScanPolicy is provided as follows to block a supply chain when CVEs with critical, high, and unknown ratings are found using `notAllowedSeverities := ["Critical","High","UnknownSeverity"]`. 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 text box 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': 'component-a'
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 := input.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 := vuln.ratings.rating[_].severity }
      not isSafe(vuln)
      msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
    }
EOF
```

2. (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.

3. 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>"
```

4. 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, `1.2.2`.

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/sample-accelerators/tanzu-java-web-app \
     --git-branch main \
     --type web \
     --label apps.tanzu.vmware.com/has-tests=true \
     --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/
   sample-accelerators/tanzu-java-web-app   True    Fetched revision: main/872ff44
   c8866b7805fb2425130edb69a9853bfdf   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/sample-accelerators/t
   anzu-java-web-app   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.

Next steps

- Configure image signing and verification in your supply chain

Configure image signing and verification in your supply chain

This how-to guide walks you through configuring your 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. Configure at least one `ClusterImagePolicy` resource to verify image signatures when deploying resources. For instructions, see Create a ClusterImagePolicy resource.

   For example:

   ```
   ---
   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-----
   ```

4. 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

Set up services for consumption by developers

This how-to guide walks service operators and application operators through setting up services for consumption by application developers. For the sake of example, you'll set up the RabbitMQ Cluster Kubernetes operator. You will learn about the `tanzu services` CLI plug-in and the most important APIs for working with services on Tanzu Application Platform.

What you will do

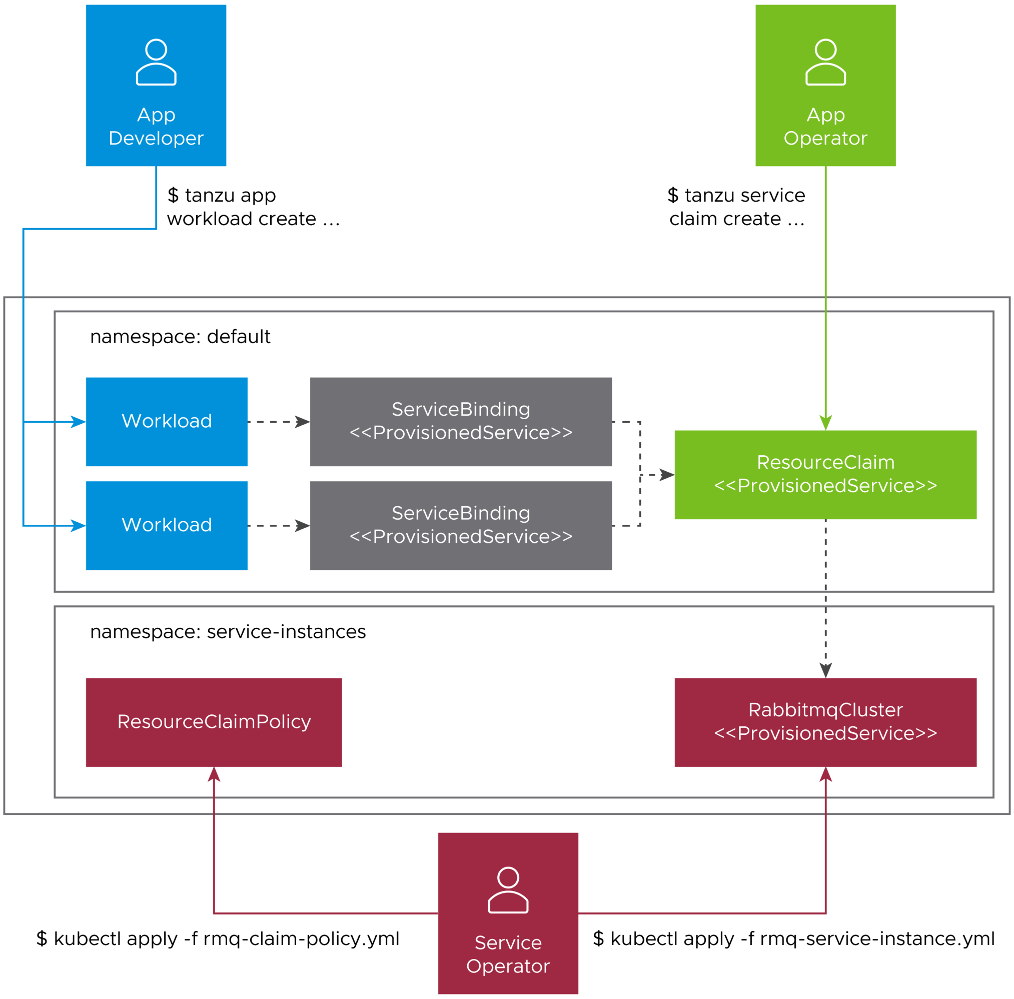
- Set up a service.
- Create a service instance.
- Claim a service instance.

This enables the application developer to bind an application workload to the service instance, as described in Consume services on Tanzu Application Platform.

Before you begin, for important background, see About consuming services on Tanzu Application Platform.

Overview

The following diagram depicts a summary of what this walkthrough covers, including binding an application workload to the service instance by the application developer.


App Developer
$ tanzu app workload create ...
App Operator
$ tanzu service claim create ...
namespace: default
Workload
Workload
ServiceBinding <<ProvisionedService>>
ServiceBinding <<ProvisionedService>>
ResourceClaim <<ProvisionedService>>
namespace: service-instances
ResourceClaimPolicy
RabbitmqCluster <<ProvisionedService>>
$ kubectl apply -f rmq-claim-policy.yml
Service Operator
$ kubectl apply -f rmq-service-instance.yml

Bear the following observations in mind as you work through this guide:

1. There is a clear separation of concerns across the various user roles.
   - The life cycle of workloads is determined by application developers.
   - The life cycle of resource claims is determined by application operators.
   - The life cycle of service instances is determined by service operators.
   - The life cycle of service bindings is implicitly tied to the life cycle of workloads.
2. Resource claims and resource claim policies are the mechanism to enable cross-namespace binding.
3. ProvisionedService is the contract allowing credentials and connectivity information to flow from the service instance, to the resource claim, to the service binding, and ultimately to the application workload. For more information, see ProvisionedService on GitHub.
4. Exclusivity of resource claims: Resource claims are considered to be mutually exclusive, meaning that service instances can be claimed by at most one resource claim.

Prerequisites

Before following this walkthrough, you must:

1. Have access to a cluster with Tanzu Application Platform installed.
2. Have downloaded and installed the Tanzu CLI and the corresponding plug-ins.
3. Ensure that your Tanzu Application Platform cluster can pull the container images required by the Kubernetes operator providing the service. For more information, see VMware RabbitMQ for Kubernetes.

Although the examples in this walkthrough use the RabbitMQ Cluster Kubernetes operator, the setup steps remain largely the same for any compatible operator.

This walkthrough uses the open source RabbitMQ Cluster operator for Kubernetes. For most real-world deployments, VMware recommends that you use the official, supported version provided by VMware. For more information, see the following VMware provided services:

- VMware RabbitMQ for Kubernetes.
- VMware SQL with Postgres for Kubernetes.
- VMware SQL with MySQL for Kubernetes.

Set up a service

This section covers the following:

- Installing the selected service Kubernetes operator.
- Creating the role-based access control (RBAC) rules to grant Tanzu Application Platform permission to interact with the APIs provided by the newly installed Kubernetes operator.
- Creating the additional supporting resources to aid with discovery of services.

For this part of the walkthrough, you assume the role of the **service operator**.

To set up a service:

1. Use `kapp` to install the open source RabbitMQ Cluster Kubernetes operator by running:

   ```
   kapp -y deploy --app rmq-operator --file https://github.com/rabbitmq/cluster-operator/releases/latest/download/cluster-operator.yml
   ```

   As a result, a new API Group (`rabbitmq.com`) and Kind (`RabbitmqCluster`) are now available in the cluster.

2. Apply RBAC rules to grant Tanzu Application Platform permission to interact with the new API.

   1. In a file named `resource-claims-rmq.yaml`, create a `ClusterRole` that defines the rules and label it so that the rules are aggregated to the appropriate controller:

      ```
      # resource-claims-rmq.yaml
      ---
      apiVersion: rbac.authorization.k8s.io/v1
      kind: ClusterRole
      metadata:
        name: resource-claims-rmq
        labels:
          servicebinding.io/controller: "true"
      rules:
      - apiGroups: ["rabbitmq.com"]
        resources: ["rabbitmqclusters"]
        verbs: ["get", "list", "watch"]
      ```

   2. Apply `resource-claims-rmq.yaml` by running:

      ```
      kubectl apply -f resource-claims-rmq.yaml
      ```

3. Make the new API discoverable to application operators.

   1. In a file named `rabbitmqcluster-clusterinstanceclass.yaml`, create a `ClusterInstanceClass` that refers to the new service, and set any additional metadata. For example:

      ```
      # rabbitmqcluster-clusterinstanceclass.yaml
      ---
      apiVersion: services.apps.tanzu.vmware.com/v1alpha1
      kind: ClusterInstanceClass
      metadata:
        name: rabbitmq
      spec:
        description:
          short: It's a RabbitMQ cluster!
        pool:
          group: rabbitmq.com
          kind: RabbitmqCluster
      ```

   2. Apply `rabbitmqcluster-clusterinstanceclass.yaml` by running:

      ```
      kubectl apply -f rabbitmqcluster-clusterinstanceclass.yaml
      ```

      After applying this resource, it is listed in the output of the `tanzu service classes list` command, and is discoverable in the `tanzu` tooling.

Create a service instance

This section covers the following:

- Using kubectl to create a `RabbitmqCluster` service instance.
- Creating a resource claim policy that permits the service instance to be claimed.

For this part of the walkthrough, you assume the role of the **service operator**.

To create a service instance:

1. Create a dedicated namespace for service instances by running:

   ```
   kubectl create namespace service-instances
   ```

   Note: Using namespaces to separate service instances from application workloads allows for greater separation of concerns, and means that you can achieve greater control over who has access to what. However, this is not a strict requirement. You can create both service instances and application workloads in the same namespace.

2. Create a `RabbitmqCluster` service instance.

   1. Create a file named `rmq-1-service-instance.yaml`:

      ```
      # rmq-1-service-instance.yaml
      ---
      apiVersion: rabbitmq.com/v1beta1
      kind: RabbitmqCluster
      metadata:
        name: rmq-1
        namespace: service-instances
      ```

   2. Apply `rmq-1-service-instance.yaml` by running:

      ```
      kubectl apply -f rmq-1-service-instance.yaml
      ```

3. Create a resource claim policy to define the namespaces the instance can be claimed and bound from.

   Note: By default, you can only claim and bind to service instances that are running in the *same* namespace as the application workloads. To claim service instances running in a different namespace, you must create a resource claim policy.

1. Create a file named `rmq-claim-policy.yaml` as follows:

   ```
   # rmq-claim-policy.yaml
   ---
   apiVersion: services.apps.tanzu.vmware.com/v1alpha1
   kind: ResourceClaimPolicy
   metadata:
     name: rabbitmqcluster-cross-namespace
     namespace: service-instances
   spec:
     consumingNamespaces:
     - '*'
     subject:
       group: rabbitmq.com
       kind: RabbitmqCluster
   ```

2. Apply `rmq-claim-policy.yaml` by running:

   ```
   kubectl apply -f rmq-claim-policy.yaml
   ```

This policy states that any resource of kind `RabbitmqCluster` on the `rabbitmq.com` API group in the `service-instances` namespace can be consumed from any namespace.

Claim a service instance

This section covers the following:

- Using `tanzu service claimable list --class` to view details about service instances claimable from a namespace.
- Using `tanzu service claim create` to create a claim for the service instance.

For this part of the walkthrough, you assume the role of the **application operator**.

Resource claims in Tanzu Application Platform are a powerful concept that serve many purposes. Arguably their most important role is to enable application operators to request services that they can use with their application workloads without having to create and manage the services themselves. For more information, see Resource Claims.

In cases where service instances are running in the same namespace as application workloads, you do not have to create a claim. You can bind to the service instance directly.

In this section you use the `tanzu service claim create` command to create a claim that the `RabbitmqCluster` service instance you created earlier can fulfill. This command requires the following information to create a claim successfully:

- `--resource-name`
- `--resource-kind`
- `--resource-api-version`
- `--resource-namespace`

To claim a service instance:

1. Find the claimable instance and the necessary claim information by running:

   ```
   tanzu service claimable list --class rabbitmq
   ```

   Expected output:

   ```
     tanzu service claimable list --class rabbitmq

     NAME    NAMESPACE            API KIND          API GROUP/VERSION
     rmq-1   service-instances    RabbitmqCluster   rabbitmq.com/v1beta1
   ```

2. Using the information from the previous command, create a claim for the service instance by running:

```
tanzu service claim create rmq-1 \
  --resource-name rmq-1 \
  --resource-namespace service-instances \
  --resource-kind RabbitmqCluster \
  --resource-api-version rabbitmq.com/v1beta1
```

You have successfully set the scene for the application developer to inspect the claim and to use it to bind to application workloads, as described in Consume services on Tanzu Application Platform.

Further use cases and reading

There are more service use cases not covered in this getting started guide. See the following:

Use Case	Short Description
Consuming AWS RDS on Tanzu Application Platform	Using the Controllers for Kubernetes (ACK) to provision an RDS instance and consume it from a Tanzu Application Platform workload. Involves making a third-party API consumable from Tanzu Application Platform.
Consuming AWS RDS on Tanzu Application Platform with Crossplane	Using Crossplane to provision an RDS instance and consume it from a Tanzu Application Platform workload. Involves making a third-party API consumable from Tanzu Application Platform.
Consuming Google Cloud SQL on Tanzu Application Platform with Config Connector	Using GCP Config Connector to provision a Cloud SQL instance and consume it from a Tanzu Application Platform workload. Involves making a third-party API consumable from Tanzu Application Platform.
Consuming Google Cloud SQL on Tanzu Application Platform with Crossplane	Using Crossplane to provision a Cloud SQL instance and consume it from a Tanzu Application Platform workload. Involves making a third-party API consumable from Tanzu Application Platform.
Direct Secret References	Binding to services running external to the cluster, for example, an in-house oracle database. Binding to services that do not conform with the binding specification.
Dedicated Service Clusters (Experimental)	Separates application workloads from service instances across dedicated clusters.

For more information about the APIs and concepts underpinning Services on Tanzu Application Platform, see the Services Toolkit Component documentation.

Next steps

Now that you've completed the Getting started guides, learn about:

- Multicluster Tanzu Application Platform

Deploy your first application on Tanzu Application Platform

This is the first in a series of Getting started how-to guides for developers. It walks you through deploying your first application on Tanzu Application Platform by using the Tanzu Application Platform GUI.

Note: This walk-through uses the Tanzu Application Platform GUI. Alternatively, you can deploy your first application on Tanzu Application Platform using the Application Accelerator Extension for VS Code.

What you will do

- Download an application accelerator, which serves as a template that codifies best practices and provides important configuration and structures ready and available for use.
- Upload it to your Git repository of choice.

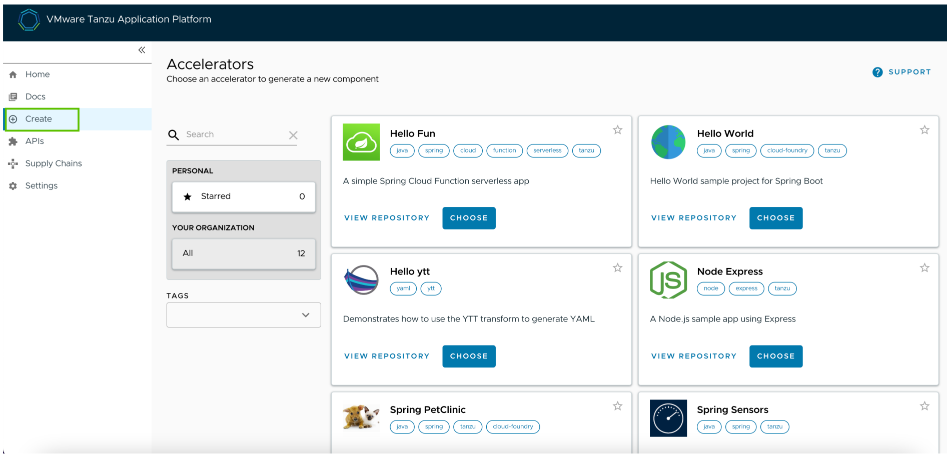
- Run a CLI command to deploy the app.
- View the build and runtime logs for your app.
- View the Web App in your browser.
- (Optional) Add your application to Tanzu Application Platform GUI software catalog.

Before you start, complete all Getting started prerequisites. For background on application accelerators, see Application Accelerator.

Deploy your application through Tanzu Application Platform GUI

To deploy your application, you must download an accelerator, upload it on your Git repository of choice, and run a CLI command. In this example, we use the `Tanzu-Java-Web-App` accelerator. We also use the Tanzu Application Platform GUI. For information about connecting to Tanzu Application Platform GUI, see Accessing Tanzu Application Platform GUI.

1. From Tanzu Application Platform GUI portal, click **Create** located on the left-hand side of the navigation bar to see the list of available accelerators.



A simple Spring Cloud Function serverless app

Hello World sample project for Spring Boot

Demonstrates how to use the YTT transform to generate YAML

A Node.js sample app using Express

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** field 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 profiles. Click **NEXT**, verify the provided information, and click **GENERATE ACCELERATOR**.
4. After the Task Activity processes complete, click **DOWNLOAD ZIP FILE**.
5. After downloading the ZIP file, expand it in a workspace directory and follow your preferred procedure for uploading the generated project files to a Git repository for your new project.
6. Deploy the Tanzu Java Web App accelerator by running the `tanzu apps workload create` command:

```
tanzu apps workload create tanzu-java-web-app \
--git-repo GIT-URL-TO-PROJECT-REPO \
--git-branch main \
--type web \
--label app.kubernetes.io/part-of=tanzu-java-web-app \
--yes \
--namespace YOUR-DEVELOPER-NAMESPACE
```

Where:

   - `GIT-URL-TO-PROJECT-REPO` is the path you uploaded to earlier.
   - `YOUR-DEVELOPER-NAMESPACE` is the namespace configured earlier.

If you bypassed step 5 or were unable to upload your accelerator to a Git repository, use the following public version to test:

```
tanzu apps workload create tanzu-java-web-app \
--git-repo https://github.com/sample-accelerators/tanzu-java-web-app \
--git-branch main \
--type web \
--label app.kubernetes.io/part-of=tanzu-java-web-app \
--yes \
--namespace YOUR-DEVELOPER-NAMESPACE
```

Where `YOUR-DEVELOPER-NAMESPACE` is the namespace configured earlier.

For more information, see Tanzu Apps Workload Create.

Note: This deployment uses an accelerator source from Git, but in later steps you use the VS Code extension to debug and live-update this application.

7. View the build and runtime logs for your app by running the `tail` command:

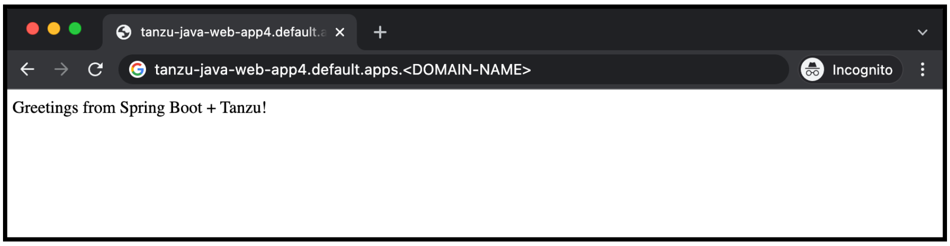
```
tanzu apps workload tail tanzu-java-web-app --since 10m --timestamp --namespace
YOUR-DEVELOPER-NAMESPACE
```

Where `YOUR-DEVELOPER-NAMESPACE` is the namespace configured earlier.

8. After the workload is built and running, you can view the Web App in your browser. View the URL of the Web App by running the following command, and then press **ctrl-click** on the Workload Knative Services URL at the bottom of the command output.

```
tanzu apps workload get tanzu-java-web-app --namespace YOUR-DEVELOPER-NAMESPACE
```

Where `YOUR-DEVELOPER-NAMESPACE` is the namespace configured earlier.

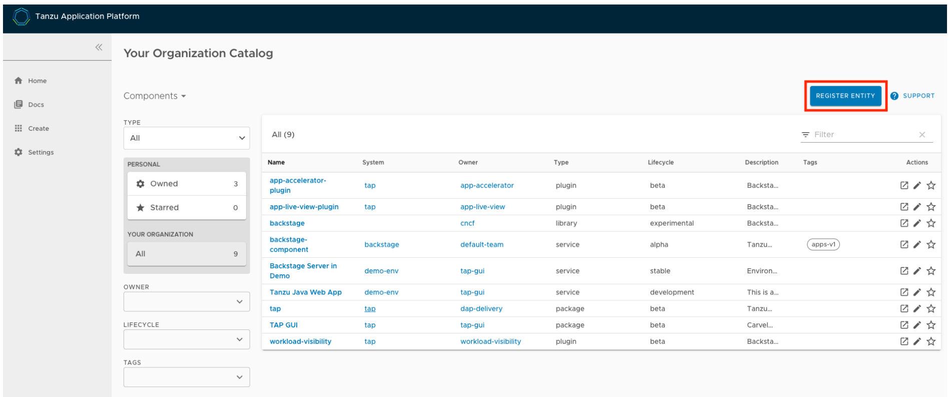


tanzu-java-web-app4.default.apps.<DOMAIN-NAME>

Greetings from Spring Boot + Tanzu!

Add your application to Tanzu Application Platform GUI software catalog

1. Navigate to the home page of Tanzu Application Platform GUI and click **Home**, located on the left navigation bar. Click **REGISTER ENTITY**.



Your Organization Catalog

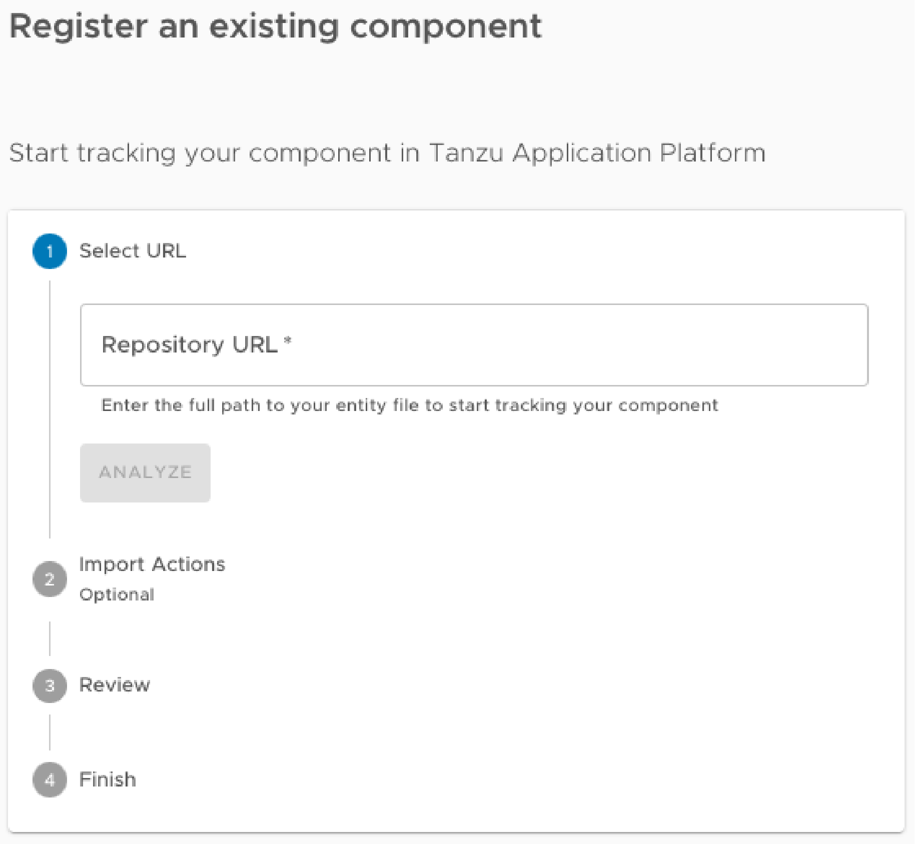
All (9)

Name	System	Owner	Type	Lifecycle	Description	Tags	Actions
app-accelerator-plugin	tap	app-accelerator	plugin	beta	Backsta...		
app-live-view-plugin	tap	app-live-view	plugin	beta	Backsta...		
backstage		cncf	library	experimental	Backsta...		
backstage-component	backstage	default-team	service	alpha	Tanzu...	apps-v1	
Backstage Server in Demo	demo-env	tap-gui	service	stable	Environ...		
Tanzu Java Web App	demo-env	tap-gui	service	development	This is a...		
tap	tap	dap-delivery	package	beta	Tanzu...		
TAP GUI	tap	tap-gui	package	beta	Carvel...		
workload-visibility	tap	workload-visibility	plugin	beta	Backsta...		

Alternatively, you can add a link to the `catalog-info.yaml` to the `tap-values.yaml` configuration file in the `tap_gui.app_config.catalog.locations` section. See Installing the Tanzu Application Platform Package and Profiles.

2. **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 field. For example, `https://github.com/USERNAME/PROJECTNAME/blob/main/catalog-info.yaml`.

3. Click **ANALYZE**.

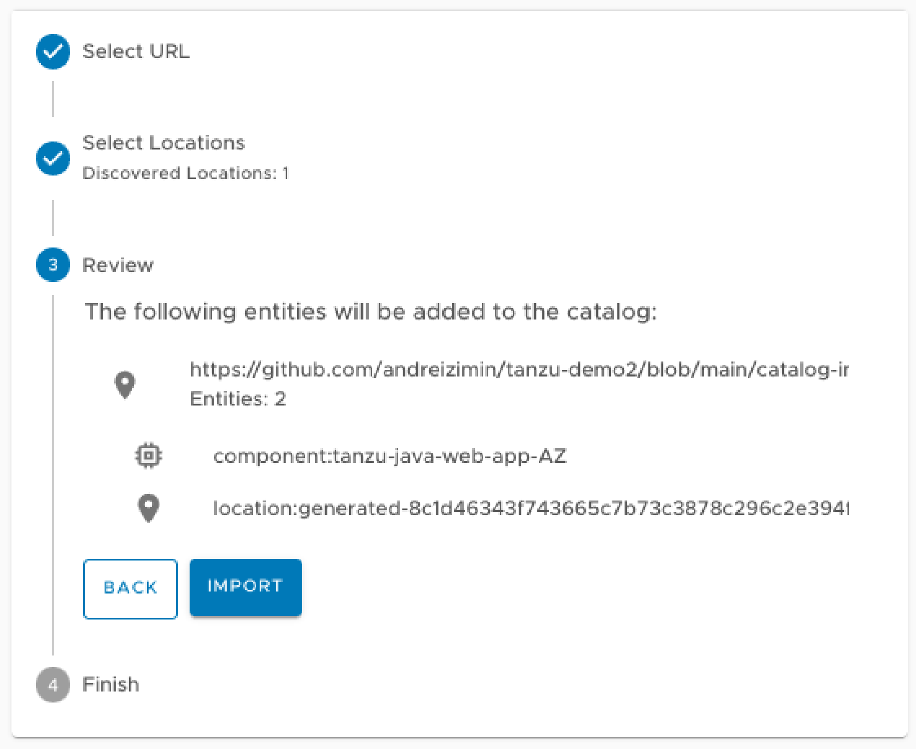


Register an existing component

Start tracking your component in Tanzu Application Platform

1
Select URL
Repository URL *
Enter the full path to your entity file to start tracking your component
ANALYZE
2
Import Actions
Optional
3
Review
4
Finish

4. Review the catalog entities to be added and click **IMPORT**.



The following entities will be added to the catalog:

https://github.com/andreizimin/tanzu-demo2/blob/main/catalog-ir
Entities: 2

component:tanzu-java-web-app-AZ

location:generated-8c1d46343f743665c7b73c3878c296c2e394‍

5. Navigate back to the home page. The catalog changes and entries are visible for further inspection.

Note: If your Tanzu Application Platform GUI instance does not have a PostgreSQL database configured, the `catalog-info.yaml` location must be re-registered after the instance is restarted or upgraded.

Next steps

- Iterate on your new application

Deploy your first application on Tanzu Application Platform

This is the first in a series of Getting started how-to guides for developers. It walks you through deploying your first application on Tanzu Application Platform by using the Tanzu Application Platform GUI.

Note: This walk-through uses the Tanzu Application Platform GUI. Alternatively, you can deploy your first application on Tanzu Application Platform using the Application Accelerator Extension for VS Code.

What you will do

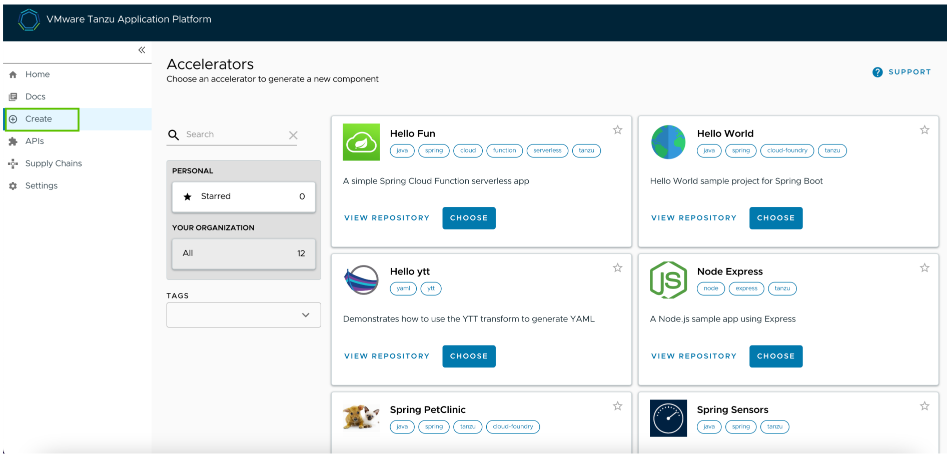
- Download an application accelerator, which serves as a template that codifies best practices and provides important configuration and structures ready and available for use.
- Upload it to your Git repository of choice.
- Run a CLI command to deploy the app.
- View the build and runtime logs for your app.
- View the Web App in your browser.
- (Optional) Add your application to Tanzu Application Platform GUI software catalog.

Before you start, complete all Getting started prerequisites. For background on application accelerators, see Application Accelerator.

Deploy your application through Tanzu Application Platform GUI

To deploy your application, you must download an accelerator, upload it on your Git repository of choice, and run a CLI command. In this example, we use the `Tanzu-Java-Web-App` accelerator. We also use the Tanzu Application Platform GUI. For information about connecting to Tanzu Application Platform GUI, see Accessing Tanzu Application Platform GUI.

1. From Tanzu Application Platform GUI portal, click **Create** located on the left-hand side of the navigation bar to see the list of available accelerators.



Accelerators

Choose an accelerator to generate a new component

Hello Fun: A simple Spring Cloud Function serverless app

Hello World: Hello World sample project for Spring Boot

Hello ytt: Demonstrates how to use the YTT transform to generate YAML

Node Express: A Node.js sample app using Express

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** field 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 profiles. Click **NEXT**, verify the provided information, and click **GENERATE ACCELERATOR**.

4. After the Task Activity processes complete, click **DOWNLOAD ZIP FILE**.

5. After downloading the ZIP file, expand it in a workspace directory and follow your preferred procedure for uploading the generated project files to a Git repository for your new project.

6. Deploy the Tanzu Java Web App accelerator by running the `tanzu apps workload create` command:

   ```
   tanzu apps workload create tanzu-java-web-app \
   --git-repo GIT-URL-TO-PROJECT-REPO \
   --git-branch main \
   --type web \
   --label app.kubernetes.io/part-of=tanzu-java-web-app \
   --yes \
   --namespace YOUR-DEVELOPER-NAMESPACE
   ```

   Where:

   - `GIT-URL-TO-PROJECT-REPO` is the path you uploaded to earlier.
   - `YOUR-DEVELOPER-NAMESPACE` is the namespace configured earlier.

   If you bypassed step 5 or were unable to upload your accelerator to a Git repository, use the following public version to test:

   ```
   tanzu apps workload create tanzu-java-web-app \
   --git-repo https://github.com/sample-accelerators/tanzu-java-web-app \
   --git-branch main \
   --type web \
   --label app.kubernetes.io/part-of=tanzu-java-web-app \
   --yes \
   --namespace YOUR-DEVELOPER-NAMESPACE
   ```

   Where `YOUR-DEVELOPER-NAMESPACE` is the namespace configured earlier.

   For more information, see Tanzu Apps Workload Create.

   Note: This deployment uses an accelerator source from Git, but in later steps you use the VS Code extension to debug and live-update this application.

7. View the build and runtime logs for your app by running the `tail` command:

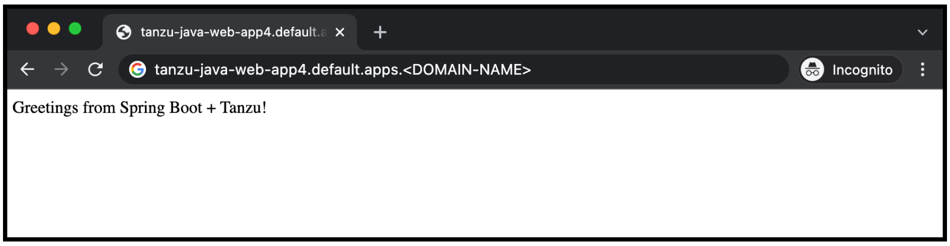
   ```
   tanzu apps workload tail tanzu-java-web-app --since 10m --timestamp --namespace
   YOUR-DEVELOPER-NAMESPACE
   ```

   Where `YOUR-DEVELOPER-NAMESPACE` is the namespace configured earlier.

8. After the workload is built and running, you can view the Web App in your browser. View the URL of the Web App by running the following command, and then press **ctrl-click** on the Workload Knative Services URL at the bottom of the command output.

   ```
   tanzu apps workload get tanzu-java-web-app --namespace YOUR-DEVELOPER-NAMESPACE
   ```

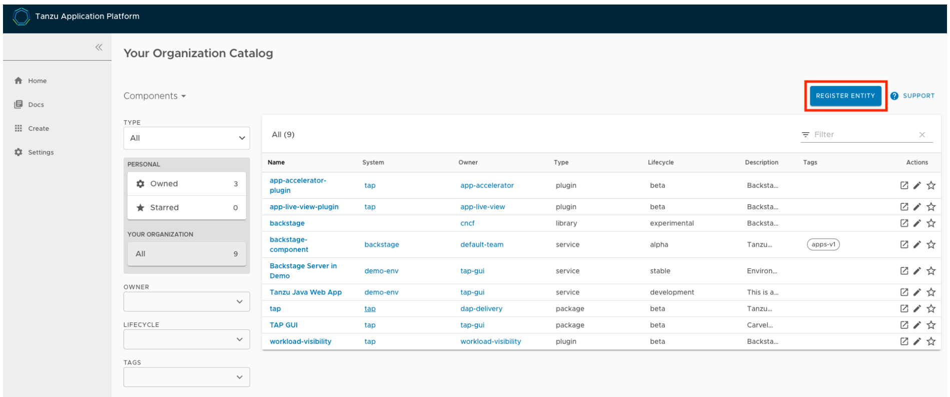
   Where `YOUR-DEVELOPER-NAMESPACE` is the namespace configured earlier.


tanzu-java-web-app4.default.a
tanzu-java-web-app4.default.apps.<DOMAIN-NAME>
Incognito

Greetings from Spring Boot + Tanzu!

Add your application to Tanzu Application Platform GUI software catalog

1. Navigate to the home page of Tanzu Application Platform GUI and click **Home**, located on the left navigation bar. Click **REGISTER ENTITY**.

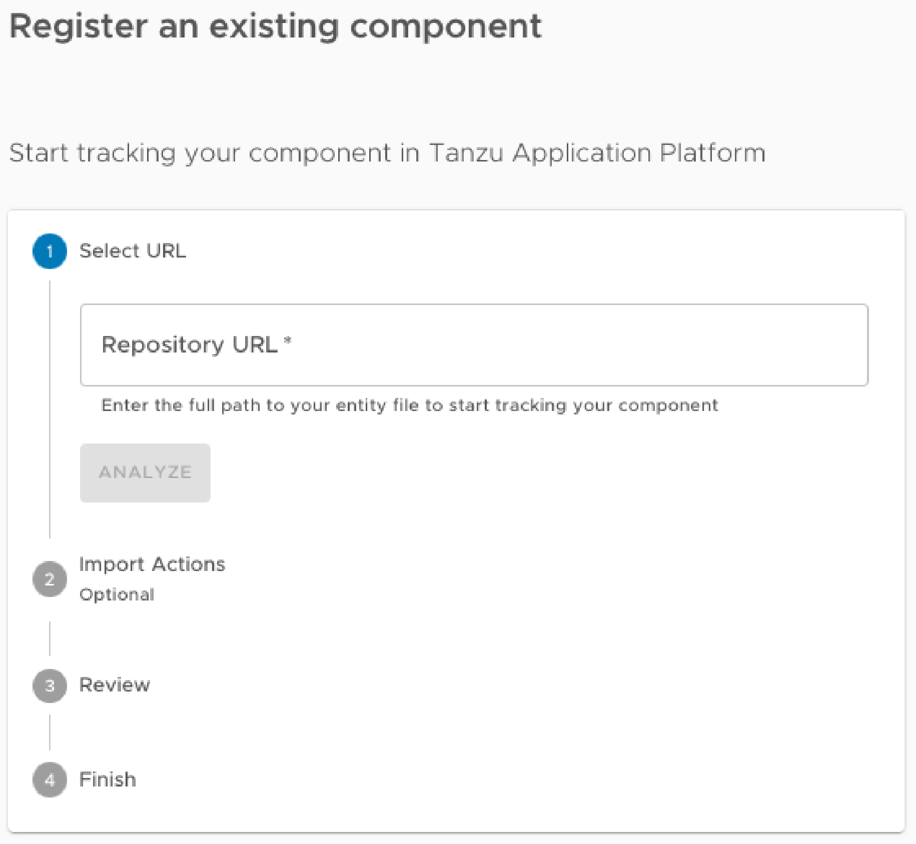
   

Name	System	Owner	Type	Lifecycle	Description	Tags	Actions
app-accelerator-plugin	tap	app-accelerator	plugin	beta	Backsta...		
app-live-view-plugin	tap	app-live-view	plugin	beta	Backsta...		
backstage		cncf	library	experimental	Backsta...		
backstage-component	backstage	default-team	service	alpha	Tanzu...	apps-v1	
Backstage Server in Demo	demo-env	tap-gui	service	stable	Environ...		
Tanzu Java Web App	demo-env	tap-gui	service	development	This is a...		
tap	tap	dap-delivery	package	beta	Tanzu...		
TAP GUI	tap	tap-gui	package	beta	Carvel...		
workload-visibility	tap	workload-visibility	plugin	beta	Backsta...		

   Alternatively, you can add a link to the `catalog-info.yaml` to the `tap-values.yaml` configuration file in the `tap_gui.app_config.catalog.locations` section. See Installing the Tanzu Application Platform Package and Profiles.

2. **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 field. For example, `https://github.com/USERNAME/PROJECTNAME/blob/main/catalog-info.yaml`.

3. Click **ANALYZE**.



Register an existing component

Start tracking your component in Tanzu Application Platform

1 Select URL

Repository URL *

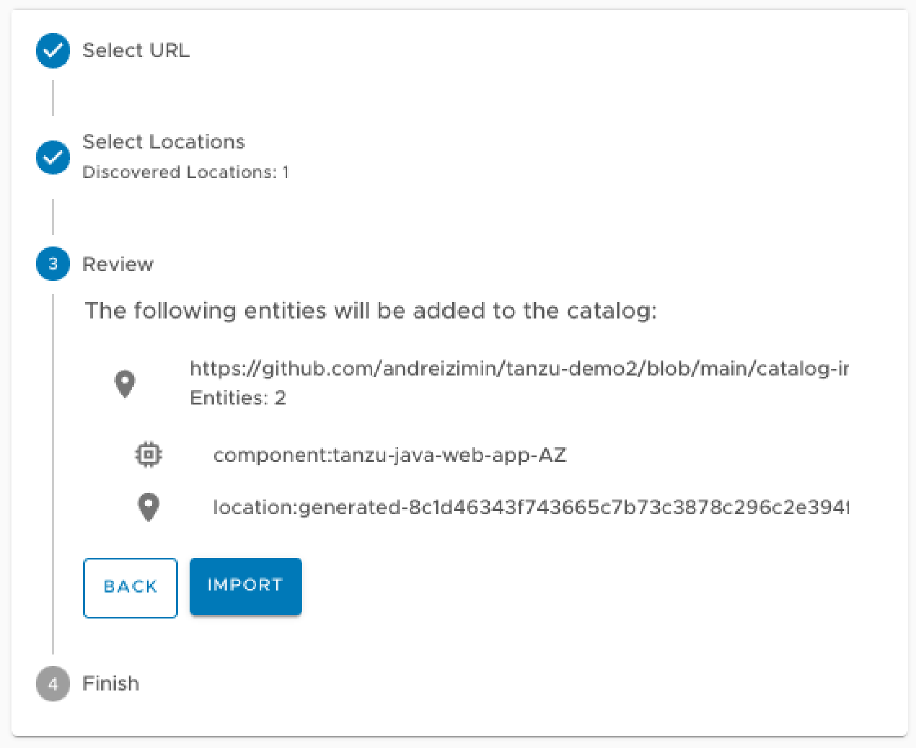
Enter the full path to your entity file to start tracking your component

2 Import Actions
Optional

3 Review

4 Finish

4. Review the catalog entities to be added and click **IMPORT**.



Select URL

Select Locations
Discovered Locations: 1

3 Review

The following entities will be added to the catalog:

https://github.com/andreizimin/tanzu-demo2/blob/main/catalog-ir
Entities: 2

component:tanzu-java-web-app-AZ

location:generated-8c1d46343f743665c7b73c3878c296c2e394(

4 Finish

5. Navigate back to the home page. The catalog changes and entries are visible for further inspection.

Note: If your Tanzu Application Platform GUI instance does not have a PostgreSQL database configured, the `catalog-info.yaml` location must be re-registered after the instance is restarted or upgraded.

Next steps

- Iterate on your new application

Iterate on your new application

This how-to guide walks you through starting to iterate on your first application on Tanzu Application Platform, which you deployed in the previous how-to, Deploy your first application.

What you will do

- Prepare your IDE to iterate on your application.
- 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.

Prepare your IDE 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 a skeleton workload developed, you are ready to begin to iterate on your new application and test code changes on the cluster.

Tanzu Developer Tools for Visual Studio 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 Dev Tools for VS Code.

Important: Use Tilt v0.27.2 or a later version for the sample application.

1. Open the Tanzu Java Web App as a project within your VS Code IDE.
2. To ensure your extension assists you with iterating on the correct project, configure its settings using the following instructions.
    1. In Visual Studio Code, navigate to **Preferences** > **Settings** > **Extensions** > **Tanzu Developer Tools**.
    2. In the **Local Path** field, provide the path to the directory containing the Tanzu Java Web App. The current directory is the default.
    3. In the **Source Image** field, provide the destination image repository to publish an image containing your workload source code. For example, `gcr.io/myteam/tanzu-java-web-app-source`.

You are now ready to iterate on your application.

Live update your application

Deploy the application to view it updating live on the cluster to see how code changes behave on a production cluster.

Follow the following steps to live update your application:

1. From the Command Palette (⇧⌘P), type in and select `Tanzu: Live Update Start`. You can view output from Tanzu Application Platform and from Tilt indicating that the container is being built and deployed.
    - You see "Live Update starting…" in the status bar at the bottom right.

  - Live update can take 1 to 3 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 status bar is visible, resolve to “Live Update Started,” navigate to `http://localhost:8080` in your browser, and 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. Navigate to your browser and refresh the page.
5. View the changes to your workload running on the cluster.
6. Either continue making changes, or stop and deactivate the live update when finished. Open the command palette (⇧⌘P), type **Tanzu**, and choose an option.

Debug your application

Debug your cluster either on the application or in your local environment.

Use the following steps to debug your cluster:

1. Set a breakpoint in your code.
2. Right-click the file `workload.yaml` within the `config` directory and select **Tanzu: Java Debug Start**. In a few moments, the workload is redeployed with debugging enabled. The “Deploy and Connect” Task completes and the debug menu actions are made available to you, indicating that the debugger has attached.
3. Navigate to `http://localhost:8080` in your browser. This hits the breakpoint within VS Code. Play to the end of the debug session using VS Code debugging controls.

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 installed successfully. For instructions, see Install Application Live View.
2. Access the Application Live View Tanzu Application Platform GUI. 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.

Next steps

- Consume services on Tanzu Application Platform

Consume services on Tanzu Application Platform

This how-to guide walks the application developer through deploying two application workloads and configuring them to communicate over RabbitMQ. You will learn about the `tanzu services` CLI plug-in and the most important APIs for working with services on Tanzu Application Platform.

Important: This walkthrough assumes that the service operator and application operator has already:

- Set up a service.
- Created a service instance.
- Claimed the service instance.

This is described in Set up services for consumption by developers.

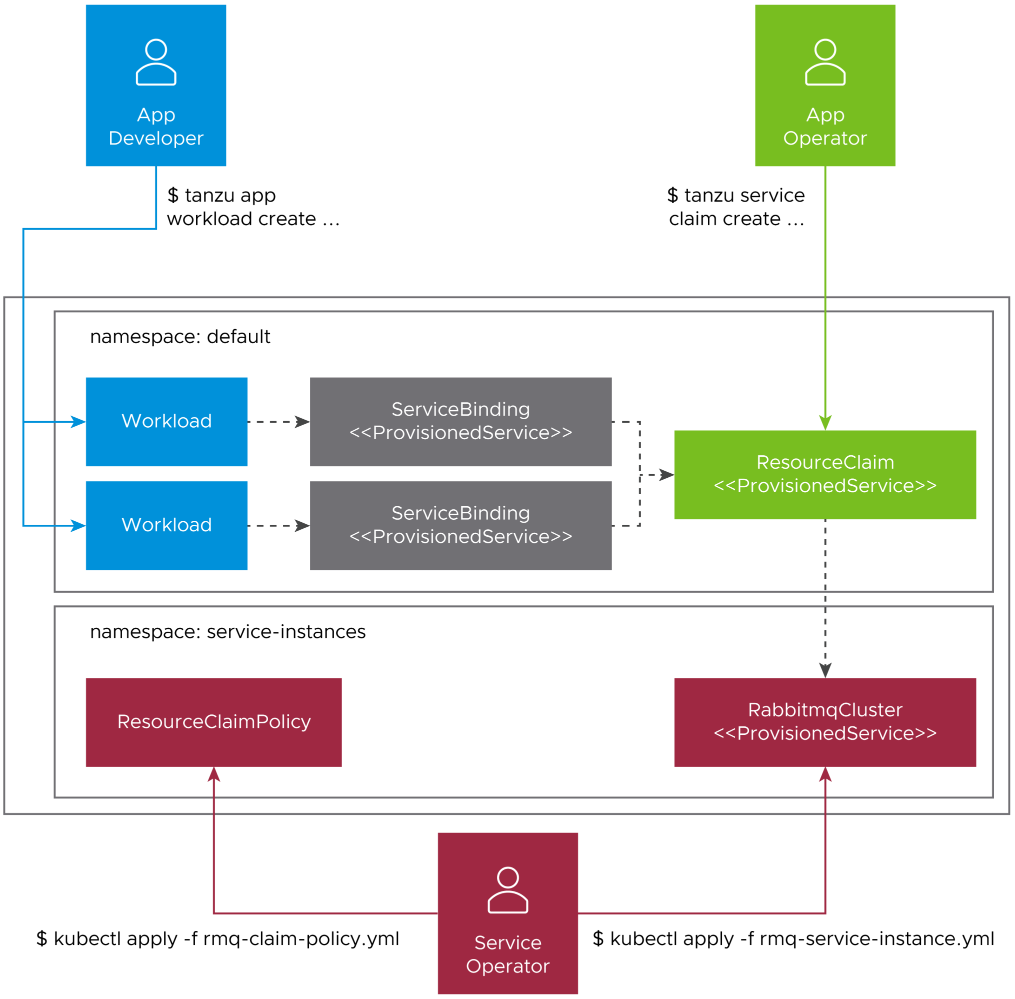
What you will do

- Inspect the resource claim created for the service instance by the application operator.
- Bind the application workload to the ResourceClaim so the workload utilizes the service instance.

Before you begin, for important background, see About consuming services on Tanzu Application Platform.

Overview

The following diagram depicts a summary of what this walkthrough covers, including the work of the service and application operators described in Set up services for consumption by developers.


App Developer
App Operator
$ tanzu app workload create ...
$ tanzu service claim create ...
namespace: default
Workload
Workload
ServiceBinding <<ProvisionedService>>
ServiceBinding <<ProvisionedService>>
ResourceClaim <<ProvisionedService>>
namespace: service-instances
ResourceClaimPolicy
RabbitmqCluster <<ProvisionedService>>
$ kubectl apply -f rmq-claim-policy.yml
Service Operator
$ kubectl apply -f rmq-service-instance.yml

Bear the following observations in mind as you work through this guide:

1. There is a clear separation of concerns across the various user roles.
    - The life cycle of workloads is determined by application developers.
    - The life cycle of resource claims is determined by application operators.
    - The life cycle of service instances is determined by service operators.
    - The life cycle of service bindings is implicitly tied to the life cycle of workloads.
2. ProvisionedService is the contract allowing credentials and connectivity information to flow from the service instance, to the resource claim, to the service binding, and ultimately to the application workload. For more information, see ProvisionedService on GitHub.

Prerequisites

Before following this walkthrough, you must:

1. Have access to a cluster with Tanzu Application Platform installed.
2. Have downloaded and installed the Tanzu CLI and the corresponding plug-ins.
3. Have set up the `default` namespace to use installed packages and use it as your developer namespace. For more information, see Set up developer namespaces to use your installed packages.
4. Ensure your Tanzu Application Platform cluster can pull source code from GitHub.
5. Ensure the service operator and application operator has completed the work of setting up the service, creating the service instance, and claiming the service instance, as described in Set up services for consumption by developers.

As application developer, you are now ready to inspect the resource claim created for the service instance by the application operator in Set up services for consumption by developers and use it to bind to application workloads.

Bind an application workload to the service instance

This section covers:

- Using `tanzu service claim list` and `tanzu service claim get` to find information about the claim to use for binding.
- Using `tanzu apps workload create` with the `--service-ref` flag to create a workload and bind it to the service instance.

You must create application workloads and bind them to the service instance using the claim.

In Tanzu Application Platform, service bindings are created when you create application workloads that specify `.spec.serviceClaims`. In this section, you create such workloads by using the `--service-ref` flag of the `tanzu apps workload create` command.

To create an application workload:

1. Inspect the claims in the developer namespace to find the value to pass to `--service-ref` command by running:

    ```
    tanzu service claim list
    ```

    Expected output:

    ```
      NAME    READY   REASON
      rmq-1   True
    ```

2. Retrieve detailed information about the claim by running:

    ```
    tanzu service claim get rmq-1
    ```

Expected output:

```
Name: rmq-1
Status:
  Ready: True
Namespace: default
Claim Reference: services.apps.tanzu.vmware.com/v1alpha1:ResourceClaim:rmq-1
Resource to Claim:
  Name: rmq-1
  Namespace: service-instances
  Group: rabbitmq.com
  Version: v1beta1
  Kind: RabbitmqCluster
```

3. Record the value of `Claim Reference` from the previous command. This is the value to pass to `--service-ref` to create the application workload.

4. Create the application workload by running:

```
tanzu apps workload create spring-sensors-consumer-web \
  --git-repo https://github.com/sample-accelerators/spring-sensors-rabbit \
  --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:ResourceClaim:rmq-
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:ResourceClaim:rmq-
1"
```

   Using the `--service-ref` flag instructs Tanzu Application Platform to bind the application workload to the service provided in the `ref`.

   Note: You are not passing a service ref to the `RabbitmqCluster` service instance directly, but rather to the resource claim that has claimed the `RabbitmqCluster` service instance. See the consuming services diagram at the beginning of this walkthrough.

5. After the workloads are ready, visit the URL of the `spring-sensors-consumer-web` app. Confirm that sensor data, passing from the `spring-sensors-producer` workload to the `create spring-sensors-consumer-web` workload using the `RabbitmqCluster` service instance, is displayed.

Further use cases and reading

There are more service use cases not covered in this getting started guide. See the following:

Use Case	Short Description
Consuming AWS RDS on Tanzu Application Platform	Using the Controllers for Kubernetes (ACK) to provision an RDS instance and consume it from a Tanzu Application Platform workload. Involves making a third-party API consumable from Tanzu Application Platform.
Consuming AWS RDS on Tanzu Application Platform with Crossplane	Using Crossplane to provision an RDS instance and consume it from a Tanzu Application Platform workload. Involves making a third-party API consumable from Tanzu Application Platform.

Consuming Google Cloud SQL on Tanzu Application Platform with Config Connector	Using GCP Config Connector to provision a Cloud SQL instance and consume it from a Tanzu Application Platform workload. Involves making a third-party API consumable from Tanzu Application Platform.
Consuming Google Cloud SQL on Tanzu Application Platform with Crossplane	Using Crossplane to provision a Cloud SQL instance and consume it from a Tanzu Application Platform workload. Involves making a third-party API consumable from Tanzu Application Platform.
Direct Secret References	Binding to services running external to the cluster, for example, an in-house oracle database. Binding to services that do not conform with the binding specification.
Dedicated Service Clusters (Experimental)	Separates application workloads from service instances across dedicated clusters.

For more information about the APIs and concepts underpinning Services on Tanzu Application Platform, see the Services Toolkit Component documentation.

Next steps

Now that you've completed the Getting started guides, learn about:

- Multicluster Tanzu Application Platform

Deploy your first air-gapped workload (beta)

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 (beta).

What you will do

- Create a workload from Git.
- Create a basic supply chain workload.

Caution: Tanzu Application Platform in an air-gapped environment is currently in beta and is intended for evaluation and test purposes only. Do not use in a production environment.

Create a workload from Git

To create a workload from Git through https, follow these steps:

1. Create a secret in your developer namespace with the caFile that matches the `gitops_ssh_secret` name in tap_values:

   ```
   kubectl create secret generic custom-ca --from-file=caFile=CA_PATH -n NAMESPACE
   ```

2. If you would like to pass in a custom settings.xml for Java, create a file called `settings-xml.yaml` similar to the following 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="http://w
   ww.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 https://maven.a
   pache.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</url>
            <mirrorOf>*</mirrorOf>
        </mirror>
    </mirrors>
    <servers>
         <server>
            <id>reposilite</id>
            <username>USERNAME</username>
            <password>PASSWORD</password>
         </server>
    </servers>
</settings>
```

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. Due to a bug, you must pass in a build environment:

```
tanzu apps workload create APP-NAME --git-repo  https://GITURL --git-branch BRANCH --type web --label app.kubernetes.io/part-of=CATALOGNAME --yes --param-yaml buildServiceBindings='[{"name": "settings-xml", "kind": "Secret"}]' --build-env "BP_MAVEN_BUILD_ARGUMENTS=-Dmaven.test.skip=true --no-transfer-progress package"
```

If you would rather pass the CA certificate in at workload create time, use the following command:

```
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 buildServiceBindings='[{"name": "settings-xml", "kind": "Secret"}]' --param "gitops_ssh_secret=git-ca" --build-env "BP_MAVEN_BUILD_ARGUMENTS=-Dmaven.test.skip=true --no-transfer-progress package"
```

Learn more about Tanzu Application Platform

The articles in this section explain concepts important to getting started with Tanzu Application Platform:

- Application Accelerator
- Supply chains on Tanzu Application Platform
- Vulnerability scanning and metadata storage for your supply chain
- About consuming services on Tanzu Application Platform

Application Accelerator

Developers can create applications and get started with feature development immediately with the help of application accelerators. Application accelerators are templates that not only codify best practices but also provide important configuration and structures ready and available for use.

Enterprise Architects use Application Accelerator to create application accelerators, which provide developers and admins in their organization with ready-made, enterprise-conformant 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 of the Tanzu Application Platform GUI helps you to discover accelerators and to enter additional 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 for using accelerators available in an Application Accelerator server.

Admins use create, update, and delete commands for managing accelerators in a Kubernetes context. When admins want to use 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 your first application on Tanzu Application Platform

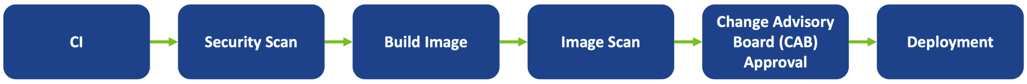
Operators:

- Create an application accelerator

Supply chains on Tanzu Application Platform

Supply chains provide a way of codifying all of the steps of your path to production, more commonly known as continuous integration/Continuous Delivery (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 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 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 additional 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 used to codify the organization's path to production are configurable, allowing their authors to add all of the steps of their application's path to production.

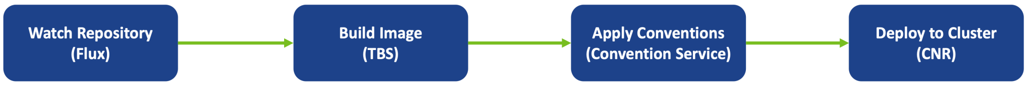
Available Supply Chains

The 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)

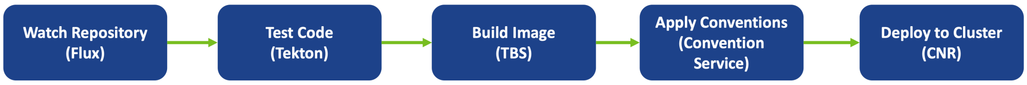
The default **OOTB Basic** supply chain and its dependencies were installed on your cluster during the Tanzu Application Platform install. The following table and diagrams provide descriptions for each of the supply chains and dependencies provided with the Tanzu Application Platform.



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 - Convention Service - Tekton - Cloud Native Runtimes - If using Service References: - Service Bindings - Services Toolkit

2: OOTB Testing

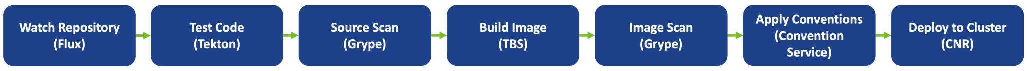
OOTB Testing supply chain runs a Tekton pipeline within the supply chain.



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.



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. - The application source code is scanned for vulnerabilities. - 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 security 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 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 which prevent critical vulnerabilities from flowing to the supply chain unresolved.

Features

Features include:

- Scan source code repositories and images for known 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 repos 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 vulernabilities.
- Supply Chain Choreographer in Tanzu Application Platform GUI visualizes the supply chain, including scans, packages, and vulnerabilities.

Next steps

Apply what you have learned:

- Add testing and security scanning to your application
- Enable CVE scan results in Supply Chain Choreographer in Tanzu Application Platform GUI

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
- TAP GUI - Troubleshooting

About consuming services on Tanzu Application Platform

As part of Tanzu Application Platform, you can work with backing services such as RabbitMQ, PostgreSQL, and MySQL among others. Binding application workloads to service instances is the most common use of services.

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 (host, port, and so on), 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, it may still be possible to provide configurations allowing such services to integrate with Tanzu Application Platform. See the following for examples of how to do this for Amazon AWS RDS.

- Consuming AWS RDS on Tanzu Application Platform (TAP) with AWS Controllers for Kubernetes (ACK)
- Consuming AWS RDS on Tanzu Application Platform (TAP) with Crossplane

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	No (might be introduced in a future release)	• Namespace and cluster topology design • Life cycle management (CRUD) of Kubernetes operators • Life cycle management (CRUD) of service instances • Life cycle management (CRUD) of resource claim policies
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:

- Consume services on Tanzu Application Platform

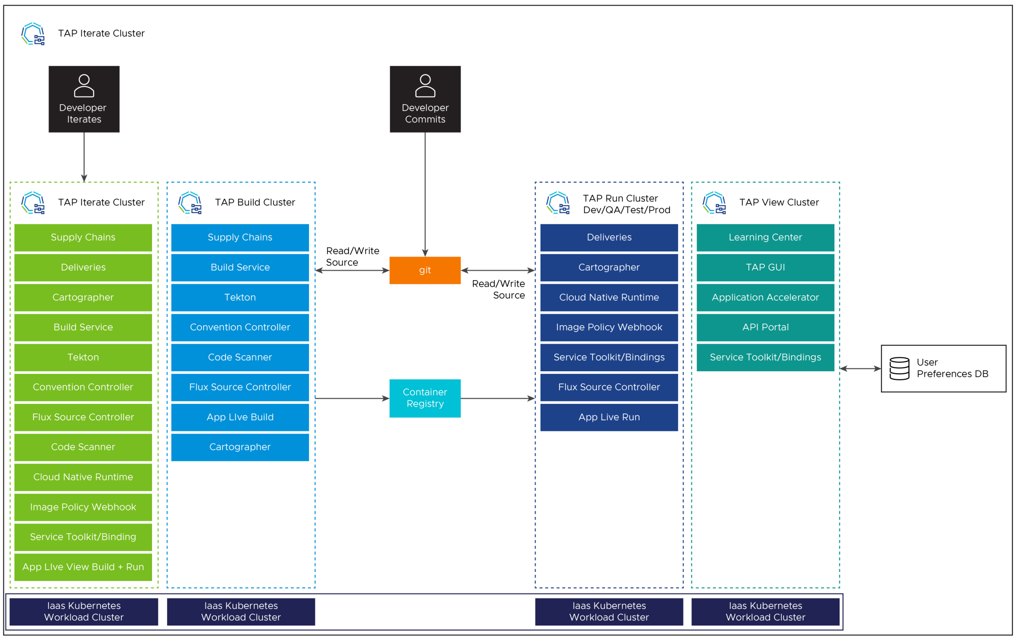
Overview of multicluster Tanzu Application Platform

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 the profile capabilities supported by Tanzu Application Platform. 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 Application Platform GUI 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

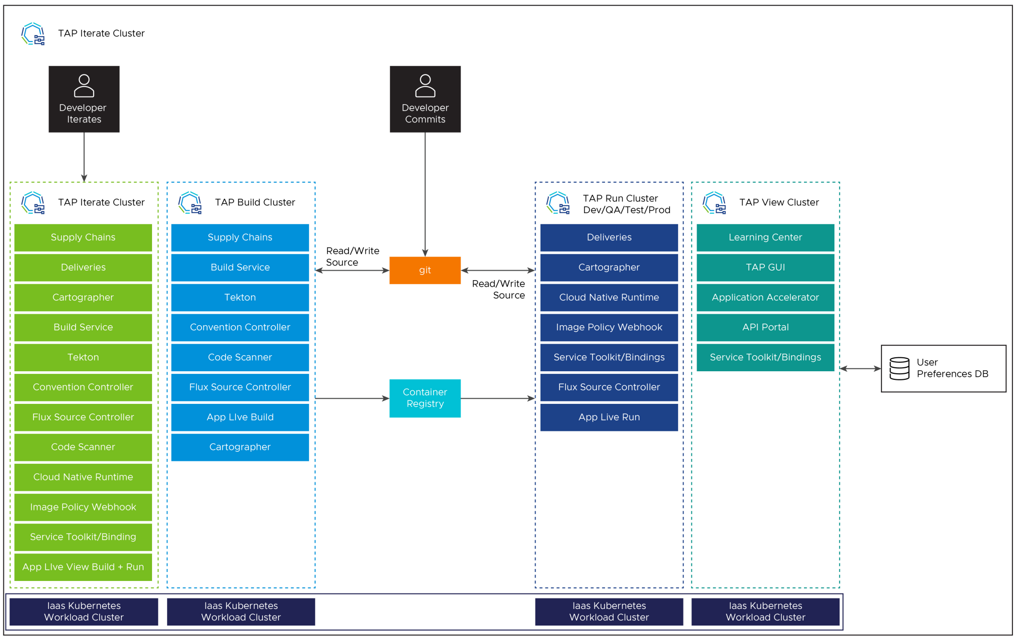
You can install Tanzu Application Platform (commonly known as TAP) in various topologies to reflect your existing landscape. VMware has tested and recommends a multicluster topology for

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 the profile capabilities supported by Tanzu Application Platform. 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 Application Platform GUI 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. Add RBAC, cluster URL, and token from Build and Run clusters as documented in Viewing resources on multiple clusters in Tanzu Application Platform GUI
5. 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 Application Platform GUI:

   ```
   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 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 Application Platform GUI 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. Deploy Supply Chain Security Tools (SCST) - Store. See Multicluster setup for more information.

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.

Add Build and Run clusters to Tanzu Application Platform GUI

After installing the Build, Run and Iterate clusters, follow the steps in View resources on multiple clusters in Tanzu Application Platform GUI to:

1. Create the `Service Accounts` that Tanzu Application Platform GUI 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 3 profiles, you're ready to run a workload by using the supply chain. See Getting started with multicluster Tanzu Application Platform.

Getting 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 Application Platform GUI (commonly known as TAP GUI) 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 3 profiles: Build, Run, and View.
2. For the sample workload, VMware uses the same Application Accelerator - Tanzu Java Web App in the non-multicluster Getting 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 sample-accelerators GitHub repository.
3. The two supply chains are `ootb-supply-chain-basic` on the Build profile and `ootb-delivery-basic` on the Run profile. For both the Build 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/sample-accelerators/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 `Deliverable` for the `Workload` by running:

   ```
   kubectl get deliverable --namespace ${DEVELOPER_NAMESPACE}
   ```

   The output should look simiar to the following:

   ```
   kubectl get deliverable --namespace default
   NAME                 SOURCE
   DELIVERY   READY   REASON             AGE
   tanzu-java-web-app   tapmulticluster.azurecr.io/tap-multi-build-dev/tanzu-java-
   web-app-default-bundle:xxxx-xxxx-xxxx-xxxx-xxxxx              False   DeliveryN
   otFound   28h
   ```

   The `Deliverable` contains the reference to the `source`. In this case, it is a bundle on the image registry you specified for the supply chain. The supply chains can also leverage Git repositories instead of ImageRepositories, but that's beyond the scope of this tutorial.

5. Create a `Deliverable` after verifying there's a `Deliver` on the build cluster. Copy its content to a file that you can take to the Run profile clusters:

   ```
   kubectl get deliverable tanzu-java-web-app --namespace ${DEVELOPER_NAMESPACE} -
   oyaml > deliverable.yaml
   ```

6. Delete the `ownerReferences` and `status` sections from the `deliverable.yaml`.

   After editing, the file will look like the following:

   ```
   apiVersion: carto.run/v1alpha1
   kind: Deliverable
   metadata:
     creationTimestamp: "2022-03-10T14:35:52Z"
     generation: 1
     labels:
       app.kubernetes.io/component: deliverable
       app.kubernetes.io/part-of: tanzu-java-web-app
       app.tanzu.vmware.com/deliverable-type: web
       apps.tanzu.vmware.com/workload-type: web
       carto.run/cluster-template-name: deliverable-template
       carto.run/resource-name: deliverable
       carto.run/supply-chain-name: source-to-url
       carto.run/template-kind: ClusterTemplate
       carto.run/workload-name: tanzu-java-web-app
       carto.run/workload-namespace: default
     name: tanzu-java-web-app
     namespace: default
     resourceVersion: "635368"
     uid: xxxx-xxxx-xxxx-xxxx-xxxx
   spec:
     source:
       image: tapmulticluster.azurecr.io/tap-multi-build-dev/tanzu-java-web-app-de
   fault-bundle:xxxx-xxxx-xxxx-xxxx-xxxx
   ```

7. Take this `Deliverable` file to the **Run** profile clusters by running:

```
kubectl apply -f deliverable.yaml --namespace ${DEVELOPER_NAMESPACE}
```

8. 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
```

9. 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!".

10. View the component in Tanzu Application Platform GUI, 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

- 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.
buildservice:
  kp_default_repository: "KP-DEFAULT-REPO"
  kp_default_repository_username: "KP-DEFAULT-REPO-USERNAME"
  kp_default_repository_password: "KP-DEFAULT-REPO-PASSWORD"
supply_chain: testing_scanning
ootb_supply_chain_testing_scanning:
  registry:
    server: "SERVER-NAME"
    repository: "REPO-NAME"
  gitops:
    ssh_secret: "SSH-SECRET-KEY"
grype:
  namespace: "MY-DEV-NAMESPACE" # (optional) Defaults to default namespace.
  targetImagePullSecret: "TARGET-REGISTRY-CREDENTIALS-SECRET"
  metadataStore:
    url: METADATA-STORE-URL-ON-VIEW-CLUSTER
    caSecret:
        name: store-ca-cert
        importFromNamespace: metadata-store-secrets
    authSecret:
        name: store-auth-token
        importFromNamespace: metadata-store-secrets
scanning:
  metadataStore:
    url: "" # Configuration is moved, so set this string to empty.
```

Where:

- `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"`.
- `SSH-SECRET-KEY` is the SSH secret key in the developer namespace for the supply chain to fetch source code from and push configuration to. See Git authentication for more information.
- `METADATA-STORE-URL-ON-VIEW-CLUSTER` is the URL of the Supply Chain Security Tools (SCST) - Store deployed on the View cluster. For example, `https://metadata-store.example.com`
- `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.

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 dependencies by upgrading Tanzu Application Platform to the latest patch, or by using an automatic update process (deprecated).

See Multicluster setup 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 inherits 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

supply_chain: basic

contour:
  envoy:
    service:
      type: LoadBalancer #NodePort can be used if your Kubernetes cluster doesn't supp
ort LoadBalancing

appliveview_connector:
  backend:
    sslDisabled: TRUE-OR-FALSE-VALUE
    host: appliveview.VIEW-CLUSTER-INGRESS-DOMAIN
```

Where:

- `INGRESS-DOMAIN` is the subdomain for the host name that you point at the `tanzu-shared-ingress` service's external IP address.
- `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.

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"

contour:
  envoy:
    service:
      type: LoadBalancer #NodePort can be used if your Kubernetes cluster doesn't supp
ort LoadBalancing
```

```
tap_gui:
  service_type: ClusterIP
  app_config:
    app:
      baseUrl: http://tap-gui.INGRESS-DOMAIN
    catalog:
      locations:
        - type: url
          target: https://GIT-CATALOG-URL/catalog-info.yaml
    backend:
      baseUrl: http://tap-gui.INGRESS-DOMAIN
      cors:
        origin: http://tap-gui.INGRESS-DOMAIN
    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
```

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. 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 Viewing resources on multiple clusters in Tanzu Application Platform GUI. Observe the order of operations laid out in the previous steps.
- `APP-LIVE-VIEW-INGRESS-DOMAIN` is the subdomain you setup to communicate with the App Live View Connectors on your Run-profile servers. This corresponds to the value key `appliveview_connector.backend.host`.

Troubleshooting 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

Troubleshooting 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 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
```

Example:

```
$ tanzu package install app-live-view -p appliveview.tanzu.vmware.com -v 0.1.0 -n tap-
install -f ./app-live-view.yml

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
```

Example:

```
$ tanzu package install app-accelerator -p accelerator.apps.tanzu.vmware.com -v 0.2.0
-n tap-install -f 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 --values-file tap-values.ya
ml -n tap-install
- 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.yml` 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
     learningcenter            learningcenter.tanzu.vmware.com
   0.1.0-build.6    Reconcile succeeded  tap-install
     learningcenter-workshops  workshops.learningcenter.tanzu.vmware.com
   0.1.0-build.7    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
learningcenter             learningcenter.tanzu.vmware.com
0.1.0             Reconcile succeeded
109m
learningcenter-workshops   workshops.learningcenter.tanzu.vmware.com
0.1.0             Reconcile succeeded
103m
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*.

Unable to add Tanzu Application Platform repo into clusters attached to Tanzu Mission Control with pre-installed Cluster Essentials v1.2.

Explanation

You cannot add Tanzu Application Platform repo into clusters attached to Tanzu Mission Control with pre-installed Cluster Essentials v1.2. Cluster Essentials must be provisioned by Tanzu Mission Control only.

Solution

Do not add a cluster with Cluster Essentials v1.2 predeployed to Tanzu Mission Control. Provision Cluster Essentials through Tanzu Mission Control.

Upgrading Tanzu Application Platform to v1.2.2 might fail

Explanation

While upgrading Tanzu Application Platform to v1.2.2 from any previous version, adding the v1.2.2 repo bundle in addition to the existing repo bundle fails.

Solution

As a workaround, to add the Tanzu Application Platform v1.2.2 repo bundle, run:

```
tanzu package repository update EXISTING-REPO-BUNDLE-NAME --url registry.tanzu.vmware.
com/tanzu-application-platform/tap-packages:1.2.2 -n NAMESPACE
```

You might observe an error with package installs `ReconcileFailed True Expected to find at least one version` until Tanzu Application Platform is upgraded to v1.2.2, but this does not affect the functionality of any components.

Troubleshoot using Tanzu Application Platform

This topic tells you how to troubleshoot using Tanzu Application Platform (commonly known as TAP).

Missing build logs after creating a workload

You create a workload, but no logs appear when you check for logs by running the following command:

```
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 each of the following commands to receive the relevant error message:

```
kubectl get clusterbuilder.kpack.io -o yaml
```

```
kubectl get image.kpack.io <workload-name> -o yaml
```

```
kubectl get build.kpack.io -o yaml
```

“Workload already exists” error after updating the workload

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

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

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 in *Using RBAC Authorization* in the 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 may not apply

If you upgrade from Tanzu Application Platform v0.4, the debug convention may 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

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

After deploying Tanzu Application Platform workloads, Tanzu Application Platform GUI 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

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

You receive an "Increase your cluster's resources" error.

Explanation

Node pressure may be caused by an insufficient number of nodes or a lack of resources on nodes necessary to deploy the workloads that you have.

Solution

Follow instructions from your cloud provider to scale out or scale up your cluster.

MutatingWebhookConfiguration prevents pod admission

Admission of all pods is prevented when the `image-policy-controller-manager` deployment pods do not start before the `MutatingWebhookConfiguration` is applied to the cluster.

Explanation

Pods can be prevented from starting if nodes in a cluster are scaled to zero and the webhook is forced to restart at the same time as other system components. A deadlock can occur when some components expect the webhook to verify their image signatures and the webhook is not yet running.

A known rare condition during Tanzu Application Platform profiles installation can cause this. If so, you may see a message similar to one of the following in component statuses:

```
Events:
  Type     Reason           Age                  From                   Message
  ----     ------           ----                 ----                   -------
  Warning  FailedCreate     4m28s                replicaset-controller  Error creati
ng: Internal error occurred: failed calling webhook "image-policy-webhook.signing.app
s.tanzu.vmware.com": Post "https://image-policy-webhook-service.image-policy-system.sv
c:443/signing-policy-check?timeout=10s": no endpoints available for service "image-pol
icy-webhook-service"
```

```
Events:
  Type     Reason           Age                  From                   Message
  ----     ------           ----                 ----                   -------
  Warning FailedCreate 10m replicaset-controller Error creating: Internal error occurr
ed: failed calling webhook "image-policy-webhook.signing.apps.tanzu.vmware.com": Post
"https://image-policy-webhook-service.image-policy-system.svc:443/signing-policy-chec
k?timeout=10s": service "image-policy-webhook-service" not found
```

Solution

Delete the MutatingWebhookConfiguration resource to resolve the deadlock and enable the system to restart. After the system is stable, restore the MutatingWebhookConfiguration resource to re-enable image signing enforcement.

Important: These steps temporarily deactivate signature verification in your cluster.

1. Back up `MutatingWebhookConfiguration` to a file by running:

   ```
   kubectl get MutatingWebhookConfiguration image-policy-mutating-webhook-configur
   ation -o yaml > image-policy-mutating-webhook-configuration.yaml
   ```

2. Delete `MutatingWebhookConfiguration` by running:

   ```
   kubectl delete MutatingWebhookConfiguration image-policy-mutating-webhook-confi
   guration
   ```

3. Wait until all components are up and running in your cluster, including the `image-policy-controller-manager pods` (namespace `image-policy-system`).

4. Re-apply `MutatingWebhookConfiguration` by running:

   ```
   kubectl apply -f image-policy-mutating-webhook-configuration.yaml
   ```

Priority class of webhook's pods preempts less privileged pods

When viewing the output of `kubectl get events`, you see events similar to the following:

```
$ kubectl get events
LAST SEEN   TYPE      REASON              OBJECT                 MESSAGE
28s         Normal    Preempted           pod/testpod            Preempted by image-polic
y-system/image-policy-controller-manager-59dc669d99-frwcp on node test-node
```

Explanation

The Supply Chain Security Tools - Sign component uses a privileged `PriorityClass` to start its pods to prevent node pressure from preempting its pods. This can cause less privileged components to have their pods preempted or evicted instead.

Solution

- **Solution 1: Reduce the number of pods deployed by the Sign component:** If your deployment of the Sign component runs more pods than necessary, scale down the deployment down as follows:
  1. Create a values file named `scst-sign-values.yaml` with the following contents:

     ```
     ---
     replicas: N
     ```

     Where `N` is an integer indicating the lowest number of pods you necessary for your current cluster configuration.
  2. Apply the new configuration by running:

     ```
     tanzu package installed update image-policy-webhook \
       --package-name image-policy-webhook.signing.apps.tanzu.vmware.com \
       --version 1.0.0-beta.3 \
       --namespace tap-install \
       --values-file scst-sign-values.yaml
     ```
  3. Wait a few minutes for your configuration to take effect in the cluster.
- **Solution 2: Increase your cluster's resources:** Node pressure may be caused by an insufficient number of nodes or a lack of resources on nodes necessary to deploy the workloads that you have. Follow instructions from your cloud provider to scale out or scale up your cluster.

CrashLoopBackOff from password authentication fails

Supply Chain Security Tools - 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 been changed between deployments. This is not supported.

Solution

Redeploy the app either with the original database password or follow these steps below 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

Supply Chain Security Tools - 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 been changed between deployments. This is not supported.

Solution

Redeploy the app either with the original database password or follow these steps below 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

When Supply Chain Security Tools - 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 postgres retains old data, even though the retention policy is set to `DELETE`.

Solution

Redeploy the app either with the original database password or follow these steps below 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

After Supply Chain Security Tools - 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 Supply Chain Security Tools - 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 Supply Chain Security Tools - Store. For example:

   ```
   # This is the storageclass that Kind uses
   kubectl apply -f https://raw.githubusercontent.com/rancher/local-path-provisioner/master/deploy/local-path-storage.yaml

   # set the storage class as default
   kubectl patch storageclass local-path -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
   ```

Failure to connect to AWS EKS clusters

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 greater and update the kubeconfig entry for your EKS cluster(s):

1. Update `aws-cli` to the latest version. See AWS documentation for more information.
2. Update the kubeconfig entry for your EKS cluster(s):

   ```
   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.

Troubleshooting Tanzu Application Platform components

For component-level troubleshooting, see these topics:

- Troubleshoot Tanzu Application Platform GUI
- Troubleshoot Learning Center
- Troubleshoot Service Bindings
- Troubleshoot Source Controller
- Troubleshoot Spring Boot Conventions
- Troubleshoot Supply Chain Security Tools - Scan
- Troubleshoot Supply Chain Security Tools - Store
- Troubleshoot Application Live View for VMware Tanzu
- Troubleshoot Cloud Native Runtimes for Tanzu
- Tanzu Build Service FAQ
- Troubleshoot Tanzu Build Service
- Troubleshoot Services Toolkit

Uninstall Tanzu Application Platform

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

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-install
     | Uninstalling package 'cloud-native-runtimes' from namespace 'tap-install'
     / Getting package install for 'cloud-native-runtimes'
     \ Deleting package install 'cloud-native-runtimes' from namespace 'tap-install'
     \ 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-rolebinding'
     | 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
```

Remove Cluster Essentials

To completely remove Cluster Essentials, see Cluster Essentials documentation.

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.

Tanzu CLI Overview

This topic tells you about the Tanzu command-line interface (CLI).

Tanzu CLI

The Tanzu CLI is a command-line interface that connects you to Tanzu. For example, you can use the Tanzu CLI to:

- Configure the Tanzu CLI itself
- Install and manage packages
- Create and manage application workloads

Tanzu CLI Architecture

The Tanzu CLI has a pluggable architecture. Plug-ins contain CLI commands. Here are the CLI plug-ins that can be installed with Tanzu Application Platform.

- Accelerator: manage accelerator's in a Kubernetes cluster
- Apps: manage application workloads running on workload clusters
- Insight: post and query image, package, source, and vulnerability data
- Package: package management
- Secret: secret management
- Services: discover service types, service instances, and manage resource claims

You can also develop your own plug-ins to add custom commands to the Tanzu CLI. For more information about plug-ins, see the Sync New Plugins, Install New Plugins, Install Local Plugins following sections.

Tanzu CLI Installation

You install and initialize the Tanzu CLI on a computer. The computer can be a laptop, host, or server.

To install the CLI :

- To use the Tanzu CLI with **Tanzu Application Platform,** see Installing the Tanzu CLI.
- To use the Tanzu CLI with **Tanzu Kubernetes Grid,** see Install the Tanzu CLI and Other Tools.

Tanzu CLI Command Groups

Tanzu CLI commands are organized into command groups. To view a list of available command groups, run `tanzu`. The list of command groups that you see depends on which CLI plug-ins are installed on your local machine.

Install New Plugins

To install a Tanzu CLI plug-in that was not automatically downloaded when running `tanzu login` or `tanzu plugin sync`, install it manually by following these steps.

1. In a terminal, run:

   ```
   tanzu plugin install PLUGIN-NAME
   ```

2. Verify that you installed the plugin successfully by running:

   ```
   tanzu plugin list
   NAME                DESCRIPTION
   SCOPE       DISCOVERY  VERSION         STATUS
   login               Login to the platform
   Standalone  default    v0.11.6         not installed
   management-cluster  Kubernetes management-cluster operations
   Standalone  default    v0.11.6         not installed
   package             Tanzu package management
   Standalone  default    v0.11.6         installed
   pinniped-auth       Pinniped authentication operations (usually not directly in
   voked)                                  Standalone  default    v0.11.6
   not installed
   secret              Tanzu secret management
   Standalone  default    v0.11.6         installed
   insight             post & query image, package, source, and vulnerability data
   Standalone             v1.2.1          installed
   test                Test the CLI
   Standalone             v0.22.0         installed
   accelerator         Manage accelerators in a Kubernetes cluster
   Standalone             v1.2.0-build.1  installed
   apps                Applications on Kubernetes
   Standalone             v0.0.0-dev      installed
   builder             Build Tanzu components
   Standalone             v0.22.0         installed
   codegen             Tanzu code generation tool
   Standalone             v0.22.0         installed
   services            Explore Service Instance Classes, discover claimable Servic
   e Instances and manage Resource Claims  Standalone             v0.3.0-rc.2
   installed
   ```

Install Local Plugins

If your network is not connected to the Internet or you want to download and inspect the Tanzu CLI plug-in binaries before installing, follow these steps:

1. Download the plug-in `tar.gz` from the release artifacts for your distribution.
2. Extract the `tar.gz` to a location on your local machine using the extraction tool of your choice. For example, the `tar -xvf` command.
3. From that location, run:

   ```
   tanzu plugin install all --local /PATH/TO/FILE/
   ```

4. Verify that you installed the plug-ins successfully by running:

```
tanzu plugin list
NAME                DESCRIPTION
SCOPE       DISCOVERY  VERSION         STATUS
login               Login to the platform
Standalone  default    v0.11.6         not installed
package             Tanzu package management
Standalone  default    v0.11.6         installed
secret              Tanzu secret management
Standalone  default    v0.11.6         installed
insight             post & query image, package, source, and vulnerability data
Standalone             v1.2.2          installed
accelerator         Manage accelerators in a Kubernetes cluster
Standalone             v1.2.0          installed
apps                Applications on Kubernetes
Standalone             v0.7.0          installed
services            Explore Service Instance Classes, discover claimable Servic
e Instances and manage Resource Claims  Standalone             v0.3.0
installed
```

Tanzu CLI Overview

This topic tells you about the Tanzu command-line interface (CLI).

Tanzu CLI

The Tanzu CLI is a command-line interface that connects you to Tanzu. For example, you can use the Tanzu CLI to:

- Configure the Tanzu CLI itself
- Install and manage packages
- Create and manage application workloads

Tanzu CLI Architecture

The Tanzu CLI has a pluggable architecture. Plug-ins contain CLI commands. Here are the CLI plug-ins that can be installed with Tanzu Application Platform.

- Accelerator: manage accelerator's in a Kubernetes cluster
- Apps: manage application workloads running on workload clusters
- Insight: post and query image, package, source, and vulnerability data
- Package: package management
- Secret: secret management
- Services: discover service types, service instances, and manage resource claims

You can also develop your own plug-ins to add custom commands to the Tanzu CLI. For more information about plug-ins, see the Sync New Plugins, Install New Plugins, Install Local Plugins following sections.

Tanzu CLI Installation

You install and initialize the Tanzu CLI on a computer. The computer can be a laptop, host, or server.

To install the CLI :

- To use the Tanzu CLI with **Tanzu Application Platform,** see Installing the Tanzu CLI.
- To use the Tanzu CLI with **Tanzu Kubernetes Grid,** see Install the Tanzu CLI and Other Tools.

Tanzu CLI Command Groups

Tanzu CLI commands are organized into command groups. To view a list of available command groups, run `tanzu`. The list of command groups that you see depends on which CLI plug-ins are installed on your local machine.

Install New Plugins

To install a Tanzu CLI plug-in that was not automatically downloaded when running `tanzu login` or `tanzu plugin sync`, install it manually by following these steps.

1. In a terminal, run:

```
tanzu plugin install PLUGIN-NAME
```

2. Verify that you installed the plugin successfully by running:

```
tanzu plugin list
NAME                DESCRIPTION
SCOPE       DISCOVERY  VERSION         STATUS
login               Login to the platform
Standalone  default    v0.11.6         not installed
management-cluster  Kubernetes management-cluster operations
Standalone  default    v0.11.6         not installed
package             Tanzu package management
Standalone  default    v0.11.6         installed
pinniped-auth       Pinniped authentication operations (usually not directly in
voked)                                  Standalone  default    v0.11.6
not installed
secret              Tanzu secret management
Standalone  default    v0.11.6         installed
insight             post & query image, package, source, and vulnerability data
Standalone             v1.2.1          installed
test                Test the CLI
Standalone             v0.22.0         installed
accelerator         Manage accelerators in a Kubernetes cluster
Standalone             v1.2.0-build.1  installed
apps                Applications on Kubernetes
Standalone             v0.0.0-dev      installed
builder             Build Tanzu components
Standalone             v0.22.0         installed
codegen             Tanzu code generation tool
Standalone             v0.22.0         installed
services            Explore Service Instance Classes, discover claimable Servic
e Instances and manage Resource Claims  Standalone             v0.3.0-rc.2
installed
```

Install Local Plugins

If your network is not connected to the Internet or you want to download and inspect the Tanzu CLI plug-in binaries before installing, follow these steps:

1. Download the plug-in `tar.gz` from the release artifacts for your distribution.
2. Extract the `tar.gz` to a location on your local machine using the extraction tool of your choice. For example, the `tar -xvf` command.
3. From that location, run:

```
tanzu plugin install all --local /PATH/TO/FILE/
```

4. Verify that you installed the plug-ins successfully by running:

```
tanzu plugin list
NAME                DESCRIPTION
SCOPE       DISCOVERY  VERSION         STATUS
login               Login to the platform
```

```
Standalone  default    v0.11.6         not installed
package             Tanzu package management
Standalone  default    v0.11.6         installed
secret              Tanzu secret management
Standalone  default    v0.11.6         installed
insight             post & query image, package, source, and vulnerability data
Standalone             v1.2.2          installed
accelerator         Manage accelerators in a Kubernetes cluster
Standalone             v1.2.0          installed
apps                Applications on Kubernetes
Standalone             v0.7.0          installed
services            Explore Service Instance Classes, discover claimable Servic
e Instances and manage Resource Claims  Standalone             v0.3.0
installed
```

Tanzu CLI plug-ins

The following plug-ins are available in Tanzu Application Platform:

- accelerator - The Application Accelerator Tanzu CLI plug-in includes commands for developers and operators to create and use accelerators.
- apps - This Tanzu 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.
- insight - The Tanzu Insight CLI plug-in enables querying vulnerability, image, and package data.

Tanzu CLI plug-ins

The following plug-ins are available in Tanzu Application Platform:

- accelerator - The Application Accelerator Tanzu CLI plug-in includes commands for developers and operators to create and use accelerators.
- apps - This Tanzu 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.
- insight - The Tanzu Insight CLI plug-in enables querying vulnerability, image, and package data.

Apps CLI plug-in overview

This topic gives you an overview of the Apps CLI plug-in. Use the Apps CLI plugin 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 developers to quickly build and test applications regardless of their familiarity with Kubernetes. Developers can turn source code into a workload that runs in a container with a URL.

A workload enables developers to choose application specifications, such as repository location, environment variables, service binding, and more. For more information on workload creation and management, see Command Reference.

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 app.

Command reference

For information about available commands, see Command Reference.

Usage and examples

For information about how to use the Apps CLI plug-in, see Usage and Examples.

Apps CLI plug-in overview

This topic gives you an overview of the Apps CLI plug-in. Use the Apps CLI plugin 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 developers to quickly build and test applications regardless of their familiarity with Kubernetes. Developers can turn source code into a workload that runs in a container with a URL.

A workload enables developers to choose application specifications, such as repository location, environment variables, service binding, and more. For more information on workload creation and management, see Command Reference.

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 app.

Command reference

For information about available commands, see Command Reference.

Usage and examples

For information about how to use the Apps CLI plug-in, see Usage and Examples.

Install Apps CLI plug-in

This topic tells you how to install the Apps CLI plug-in on Tanzu Application Platform (commonly known as TAP).

> **Note**
>
> Follow the steps in this topic if you do not want to use a profile to install Apps CLI plug-in. For more information about profiles, see About Tanzu Application Platform components and profiles.

Prerequisites

Before you install the Apps CLI plug-in:

- Follow the instructions to Install or update the Tanzu CLI and plug-ins.

Install

From VMware Tanzu Network

To install the Apps CLI plug-in:

1. From the `$HOME/tanzu` directory, run:

```
tanzu plugin install --local ./cli apps
```

2. To verify that the CLI is installed correctly, run:

   ```
   tanzu apps version
   ```

   A version is displayed in the output.

From Release

Download the latest release from the apps-cli-plugin release page. Each of these releases has the *Assets* section where the packages for each *system-architecture* are placed.

To install the Apps CLI plug-in:

Download binary executable file `tanzu-apps-plugin-{OS_ARCH}-{version}.tar.gz` For example, run these commands on macOS with plugin version v0.11.0

```
tar -xvf tanzu-apps-plugin-darwin-amd64-v0.11.0.tar.gz
tanzu plugin install apps --local ./tanzu-apps-plugin-darwin-amd64-v0.11.0 --version v0.11.0
```

Create a workload

This topic tells you how to create a workload from example source code with Tanzu Application Platform (commonly known as TAP).

Prerequisites

The following prerequisites are required to use workloads with Tanzu Application Platform:

- Install kubectl.
- Install Tanzu Application Platform components on a Kubernetes cluster. See Installing Tanzu Application Platform.
- Set your kubeconfig context to the prepared cluster `kubectl config use-context CONTEXT_NAME`.
- Install Tanzu CLI. See Install or update the Tanzu CLI and plug-ins.
- Install the apps plug-in. See the Install Apps plug-in.
- Set up developer namespaces to use your installed packages.

Get started with an example workload

Create a workload from GitHub repository

Create a workload from an existing Git repository by setting the flags `--git-repo`, `--git-branch`, `--git-tag`, and `--git-commit`. 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 create pet-clinic --git-repo https://github.com/sample-accelerators/spring-petclinic --git-tag tap-1.1 --type web
```

Respond `Y` to prompts to complete process.

Where:

- `pet-clinic` is the name of the workload.
- `--git-repo` is the location of the code to build the workload from.
- `--git-branch` (optional) specifies which branch in the repository to pull the code from.

- `--type` used to distinguish the workload type.

You can find the options available for specifying the workload in the command reference for `workload create`, or you can run `tanzu apps workload create --help`.

Create a workload from local source code

Tanzu Application Platform supports creating a workload from an existing local project by setting the flags `--local-path` and `--source-image`, this allows the supply chain to generate an image (carvel-imgpkg) and push it to the given registry to be used in the workload.

- To create a named workload and specify where the local source code is, run:

  ```
  tanzu apps workload create pet-clinic --local-path /path/to/my/project --source
  -image springio/petclinic
  ```

  Respond `Y` to the prompt to publish the local source code and update the image.

  Where:

  - `pet-clinic` is the name of the workload.
  - `--local-path` points to the directory where the source code is located.
  - `--source-image` is the registry path where the local source code is uploaded as an image.

  Exclude Files When working with local source code, you can exclude files from the source code to be uploaded within the image by creating a file `.tanzuignore` at the root of the source code. You can find the options available to specify the workload in the command reference for `workload create`, or run `tanzu apps workload create --help`.

  The file must contain a list of file paths to exclude from the image including the file itself and the directories must not end with the system path separator (`/` or `\`).

  If the file contains files or directories that are not in the source code, they are ignored.

  If a line in the file starts with a `#` hashtag , the line is ignored.

  Example

  ```
  # This is a comment
  this/is/a/folder/to/exclude

  this-is-a-file.ext
  ```

Create workload from an existing image

Creating a workload from an existing image by setting the flag `--image`. This allows the supply chain to get the given image from the registry to deploy the application.

An example on how to create a workload from image is as follows:

```
tanzu apps workload create petclinic-image --image springcommunity/spring-framework-pe
tclinic
```

Respond `Y` to prompts to complete process.

Where:

- `petclinic-image` is the name of the workload.
- `--image` is an existing image, pulled from a registry, that contains the source that the workload is going to use to create the application.

Create a workload from Maven repository artifact

Tanzu Application Platform supports creating a workload from a Maven repository artifact (Source-Controller) by setting some specific properties as yaml parameters in the workload when using the supply chain.

The maven repository URL is being set when the supply chain is created.

- 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:

```
# YAML
tanzu apps workload create petclinic-image --param-yaml maven=$"artifactId:hello-world
\ntype: jar\nversion: 0.0.1\ngroupId: carto.run"

# JSON
tanzu apps workload create petclinic-image --param-yaml maven="{"artifactId":"hello-wo
rld", "type": "jar", "version": "0.0.1", "groupId": "carto.run"}"
```

Bind a service to a workload

Tanzu Application Platform supports creating a workload with binding to multiple services (Service Binding). The cluster supply chain is in charge of provisioning those services.

The intent of these bindings is to provide information from a service resource to an application.

- To bind a database service to a workload, run:

```
tanzu apps workload update pet-clinic --service-ref "database=services.tanzu.vm
ware.com/v1alpha1:MySQL:my-prod-db"
```

  Where:

  - `pet-clinic` is the name of the workload to be updated.
  - `--service-ref` references the service using the format {service-ref-name}={apiVersion}:{kind}:{service-binding-name}.

Check services consumption documentation to get more information on how to bind a service to a workload.

Next steps

You can check workload details and status, add environment variables, export definitions, bind services, and use flags with these commands. For more information,Command reference.

1. To check workload status and details, use `workload get` command and to get workload logs, use `workload tail` command. For more information about these, refer to debug workload section.
2. To add environment variables, run:

```
tanzu apps workload update pet-clinic --env foo=bar
```

3. To export the workload definition into Git, or to migrate to another environment, run:

   ```
   tanzu apps workload get pet-clinic --export
   ```

4. To bind a service to a workload, see the –service-ref flag.
5. To see flags available for the workload commands, run:

   ```
   tanzu apps workload -h
   tanzu apps workload get -h
   tanzu apps workload create -h
   ```

Debug workloads

This topic tells you how to debug a workload in Tanzu Application Platform (commonly known as TAP).

Check build logs

After the workload is created, you can tail the workload to view the build and runtime logs. For more information about `tail` command, see workload tail.

- Check logs by running:

  ```
  tanzu apps workload tail pet-clinic --since 10m --timestamp
  ```

  Where:

  - `pet-clinic` is the name you gave the workload.
  - `--since` (optional) the amount of time to go back to begin streaming logs. The default is 1 second.
  - `--timestamp` (optional) prints the timestamp with each log line.

Get the workload status and details

After the workload build process is complete, create a Knative service to run the workload. You can view workload details at anytime in the process. Some details, such as the workload URL, are only available after the workload is running.

1. To check the workload details, run:

   ```
   tanzu apps workload get pet-clinic
   ```

   Where:

   - `pet-clinic` is the name of the workload you want details about.

2. You can now see the running workload. When the workload is created, `tanzu apps workload get` includes the URL for the running workload. Some terminals allow you to `ctrl`+click the URL to view it. You can also copy and paste the URL into your web browser to see the workload.

Common workload errors

A workload can either be ready, on error or with an unknown status.

There are known errors that will make the workload enter in an error or unknown status. The most common are:

- *Local Path Development Error Cases*
  - *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 plugin 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`. 400 Bad request is sent when either the project does not exists, 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 repo 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 csc list` and `tanzu apps csc get <supply-chain>` commands to see the selector criterias for the supply chains. You can apply the missing labels to a workload by using `tanzu apps workload apply`
    - e.g. `tanzu apps workload apply workload-name --type web`
    - e.g. `tanzu apps workload apply workload-name --label apps.tanzu.vmware.com/workload-type=web`
- *MissingValueAtPath*
  - *Message*: Waiting to read value [.status.artifact.url] from resource gitrepository.source.toolkit.fluxcd.io in namespace [ns]
  - *Possible Causes*:
    - The git `url/tag/branch/commit` params passed in the workload are not valid.
      - *Resolution*: Fix the invalid git param by using *tanzu apps workload apply*
    - The git repository is not accessible from the cluster
      - *Resolution*: Configure your cluster networking or your Git repo networking so that they can communicate with each other.
    - The namespace is missing the Git secret for communicating with the private repository
      - *Resolution*: Checkout this page on how to setup Git Authentication for Private repositories [Link to 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 permissions to create objects that are stamped out by the supply chain.

        - *Resolution*: This can be fixed by setting up the Set up developer namespaces to use your installed packages with the required service account and permissions.

Command reference

- Tanzu apps
    - Workload
        - Workload apply
        - Workload create
        - Workload update
            - Workload create/update/apply flags and usage examples
        - Workload get
            - Workload get flags and usage examples
        - Workload delete
            - Workload delete flags and usage examples
        - Workloads list
            - Workload list flags and usage examples
        - Workload tail
            - Workload tail flags and usage examples
- Cluster supply chain
    - Get cluster supply chain cluster supply chain get flags and usage examples
    - List cluster supply chain

Command reference

- Tanzu apps
    - Workload
        - Workload apply
        - Workload create
        - Workload update
            - Workload create/update/apply flags and usage examples
        - Workload get
            - Workload get flags and usage examples
        - Workload delete
            - Workload delete flags and usage examples
        - Workloads list
            - Workload list flags and usage examples
        - Workload tail
            - Workload tail flags and usage examples
- Cluster supply chain
    - Get cluster supply chain cluster supply chain get flags and usage examples
    - List cluster supply chain

Commands Details

The proceeding topics shows detailed examples of how to use flags on the Tanzu CLI.

Tanzu Apps Cluster Supply Chain Get

Use the `tanzu apps cluster-supply-chain get` command to get a detailed information about the cluster supply chain.

Default view

The default view of `get` command shows the status of the supply chain, and the selectors that a workload must match so it is taken by that workload

For example:

```
$ tanzu apps cluster-supply-chain get source-to-url
---
# source-to-url: Ready
---
Supply Chain Selectors
TYPE     KEY                                   OPERATOR   VALUE
labels   apps.tanzu.vmware.com/workload-type              web
```

tanzu apps workload apply

Use the `tanzu apps workload apply` to create and update workloads that are deployed in a cluster through a supply chain.

Default view

In the output of the `tanzu apps workload apply` command, the specification for the workload is shown in YAML format.

▶ Example

In the first section, the definition of workload is displayed. It's followed by a dialog box asking whether to create or update the workload. In the last section, if a workload is created or updated, some hints are displayed about the next steps. Each flag used in this example is explained in detail in the following section.

Workload Apply flags

`--annotation`

Set the annotations to be applied to the workload, to specify more than one annotation set the flag multiple times. These annotations are passed as parameters to be processed in the supply chain.

▶ Example

To delete an annotation, use `-` after its name.

▶ Example

`--app`

This is the application the workload is part of. This is part of the workload metadata section.

▶ Example

`--build-env`

Sets environment variables to use in the build phase by the build resources in the supply chain where some build-specific behavior can be set or changed.

▶ Example

To delete a build environment variable, use `-` after its name.

▶ Example

`--debug`

Sets the parameter variable debug to true in workload.

▶ Example

`--dry-run`

This flag prepares all the steps to submit the workload to the cluster but stops before sending it, and shows an output of the final structure of the workload.

▶ Example

`--env`

Set the environment variables to the workload so the supply chain resources can use it to properly deploy the workload application

▶ Example

`--file`, `-f`

Set a workload specification file to create the workload from. Any other workload specification passed by flags to the command are set or override whatever is in the file. Another way to use this flag is using `-` in the command, to receive workload definition through standard input. For more information, see Working with Yaml Files.

▶ Example

`--git-repo`

Source Git repository for the workload. Specify `--git-tag`, `--git-commit` or `--git-branch`.

`--git-branch`

Branch in a Git repository from where the workload is created. You can specify this as part of a commit or tag.

▶ Example

`--git-tag`

Tag in the workload source Git repository. Specify `--git-commit` or `--git-branch`.

`--git-commit`

Commit in the workload source Git repository. Specify `--git-branch` or `git-tag`.

▶ Example

`--image`

Sets the OSI image to be used as the workload application source instead of a Git repository

▶ Example

`--label`

Sets the label to be applied to the workload. To specify more than one label, set the flag multiple times.

▶ Example

To unset labels, use `-` after their name.

▶ Example

`--limit-cpu`

Refers to the maximum CPU the workload pods are allowed to use.

▶ Example

`--limit-memory`

Refers to the maximum memory the workload pods are allowed to use.

▶ Example

`--live-update`

Enable this to deploy a workload once, save changes to the code, and see those changes reflected within seconds in the workload running on the cluster.

▶ Example

`--local-path`

Sets the path to a source in the local machine from where the workload creates an image to use as an application source. The local path can be a directory, a JAR, a ZIP, or a WAR file. Java/Spring Boot compiled binaries are also supported. This flag must be used with `--source-image` flag.

Note If Java/Spring compiled binary is passed instead of source code, the command takes less time to apply the workload since the buildpack skips the compiling steps and starts uploading the image.

When working with local source code, you can exclude files from the source code to be uploaded within the image by creating a file `.tanzuignore` at the root of the source code. The `.tanzuignore` file contains a list of file paths to exclude from the image including the file itself and the directories must not end with the system path separator (`/` or `\`). Directories that are not in the source code, and lines starting with the hashtag `#` character are ignored.

`--source-image`, `-s`

Registry path where the local source code is uploaded as an image.

▶ Example

`--namespace`, `-n`

Specifies the namespace in which the workload is to be created or updated.

▶ Example

`--param`

Additional parameters to be sent to the supply chain, the value is sent as a string, for complex yaml/json objects use `--param-yaml`

▶ Example

To unset parameters, use `-` after their name.

▶ Example

`--param-yaml`

Additional parameters to be sent to the supply chain, the value is sent as a complex object.

▶ Example

To unset parameters, use `-` after their name.

▶ Example

`--request-cpu`

Refers to the minimum CPU the workload pods are requesting to use.

▶ Example

`--request-memory`

Refers to the minimum memory the workload pods are requesting to use.

▶ Example

`--service-account`

Refers to the service account associated with the workload. A service account provides an identity for a workload object.

▶ Example

To unset a service account, pass an empty string.

▶ Example

`--service-ref`

Binds a service to a workload to provide the information from a service resource to an application.

Note For more information see Tanzu Application Platform documentation.

▶ Example

To delete service binding, use the service name followed by `-`.

▶ Example

`--sub-path`

Defines which path is used as root to create and update workloads.

▶ Example

`--tail`

Prints the logs of the workload creation in every step.

▶ Example

`--tail-timestamp`

Prints the logs of the workload creation in every step adding the time in which the log is occurring.

▶ Example

`--type`

Sets the type of workload by adding the label `apps.tanzu.vmware.com/workload-type`, which is very common to be used as a matcher by supply chains.

▶ Example

`--wait`

Holds until workload is ready.

▶ Example

`--wait-timeout`

Sets a timeout to wait for the workload to be ready.

▶ Example

`--yes`, `-y`

Assume `yes` on all the survey prompts

▶ Example

tanzu apps workload delete

This command deletes workloads in a cluster. Deleting a workload does not delete the images published in the registry.

Default view

When attempting to delete a workload without the `--yes` flag, a message asking if it is really to be deleted appears in the terminal and, if the user responses `"Y"`, then the workload starts a deletion process inside the cluster.

```
tanzu apps workload delete spring-pet-clinic
? Really delete the workload "spring-pet-clinic"? Yes
Deleted workload "spring-pet-clinic"
```

```
tanzu apps workload delete spring-pet-clinic --yes
Deleted workload "spring-pet-clinic"
```

Workload Delete flags

`--all`

Deletes all workloads in a namespace.

```
tanzu apps workload delete --all
? Really delete all workloads in the namespace "default"? (y/N) Y
Deleted workloads in namespace "default"
```

```
tanzu apps workload delete --all -n my-namespace
? Really delete all workloads in the namespace "my-namespace"? Yes
Deleted workloads in namespace "my-namespace"
```

`--file`, `-f`

Path to a file that contains the specification of the workload to be deleted.

```
tanzu apps workload delete -f path/to/file/spring-petclinic.yaml
? Really delete the workload "spring-petclinic"? Yes
Deleted workload "spring-petclinic"
```

`--namespace`, `-n`

Specifies the namespace in which the workload is to be deleted.

```
tanzu apps workload delete spring-petclinic -n spring-petclinic-ns
? Really delete the workload "spring-petclinic"? Yes
Deleted workload "spring-petclinic"
```

`wait`

Waits until workload is deleted.

```
tanzu apps workload delete -f path/to/file/spring-petclinic.yaml --wait
? Really delete the workload "spring-petclinic"? Yes
Deleted workload "spring-petclinic"
Waiting for workload "spring-petclinic" to be deleted...
Workload "spring-petclinic" was deleted
```

`--wait-timeout`

Sets a timeout to wait for workload to be deleted.

```
tanzu apps workload delete -f path/to/file/spring-petclinic.yaml --wait --wait-timeout
1m
? Really delete the workload "spring-petclinic"? Yes
Deleted workload "spring-petclinic"
Waiting for workload "spring-petclinic" to be deleted...
Workload "spring-petclinic" was deleted
```

```
tanzu apps workload delete spring-petclinic -n spring-petclinic-ns --wait --wait-timeo
ut 1m
? Really delete the workload "spring-petclinic"? Yes
Deleted workload "spring-petclinic"
Waiting for workload "spring-petclinic" to be deleted...
Error: timeout after 1m waiting for "spring-petclinic" to be deleted
To view status run: tanzu apps workload get spring-petclinic --namespace spring-petcli
nic-ns
Error: exit status 1

✖  exit status 1
```

`--yes`, `-f`

Assume yes on all the survey prompts.

```
tanzu apps workload delete spring-petclinic --yes
Deleted workload "spring-petclinic"
```

tanzu apps workload get

Use the `tanzu apps workload get` command to retrieve information and status about a workload.

Some available details are:

- The status of the workload.
- The source of the workload application.
- The supply chain which took care of the workload.
- The supply chain resources which interact with the workload.
- If there is any issue while deploying the workload and finally which *pods* the workload generates and the knative services related to the workload.
- if the supply chain is using knative.

Default view

There are multiple sections in the workload get command output. Following data is displayed:

- Name of the workload and its status.
- Display source information of workload.
- If the workload was matched with a supply chain, the information of its name and status is displayed.
- Information and status of the individual steps that's defined in the supply chain for workload.
- Any issue with the workload, the name and the corresponding message.
- Workload related resource information and status like services claims, related pods, knative services.

At the very end of the command output, a hint to follow up commands is also displayed.

```
tanzu apps workload get rmq-sample-app
---
# rmq-sample-app: Ready
---
Source
```

```
type:     git
url:      https://github.com/jhvhs/rabbitmq-sample
branch:   main

Supply Chain
name:          source-to-url
last update:   94s
ready:         True

RESOURCE          READY   TIME
source-provider   True    3m51s
deliverable       True    3m55s
image-builder     True    101s
config-provider   True    94s
app-config        True    94s
config-writer     True    94s

Issues
No issues reported.

Services
CLAIM   NAME                         KIND              API VERSION
rmq     example-rabbitmq-cluster-1   RabbitmqCluster   rabbitmq.com/v1beta1

Pods
NAME                                              STATUS      RESTARTS   AGE
rmq-sample-app-00001-deployment-78fc86b47c-r5jws   Running     0          45s
rmq-sample-app-build-1-build-pod                  Succeeded   0          3m50s
rmq-sample-app-config-writer-pbshl-pod            Succeeded   0          94s

Knative Services
NAME             READY   URL
rmq-sample-app   Ready   http://rmq-sample-app.default.example.com

To see logs: "tanzu apps workload tail rmq-sample-app"
```

--export

Exports the submitted workload in yaml format. This flag can also be used with --output flag. With export, the output is shortened because some fields are removed.

```
tanzu apps workload get pet-clinic --export

---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
labels:
    apps.tanzu.vmware.com/workload-type: web
    autoscaling.knative.dev/min-scale: "1"
name: pet-clinic
namespace: default
spec:
source:
    git:
    ref:
        tag: tap-1.2
    url: https://github.com/sample-accelerators/spring-petclinic
```

--output/-o

Configures how the workload is being shown. This supports the values yaml, yml and json, where yaml and yml are equal. It shows the actual workload in the cluster.

- yaml/yml

  ```
  tanzu apps workload get pet-clinic -o yaml
  ---
  apiVersion: carto.run/v1alpha1
  kind: Workload
  ```

```
metadata:
creationTimestamp: "2022-06-03T18:10:59Z"
generation: 1
labels:
    apps.tanzu.vmware.com/workload-type: web
    autoscaling.knative.dev/min-scale: "1"
...
spec:
source:
    git:
    ref:
        tag: tap-1.1
    url: https://github.com/sample-accelerators/spring-petclinic
status:
    conditions:
    - lastTransitionTime: "2022-06-03T18:10:59Z"
        message: ""
        reason: Ready
        status: "True"
        type: SupplyChainReady
    - lastTransitionTime: "2022-06-03T18:14:18Z"
        message: ""
        reason: ResourceSubmissionComplete
        status: "True"
        type: ResourcesSubmitted
    - lastTransitionTime: "2022-06-03T18:14:18Z"
        message: ""
        reason: Ready
        status: "True"
        type: Ready
    observedGeneration: 1
    resources:
    ...
    supplyChainRef:
        kind: ClusterSupplyChain
        name: source-to-url
        ...
```

- json

```
tanzu apps workload get pet-clinic -o json
{
    "kind": "Workload",
    "apiVersion": "carto.run/v1alpha1",
    "metadata": {
        "name": "pet-clinic",
        "namespace": "default",
        "uid": "937679ca-9c72-4e23-bfef-6334e6c003a7",
        "resourceVersion": "111637840",
        "generation": 1,
        "creationTimestamp": "2022-06-03T18:10:59Z",
        "labels": {
            "apps.tanzu.vmware.com/workload-type": "web",
            "autoscaling.knative.dev/min-scale": "1"
        },
...
}
"spec": {
        "source": {
            "git": {
                "url": "https://github.com/sample-accelerators/spring-petclini
c",
                "ref": {
                    "tag": "tap-1.1"
                }
            }
        }
    },
    "status": {
        "observedGeneration": 1,
        "conditions": [
            {
                "type": "SupplyChainReady",
                "status": "True",
                "lastTransitionTime": "2022-06-03T18:10:59Z",
                "reason": "Ready",
                "message": ""
            },
            {
                "type": "ResourcesSubmitted",
                "status": "True",
                "lastTransitionTime": "2022-06-03T18:14:18Z",
                "reason": "ResourceSubmissionComplete",
                "message": ""
            },
            {
                "type": "Ready",
                "status": "True",
                "lastTransitionTime": "2022-06-03T18:14:18Z",
                "reason": "Ready",
                "message": ""
            }
        ],
        "supplyChainRef": {
            "kind": "ClusterSupplyChain",
            "name": "source-to-url"
        },
        "resources": [
            {
                "name": "source-provider",
                "stampedRef": {
                    "kind": "GitRepository",
                    "namespace": "default",
                    "name": "pet-clinic",
                    ...
                }
            }
        ]
        ...
    }
    ...
}
```

`--namespace`/`-n`

Specifies the namespace where the workload is deployed.

```
tanzu apps workload get pet-clinic -n development

---
# pet-clinic: Ready
---
Source
type:   git
url:    https://github.com/sample-accelerators/spring-petclinic
tag:    tap-1.2

Supply Chain
name:          source-to-url
last update:   10d
ready:         True

RESOURCE          READY   TIME
source-provider
deliverable
image-builder
config-provider
app-config
config-writer

Issues
No issues reported.
```

```
Pods
NAME                                           STATUS      RESTARTS   AGE
pet-clinic-00001-deployment-6445565f7b-ts815   Running     0          102s
pet-clinic-build-1-build-pod                   Succeeded   0          102s
pet-clinic-config-writer-8c9zv-pod             Succeeded   0          2m7s
Knative Services
NAME         READY   URL
pet-clinic   Ready   http://pet-clinic.default.apps.34.133.168.14.nip.io

To see logs: "tanzu apps workload tail pet-clinic"
```

tanzu apps workload list

The `tanzu apps workload list` command gets the workloads present in the cluster, either in the current namespace, in another namespace, or in all namespaces.

Default view

The default view for workload list is a table with the workloads present in the cluster in the specified namespace. This table has, in each row, the name of the workload, the application it is related to, its status, and how long it has been in the cluster.

For example, in the default namespace

```
tanzu apps workload list

NAME                APP                 READY                    AGE
nginx4              <empty>             Ready                    7d9h
petclinic2          <empty>             Ready                    29h
rmq-sample-app      <empty>             Ready                    164m
rmq-sample-app4     <empty>             WorkloadLabelsMissing    29d
spring-pet-clinic   <empty>             Unknown                  166m
spring-petclinic2   spring-petclinic    Unknown                  29d
spring-petclinic3   spring-petclinic    Ready                    29d
```

>Workload List flags

`--all-namespaces`, `-A`

Shows workloads in all namespaces in cluster.

```
tanzu apps workload list -A

NAMESPACE   NAME                APP                 READY                           AGE
default     nginx4              <empty>             Ready                           7d9h
default     petclinic2          <empty>             Ready                           30h
default     rmq-sample-app      <empty>             Ready                           179m
default     rmq-sample-app4     <empty>             WorkloadLabelsMissing           29d
default     spring-pet-clinic   <empty>             Unknown                         3h1m
default     spring-petclinic2   spring-petclinic    Unknown                         29d
default     spring-petclinic3   spring-petclinic    Ready                           29d
nginx-ns    nginx2              <empty>             TemplateRejectedByAPIServer     8d
nginx-ns    nginx4              <empty>             TemplateRejectedByAPIServer     8d
```

`--app`

Shows workloads which app is the one specified in the command.

```
tanzu apps workload list --app spring-petclinic

NAME                READY     AGE
spring-petclinic2   Unknown   29d
spring-petclinic3   Ready     29d
```

`--namespace`, `-n`

Lists all the workloads present in the specified namespace.

```
tanzu apps workload list -n my-namespace

NAME     APP       READY                         AGE
app1     <empty>   TemplateRejectedByAPIServer   8d
app2     <empty>   Ready                         8d
app3     <empty>   Unknown                       8d
```

`--output`, `-o`

Lists all workloads in the specified namespace in yaml, yml or json format.

- yaml/yml

```
---
- apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
    creationTimestamp: "2022-05-17T22:06:49Z"
    generation: 1
    labels:
    app.kubernetes.io/part-of: spring-petclinic
    apps.tanzu.vmware.com/workload-type: web
    managedFields:
    ...
    ...
    manager: cartographer
    operation: Update
    time: "2022-05-17T22:06:52Z"
name: spring-petclinic3
namespace: default
resourceVersion: "106252670"
uid: fcca2d4b-c713-43a5-9a53-9f1ebb214726
spec:
    source:
        git:
            ref:
                tag: tap-1.1
            url: https://github.com/sample-accelerators/spring-petclinic
...
...
---
- apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
    creationTimestamp: "2022-05-17T22:06:49Z"
    generation: 1
    labels:
    app.kubernetes.io/part-of: spring-petclinic
    apps.tanzu.vmware.com/workload-type: web
    managedFields:
    ...
    ...
    manager: cartographer
    operation: Update
    time: "2022-05-17T22:06:52Z"
name: spring-petclinic2
namespace: default
resourceVersion: "106252670"
uid: fcca2d4b-c713-43a5-9a53-9f1ebb214726
spec:
    source:
        git:
            ref:
                tag: tap-1.1
            url: https://github.com/sample-accelerators/spring-petclinic
...
...
```

- json

```
[
    {
        "kind": "Workload",
        "apiVersion": "carto.run/v1alpha1",
        "metadata": {
            "name": "spring-petclinic3",
            "namespace": "default",
            "uid": "fcca2d4b-c713-43a5-9a53-9f1ebb214726",
            "resourceVersion": "106252670",
            "generation": 1,
            "creationTimestamp": "2022-05-17T22:06:49Z",
            "labels": {
                "app.kubernetes.io/part-of": "spring-petclinic",
                "apps.tanzu.vmware.com/workload-type": "web"
            },
        ...
        }
    ...
    },
    {
        "kind": "Workload",
        "apiVersion": "carto.run/v1alpha1",
        "metadata": {
            "name": "spring-petclinic2",
            "namespace": "default",
            "uid": "fcca2d4b-c713-43a5-9a53-9f1ebb214726",
            "resourceVersion": "106252670",
            "generation": 1,
            "creationTimestamp": "2022-05-17T22:06:49Z",
            "labels": {
                "app.kubernetes.io/part-of": "spring-petclinic",
                "apps.tanzu.vmware.com/workload-type": "web"
            },
        ...
        }
    ...
    },
...
...
]
```

tanzu apps workload tail

The `tanzu apps workload tail` command checks the runtime logs of a workload.

Default view

Without timestamp set, workload tail will show the stage where it is and the log related.

```
+ spring-pet-clinic-build-1-build-pod › prepare
+ spring-pet-clinic-build-1-build-pod › detect
+ spring-pet-clinic-build-1-build-pod › analyze
+ spring-pet-clinic-build-1-build-pod › build
+ spring-pet-clinic-build-1-build-pod › restore
spring-pet-clinic-build-1-build-pod[detect] ======== Output: tanzu-buildpacks/poetry@
0.1.0 ========
spring-pet-clinic-build-1-build-pod[detect] pyproject.toml must include [tool.poetry.d
ependencies.python], see https://python-poetry.org/docs/pyproject/#dependencies-and-de
v-dependencies
spring-pet-clinic-build-1-build-pod[analyze] Restoring data for sbom from previous ima
ge
spring-pet-clinic-build-1-build-pod[detect] err:  tanzu-buildpacks/poetry@0.1.0 (1)
spring-pet-clinic-build-1-build-pod[detect] ======== Output: tanzu-buildpacks/poetry@
0.1.0 ========
spring-pet-clinic-build-1-build-pod[detect] pyproject.toml must include [tool.poetry.d
ependencies.python], see https://python-poetry.org/docs/pyproject/#dependencies-and-de
v-dependencies
spring-pet-clinic-build-1-build-pod[detect] err:  tanzu-buildpacks/poetry@0.1.0 (1)
spring-pet-clinic-build-1-build-pod[detect] 10 of 38 buildpacks participating
```

```
spring-pet-clinic-build-1-build-pod[detect] paketo-buildpacks/ca-certificates   3.1.0
spring-pet-clinic-build-1-build-pod[detect] paketo-buildpacks/bellsoft-liberica 9.2.0
spring-pet-clinic-build-1-build-pod[detect] paketo-buildpacks/syft              1.10.0
spring-pet-clinic-build-1-build-pod[detect] paketo-buildpacks/gradle            6.4.1
spring-pet-clinic-build-1-build-pod[detect] paketo-buildpacks/maven             6.4.0
spring-pet-clinic-build-1-build-pod[detect] paketo-buildpacks/executable-jar    6.1.0
spring-pet-clinic-build-1-build-pod[detect] paketo-buildpacks/apache-tomcat     7.2.0
spring-pet-clinic-build-1-build-pod[detect] paketo-buildpacks/dist-zip          5.2.0
spring-pet-clinic-build-1-build-pod[detect] paketo-buildpacks/spring-boot       5.8.0
spring-pet-clinic-build-1-build-pod[detect] paketo-buildpacks/image-labels      4.1.0
...
...
...
```

>Workload Tail flags

`--component`

Set the component from which the tail command should stream the logs, the values that the flag can take depend on the final deployed pods label `app.kubernetes.io/component`, for example, `build`, `run` and `config-writer`

```
tanzu apps workload tail pet-clinic --component build

pet-clinic-build-1-build-pod[export] Adding label 'io.buildpacks.project.metadata'
pet-clinic-build-1-build-pod[export] Adding label 'org.opencontainers.image.title'
pet-clinic-build-1-build-pod[export] Adding label 'org.opencontainers.image.version'
pet-clinic-build-1-build-pod[export] Adding label 'org.springframework.boot.version'
pet-clinic-build-1-build-pod[export] Adding label 'org.opencontainers.image.source'
pet-clinic-build-1-build-pod[export] Setting default process type 'web'
pet-clinic-build-1-build-pod[export] Saving gcr.io/dalfonso-tanzu-dev-frmwrk/pet-clini
c-default...
pet-clinic-build-1-build-pod[export] *** Images (sha256:2ae6154c4433d870a330a0c2fc8253
40c3ead2603e3d1526e47c47cb6297fffe):
pet-clinic-build-1-build-pod[export]       gcr.io/dalfonso-tanzu-dev-frmwrk/pet-clinic
-default
pet-clinic-build-1-build-pod[export]       gcr.io/dalfonso-tanzu-dev-frmwrk/pet-clinic
-default:b1.20220603.181107
pet-clinic-build-1-build-pod[export] Adding cache layer 'paketo-buildpacks/bellsoft-li
berica:jdk'
pet-clinic-build-1-build-pod[export] Adding cache layer 'paketo-buildpacks/syft:syft'
pet-clinic-build-1-build-pod[export] Adding cache layer 'paketo-buildpacks/maven:appli
cation'
pet-clinic-build-1-build-pod[export] Adding cache layer 'paketo-buildpacks/maven:cach
e'
pet-clinic-build-1-build-pod[export] Adding cache layer 'cache.sbom'
```

`--namespace`, `-n`

Specifies the namespace where the workload was deployed to get logs from.

```
tanzu apps workload tail pet-clinic -n development

pet-clinic-00004-deployment-6445565f7b-ts8l5[workload] 2022-06-14 16:28:52.684  INFO 1
--- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engi
ne: [Apache Tomcat/9.0.63]
+ pet-clinic-build-3-build-pod › export
pet-clinic-00004-deployment-6445565f7b-ts8l5[workload] 2022-06-14 16:28:52.699  INFO 1
--- [           main] o.a.c.c.C.[Tomcat-1].[localhost].[/]     : Initializing Spring e
mbedded WebApplicationContext
pet-clinic-00004-deployment-6445565f7b-ts8l5[workload] 2022-06-14 16:28:52.699  INFO 1
--- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationCo
ntext: initialization completed in 131 ms
pet-clinic-00004-deployment-6445565f7b-ts8l5[workload] 2022-06-14 16:28:52.755  INFO 1
--- [           main] o.s.b.a.e.web.EndpointLinksResolver      : Exposing 13 endpoint
(s) beneath base path '/actuator'
pet-clinic-00004-deployment-6445565f7b-ts8l5[workload] 2022-06-14 16:28:53.059  INFO 1
--- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on por
t(s): 8081 (http) with context path ''
```

```
pet-clinic-00004-deployment-6445565f7b-ts8l5[workload] 2022-06-14 16:28:53.074  INFO 1
--- [           main] o.s.s.petclinic.PetClinicApplication     : Started PetClinicAppl
ication in 8.373 seconds (JVM running for 8.993)
pet-clinic-00004-deployment-6445565f7b-ts8l5[workload] 2022-06-14 16:28:53.229  INFO 1
--- [nio-8081-exec-1] o.a.c.c.C.[Tomcat-1].[localhost].[/]     : Initializing Spring D
ispatcherServlet 'dispatcherServlet'
pet-clinic-00004-deployment-6445565f7b-ts8l5[workload] 2022-06-14 16:28:53.229  INFO 1
--- [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet
'dispatcherServlet'
pet-clinic-00004-deployment-6445565f7b-ts8l5[workload] 2022-06-14 16:28:53.231  INFO 1
--- [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initializat
ion in 2 ms
```

--since

Sets the time duration to start reading logs from, this is set in seconds (`s`), minutes(`m`) or hours (`h`) in the format `0h0m0s`. It is not necessary to indicate a `0` duration, for example, 1 hour, 0 minutes and 1 second is `1h1s`. The default value for this flag is 1 second `1s`.

```
tanzu apps workload tail pet-clinic --since 1h1s

pet-clinic-config-writer-9fbk6-pod[place-tools] 2022/06/14 16:28:04 Copied /ko-app/ent
rypoint to /tekton/bin/entrypoint
pet-clinic-config-writer-9fbk6-pod[place-scripts] 2022/06/14 16:28:06 Decoded script /
tekton/scripts/script-0-dz84w
pet-clinic-config-writer-9fbk6-pod[step-init] 2022/06/14 16:28:05 Setup /step director
ies
pet-clinic-config-writer-9fbk6-pod[step-main] ++ mktemp -d
pet-clinic-config-writer-9fbk6-pod[step-main] + cd /tmp/tmp.n4ObHYVxpl
pet-clinic-config-writer-9fbk6-pod[step-main] + echo -e eyJkZWxpdmVyeS55bWwiOiJhcGlWZX
JzaW9uOiBzZXJ2aW5nLmtuYXRpdmUuZGV2L3YxXG5raW5kOiBTZXJ2aWNlXG5tZXRhZGF0YTpcbiAgbmFtZTog
cGV0LWNsaW5pY1xuICBsYWJlbHM6XG4gICAgYXBwcy50YW56dS52bXdhcmUuY29tL3dvcmtsb2FkLXR5cGU6IH
dlYlxuICAgIGF1dG9zY2FsaW5nLmtuYXRpdmUuZGV2L21pbi1zY2FsZTogXCIxXCJcbiAgICBhcHAua3ViZXJu
ZXRlcy5pby9jb21wb25lbnQ6IHJ1blxuICAgIGNhcnRvLnJ1bi93b3JrbG9hZC1uYW1lOiBwZXQtY2xpbmljXG
5zcGVjOlxuICB0ZW1wbGF0ZTpcbiAgICBtZXRhZGF0YTpcbiAgICAgIGFubm90YXRpb25zOlxuICAgICAgICBi
b290LnNwcmluZy5pby9hY3R1YXRvcjogaHR0cDovLzo4MDgxL2FjdHVhdG9yXG4gICAgICAgIGJvb3Quc3ByaW
5nLmlvL3ZlcnNpb246IDIuNi44XG4gICAgICAgIGNvbnZlbnRpb25zLmFwcHMudGFuenUudm13YXJlLmNvbS9h
cHBsaWVkLWNvbnZlbnRpb25zOiB8LVxuICAgICAgICAgIHNwcmluZy1ib290LWNvbnZlbnRpb24vc3ByaW5nLW
Jvb3RcbiAgICAgICAgICBzcHJpbmctYm9vdC1jb252ZW50aW9uL3NwcmluZy1ib290LWdyYWNlZnVsLXNodXRk
b3duXG4gICAgICAgICAgc3ByaW5nLWJvb3QtY29udmVudGlvbi9zcHJpbmctYm9vdC13ZWJcbiAgICAgICAgIC
BzcHJpbmctYm9vdC1jb252ZW50aW9uL3NwcmluZy1ib290LWFjdHVhdG9yXG4gICAgICAgICAgc3ByaW5nLWJv
b3QtY29udmVudGlvbi9zcHJpbmctYm9vdC1hY3R1YXRvci1wcm9iZXNcbiAgICAgICAgICBzcHJpbmctYm9vdC
1jb252ZW50aW9uL3NlcnZpY2UtaW50ZW50LW15c3FsXG4gICAgICAgICAgc3ByaW5nLWJvb3QtY29udmVudGlv
bi9zZXJ2aWNlLWludGVudC1wb3N0Z3Jlc1xuICAgICAgICAgIGFwcGxpdmV2aWV3LXNhbXBsZS9hcHAtbGl2ZS
12aWV3LWNvbm5lY3RvclxuICAgICAgICAgIGFwcGxpdmV2aWV3LXNhbXBsZS9hcHAtbGl2ZS12aWV3LWFwcGZs
YXZvdXJzXG4gICAgICAgICAgYXBwbGl2ZXZpZXctc2FtcGxlL2FwcC1saXZlLXZpZXctc3lzdGVtcHJvcGVydG
llc1xuICAgICAgICBkZXZlbG9wZXIuY29udmVudGlvbnMvdGFyZ2V0LWNvbnRhaW5lcnM6IHdvcmtsb2FkXG4g
ICAgICAgIHNlcnZpY2VzLmNvbnZlbnRpb25zLmFwcHMudGFuenUudm13YXJlLmNvbS9teXNxbDogbXlzcWwtY2
9ubmVjdG9yLWphdmEvOC4wLjI5XG4gICAgICAgIHNlcnZpY2VzLmNvbnZlbnRpb25zLmFwcHMudGFuenUudm13
YXJlLmNvbS9wb3N0Z3JlczogcG9zdGdyZXNxbC80Mi4zLjVcbiAgICAgIGxhYmVsczpcbiAgICAgICAgYXBwLm
t1YmVybmV0ZXMuaW8vY29tcG9uZW50OiBydW5cbiAgICAgICAgYXBwcy50YW56dS52bXdhcmUuY29tL3dvcmts
b2FkLXR5cGU6IHdlYlxuICAgICAgICBjYXJ0by5ydW4vd29ya2xvYWQtbmFtZTogcGV0LWNsaW5pY1xuICAgIC
AgICBjb252ZW50aW9ucy5hcHBzLnRhbnp1LnZtd2FyZS5jb20vZnJhbWV3b3JrOiBzcHJpbmctYm9vdFxuICAg
ICAgICBzZXJ2aWNlcy5jb252ZW50aW9ucy5hcHBzLnRhbnp1LnZtd2FyZS5jb20vbXlzcWw6IHdvcmtsb2FkXG
4gICAgICAgIHNlcnZpY2VzLmNvbnZlbnRpb25zLmFwcHMudGFuenUudm13YXJlLmNvbS9wb3N0Z3JlczogdG9y
a2xvYWRcbiAgICAgICAgdGFuenUuYXBwLmxpdmUudmlldzogXCJ0cnVlXCJcbiAgICAgICAgdGFuenUuYXBwLm
xpdmUudmlldy5hcHBsaWNhdGlvbi5hY3R1YXRvci5wb3J0OiBcIjgwODFcIlxuICAgICAgICB0YW56dS5hcHAu
bGl2ZS52aWV3LmFwcGxpY2F0aW9uLmZsYXZvdXJzOiBzcHJpbmctYm9vdFxuICAgICAgICB0YW56dS5hcHAubG
l2ZS52aWV3LmFwcGxpY2F0aW9uLm5hbWU6IHBldGNsaW5pY1xuICAgIHNwZWM6XG4gICAgICBjb250YWluZXJz
OlxuICAgICAgLSBlbnY6XG4gICAgICAgIC0gbmFtZTogSkFWQV9UT09MX09QVElPTlNcbiAgICAgICAgICB2YW
x1ZTogLURtYW5hZ2VtZW50LmVuZHBvaW50LmhlYWx0aC5wcm9iZXMuYWRkLWFkZGl0aW9uYWwtcGF0aHM9XCJ0
cnVlXCIgLURtYW5hZ2VtZW50LmVuZHBvaW50LmhlYWx0aC5zaG93LWRldGFpbHM9YWx3YXlzIC1EbWFuYWdlbW
VudC5lbmRwb2ludHMud2ViLmJhc2UtcGF0aD1cIi9hY3R1YXRvclwiIC1EbWFuYWdlbWVudC5lbmRwb2ludHMu
d2ViLmV4cG9zdXJlLmluY2x1ZGU9KiAtRG1hbmFnZW1lbnQuaGVhbHRoLnByb2Jlcy5lbmFibGVkPVwidHJ1ZV
wiIC1EbWFuYWdlbWVudC5zZXJ2ZXIucG9ydD1cIjgwODFcIiAtRHNlcnZlci5wb3J0PVwiODA4MFwiIC1Ec2Vy
dmVyLnNodXRkb3duLmdyYWNlLXBlcmlvZD1cIjI0c1wiXG4gICAgICAgIGltYWdlOiBnY3IuaW8vZGFsZm9uc2
8tdGFuenUtZGV2LWZybXdyay9wZXQtY2xpbmljLWRlZmF1bHRAc2hhMjU2OjM5NjRiNTQwNTVlZjNkNmFiNWQ3
YTM5MmVjOGU3OWJhOTg2NjczODU2NmIyOGE2OGY4ZDM2YWY5YjkyMGJhODNcbiAgICAgICAgbGl2ZW5lc3NQcm
9iZTpcbiAgICAgICAgICBodHRwR2V0OlxuICAgICAgICAgICAgcGF0aDogL2xpdmV6XG4gICAgICAgICAgICBw
b3J0OiA4MDgwXG4gICAgICAgICAgICBzY2hlbWU6IEhUVFBBcbiAgICAgICAgbmFtZTogd29ya2xvYWRcbiAgIC
AgICAgcG9ydHM6XG4gICAgICAgIC0gY29udGFpbmVyUG9ydDogODA4MFxuICAgICAgICAgIHByb3RvY29sOiBU
```

```
Q1BcbiAgICAgICAgcmVhZGluZXNzUHJvYmU6XG4gICAgICAgICAgaHR0cEdldDpcbiAgICAgICAgICAgIHBhdG
g6IC9yZWFkeXpcbiAgICAgICAgICAgIHBvcnQ6IDgwODBcbiAgICAgICAgIHNjaGVtZTogSFRUUFxuICAg
ICAgICByZXNvdXJjZXM6IHt9XG4gICAgICAgIHNlY3VyaXR5Q29udGV4dDpcbiAgICAgICAgICBydW5Bc1VzZX
I6IDEwMDBcbiAgICAgIHNlcnZpY2VBY2NvdW50TmFtZTogZGVmYXVsdFxuIn0=
pet-clinic-config-writer-9fbk6-pod[step-main] + base64 --decode
pet-clinic-config-writer-9fbk6-pod[step-main] ++ cat files.json
+ pet-clinic-config-writer-kpmc6-pod › place-tools
pet-clinic-config-writer-9fbk6-pod[step-main] ++ jq -r 'to_entries | .[] | @sh "mkdir
-p $(dirname \(.key)) && echo \(.value) > \(.key)"'
+ pet-clinic-config-writer-kpmc6-pod › step-main
+ pet-clinic-config-writer-kpmc6-pod › step-init
+ pet-clinic-config-writer-kpmc6-pod › place-scripts
pet-clinic-config-writer-9fbk6-pod[step-main] + eval 'mkdir -p $(dirname '\''delivery.
yml'\'') && echo '\''apiVersion: serving.knative.dev/v1'
pet-clinic-config-writer-9fbk6-pod[step-main] kind: Service
pet-clinic-config-writer-9fbk6-pod[step-main] metadata:
pet-clinic-config-writer-9fbk6-pod[step-main]   name: pet-clinic
pet-clinic-config-writer-9fbk6-pod[step-main]   labels:
pet-clinic-config-writer-9fbk6-pod[step-main]     apps.tanzu.vmware.com/workload-type:
web
pet-clinic-config-writer-9fbk6-pod[step-main]     autoscaling.knative.dev/min-scale:
"1"
pet-clinic-config-writer-9fbk6-pod[step-main]     app.kubernetes.io/component: run
pet-clinic-config-writer-9fbk6-pod[step-main]     carto.run/workload-name: pet-clinic
```

`--timestamp`, `-t`

Adds the timestamp to the beginning of each log message

```
tanzu apps workload tail pet-clinic -t

pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645910625-0
5:00
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645942876-0
5:00
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645951930-0
5:00               |\      _,,,--,,_
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645957151-0
5:00              /,`.-'`'   ._  \-;;,_
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645961411-0
5:00   _______ __|,4-  ) )_   .;.(__`'-'__     ___ __    _ ___ _______
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645967316-0
5:00  |       | '---''(_/._)-'(_\_)   |   |   |   |  |  | |   |       |
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645971010-0
5:00  |    _  |    ___|_     _|       |   |   |   |   |_| |   |       | __ _ _
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645976591-0
5:00  |   |_| |   |___  |   | |       |   |   |   |       |   |       | \ \ \ \
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645986474-0
5:00  |    ___|    ___| |   | |      _|   |___|   |  _    |   |      _|  \ \ \ \
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645990521-0
5:00  |   |   |   |___  |   | |     |_|       |   | | |   |   |     |_    ) ) ) )
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645994112-0
5:00  |___|   |_______| |___| |_______|_______|___|_|  |__|___|_______|  / / / /
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645998053-0
5:00  ==================================================================/_/_/_/
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.646001577-0
5:00
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.646005296-0
5:00 :: Built with Spring Boot :: 2.6.8
```

Tanzu apps

This topic includes a description of applications (apps) available on Kubernetes.

Options

```
      --context name      name of the kubeconfig context to use (default is current-co
ntext defined by kubeconfig)
```

```
  -h, --help                help for apps
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
      --no-color          deactivate color output in terminals
  -v, --verbose int32     number for the log level verbosity (default 1)
```

See also

- Tanzu Apps Cluster Supply Chain - Patterns for building and configuring workloads
- Tanzu Apps Workload - Workload life cycle management

Tanzu apps

This topic includes a description of applications (apps) available on Kubernetes.

Options

```
      --context name      name of the kubeconfig context to use (default is current-co
ntext defined by kubeconfig)
  -h, --help              help for apps
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
      --no-color          deactivate color output in terminals
  -v, --verbose int32     number for the log level verbosity (default 1)
```

See also

- Tanzu Apps Cluster Supply Chain - Patterns for building and configuring workloads
- Tanzu Apps Workload - Workload life cycle management

Tanzu apps workload

This topic helps you with workload life cycle management.

A workload can run as a Knative service, Kubernetes deployment, or other runtime. Workloads can be grouped together with other related resources, such as storage or credential objects as a logical application for easier management.

Workload configuration includes:

- Source code to build
- Runtime resource limits
- Environment variables
- Services to bind

Options

```
  -h, --help   help for workload
```

Options inherited from parent commands

```
      --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)
      --no-color          deactivate color output in terminals
  -v, --verbose int32     number for the log level verbosity (default 1)
```

See also

- Tanzu applications - Applications on Kubernetes

- Tanzu apps workload apply - Apply configuration to a new or existing workload
- Tanzu apps workload create - Create a workload with specified configuration
- Tanzu apps workload delete - Delete workload(s)
- Tanzu apps workload get - Get details from a workload
- Tanzu apps workload list - Table listing of workloads
- Tanzu apps workload tail - Watch workload-related logs
- Tanzu apps workload update - Update configuration of an existing workload

Tanzu apps workload

This topic helps you with workload life cycle management.

A workload can run as a Knative service, Kubernetes deployment, or other runtime. Workloads can be grouped together with other related resources, such as storage or credential objects as a logical application for easier management.

Workload configuration includes:

- Source code to build
- Runtime resource limits
- Environment variables
- Services to bind

Options

```
  -h, --help   help for workload
```

Options inherited from parent commands

```
      --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)
      --no-color          deactivate color output in terminals
  -v, --verbose int32     number for the log level verbosity (default 1)
```

See also

- Tanzu applications - Applications on Kubernetes
- Tanzu apps workload apply - Apply configuration to a new or existing workload
- Tanzu apps workload create - Create a workload with specified configuration
- Tanzu apps workload delete - Delete workload(s)
- Tanzu apps workload get - Get details from a workload
- Tanzu apps workload list - Table listing of workloads
- Tanzu apps workload tail - Watch workload-related logs
- Tanzu apps workload update - Update configuration of an existing workload

Tanzu apps workload apply

This topic helps you apply configurations to a new or existing workload.

Synopsis

Apply configurations to a new or existing workload. If the resource does not exist, it is created.

Workload configuration options include:

- Source code to build (if there is a `.tanzuignore` file, file paths listed there will be ignored in the build)
- runtime resource limits
- environment variables
- services to bind
- Set complex params with `--param-yaml` (use `$` as prefix in value when escape characters `\` within)

```
tanzu apps workload apply [name] [flags]
```

Examples

```
tanzu apps workload apply --file workload.yaml
tanzu apps workload apply my-workload --param-yaml maven=$"artifactId:hello-world\ntyp
e: jar\nversion: 0.0.1\ngroupId: carto.run"
```

Options

```
      --annotation "key=value" pair    annotation is represented as a "key=value" pair
("key-" to remove, flag can be used multiple times)
      --app name                       application name the workload is a part of
      --build-env "key=value" pair     build environment variables represented as a "k
ey=value" pair ("key-" to remove, flag can be used multiple times)
      --debug                          put the workload in debug mode (--debug=false t
o deactivate)
      --dry-run                        print kubernetes resources to stdout rather tha
n apply them to the cluster, messages normally on stdout will be sent to stderr
      --env "key=value" pair           environment variables represented as a "key=val
ue" pair ("key-" to remove, flag can be used multiple times)
  -f, --file file path                 file path containing the description of a singl
e workload, other flags are layered on top of this resource. Use value "-" to read fro
m stdin
      --git-branch branch              branch within the Git repository to checkout
      --git-commit SHA                 commit SHA within the Git repository to checkou
t
      --git-repo url                   git url to remote source code
      --git-tag tag                    tag within the Git repository to checkout
  -h, --help                           help for apply
      --image image                    pre-built image, skips the source resolution an
d build phases of the supply chain
      --label "key=value" pair         label is represented as a "key=value" pair ("ke
y-" to remove, flag can be used multiple times)
      --limit-cpu cores                the maximum amount of cpu allowed, in CPU cores
(500m = .5 cores)
      --limit-memory bytes             the maximum amount of memory allowed, in bytes
(500Mi = 500MiB = 500 * 1024 * 1024)
      --live-update                    put the workload in live update mode (--live-up
date=false to deactivate)
      --local-path path                path to a directory, .zip, .jar or .war file co
ntaining workload source code
  -n, --namespace name                 kubernetes namespace (defaulted from kube confi
g)
      --param "key=value" pair         additional parameters represented as a "key=val
ue" pair ("key-" to remove, flag can be used multiple times)
      --param-yaml "key=value" pair    specify nested parameters using YAML or JSON fo
rmatted values represented as a "key=value" pair ("key-" to remove, flag can be used m
ultiple times)
      --request-cpu cores              the minimum amount of cpu required, in CPU core
s (500m = .5 cores)
      --request-memory bytes           the minimum amount of memory required, in bytes
(500Mi = 500MiB = 500 * 1024 * 1024)
      --service-account string         name of service account permitted to create res
ources submitted by the supply chain (to unset, pass empty string "")
```

```
      --service-ref object reference   object reference for a service to bind to the w
orkload "service-ref-name=apiVersion:kind:service-binding-name" ("service-ref-name-" t
o remove, flag can be used multiple times)
  -s, --source-image image             destination image repository where source code
is staged before being built
      --sub-path path                  relative path inside the repository or image to
treat as application root (to unset, pass empty string "")
      --tail                           show logs while waiting for workload to become
ready
      --tail-timestamp                 show logs and add timestamp to each log line wh
ile waiting for workload to become ready
      --type type                      distinguish workload type
      --wait                           waits for workload to become ready
      --wait-timeout duration          timeout for workload to become ready when waiti
ng (default 10m0s)
  -y, --yes                            accept all prompts
```

Options inherited from parent commands

```
      --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)
      --no-color          deactivate color output in terminals
  -v, --verbose int32     number for the log level verbosity (default 1)
```

See also

- Tanzu Apps Workload - Workload life cycle management

Tanzu apps workload create

This topic helps you create a workload with the specified configuration.

Synopsis

Create a workload with the specified configuration.

Workload configuration options include:

- Source code to build (if there is a `.tanzuignore` file, filepaths listed there will be ignored in the build)
- Runtime resource limits
- Environment variables
- Services to bind
- Set complex params with `--param-yaml` (use `$` as prefix in value when escape characters `\` within)

```
tanzu apps workload create [name] [flags]
```

Examples

```
tanzu apps workload create my-workload --git-repo https://example.com/my-workload.git
tanzu apps workload create my-workload --local-path . --source-image registry.example/
repository:tag
tanzu apps workload create --file workload.yaml
tanzu apps workload create my-workload --param-yaml maven=$"artifactId:hello-world\nty
pe: jar\nversion: 0.0.1\ngroupId: carto.run"
```

Options

```
      --annotation "key=value" pair    annotation is represented as a "key=value" pair
("key-" to remove, flag can be used multiple times)
      --app name                       application name the workload is a part of
      --build-env "key=value" pair     build environment variables represented as a "k
ey=value" pair ("key-" to remove, flag can be used multiple times)
      --debug                          put the workload in debug mode (--debug=false t
o deactivate)
      --dry-run                        print kubernetes resources to stdout rather tha
n apply them to the cluster, messages normally on stdout will be sent to stderr
      --env "key=value" pair           environment variables represented as a "key=val
ue" pair ("key-" to remove, flag can be used multiple times)
  -f, --file file path                 file path containing the description of a singl
e workload, other flags are layered on top of this resource. Use value "-" to read fro
m stdin
      --git-branch branch              branch within the Git repository to checkout
      --git-commit SHA                 commit SHA within the Git repository to checkou
t
      --git-repo url                   git url to remote source code
      --git-tag tag                    tag within the Git repository to checkout
  -h, --help                           help for create
      --image image                    pre-built image, skips the source resolution an
d build phases of the supply chain
      --label "key=value" pair         label is represented as a "key=value" pair ("ke
y-" to remove, flag can be used multiple times)
      --limit-cpu cores                the maximum amount of cpu allowed, in CPU cores
(500m = .5 cores)
      --limit-memory bytes             the maximum amount of memory allowed, in bytes
(500Mi = 500MiB = 500 * 1024 * 1024)
      --live-update                    put the workload in live update mode (--live-up
date=false to deactivate)
      --local-path path                path to a directory, .zip, .jar or .war file co
ntaining workload source code
  -n, --namespace name                 kubernetes namespace (defaulted from kube confi
g)
      --param "key=value" pair         additional parameters represented as a "key=val
ue" pair ("key-" to remove, flag can be used multiple times)
      --param-yaml "key=value" pair    specify nested parameters using YAML or JSON fo
rmatted values represented as a "key=value" pair ("key-" to remove, flag can be used m
ultiple times)
      --request-cpu cores              the minimum amount of cpu required, in CPU core
s (500m = .5 cores)
      --request-memory bytes           the minimum amount of memory required, in bytes
(500Mi = 500MiB = 500 * 1024 * 1024)
      --service-account string         name of service account permitted to create res
ources submitted by the supply chain (to unset, pass empty string "")
      --service-ref object reference   object reference for a service to bind to the w
orkload "service-ref-name=apiVersion:kind:service-binding-name" ("service-ref-name-" t
o remove, flag can be used multiple times)
  -s, --source-image image             destination image repository where source code
is staged before being built
      --sub-path path                  relative path inside the repo or image to treat
as application root (to unset, pass empty string "")
      --tail                           show logs while waiting for workload to become
ready
      --tail-timestamp                 show logs and add timestamp to each log line wh
ile waiting for workload to become ready
      --type type                      distinguish workload type
      --wait                           waits for workload to become ready
      --wait-timeout duration          timeout for workload to become ready when waiti
ng (default 10m0s)
  -y, --yes
```

Options inherited from parent commands

```
      --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)
      --no-color          deactivate color output in terminals
  -v, --verbose int32     number for the log level verbosity (default 1)
```

See also

- Tanzu Apps Workload - Workload life cycle management

Tanzu apps workload update

This topic helps you update the configuration of an existing workload.

Synopsis

Update the configuration of an existing workload.

Workload configuration options include:

- Source code to build (if there is a `.tanzuignore` file, file paths listed there are ignored in the build)
- runtime resource limits
- environment variables
- services to bind
- Set complex params with `--param-yaml` (use `$` as prefix in value when escape characters `\` within)

```
tanzu apps workload update [name] [flags]
```

Examples

```
tanzu apps workload update my-workload --debug=false
tanzu apps workload update my-workload --local-path .
tanzu apps workload update my-workload --env key=value
tanzu apps workload update my-workload --build-env key=value
tanzu apps workload update --file workload.yaml
tanzu apps workload update my-workload --param-yaml maven=$"artifactId:hello-world\nty
pe: jar\nversion: 0.0.1\ngroupId: carto.run"
```

Options

```
      --annotation "key=value" pair    annotation is represented as a "key=value" pair
("key-" to remove, flag can be used multiple times)
      --app name                       application name the workload is a part of
      --build-env "key=value" pair     build environment variables represented as a "k
ey=value" pair ("key-" to remove, flag can be used multiple times)
      --debug                          put the workload in debug mode (--debug=false t
o deactivate)
      --dry-run                        print kubernetes resources to stdout rather tha
n apply them to the cluster, messages normally on stdout will be sent to stderr
      --env "key=value" pair           environment variables represented as a "key=val
ue" pair ("key-" to remove, flag can be used multiple times)
  -f, --file file path                 file path containing the description of a singl
e workload, other flags are layered on top of this resource. Use value "-" to read fro
m stdin
      --git-branch branch              branch within the Git repository to checkout
      --git-commit SHA                 commit SHA within the Git repository to checkou
t
      --git-repo url                   git url to remote source code
      --git-tag tag                    tag within the Git repository to checkout
  -h, --help                           help for update
      --image image                    pre-built image, skips the source resolution an
d build phases of the supply chain
      --label "key=value" pair         label is represented as a "key=value" pair ("ke
y-" to remove, flag can be used multiple times)
      --limit-cpu cores                the maximum amount of cpu allowed, in CPU cores
(500m = .5 cores)
      --limit-memory bytes             the maximum amount of memory allowed, in bytes
(500Mi = 500MiB = 500 * 1024 * 1024)
```

```
      --live-update                    put the workload in live update mode (--live-up
date=false to deactivate)
      --local-path path                path to a directory, .zip, .jar or .war file co
ntaining workload source code
  -n, --namespace name                 kubernetes namespace (defaulted from kube confi
g)
      --param "key=value" pair         additional parameters represented as a "key=val
ue" pair ("key-" to remove, flag can be used multiple times)
      --param-yaml "key=value" pair    specify nested parameters using YAML or JSON fo
rmatted values represented as a "key=value" pair ("key-" to remove, flag can be used m
ultiple times)
      --request-cpu cores              the minimum amount of cpu required, in CPU core
s (500m = .5 cores)
      --request-memory bytes           the minimum amount of memory required, in bytes
(500Mi = 500MiB = 500 * 1024 * 1024)
      --service-account string         name of service account permitted to create res
ources submitted by the supply chain (to unset, pass empty string "")
      --service-ref object reference   object reference for a service to bind to the w
orkload "service-ref-name=apiVersion:kind:service-binding-name" ("service-ref-name-" t
o remove, flag can be used multiple times)
  -s, --source-image image             destination image repository where source code
is staged before being built
      --sub-path path                  relative path inside the repo or image to treat
as application root (to unset, pass empty string "")
      --tail                           show logs while waiting for workload to become
ready
      --tail-timestamp                 show logs and add timestamp to each log line wh
ile waiting for workload to become ready
      --type type                      distinguish workload type
      --wait                           waits for workload to become ready
      --wait-timeout duration          timeout for workload to become ready when waiti
ng (default 10m0s)
  -y, --yes                            accept all prompts
```

Options inherited from parent commands

```
      --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)
      --no-color          deactivate color output in terminals
  -v, --verbose int32     number for the log level verbosity (default 1)
```

See also

- Tanzu Apps Workload - Workload life cycle management

Tanzu apps workload get

This topic helps you get details from a workload.

```
tanzu apps workload get <name> [flags]
```

Examples

```
tanzu apps workload get my-workload
```

Options

```
      --export           export workload in yaml format
  -h, --help             help for get
  -n, --namespace name   kubernetes namespace (defaulted from kube config)
  -o, --output string    output the Workload formatted. Supported formats: "json", "ya
ml", "yml"
```

Options inherited from parent commands

```
      --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)
      --no-color          deactivate color output in terminals
  -v, --verbose int32     number for the log level verbosity (default 1)
```

See also

- Tanzu apps workload - Workload life cycle management

Tanzu apps workload delete

This topic helps you delete one or more workloads by name or all workloads within a namespace.

Deleting a workload prevents new builds while preserving built images in the registry.

```
tanzu apps workload delete <name(s)> [flags]
```

Examples

```
tanzu apps workload delete my-workload
tanzu apps workload delete --all
```

Options

```
      --all                      delete all workloads within the namespace
  -f, --file file path           file path containing the description of a single workl
oad; other flags are layered on top of this resource. Use value "-" to read from stdin
  -h, --help                     help for delete
  -n, --namespace name           kubernetes namespace (defaulted from kube config)
      --wait                     waits for workload to be deleted
      --wait-timeout duration    timeout for workload to be deleted when waiting (defau
lt 1m0s)
  -y, --yes                      accept all prompts
```

Options inherited from parent commands

```
      --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)
      --no-color          deactivate color output in terminals
  -v, --verbose int32     number for the log level verbosity (default 1)
```

See also

- Tanzu Apps Workload - Workload life cycle management

Tanzu apps workload list

This topic will help you list workloads in a namespace or across all namespaces.

```
tanzu apps workload list [flags]
```

Examples

```
tanzu apps workload list
tanzu apps workload list --all-namespaces
```

Options

```
  -A, --all-namespaces   use all kubernetes namespaces
      --app name         application name the workload is a part of
  -h, --help             help for list
  -n, --namespace name   kubernetes namespace (defaulted from kube config)
```

Options inherited from parent commands

```
      --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)
      --no-color          deactivate color output in terminals
  -v, --verbose int32     number for the log level verbosity (default 1)
```

See also

- Tanzu Apps Workload - Workload life cycle management

Tanzu apps workload tail

This topic will help you to watch workload related logs.

You can stream logs for a workload until canceled. To cancel, press Ctl-c in the shell or stop the process. As new workload pods are started, the logs are displayed. To show historical logs use – since.

```
tanzu apps workload tail <name> [flags]
```

Examples

```
tanzu apps workload tail my-workload
tanzu apps workload tail my-workload --since 1h
```

Options

```
      --component name   workload component name (e.g. build)
  -h, --help             help for tail
  -n, --namespace name   kubernetes namespace (defaulted from kube config)
      --since duration   time duration to start reading logs from (default 1s)
  -t, --timestamp        print timestamp for each log line
```

Options inherited from parent commands

```
      --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)
      --no-color          deactivate color output in terminals
  -v, --verbose int32     number for the log level verbosity (default 1)
```

See also

- Tanzu Apps Workload - Workload life cycle management

Tanzu apps cluster supply chain

This topic includes patterns for building and configuring workloads.

Options

```
  -h, --help   help for cluster-supply-chain
```

Options inherited from parent commands

```
      --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)
      --no-color          deactivate color output in terminals
  -v, --verbose int32     number for the log level verbosity (default 1)
```

See also

- Tanzu applications - Applications on Kubernetes
- tanzu apps cluster-supply-chain get - Get details from a cluster supply chain
- Tanzu apps cluster supply chain list - Table listing of cluster supply chains

Tanzu apps cluster supply chain

This topic includes patterns for building and configuring workloads.

Options

```
  -h, --help   help for cluster-supply-chain
```

Options inherited from parent commands

```
      --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)
      --no-color          deactivate color output in terminals
  -v, --verbose int32     number for the log level verbosity (default 1)
```

See also

- Tanzu applications - Applications on Kubernetes
- tanzu apps cluster-supply-chain get - Get details from a cluster supply chain
- Tanzu apps cluster supply chain list - Table listing of cluster supply chains

Tanzu apps cluster supply chain list

This topic helps you list cluster supply chains.

```
tanzu apps cluster-supply-chain list [flags]
```

Examples

```
tanzu apps cluster-supply-chain list
```

Options

```
  -h, --help   help for list
```

Options inherited from parent commands

```
      --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)
      --no-color          deactivate color output in terminals
  -v, --verbose int32     number for the log level verbosity (default 1)
```

See also

- Tanzu apps cluster supply chain - Patterns for building and configuring workloads

Usage and examples

Changing clusters

The Apps CLI plug-in refers to the default kubeconfig file to access a Kubernetes cluster. When a `tanzu apps` command is run, the plug-in uses the default context that's defined in that kubeconfig file (located by default at `$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 will target the cluster defined in the new default kubeconfig context.
2. Include the `--context <context-name>` flag when running any `tanzu apps` command. All subsequent `tanzu apps` commands without the `--context <context-name>` flag will continue to use the default context set in the kubeconfig.

There are also two ways to override the default kubeconfig:

1. Set the env var `KUBECONFIG=<path>` to change the kubeconfig the Apps CLI plug-in will reference. All subsequent `tanzu apps` commands will reference the non-default kubeconfig assigned to the env var.
2. Include the `--kubeconfig <path>` flag when running any `tanzu apps` command. All subsequent `tanzu apps` commands without the `--kubeconfig <path>` flag will continue to use the default kubeconfig.

For more information about kubeconfig, see Configure Access to Multiple Clusters.

Checking update status

You can use the Apps CLI plug-in to create or update a workload. After you've successfully submitted your changes to the platform, the CLI command exits. Depending on the changes you submitted, it might take time for them to be executed on the platform. Run `tanzu apps workload get` to check the status of your changes. For more information on this command, see Tanzu Apps Workload Get.

Working with YAML files

In many cases, you can manage workload life cycles through CLI commands. However, you might find cases where you want to manage a workload by using a `yaml` file. The Apps CLI plug-in supports using `yaml` files.

The plug-in is designed to manage one workload at a time. When you manage a workload using a `yaml` file, that file must contain a single workload definition. Plug-in commands support only one file per command.

For example, a valid file looks similar to the following example:

```
---
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:
  source:
    git:
      url: https://github.com/sample-accelerators/spring-petclinic
      ref:
        tag: tap-1.1
```

To create a workload from a file like the one just shown, run:

```
tanzu apps workload create -f my-workload-file.yaml
```

Another way to create a workload from `yaml` is passing the definition through `stdin`. For example, run:

```
tanzu apps workload create -f - --yes
```

The console remains waiting for some input, and the content with a valid `yaml` definition for a workload can be either written or pasted, then press `ctrl`+D three times to start workload creation. This can also be done with `workload update` and `workload apply` commands.

Note: to pass workload through `stdin`, `--yes` flag is needed. If not used, command will fail.

Autocompletion

To enable command autocompletion, the Tanzu CLI offers the `tanzu completion` command.

Add the following command to the shell config file according to the current setup. Use one of the following options:

Bash

```
tanzu completion bash >  $HOME/.tanzu/completion.bash.inc
```

Zsh

```
echo "autoload -U compinit; compinit" >> ~/.zshrc
tanzu completion zsh > "${fpath[1]}/_tanzu"
```

Application Accelerator CLI plug-in overview

The Application Accelerator Tanzun CLI plug-in includes commands for developers and operators to create and use accelerators.

Server API connections for operators and developers

The Application 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 acccelerator 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. The URL depends on the configuration settings for Application Accelerator:

- For installations configured with a **shared 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.

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.

Note: this URL can be used for the VScode extension `acc server url` config

Installation

For information about installing the Tanzu CLI accelerator plug-in, see Install Accelerator CLI plug-in.

Command reference

For information about available commands, see Command Reference.

Application Accelerator CLI plug-in overview

The Application Accelerator Tanzun CLI plug-in includes commands for developers and operators to create and use accelerators.

Server API connections for operators and developers

The Application 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 acccelerator 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. The URL depends on the configuration settings for Application Accelerator:

- For installations configured with a **shared 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.

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.

Note: this URL can be used for the VScode extension `acc server url` config

Installation

For information about installing the Tanzu CLI accelerator plug-in, see Install Accelerator CLI plug-in.

Command reference

For information about available commands, see Command Reference.

Install Accelerator CLI plug-in

This topic tells you how to install the Accelerator CLI plug-in.

> **Note**
>
> Follow the steps in this topic if you do not want to use a profile to install Accelerator CLI plug-in. For more information about profiles, see About Tanzu Application Platform components and profiles.

Prerequisites

Before you install the Accelerator CLI plug-in:

- Follow the instructions to Install or update the Tanzu CLI and plug-ins.

Install

To install the Accelerator CLI plug-in:

1. From the `$HOME/tanzu` directory, run:

   ```
   tanzu plugin install --local ./cli accelerator
   ```

2. To verify that the CLI is installed correctly, run:

   ```
   tanzu accelerator version
   ```

   A version will be displayed in the output.

   If the following error is displayed during installation:

```
Error: could not find plug-in "accelerator" in any known repositories

✖  could not find plug-in "accelerator" in any known repositories
```

Verify that there is an `accelerator` entry in the `cli/manifest.yaml` file. It will look similar to this:

```
plugins:
...
    - name: accelerator
    description: Manage accelerators in a Kubernetes cluster
    versions: []
```

Command reference

- tanzu accelerator
  - tanzu accelerator apply
  - tanzu accelerator create
  - tanzu accelerator delete
  - tanzu accelerator fragment
  - tanzu accelerator fragment create
  - tanzu accelerator fragment delete
  - tanzu accelerator fragment get
  - tanzu accelerator fragment list
  - tanzu accelerator fragment update
  - tanzu accelerator generate
  - tanzu accelerator get
  - tanzu accelerator list
  - tanzu accelerator push
  - tanzu accelerator update

Command reference

- tanzu accelerator
  - tanzu accelerator apply
  - tanzu accelerator create
  - tanzu accelerator delete
  - tanzu accelerator fragment
  - tanzu accelerator fragment create
  - tanzu accelerator fragment delete
  - tanzu accelerator fragment get
  - tanzu accelerator fragment list
  - tanzu accelerator fragment update
  - tanzu accelerator generate
  - tanzu accelerator get
  - tanzu accelerator list
  - tanzu accelerator push
  - tanzu accelerator update

tanzu accelerator

This command manages accelerators in a Kubernetes cluster.

Options

```
      --context name      name of the kubeconfig context to use (default is current-co
ntext defined by kubeconfig)
  -h, --help              help for accelerator
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
```

SEE ALSO

- tanzu accelerator apply - Apply accelerator resource
- tanzu accelerator create - Create a new accelerator
- tanzu accelerator delete - Delete an accelerator
- tanzu accelerator fragment - Fragment commands
- tanzu accelerator generate - Generate project from accelerator
- tanzu accelerator get - Get accelerator information.
- tanzu accelerator list - List accelerators
- tanzu accelerator push - Push local path to source image
- tanzu accelerator update - Update an accelerator

tanzu accelerator

This command manages accelerators in a Kubernetes cluster.

Options

```
      --context name      name of the kubeconfig context to use (default is current-co
ntext defined by kubeconfig)
  -h, --help              help for accelerator
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
```

SEE ALSO

- tanzu accelerator apply - Apply accelerator resource
- tanzu accelerator create - Create a new accelerator
- tanzu accelerator delete - Delete an accelerator
- tanzu accelerator fragment - Fragment commands
- tanzu accelerator generate - Generate project from accelerator
- tanzu accelerator get - Get accelerator information.
- tanzu accelerator list - List accelerators
- tanzu accelerator push - Push local path to source image
- tanzu accelerator update - Update an accelerator

tanzu accelerator apply

tanzu accelerator apply

This command creates or updates accelerators.

Synopsis

Create or update accelerator resource using specified manifest file.

```
tanzu accelerator apply [flags]
```

Examples

```
tanzu accelerator apply --filename <path-to-resource-manifest>
```

Options

```
  -f, --filename string    path of manifest file for the resource
  -h, --help               help for apply
  -n, --namespace string   namespace for the resource (default "accelerator-system")
```

Options inherited from parent commands

```
      --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)
```

SEE ALSO

- tanzu accelerator - Manage accelerators in a Kubernetes cluster

tanzu accelerator create

This command creates a new accelerator.

Synopsis

Create a new accelerator resource with specified configuration.

Accelerator configuration options include: - Git repository URL and branch/tag where accelerator code and metadata is defined - Metadata like description, display-name, tags and icon-url

The Git repository option is required. Metadata options are optional and will override any values for the same options specified in the accelerator metadata retrieved from the Git repository.

```
tanzu accelerator create [flags]
```

Examples

```
tanzu accelerator create <accelerator-name> --git-repository <URL> --git-branch <branc
h>
```

Options

```
      --description string    description of this accelerator
      --display-name string   display name for the accelerator
      --git-branch string     Git repository branch to be used
      --git-repo string       Git repository URL for the accelerator
      --git-sub-path string   Git repository subPath to be used
      --git-tag string        Git repository tag to be used
  -h, --help                  help for create
      --icon-url string       URL for icon to use with the accelerator
      --interval string       interval for checking for updates to Git or image reposi
tory
      --local-path string     path to the directory containing the source for the acce
lerator
  -n, --namespace string      namespace for accelerator system (default "accelerator-s
ystem")
      --secret-ref string     name of secret containing credentials for private Git or
```

```
image repository
      --source-image string   name of the source image for the accelerator
      --tags strings          tags that can be used to search for accelerators
```

Options inherited from parent commands

```
      --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)
```

SEE ALSO

- tanzu accelerator - Manage accelerators in a Kubernetes cluster

tanzu accelerator delete

tanzu accelerator delete

This command deletes an accelerator.

Synopsis

Delete the accelerator resource with the specified name.

```
tanzu accelerator delete [flags]
```

Examples

```
tanzu accelerator delete <accelerator-name>
```

Options

```
  -h, --help               help for delete
  -n, --namespace string   namespace for accelerator system (default "accelerator-syst
em")
```

Options inherited from parent commands

```
      --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)
```

SEE ALSO

- tanzu accelerator - Manage accelerators in a Kubernetes cluster

tanzu accelerator fragment

This command manages fragments.

Synopsis

Commands to manage accelerator fragments

Examples

```
tanzu accelerator fragment --help
```

Options

```
  -h, --help   help for fragment
```

Options inherited from parent commands

```
      --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)
```

SEE ALSO

- tanzu accelerator - Manage accelerators in a Kubernetes cluster
- tanzu accelerator fragment create - Create a new accelerator fragment
- tanzu accelerator fragment delete - Delete an accelerator fragment
- tanzu accelerator fragment get - Get accelerator fragment information
- tanzu accelerator fragment list - List accelerator fragments
- tanzu accelerator fragment update - Update an accelerator fragment

tanzu accelerator fragment create

tanzu accelerator fragment create

This command creates a new accelerator fragment.

Synopsis

Create a new accelerator fragment resource with specified configuration.

Accelerator configuration options include: - Git repository URL and branch/tag where accelerator code and metadata is defined - Metadata like description, display-name, tags and icon-url

The Git repository option is required. Metadata options are optional and will override any values for the same options specified in the accelerator metadata retrieved from the Git repository.

```
tanzu accelerator fragment create [flags]
```

Examples

```
tanzu acceleratorent fragm create <fragment-name> --git-repository <URL> --git-branch
<branch> --git-sub-path <sub-path>
```

Options

```
      --display-name string   display name for the accelerator
      --git-branch string     Git repository branch to be used
      --git-repo string       Git repository URL for the accelerator
      --git-sub-path string   Git repository subPath to be used
      --git-tag string        Git repository tag to be used
  -h, --help                  help for create
      --interval string       interval for checking for updates to Git or image reposi
tory
  -n, --namespace string      namespace for accelerator system (default "accelerator-s
ystem")
      --secret-ref string     name of secret containing credentials for private Git or
image repository
```

Options inherited from parent commands

```
      --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)
```

SEE ALSO

- tanzu accelerator fragment - Fragment commands

tanzu accelerator fragment delete

tanzu accelerator fragment delete

This command deletes an accelerator fragment.

Synopsis

Delete the accelerator fragment resource with the specified name.

```
tanzu accelerator fragment delete [flags]
```

Examples

```
tanzu accelerator fragment delete <fragment-name>
```

Options

```
  -h, --help                help for delete
  -n, --namespace string    namespace for accelerator system (default "accelerator-syst
em")
```

Options inherited from parent commands

```
      --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)
```

SEE ALSO

- tanzu accelerator fragment - Fragment commands

tanzu accelerator fragment get

This command gets accelerator fragment information.

Synopsis

Get accelerator fragment information.

```
tanzu accelerator fragment get [flags]
```

Examples

```
tanzu accelerator get <fragment-name>
```

Options

```
  -h, --help                help for get
  -n, --namespace string    namespace for accelerator system (default "accelerator-syst
em")
```

Options inherited from parent commands

```
      --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)
```

SEE ALSO

- tanzu accelerator fragment - Fragment commands

tanzu accelerator fragment list

This command lists accelerator fragments.

Synopsis

List all accelerator fragments.

```
tanzu accelerator fragment list [flags]
```

Examples

```
tanzu accelerator fragment list
```

Options

```
  -h, --help               help for list
  -n, --namespace string   namespace for accelerator system (default "accelerator-syst
em")
```

Options inherited from parent commands

```
      --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)
```

SEE ALSO

- tanzu accelerator fragment - Fragment commands

tanzu accelerator fragment update

This command updates an accelerator fragment.

Synopsis

Update an accelerator fragment resource with the specified name using the specified configuration.

Accelerator configuration options include: - Git repository URL and branch/tag where accelerator code and metadata is defined - Metadata like display-name

The update command also provides a –reconcile flag that will force the accelerator fragment to be refreshed with any changes made to the associated Git repository.

```
tanzu accelerator fragment update [flags]
```

Examples

```
tanzu accelerator update <accelerator-name> --description "Lorem Ipsum"
```

Options

```
      --display-name string   display name for the accelerator fragment
      --git-branch string     Git repository branch to be used
      --git-repo string       Git repository URL for the accelerator fragment
      --git-sub-path string   Git repository subPath to be used
      --git-tag string        Git repository tag to be used
  -h, --help                  help for update
      --interval string       interval for checking for updates to Git repository
  -n, --namespace string      namespace for accelerator fragments (default "accelerato
r-system")
      --reconcile             trigger a reconciliation including the associated GitRep
ository resource
      --secret-ref string     name of secret containing credentials for private Git re
pository
```

Options inherited from parent commands

```
      --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)
```

SEE ALSO

- tanzu accelerator fragment - Fragment commands

tanzu accelerator generate

tanzu accelerator generate

This command generates a project from an accelerator.

Synopsis

Generate a project from an accelerator using provided options and download project artifacts as a ZIP file.

Generation options are provided as a JSON string and should match the metadata options that are specified for the accelerator used for the generation. The options can include “projectName” which defaults to the name of the accelerator. This “projectName” will be used as the name of the generated ZIP file.

You can see the available options by using the “tanzu accelerator list ” command.

Here is an example of an options JSON string that specifies the “projectName” and an “includeKubernetes” boolean flag:

```
--options '{"projectName":"test", "includeKubernetes": true}'
```

You can also provide a file that specifies the JSON string using the –options-file flag.

The generate command needs access to the Application Accelerator server. You can specify the –server-url flag or set an ACC_SERVER_URL environment variable. If you specify the –server-url flag it will override the ACC_SERVER_URL environmnet variable if it is set.

```
tanzu accelerator generate [flags]
```

Examples

```
tanzu accelerator generate <accelerator-name> --options '{"projectName":"test"}'
```

Options

```
  -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
```

Options inherited from parent commands

```
      --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)
```

SEE ALSO

- tanzu accelerator - Manage accelerators in a Kubernetes cluster

tanzu accelerator get

This command gets accelerator information.

Synopsis

Get accelerator information.

You can choose to get the accelerator from the Application Accelerator server using –server-url flag or from a Kubernetes context using –from-context flag. The default is to get accelerators from the Kubernetes context. To override this, you can set the ACC_SERVER_URL environment variable with the URL for the Application Accelerator server you want to access.

```
tanzu accelerator get [flags]
```

Examples

```
tanzu accelerator get <accelerator-name> --from-context
```

Options

```
      --from-context        retrieve resources from current context defined in kubecon
fig
  -h, --help                help for get
  -n, --namespace string    namespace for accelerator system (default "accelerator-sys
tem")
      --server-url string   the URL for the Application Accelerator server
```

Options inherited from parent commands

```
      --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)
```

SEE ALSO

- tanzu accelerator - Manage accelerators in a Kubernetes cluster

tanzu accelerator list

This command lists accelerators.

Synopsis

List all accelerators.

You can choose to list the accelerators from the Application Accelerator server using –server-url flag or from a Kubernetes context using –from-context flag. The default is to list accelerators from the Kubernetes context. To override this, you can set the ACC_SERVER_URL environment variable with the URL for the Application Accelerator server you want to access.

```
tanzu accelerator list [flags]
```

Examples

```
tanzu accelerator list
```

Options

```
      --from-context        retrieve resources from current context defined in kubecon
fig
  -h, --help                help for list
  -n, --namespace string    namespace for accelerator system (default "accelerator-sys
tem")
      --server-url string   the URL for the Application Accelerator server
```

Options inherited from parent commands

```
      --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)
```

SEE ALSO

- tanzu accelerator - Manage accelerators in a Kubernetes cluster

tanzu accelerator push

tanzu accelerator push

This command pushes source code from local path to source image.

Synopsis

Push source code from local path to source image used by an accelerator

```
tanzu accelerator push [flags]
```

Examples

```
tanzu accelerator push --local-path <local path> --source-image <image>
```

Options

```
  -h, --help                  help for push
      --local-path string     path to the directory containing the source for the acce
```

```
lerator
      --source-image string   name of the source image for the accelerator
```

Options inherited from parent commands

```
      --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)
```

SEE ALSO

- tanzu accelerator - Manage accelerators in a Kubernetes cluster

tanzu accelerator update

This command updates an accelerator.

Synopsis

Update an accelerator resource with the specified name using the specified configuration.

Accelerator configuration options include: - Git repository URL and branch/tag where accelerator code and metadata is defined - Metadata like description, display-name, tags and icon-url

The update command also provides a –reoncile flag that will force the accelerator to be refreshed with any changes made to the associated Git repository.

```
tanzu accelerator update [flags]
```

Examples

```
tanzu accelerator update <accelerator-name> --description "Lorem Ipsum"
```

Options

```
      --description string    description of this accelerator
      --display-name string   display name for the accelerator
      --git-branch string     Git repository branch to be used
      --git-repo string       Git repository URL for the accelerator
      --git-sub-path string   Git repository subPath to be used
      --git-tag string        Git repository tag to be used
  -h, --help                  help for update
      --icon-url string       URL for icon to use with the accelerator
      --interval string       interval for checking for updates to Git or image reposi
tory
  -n, --namespace string      namespace for accelerator system (default "accelerator-s
ystem")
      --reconcile             trigger a reconciliation including the associated GitRep
ository resource
      --secret-ref string     name of secret containing credentials for private Git or
image repository
      --source-image string   name of the source image for the accelerator
      --tags strings          tags that can be used to search for accelerators
```

Options inherited from parent commands

```
      --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)
```

SEE ALSO

- tanzu accelerator - Manage accelerators in a Kubernetes cluster

Tanzu Insight plug-in overview

The Tanzu Insight CLI plug-in enables querying vulnerability, image, and package data.

Follow the below steps to install, configure, and use the 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. If the `insight` plug-in is not already installed, see Install the Tanzu Insight plug-in
2. Configure insight

Once `tanzu insight` CLI plug-in is set up:

1. Add data
2. Query data

Tanzu Insight plug-in overview

The Tanzu Insight CLI plug-in enables querying vulnerability, image, and package data.

Follow the below steps to install, configure, and use the 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. If the `insight` plug-in is not already installed, see Install the Tanzu Insight plug-in
2. Configure insight

Once `tanzu insight` CLI plug-in is set up:

1. Add data
2. Query data

Install the Tanzu Insight CLI plug-in

Note

By following the instructions to install the Tanzu CLI and all the plug-ins, the Tanzu Insight plug-in is also installed.

This topic explains how to install the Tanzu Insight plug-in by itself, after the user has installed the Tanzu CLI.

Note

Follow the steps in this topic if you do not want to use a profile to install the Tanzu Insight CLI plug-in. For more information about profiles, see About Tanzu Application Platform components and profiles.

1. From your `tanzu` directory, install the local version of the Tanzu Insight plug-in you downloaded by running:

   ```
   cd $HOME/tanzu
   tanzu plugin install insight --local cli
   ```

2. Follow the steps in Configure the Tanzu Insight CLI plug-in.

Configure the Tanzu Insight CLI plug-in

This topic explains how to configure the Tanzu Insight plug-in.

Set the target and certificate authority (CA) certificate

Note These instructions are for the recommended configuration where ingress is enabled. For instructions on non-ingress setups, see Configure target endpoint and certificate for more details.

The endpoint host should be set to `metadata-store.<ingress-domain>` (such as `metadata-store.example.domain.com`), where `<ingress-domain>` should match the 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 ingress-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 --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 describes 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 (SBoM) files.

Supported use cases

The following are a few use cases supported by the CLI:

- What packages and CVEs exist in a particular image? (`image`)
- What packages and CVEs exist in my source code? (`source`)
- What dependencies are affected by a specific CVE? (`vulnerabilities`)

Query using the Tanzu Insight CLI plug-in

Install the Tanzu Insight CLI plug-in if you have not already done so.

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 vulnerabilties.
- `vulnerabilities` - Query vulnerabilities by image, package, or source code.

Use `tanzu insight -h` or for more information see Tanzu Insight Details.

Example #1: What packages & CVEs does a specific image contain?

Run:

```
tanzu insight image get --digest DIGEST
```

Where:

- `DIGEST` is the component version or image digest.

For example:

```
$ tanzu insight image get --digest sha256:407d7099d6ce7e3632b6d00682a43028d75d3b088600
797a833607bd629d1ed5
Registry:       docker.io
Image Name:     checkr/flagr:1.1.12
```

```
Digest:         sha256:407d7099d6ce7e3632b6d00682a43028d75d3b088600797a833607bd629d1ed
5
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 #2: What packages & CVEs does my source code contain?

Determining source code org, repository, and commit SHA

In order to query a source scan for vulnerabilities, you need a Git org and Git repository, or the commit SHA. Find these by examining the source scan resource.

Run:

```
kubectl describe sourcescan <workload name> -n <workload namespace>
```

For example:

```
kubectl describe sourcescan tanzu-java-web-app -n my-apps
```

In the resource look for the `Spec.Blob` field. Within, there's `Revision` and `URL`.

For example:

```
Spec:
  Blob:
    Revision:     master/c7e4c27ba43250a4b7c46f030355c108aa73cc39
    URL:          http://source-controller.flux-system.svc.cluster.local./gitrepositor
y/my-apps/tanzu-java-web-app-gitops/c7e4c27ba43250a4b7c46f030355c108aa73cc39.tar.gz
```

In the earlier example, the URL is parsed and split into the org and repo. Revision is parsed as the commit SHA.

- Org is parsed as `gitrepository`
- Repo is parsed as `my-apps/tanzu-java-web-app-gitops/c7e4c27ba43250a4b7c46f030355c108aa73cc39.tar.gz`
- Commit SHA is parsed as `master/c7e4c27ba43250a4b7c46f030355c108aa73cc39`

Use this information to perform your search.

Source code query with repo & org

Run:

```
tanzu insight source get --repo REPO --org ORG
```

Where:

- `REPO` specifies the repository
  - E.g., java-web-app
  - E.g., my-apps/java-web-app/c7ls8bakd87sakjda8d7.tar.gz
- `ORG` is the source code's organization
  - E.g., gitrepository

    - E.g., gitrepositiory-kj32kal8

For example:

```
$ tanzu insight source get --repo my-apps/java-web-app/c7ls8bakd87sakjda8d7.tar.gz --o
rg gitrepository
ID:             1
Repository:  my-apps/java-web-app/c7ls8bakd87sakjda8d7.tar.gz
Commit:  c7e4c27ba43250a4b7c46f030355c108aa73cc39
Organization:   gitrepository
Packages:
                1. go.uber.org/atomic@v1.7.0
                CVEs:
                        1. CVE-2022-42322 (Low)
                2. golang.org/x/crypto@v0.0.0-20220518034528-6f7dac969898
                3. github.com/valyala/bytebufferpool@v1.0.0
```

Source code query with commit SHA

Run:

```
tanzu insight source get --commit COMMIT
```

Where:

- `COMMIT` specifies the commit
    - E.g., d7e4c27ba43250a4b7c46f030355c108aa73cc39
    - E.g., master/d7e4c27ba43250a4b7c46f030355c108aa73cc39

For example:

```
$ tanzu insight source get --commit b66668e
ID:             2
Repository:  kpack
Commit:  b66668e
Organization:   pivotal
Packages:
                1. cloud.google.com/go/kms@v1.0.0
                2. github.com/BurntSushi/toml@v3.1.1
                CVEs:
                        1. CVE-2021-30999 (Low)
                3. github.com/Microsoft/go-winio@v0.5.2
```

Example #3: 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

This topic describes how to add vulnerability scan reports or Software Bill of Materials (SBoM) files to the Supply Chain Security Tools - 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).

Note: 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`

Note: 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.

Command reference

The Tanzu Insight CLI plug-in is used to post data and query the Supply Chain Security Tools - Store database.

Synopsis

This CLI plug-in is used to post data and query the Supply Chain Security Tools - Store through its secure REST API. Source commit and image vulnerability reports can be uploaded using CycloneDX format (XML and JSON) and SPDX format (JSON). Source commit, image, package, and vulnerabilities can be queried and outputted in CycloneDX XML, JSON, and human-readable text formats.

Options

```
  -h, --help   help for tanzu insight
```

See also

- Tanzu insight config - Config commands
- Tanzu insight health - Checks if endpoint is reachable
- Tanzu insight image - Image commands
- Tanzu insight package - Package commands
- Tanzu insight source - Source commands
- Tanzu insight version - Display Tanzu Insight version
- Tanzu insight vulnerabilities - Vulnerabilities commands

Command reference

The Tanzu Insight CLI plug-in is used to post data and query the Supply Chain Security Tools - Store database.

Synopsis

This CLI plug-in is used to post data and query the Supply Chain Security Tools - Store through its secure REST API. Source commit and image vulnerability reports can be uploaded using CycloneDX format (XML and JSON) and SPDX format (JSON). Source commit, image, package, and vulnerabilities can be queried and outputted in CycloneDX XML, JSON, and human-readable text formats.

Options

```
  -h, --help   help for tanzu insight
```

See also

- Tanzu insight config - Config commands
- Tanzu insight health - Checks if endpoint is reachable
- Tanzu insight image - Image commands
- Tanzu insight package - Package commands
- Tanzu insight source - Source commands
- Tanzu insight version - Display Tanzu Insight version
- Tanzu insight vulnerabilities - Vulnerabilities commands

Tanzu insight config set-target

Tanzu insight config set-target

Set the metadata store endpoint.

Synopsis

Set the target endpoint for the metadata store.

```
tanzu insight config set-target <endpoint> [--ca-cert <ca certificate path to verify p
eer against>] [--access-token <kubernetes service account access token>] [flags]
```

Examples

```
tanzu insight config set-target https://localhost:8443 --ca-cert=/tmp/ca.crt --access-
token eyJhbGc...
```

Options

```
      --access-token string   Kubernetes access token. It is recommended to use the En
vironment Variable METADATA_STORE_ACCESS_TOKEN during the API calls, this will overrid
e access token flag. Note: using the the access-token flag stores the token on disk, t
he Environment Variable is retrieved at the time of the API call
      --ca-cert string        trusted ca certificate
  -h, --help                  help for set-target
```

See also

- Tanzu insight config - Config commands

Tanzu insight config

Config commands are as follows:

Options

```
  -h, --help   help for config
```

See also

- Tanzu insight - This CLI is used to post data and make queries to the metadata store.
- Tanzu insight config set-target - Set metadata store endpoint.

Tanzu insight health

Tanzu insight health

Checks if endpoint is reachable.

Synopsis

Checks if endpoint is reachable.

```
tanzu insight health [flags]
```

Examples

```
tanzu insight health
```

Options

```
  -h, --help   help for health
```

See also

- Tanzu insight

Tanzu insight image

Image commands are as follows:

Options

```
  -h, --help   help for image
```

See also

- Tanzu insight - This CLI is used to post data and query the metadata store.
- Tanzu insight image add - Add an image report.
- Tanzu insight image get - Get image by digest.
- Tanzu insight image packages - Get image packages.
- Tanzu insight image vulnerabilities - Get image vulnerabilities.

Tanzu insight image

Image commands are as follows:

Options

```
  -h, --help   help for image
```

See also

- Tanzu insight - This CLI is used to post data and query the metadata store.
- Tanzu insight image add - Add an image report.
- Tanzu insight image get - Get image by digest.
- Tanzu insight image packages - Get image packages.
- Tanzu insight image vulnerabilities - Get image vulnerabilities.

Tanzu insight image add

Add an image report from a report file:

```
tanzu insight image add [--cyclonedxtype <json|xml>] [--spdxtype json] --path <filepath>
```

If report type is not specified, it will be defaulted to `--cyclonedxtype=xml`

Examples

```
tanzu insight image add --cyclonedxtype json --path /path/to/file.json
```

Options

```
      --cyclonedxtype string   cyclonedx file type(xml/json, default: xml)
  -h, --help                   help for add
      --path string            path to file
      --spdxtype string        spdx file type(json)
```

See also

- Tanzu insight image - Image commands

Tanzu insight image get

Get image by digest.

Synopsis

Get image by digest.

```
tanzu insight image get --digest <image-digest> [--format <image-format>] [flags]
```

Examples

```
tanzu insight image get --digest sha256:a86859ac1946065d93df9ecb5cb7060adeeb0288fad610b1b659907 --format json
```

Options

```
  -d, --digest string   image digest
  -f, --format string   output format (default "text")
  -h, --help            help for get
```

See Also

- Tanzu insight image - Image commands

Tanzu insight image packages

Get image packages.

Synopsis

Get image packages.

```
tanzu insight image packages [--digest <image-digest>] [--name <name>] [--format <imag
e-format>] [flags]
```

Examples

```
tanzu insight image packages --digest sha256:a86859ac1946065d93df9ecb5cb7060adeeb0288f
ad610b1b659907 --format json
```

Options

```
  -d, --digest string   image digest
  -f, --format string   output format (default "text")
  -h, --help            help for packages
  -n, --name string     image name
```

See also

- Tanzu insight image - Image commands

Tanzu insight image vulnerabilities

Get image vulnerabilities:

```
tanzu insight image vulnerabilities --digest <image-digest> [--format <image-format>]
[flags]
```

Examples

```
tanzu insight image vulnerabilities --digest sha256:a86859ac1946065d93df9ecb5cb7060ade
eb0288fad610b1b659907 --format json
```

Options

```
  -d, --digest string   image digest
  -f, --format string   output format (default "text")
  -h, --help            help for vulnerabilities
```

See also

- Tanzu insight image - Image commands

Tanzu insight package

Package commands are as follows:

Options

```
  -h, --help   help for package
```

See also

- Tanzu insight - This CLI is used to post data and query the metadata store.
- Tanzu insight package get - Get package by name, version, and package manager.
- Tanzu insight package images - Get images that contain the given package by name.
- Tanzu insight package sources - Get sources that contain the given package by name.
- Tanzu insight package vulnerabilities - Get package vulnerabilities.

Tanzu insight package

Package commands are as follows:

Options

```
  -h, --help   help for package
```

See also

- Tanzu insight - This CLI is used to post data and query the metadata store.
- Tanzu insight package get - Get package by name, version, and package manager.
- Tanzu insight package images - Get images that contain the given package by name.
- Tanzu insight package sources - Get sources that contain the given package by name.
- Tanzu insight package vulnerabilities - Get package vulnerabilities.

Tanzu insight package get

Get package by name, version, and package manager.

Synopsis

Get package by name, version, and package manager.

```
tanzu insight package get --name <package name> --version <package version> --pkgmngr
Unknown [--format <format>] [flags]
```

Examples

```
tanzu insight package get --name client --version 1.0.0a --pkgmngr Unknown
```

Options

```
  -f, --format string    output format which can be in 'json' or 'text'. If not presen
t, defaults to text. (default "text")
```

```
  -h, --help             help for get
  -n, --name string      name of the package
  -p, --pkgmngr string   Package manager of the dependency. 'Unknown' is currently the
only supported value (default "Unknown")
  -v, --version string   version of the package
```

See also

- Tanzu insight package - Package commands

Tanzu insight Package Images

Get images that contain the given package by name.

Synopsis

Get images that contain the given package by name.

```
tanzu insight package images --name <package name> [flags]
```

Examples

```
tanzu insight package images --name client
```

Options

```
  -f, --format string   output format which can be in 'json' or 'text'. If not presen
t, defaults to text. (default "text")
  -h, --help            help for images
  -n, --name string     name of the package
```

See also

- Tanzu insight package - Package commands

Tanzu insight package sources

Get sources that contain the given package by name.

Synopsis

Get sources that contain the given package by name.

```
tanzu insight package sources --name <package name> [flags]
```

Examples

```
tanzu insight package sources --name client
```

Options

```
  -f, --format string   output format which can be in 'json' or 'text'. If not presen
t, defaults to text. (default "text")
  -h, --help            help for sources
  -n, --name string     name of the package
```

See also

- Tanzu insight package - Package commands

Tanzu insight Package Vulnerabilities

Get package vulnerabilities.

Synopsis

Get package vulnerabilities.

```
tanzu insight package vulnerabilities --name <package name> [flags]
```

Examples

```
tanzu insight package vulnerabilities --name client
```

Options

```
  -f, --format string   output format which can be in 'json' or 'text'. If not presen
t, defaults to text. (default "text")
  -h, --help            help for vulnerabilities
  -n, --name string     name of the package
```

See also

- Tanzu insight package - Package commands

Tanzu insight source

Source commands are as follows:

Options

```
  -h, --help   help for source
```

See also

- Tanzu insight - This CLI is used to post data and query the metadata store.
- Tanzu insight source add - Add a source report.
- Tanzu insight source get - Get sources by repository, commit, or organization.
- Tanzu insight source packages - Get source packages.
- Tanzu insight source vulnerabilities - Get source vulnerabilities.

Tanzu insight source

Source commands are as follows:

Options

```
  -h, --help   help for source
```

See also

- Tanzu insight - This CLI is used to post data and query the metadata store.
- Tanzu insight source add - Add a source report.
- Tanzu insight source get - Get sources by repository, commit, or organization.
- Tanzu insight source packages - Get source packages.
- Tanzu insight source vulnerabilities - Get source vulnerabilities.

Tanzu insight source add

Add a source report from a report file:

```
tanzu insight source add [--cyclonedxtype <json|xml>] [--spdxtype json] --path <filepa
th>
```

If report type is not specified, it will be defaulted to `--cyclonedxtype=xml`

Examples

```
tanzu insight source add --cyclonedxtype json --path  /path/to/file.json
```

Options

```
      --cyclonedxtype string   cyclonedx file type (xml/json, default: xml)
  -h, --help                   help for add
      --path string            path to file
      --spdxtype string        spdx file type (json)
```

See also

- Tanzu insight source - Source commands

Tanzu insight source get

Get sources by repository, commit, or organization.

Synopsis

Get sources by repository, commit, or organization.

```
tanzu insight source get --repo <repository> --commit <commit-hash> --org <organizatio
n-name> [--format <format>] [flags]
```

Examples

```
tanzu insight source get --repo github.com/org/example --commit b33dfee51 --org compan
y
```

Options

```
  -c, --commit string   commit hash
  -f, --format string   output format which can be in 'json' or 'text'. If not presen
t, defaults to text. (default "text")
  -h, --help            help for get
  -o, --org string      organization that owns the source
  -r, --repo string     repository name
```

See also

- Tanzu insight source - Source commands

Tanzu insight source packages

Get source packages.

Synopsis

Get source packages.

```
tanzu insight source packages [--commit <commit-hash>] [--repo <repo-url>] [--format <
format>] [flags]
```

Examples

```
tanzu insight sources packages --commit 0b1b659907 --format json
```

Options

```
  -c, --commit string   commit hash
  -f, --format string   output format (default "text")
  -h, --help            help for packages
  -r, --repo string     source repository url
```

See also

- Tanzu insight source - Source commands

Tanzu insight source vulnerabilities

Get source vulnerabilities.

Synopsis

Get source vulnerabilities. You can specify either commit or repo.

```
tanzu insight source vulnerabilities [--commit <commit-hash>] [--repo <repo-url>] [--f
ormat <format>] [flags]
```

Examples

```
tanzu insight sources vulnerabilities --commit eb55fc13
```

Options

```
  -c, --commit string   commit hash
  -f, --format string   output format which can be in 'json' or 'text'. If not presen
t, defaults to text. (default "text")
  -h, --help            help for vulnerabilities
  -r, --repo string     source repository url
```

See also

- Tanzu insight source - Source commands

Tanzu insight version

To display tanzu insight version:

```
tanzu insight version [flags]
```

Options

```
  -h, --help   help for version
```

See also

- Tanzu insight - This CLI is used to post data and query the metadata store.

Tanzu insight vulnerabilities

Vulnerabilities commands are as follows:

Options

```
  -h, --help   help for vulnerabilities
```

See also

- Tanzu insight - This CLI is used to post data and query the metadata store.
- Tanzu insight vulnerabilities get - Get vulnerability by CVE id.
- Tanzu insight vulnerabilities images - Get images with a given vulnerability.
- Tanzu insight vulnerabilities packages - Get packages with a given vulnerability.
- Tanzu insight vulnerabilities sources - Get sources with a given vulnerability.

Tanzu insight vulnerabilities

Vulnerabilities commands are as follows:

Options

```
  -h, --help   help for vulnerabilities
```

See also

- Tanzu insight - This CLI is used to post data and query the metadata store.
- Tanzu insight vulnerabilities get - Get vulnerability by CVE id.
- Tanzu insight vulnerabilities images - Get images with a given vulnerability.
- Tanzu insight vulnerabilities packages - Get packages with a given vulnerability.
- Tanzu insight vulnerabilities sources - Get sources with a given vulnerability.

Tanzu insight vulnerabilities get

Get vulnerability by CVE id.

Synopsis

Get vulnerability by CVE id.

```
tanzu insight vulnerabilities get --cveid <cve-id> [--format <format>] [flags]
```

Examples

```
tanzu insight vulnerabilities get --cveid CVE-123123-2021
```

Options

```
  -c, --cveid string    CVE id
  -f, --format string   output format which can be in 'json' or 'text'. If not presen
t, defaults to text. (default "text")
  -h, --help            help for get
```

See also

- Tanzu insight vulnerabilities - Vulnerabilities commands

Tanzu insight vulnerabilities images

Get images with a given vulnerability.

Synopsis

Get images with a given vulnerability.

```
tanzu insight vulnerabilities images --cveid <cve-id> [--format <format>] [flags]
```

Examples

```
tanzu insight vulnerabilities images --cveid CVE-123123-2021
```

Options

```
  -c, --cveid string    CVE id
  -f, --format string   output format which can be in 'json' or 'text'. If not presen
t, defaults to text. (default "text")
  -h, --help            help for images
```

See also

- Tanzu insight vulnerabilities - Vulnerabilities commands

Tanzu insight vulnerabilities packages

Get packages with a given vulnerability.

Synopsis

Get packages with a given vulnerability.

```
tanzu insight vulnerabilities packages --cveid <cve-id> [--format <format>] [flags]
```

Examples

```
tanzu insight vulnerabilities packages --cveid CVE-123123-2021
```

Options

```
  -c, --cveid string    CVE id
  -f, --format string   output format which can be in 'json' or 'text'. If not presen
t, defaults to text. (default "text")
  -h, --help            help for packages
```

See also

- Tanzu insight vulnerabilities - Vulnerabilities commands

Tanzu insight vulnerabilities sources

Get sources with a given vulnerability.

Synopsis

Get sources with a given vulnerability.

```
tanzu insight vulnerabilities sources --cveid <cve-id> [--format <format>] [flags]
```

Examples

```
tanzu insight vulnerabilities sources --cveid CVE-123123-2021
```

Options

```
  -c, --cveid string    CVE id
  -f, --format string   output format which can be in 'json' or 'text'. If not presen
t, defaults to text. (default "text")
  -h, --help            help for sources
```

See also

- Tanzu insight vulnerabilities - Vulnerabilities commands

Overview of Tanzu Application Platform authentication and authorization

Tanzu Application Platform (commonly known as TAP) v1.2 includes:

- Five new default roles to help you set up permissions for users and service accounts within a namespace on a cluster that runs one of the Tanzu Application Platform profiles.
- A Tanzu CLI RBAC (role-based access control) plug-in for role binding. For more information, see Bind a user or group to a default role.
- Documentation for integrating with your existing identity management solution.

Default roles

Three roles are for users:

- app-editor
- app-viewer
- app-operator

Two roles are for service accounts associated with the Tanzu Supply Chain:

- workload
- 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`. For an overview of the different roles and their permissions, see Role Descriptions.

Working with roles using the RBAC CLI plug-in

For more information about working with roles, see Bind a user or group to a default role.

Disclaimer

Tanzu Application Platform GUI does not make use of the described roles. Instead, it provides the user with view access for each cluster.

Overview of Tanzu Application Platform authentication and authorization

Tanzu Application Platform (commonly known as TAP) v1.2 includes:

- Five new default roles to help you set up permissions for users and service accounts within a namespace on a cluster that runs one of the Tanzu Application Platform profiles.
- A Tanzu CLI RBAC (role-based access control) plug-in for role binding. For more information, see Bind a user or group to a default role.
- Documentation for integrating with your existing identity management solution.

Default roles

Three roles are for users:

- app-editor
- app-viewer
- app-operator

Two roles are for service accounts associated with the Tanzu Supply Chain:

- workload
- 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`. For an overview of the different roles and their permissions, see Role Descriptions.

Working with roles using the RBAC CLI plug-in

For more information about working with roles, see Bind a user or group to a default role.

Disclaimer

Tanzu Application Platform GUI 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.

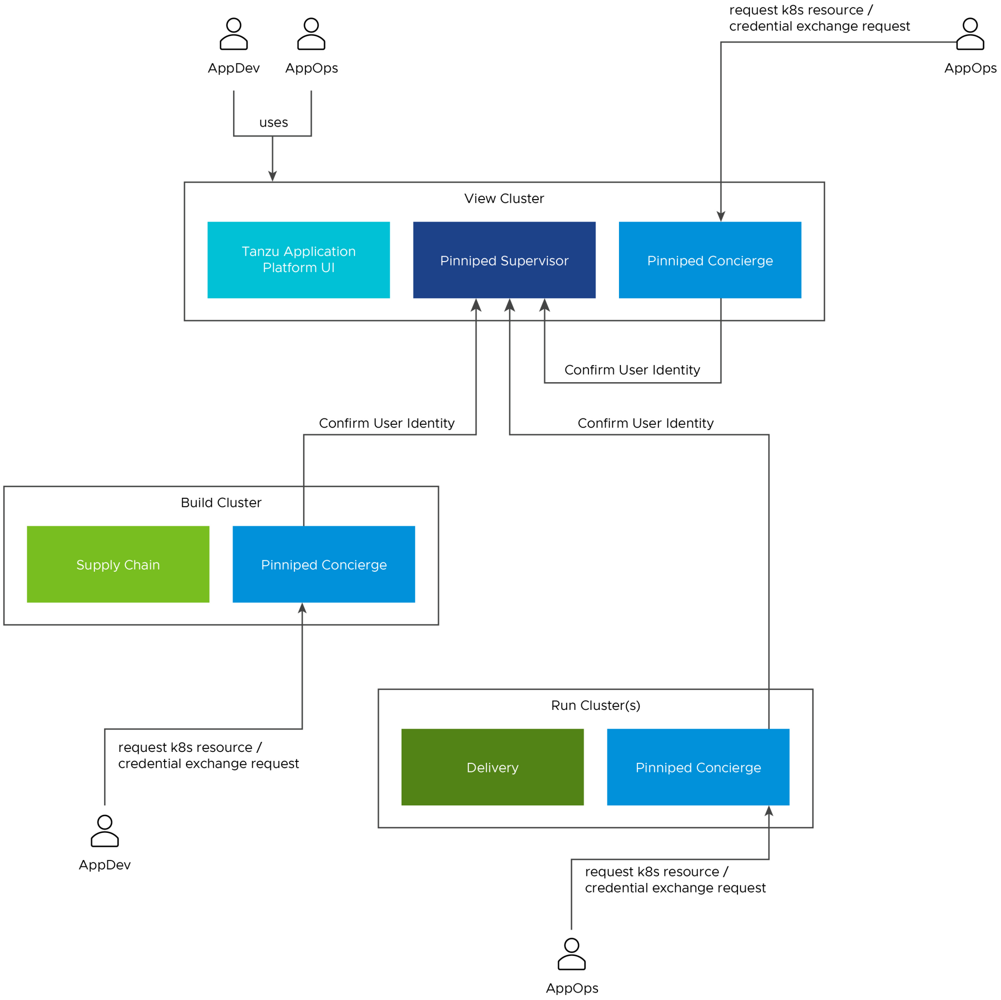
Environment planning

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
  routes:
  - 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.loadBalancer.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
  routes:
  - 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

Meet these prerequisites:

- Download and install the Azure CLI
- Download and install the Tanzu CLI
- Download and install the Tanzu CLI RBAC plug-in

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

Meet these prerequisites:

- Download and install the Tanzu CLI
- Download and install the Tanzu CLI RBAC plug-in
- 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"]
        allowPasswordGrant: false

      # Specify how claims are mapped to Kubernetes identities.
      claims:
        username: email
        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 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

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/concierge-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.

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 neeeded 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.

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 neeeded 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","clusterdeploymenttemplates","clusterimagetemplates","clusterruntemplates","clustersourcetemplates","clustersupplychains","clustertemplates"]
  verbs: ["get","list","watch"]
```

apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access

```
- apiGroups: ["carto.run"]
  resources: ["clusterconfigtemplates","clusterconfigtemplates","clusterdeliveries","clusterdeploymenttemplates","clusterimagetemplates","clusterruntemplates","clustersourcetemplates","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: ["eventing.knative.dev"]
  resources: ["brokers","triggers"]
  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"]
```

`apps.tanzu.vmware.com/aggregate-to-app-operator: "true"`

```
- apiGroups: ["eventing.knative.dev"]
  resources: ["brokers"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["sources.*"]
  resources: ["(many)"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
```

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 — Store

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

You can choose one of the following two approaches to bind a user or group to a default role:

- Use the Tanzu Application Platform RBAC CLI plug-in, which only supports binding Tanzu Application Platform (commonly known as TAP) default roles.
- Use Kubernetes role-based access control (RBAC) role binding.

VMware recommends that you use the Tanzu Application Platform RBAC CLI plug-in. This CLI plug-in simplifies the process by binding the cluster-scoped resource permissions at the same time as the namespace-scoped resource permissions, where applicable, for each default role. The following sections cover the Tanzu Application Platform RBAC CLI plug-in.

Prerequisites

1. Download the latest Tanzu CLI version.
2. Download the Tanzu Application Platform RBAC CLI plug-in `tar.gz` file from Tanzu Network.
3. Ensure you have admin access to the cluster.
4. Ensure you have configured an authentication solution for the cluster. You can use Pinniped or the authentication service native to your Kubernetes distribution.

Install the Tanzu Application Platform RBAC CLI plug-in

Follow these steps to install the Tanzu Application Platform RBAC CLI plug-in:

Caution: The Tanzu Application Platform RBAC CLI plug-in is currently in beta and is intended for evaluation and test purposes only.

1. Untar the `tar.gz` file:

   ```
   tar -zxvf NAME-OF-THE-TAR
   ```

2. Install the Tanzu Application Platform RBAC CLI plug-in locally on your operating system:

   macOS

```
tanzu plugin install rbac --local darwin-amd64
```

Linux

```
tanzu plugin install rbac --local linux-amd64
```

Windows

```
tanzu plugin install rbac --local windows-amd64
```

(Optional) Use a different kubeconfig location

You can use a different kubeconfig location by running:

```
tanzu rbac --kubeconfig PATH-OF-KUBECONFIG binding add --user USER --role ROLE --namespace NAMESPACE
```

Note

The environment variable `KUBECONFIG` is not implemented. You must use the `--kubeconfig` flag to enter a different location. Otherwise the default `~/.kube/config` is used.

For example:

```
$ tanzu rbac --kubeconfig /tmp/pinniped_kubeconfig.yaml binding add --user username@vmware.com --role app-editor --namespace user-ns
```

Add the specified user or group to a role

Add a user or group to a role by running:

```
tanzu rbac binding add --user USER --role ROLE --namespace NAMESPACE
```

```
tanzu rbac binding add --group GROUP --role ROLE --namespace NAMESPACE
```

For example:

```
$ tanzu rbac binding add --user username@vmware.com --role app-editor --namespace user-ns
```

Get a list of users and groups from a role

Get a list of users and groups from a role by running:

```
tanzu rbac binding get --role ROLE --namespace NAMESPACE
```

For example:

```
$ tanzu rbac binding get --role app-editor --namespace user-ns
```

Remove the specified user or group from a role

Remove a user or group from a role by running:

```
tanzu rbac binding delete --user USER --role ROLE --namespace NAMESPACE
tanzu rbac binding delete --group GROUP --role ROLE --namespace NAMESPACE
```

For example:

```
$ tanzu rbac binding delete --user username@vmware.com --role app-editor --namespace u
ser-ns
```

Error logs

Authorization error logs might include the following errors:

- Permission Denied:

  The current user does not have permissions to create or edit rolebinding objects. Use an admin account when using the RBAC CLI.

  ```
  Error: rolebindings.rbac.authorization.k8s.io "app-operator" is forbidden: User
  "<subject>" cannot get resource "rolebindings" in API group "rbac.authorizatio
  n.k8s.io" in the namespace "namespace"
  Usage:
  tanzu rbac binding add [flags]
  Flags:
  -g, --group string User Group
  -h, --help help for add
  -n, --namespace string Namespace
  -r, --role string Role
  -u, --user string User Name

  Global Flags:
  --kubeconfig string kubeconfig file
  ```

- Already Bound Error:

  Adding a subject, user or group, to a role that already has the subject produces the following error:

  ```
  Error: User ‘test-user’ is already bound to 'app-operator' role
  Usage:
  tanzu rbac binding add [flags]
  Flags:
  -g, --group string User Group
  -h, --help help for add
  -n, --namespace string Namespace
  -r, --role string Role
  -u, --user string User Name

  Global Flags:
  --kubeconfig string kubeconfig file
  ```

- Could Not Find Error:

  When removing a subject from a role, this error can occur in the following two scenarios:

  1. The rolebinding does not exist.
  2. The subject does not exist in the rolebinding.

  Ensure the rolebinding exists and that the subject name is correctly spelled.

  ```
  Error: Did not find User 'test-user' in RoleBinding 'app-operator'
  Usage:
  tanzu rbac binding delete [flags]

  Flags:
  -g, --group string User Group
  -h, --help help for delete
  -n, --namespace string Namespace
  -r, --role string Role
  -u, --user string User Name
  ```

```
Global Flags:
--kubeconfig string kubeconfig file
```

- Object Has Been Modified Error:

  This error is a race condition caused by running multiple RBAC CLI actions at the same time. Rerunning the RBAC CLI might fix the issue.

  ```
  Removed User 'test-user' from RoleBinding 'app-operator'
  Removed User 'test-user' from ClusterRoleBinding 'app-operator-cluster-access'
  Error: Operation cannot be fulfilled on rolebindings.rbac.authorization.k8s.io
  "app-operator": the object has been modified; please apply your changes to the
  latest version and try again
  Usage:
  tanzu rbac binding delete [flags]

  Flags:
  -g, --group string User Group
  -h, --help help for delete
  -n, --namespace string Namespace
  -r, --role string Role
  -u, --user string User Name
  ```

Troubleshooting

1. Get a list of permissions for a user or a group:

   ```
   export NAME=SUBJECT-NAME
   kubectl get rolebindings,clusterrolebindings -A -o json | jq -r ".items[] | sel
   ect(.subjects[]?.name == \"${NAME}\") | .roleRef.name" | xargs -n1 kubectl desc
   ribe clusterroles
   ```

2. Get a list of user or group for a specific role:

   ```
   tanzu rbac binding get --role ROLE --namespace NAMESPACE
   ```

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 users access to resources by using `rolebindings`. It can be done with the `tanzu rbac` plug-in. 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 Tanzu Application Platform authentication and authorization overview to get started.

Install

Defaults roles are released as part of Tanzu Application Platform. Alternatively, you can also Install default roles independently.

Note: The `tanzu rbac` CLI plug-in requires a separate installation.

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 Tanzu Application Platform authentication and authorization overview to get started.

Install

Defaults roles are released as part of Tanzu Application Platform. Alternatively, you can also Install default roles independently.

Note: The `tanzu rbac` CLI plug-in requires a separate installation.

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-name 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-name tap-auth.tanzu.vmware.com \
     --version 1.0.1 \
     --namespace tap-install
   ```

Application Accelerator Overview

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.

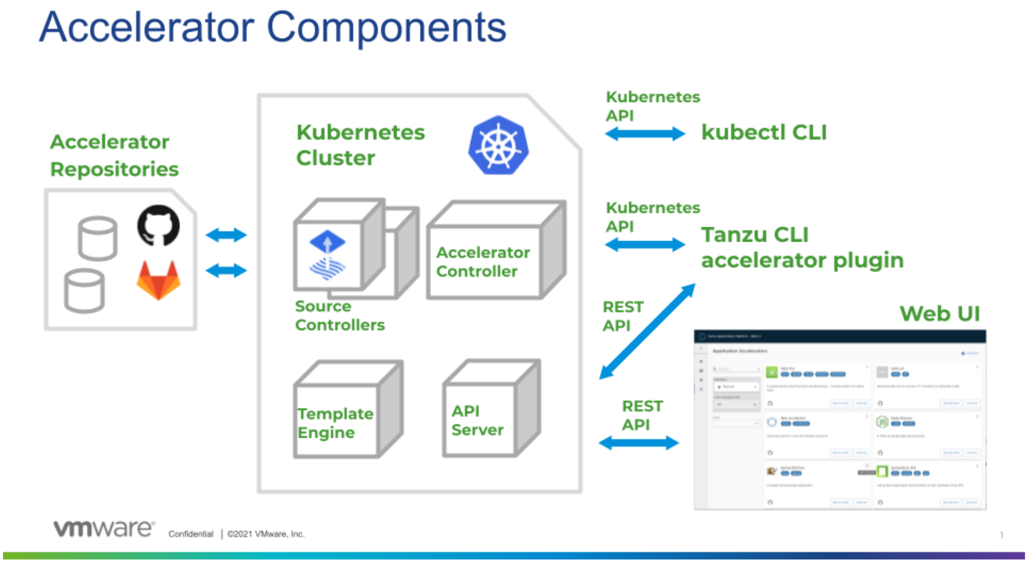
Enterprise Architects author and publish accelerator projects that provide developers and operators in their organization ready-made, enterprise-conformant code and configurations.

Published accelerators projects are maintained in Git repositories. You can then use Application Accelerator to create new projects based on those accelerator projects.

The Application Accelerator user interface(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 the Application Accelerator UI. 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 input fields for any input options.

Application Accelerator sends the input values to the Accelerator Engine for processing. The Engine then returns the project in a ZIP file. You can open the project in your favorite integrated development environment (IDE) to develop further.

Next steps

Learn more about:

- Creating Accelerators

Application Accelerator Overview

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.

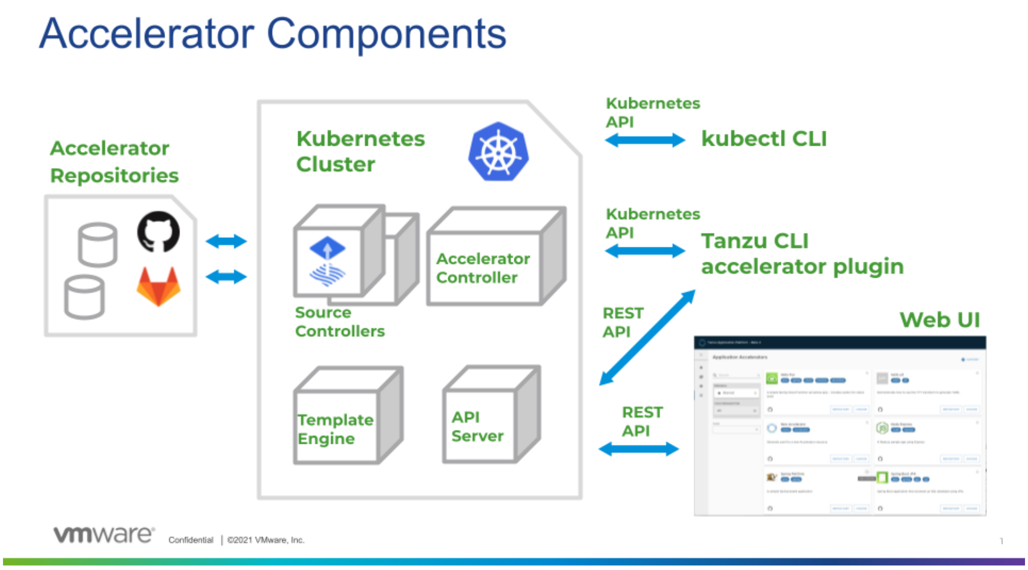
Enterprise Architects author and publish accelerator projects that provide developers and operators in their organization ready-made, enterprise-conformant code and configurations.

Published accelerators projects are maintained in Git repositories. You can then use Application Accelerator to create new projects based on those accelerator projects.

The Application Accelerator user interface(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 the Application Accelerator UI. 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 input fields for any input options.

Application Accelerator sends the input values to the Accelerator Engine for processing. The Engine then returns the project in a ZIP file. You can 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 cert-manager, Contour, and Flux CD Source Controller.
- Install Source Controller on the cluster. See Install Source Controller.

Configure properties and resource usage

When you install the Application Accelerator, you can configure the following optional properties:

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, defaults to `shared.ingress_domain`
tls.secret_n_ame	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	Should the system record each engine invocation when generating files for an accelerator?

VMware recommends that you do not override the defaults 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

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 ta
   p-install
   - Retrieving package versions for accelerator.apps.tanzu.vmware.com...
     NAME                               VERSION  RELEASED-AT
     accelerator.apps.tanzu.vmware.com  1.2.1    2022-06-22 13:00:00 -0400 EDT
   ```

2. (Optional) To make changes to the default installation settings, run:

   ```
   tanzu package available get accelerator.apps.tanzu.vmware.com/VERSION-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 accelerator.apps.tanzu.vmware.com/1.2.1 --values-sc
   hema --namespace tap-install
   ```

   For more information about values schema options, see the properties listed earlier.

3. Create an `app-accelerator-values.yaml` using the following example code:

   ```
   server:
     service_type: "LoadBalancer"
     watched_namespace: "accelerator-system"
   samples:
     include: true
   ```

   Edit the values if needed or leave the default values.

   For clusters that do not support the `LoadBalancer` service type, override the default value for `server.service_type`.

4. Install the package by running:

   ```
   tanzu package install app-accelerator -p accelerator.apps.tanzu.vmware.com -v V
   ERSION-NUMBER -n tap-install -f app-accelerator-values.yaml
   ```

   Where `VERSION-NUMBER` is the version included in the Tanzu Application Platform installation.

For example:

```
$ tanzu package install app-accelerator -p accelerator.apps.tanzu.vmware.com -v
1.2.1 -n tap-install -f 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'
```

5. 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:         1.2.1
STATUS:                  Reconcile succeeded
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`.

6. To see the IP address for the Application Accelerator API when the `server.service_type` is set to `LoadBalancer`, run the following command:

```
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` command.

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.

Overview

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, then you do not have to configure anything additional. Accelerators are created either using the Tanzu CLI or by applying a YAML manifest using `kubectl`.

Examples for creating accelerators

A minimal example for creating an accelerator

A minimal example could look like the following manifest:

hello-fun.yaml

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: hello-fun
spec:
  git:
    url: https://github.com/sample-accelerators/hello-fun
    ref:
      branch: main
```

This minimal example creates an accelerator named `hello-fun`. The `displayName`, `description`, `iconUrl`, and `tags` fields 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 https://github.com/sample-accelerators/hello-fun. For example:

accelerator.yaml

```
accelerator:
  displayName: Hello Fun
  description: A simple Spring Cloud Function serverless app
  iconUrl: https://raw.githubusercontent.com/simple-starters/icons/master/icon-cloud.p
ng
  tags:
  - java
  - spring
  - cloud
  - function
  - serverless
  - tanzu
...
```

To create this accelerator with `kubectl` run:

```
kubectl apply --namespace --accelerator-system --filename hello-fun.yaml
```

Or, you could use the Tanzu CLI and run:

```
tanzu accelerator create hello-fun --git-repo https://github.com/sample-accelerators/h
ello-fun --git-branch main
```

An example for creating an accelerator with customized properties

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

my-hello-fun.yaml

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: my-hello-fun
spec:
  displayName: My Hello Fun
  description: My own Spring Cloud Function serverless app
  iconUrl: https://raw.githubusercontent.com/sample-accelerators/icons/master/icon-tan
zu-light.png
  tags:
    - spring
    - cloud
    - function
    - serverless
  git:
    ignore: ".git/, bin/"
    url: https://github.com/sample-accelerators/hello-fun
    ref:
      branch: test
```

To create this accelerator with `kubectl` you could run:

```
kubectl apply --namespace --accelerator-system --filename my-hello-fun.yaml
```

Or, you could use the Tanzu CLI and run:

```
tanzu accelerator create my-hello-fun --git-repo https://github.com/sample-accelerator
s/hello-fun --git-branch main \
  --description "My own Spring Cloud Function serverless app" \
  --display-name "My Hello Fun" \
  --icon-url https://raw.githubusercontent.com/sample-accelerators/icons/master/icon-t
anzu-light.png \
  --tags "spring,cloud,function,serverless"
```

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

Using non-public repositories

For Git repositories that aren't accessible anonymously, you need to 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. 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. See fluxcd/source-controller HTTPS Certificate Authority
- For SSH repositories, the secret must contain identity, identity.pub and known_hosts fields. See fluxcd/source-controller SSH authentication.
- For Image repositories that aren't publicly available, an image pull secret may be provided. For more information, see 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.

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>
```

This creates a secret such as the following:

https-credentials.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: https-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 private repository
  git:
    url: <repository-URL>
    ref:
      branch: main
    secretRef:
      name: https-credentials
```

If you are using the Tanzu CLI, then 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, first create a secret with the HTTP credentials and the certificate.

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>
```

This creates a secret that looks like this:

https-ca-credentials.yaml

```
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 private repository
  git:
    url: <repository-URL>
    ref:
      branch: main
    secretRef:
      name: https-ca-credentials
```

If you are using the Tanzu CLI, then 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, first create a secret with the SSH credentials like 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
```

This example assumes that you don't have a key file already created. If you do, skip the `ssh-keygen` and `ssh-keyscan` steps and replace the values for the `kubectl create secret` command using the following:

- `--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>`

The secret that is created will look like this:

ssh-credentials.yaml

```
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 private repository
  git:
    url: <repository-URL>
    ref:
      branch: main
    secretRef:
      name: ssh-credentials
```

If you are using the Tanzu CLI, then 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. The easiest way to do that is to 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, first create a secret with the image-pull credentials:

```
create secret generic registry-credentials \
    --namespace accelerator-system \
    --from-literal=username=<user> \
    --from-literal=password=<password>
```

This creates a secret that looks like this:

https-credentials.yaml

```
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 private repository
  source:
    image: "registry.example.com/test/private-acc-src:latest"
    imagePullSecrets:
    - name: registry-credentials
```

If you are using the Tanzu CLI, then add the `--secret-ref` flag to your `tanzu accelerator create` command and provide the name of the secret for that flag.

Next steps

- Creating accelerators

Create accelerators

This topic tells you how to create an accelerator in Tanzu Application Platform GUI.

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 Application Platform GUI from a browser. For more information, see the “Tanzu Application Platform GUI” section in the most recent release for Tanzu Application Platform documentation
- kubectl v1.20 and later. The Kubernetes command line tool (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 for 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-559124_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.

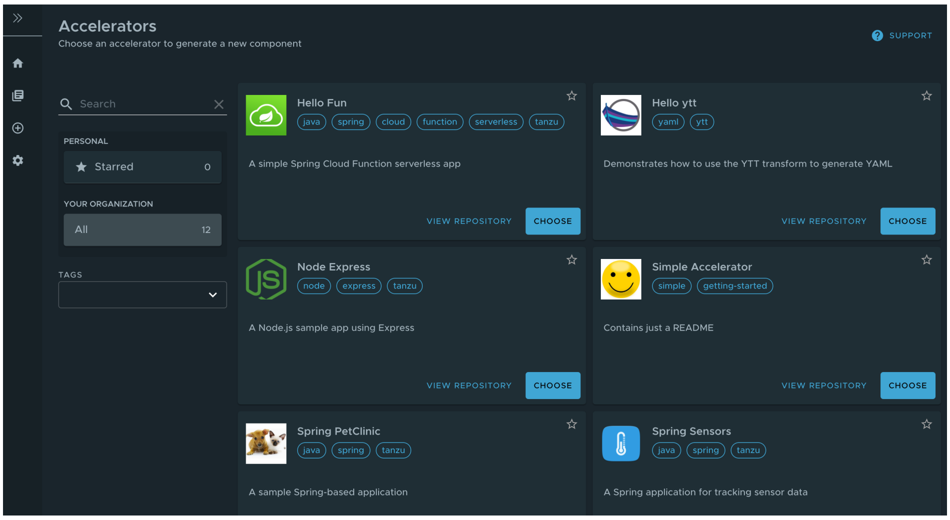
Publishing the new accelerator

1. To publish your new accelerator, run this command in your terminal:

   ```
   tanzu accelerator create simple --git-repository YOUR-GIT-REPOSITORY-URL --git-branch YOUR-GIT-BRANCH
   ```

   Where:

   - `YOUR-GIT-REPOSITORY-URL` is the URL for your Git repository.
   - `YOUR-GIT-BRANCH` is the name of the branch where you pushed the new `accelerator.yaml` file.
2. Refresh Tanzu Application Platform GUI to reveal the newly published accelerator. It might take a few seconds for Tanzu Application Platform GUI to refresh the catalog and add an entry for your new accelerator.



A simple Spring Cloud Function serverless app

Demonstrates how to use the YTT transform to generate YAML

A Node.js sample app using Express

Contains just a README

A sample Spring-based application

A Spring application for tracking sensor data

An alternative to using the Tanzu CLI is 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 may be imported to accelerators using an `import` entry and the transforms from the fragment may 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 the sample-accelerators/fragments Git repository for the content of these fragments.

To discover what fragments are available to use, you can run the following command:

```
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 + ''''"
    - 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.

To deploy new fragments to the accelerator system you can use the new `tanzu accelerator fragment create` CLI command or you can 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:
      branch: tap-1.2
    url: https://github.com/sample-accelerators/fragments.git
    subPath: java-version
```

To create the fragment (we can save the above manifest in a `java-version.yaml` file) and use:

```
tanzu accelerator apply -f ./java-version.yaml
```

The `accelerator apply` command can be used to apply both Accelerator and Fragment resources.

To avoid having to create a separate manifest file, you can use the following command instead:

```
tanzu accelerator fragment create java-version \
  --git-repo https://github.com/sample-accelerators/fragments.git \
  --git-branch main \
  --git-tag tap-1.2 \
  --git-sub-path java-version
```

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 acelerator 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 invoke 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 Application Platform GUI.

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 Application Platform GUI from a browser. For more information, see the “Tanzu Application Platform GUI” section in the most recent release for Tanzu Application Platform documentation
- kubectl v1.20 and later. The Kubernetes command line tool (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 for 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.

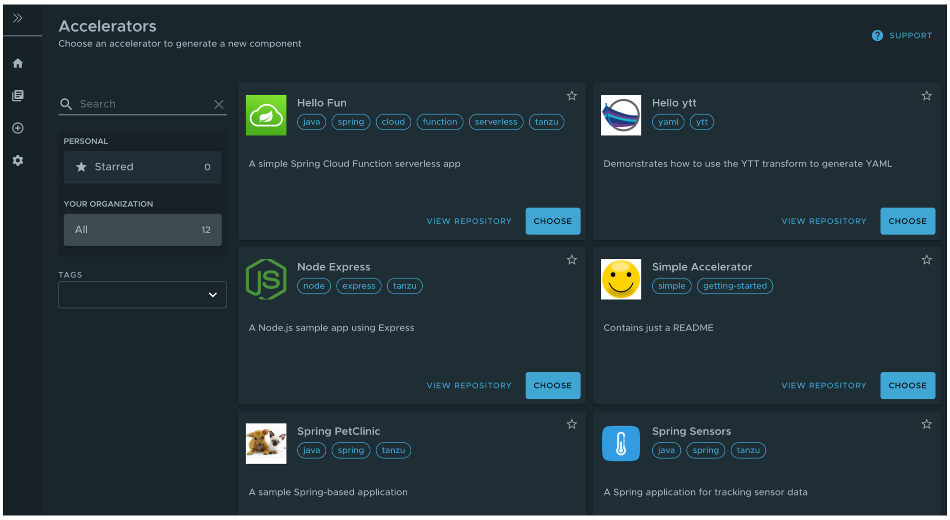
Publishing the new accelerator

1. To publish your new accelerator, run this command in your terminal:

   ```
   tanzu accelerator create simple --git-repository YOUR-GIT-REPOSITORY-URL --git-
   branch YOUR-GIT-BRANCH
   ```

   Where:

   - `YOUR-GIT-REPOSITORY-URL` is the URL for your Git repository.
   - `YOUR-GIT-BRANCH` is the name of the branch where you pushed the new `accelerator.yaml` file.
2. Refresh Tanzu Application Platform GUI to reveal the newly published accelerator. It might take a few seconds for Tanzu Application Platform GUI to refresh the catalog and add an entry for your new accelerator.



Accelerators

Choose an accelerator to generate a new component

Hello Fun — java, spring, cloud, function, serverless, tanzu — A simple Spring Cloud Function serverless app

Hello ytt — yaml, ytt — Demonstrates how to use the YTT transform to generate YAML

Node Express — node, express, tanzu — A Node.js sample app using Express

Simple Accelerator — simple, getting-started — Contains just a README

Spring PetClinic — java, spring, tanzu — A sample Spring-based application

Spring Sensors — java, spring, tanzu — A Spring application for tracking sensor data

An alternative to using the Tanzu CLI is 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 may be imported to accelerators using an `import` entry and the transforms from the fragment may 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 the sample-accelerators/fragments Git repository for the content of these fragments.

To discover what fragments are available to use, you can run the following command:

```
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 + ''''"
    - 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.

To deploy new fragments to the accelerator system you can use the new `tanzu accelerator fragment create` CLI command or you can 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:
      branch: tap-1.2
    url: https://github.com/sample-accelerators/fragments.git
    subPath: java-version
```

To create the fragment (we can save the above manifest in a `java-version.yaml` file) and use:

```
tanzu accelerator apply -f ./java-version.yaml
```

The `accelerator apply` command can be used to apply both Accelerator and Fragment resources.

To avoid having to create a separate manifest file, you can use the following command instead:

```
tanzu accelerator fragment create java-version \
  --git-repo https://github.com/sample-accelerators/fragments.git \
  --git-branch main \
  --git-tag tap-1.2 \
  --git-sub-path java-version
```

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 invoke 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 Appplication 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/simple-starters/icons/master/icon-cloud.p
ng
  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`.
- **dataType**: Data types that work with the UI are `string`, `boolean`, `number`, and arrays of those, as in `[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 a prerequisite. See **required** for that purpose.

  This regex is used by 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 that allows line breaks.
  - `checkbox`: Ideal for Boolean values or multivalue 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 should not be used to trigger several other options unfolding, one for each value. Instead, use several single-valued options:

Instead of

```
options:
  - name: toppings
```

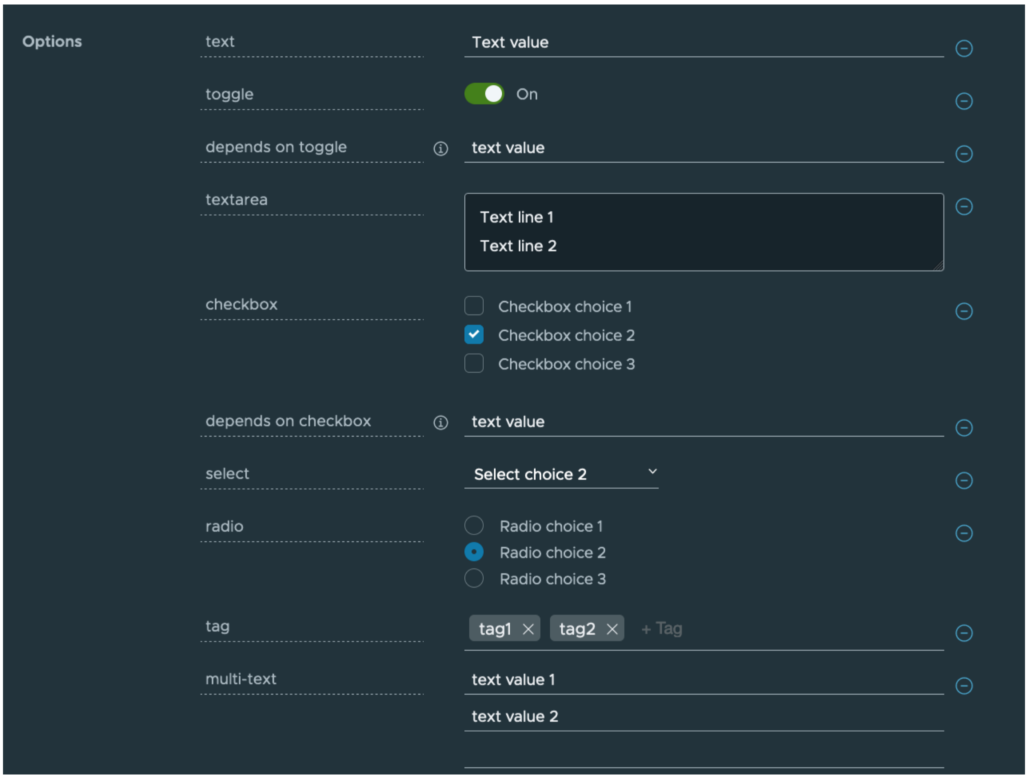
```
    dataType: [string]
    inputType: checkbox
    choices:
      - value: vegetables
        text: Vegetables
      - value: meat
        text: Meat
        ...
  - name: vegType
    dependsOn:
      name: toppings
      value: [vegetables] # or vegetables, this won't do what you want either
  - 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 screenshot and `accelerator.yaml` file snippet that follows demonstrates each `inputType`. You can also see the GitHub sample demo-input-types.



Options

text: Text value

toggle: On

depends on toggle: text value

textarea:
Text line 1
Text line 2

checkbox:
□ Checkbox choice 1
☑ Checkbox choice 2
□ Checkbox choice 3

depends on checkbox: text value

select: Select choice 2

radio:
○ Radio choice 1
● Radio choice 2
○ Radio choice 3

tag: tag1 × tag2 × + Tag

multi-text:
text value 1
text value 2

```
accelerator:
  displayName: Demo Input Types
  description: "Accelerator with options for each inputType"
  iconUrl: https://raw.githubusercontent.com/sample-accelerators/icons/master/icon-tan
zu-light.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>`.

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.

As such, 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. Unfortunately, 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/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 swiss army knife 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 in a “logical way” combining `Merge` and `Chain` under the hood.

For example:

```
include: ['**/*.txt', '**.md']
chain:
- type: ReplaceText
  ...
```

Is the same as:

```
type: Chain
transformations:
- type: Include
  patterns: ['**/*.txt', '**.md']
- type: Chain
  trasformations:
```

```
  - 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 this all sounds confusing, fortunately there is a basic guideline for understanding and predicting the effect of a deactivated transform in the context of your accelerator definition. Namely, if a transform's condition evaluates to false, pretend it isn't there. In other words, 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'
```

The `JsonPrettyPrint` transform type is purely hypothetical. There *could* be such a transform, but VMware doesn’t currently provide it.

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.

This behavior accords with our guideline. For example, if **transform A**’s condtion is `false`, it behaves as if **transform A** wasn’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**. This is precisely what would happen were **transform A** not there.

A small gotcha with using conditionals in merge transforms

As mentioned earlier, it is a useful pattern to use merges with overlapping contents. Yet you must be careful using this in combination with conditional transforms.

For example:

```
engine:
  merge:
  - include: ["**/*"]
  - include: ["**/pom.xml"]
    chain:
    - type: ReplaceText
      subsitutions: ...
```

Now add a little twist. Say you only wanted to include pom files if the user selects a `useMaven` option. You might be tempted to add a ‘condition’ to **transform B** to deactivate it when that option isn’t selected:

```
engine:
  merge:
  - include: "**/*"
  - condition: '#useMaven'
    include: ["**/pom.xml"]
    chain:
    - type: ReplaceText
      subsitutions: ...
```

However, this doesn't do what you might expect. The final result *still* contains `pom.xml` files. To understand why, recall the guideline for deactivated transforms: If a transform is deactivated, pretend it isn't there. 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 pitfall. 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. Again, 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. To this end, 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 also comes with some convenient built-in support for conflict resolution. It automatically selects the `UseLast` strategy if none is explicitly supplied. So in practice, you rarely, if ever, need to explicitly 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.

As an accelerator author, better to avoid relying on the file order produced from reading directly from a file system. So it’s better to avoid doing something like the preceding `RewritePath` example, *unless* you do not care about the ordering of the various sections of the produced `README.md` file.

If you do care and want to control the order explicitly, you use the fact that `Merge` processes its children in order and reflects this order in the resulting output set of files. 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` definitely comes before `DEPLOYMENT.md` from the second child of `merge`. So you can control the merge order directly by changing the order of the merge children.

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.

To learn more, read the following more detailed documents:

- An exhaustive Reference of all built-in transform types
- A sample, commented accelerator.yaml to learn from a concrete example

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 but with similar aspects, which requires 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
- contains an `accelerator.yaml` descriptor, with options declarations and a root Transform.

The only differences reside in the way they are declared to the system — they are filed as **Fragments** custom resources — and the way they deal with files: because they deal with their own files and files from the accelerator using them, they typically use dedicated conflict resolution strategies (more on that later).

Fragments may be thought of as “functions” in programming languages: once defined and referenced, they may be “called” at various points in the main accelerator. The composition feature is designed with ease-of-use and “common use case first” in mind, so these “functions” are typically called with as little noise as possible, you can also call them with 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,
- and the, `InvokeFragment` Transform, to be used alongside any other Transform.

| The `imports` section explained

To be usable in composition, a fragment MUST be *imported* in 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 (this is why 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 above, 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 of the metadata about options (type, default value, description, choices if applicable, *etc.*) is coming along when being imported.

As a consequence 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 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 straight forward: 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: "*"

    - name: yet-another-fragment
      expose:
        - name: someOption

        - name: someOtherOption
          as: aDifferentName
engine:
  ...
```

As shown earlier, the `imports` section calls a list of fragments to import and by default all of their options become options 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 may 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) may be exposed with their original name.
- The default value for `expose` is `[{name: "*"}]`, *i.e.* by default expose all options with their original name.
- As soon as a single remap rule appears, the default is overridden (*i.e.* to override some names AND expose the others unchanged, the `*` must be explicitly re-added)

| Using `dependsOn` in the `imports` section<

Lastly, as a convenience for 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 expample:

````
...
engine:
  ...
    type: InvokeFragment
    condition: "#deploymentType == 'workload'"
    reference: tap-initialize```
````

Application Accelerator transforms reference

This topic provides 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.

See also

- Conflict Resolution

Application Accelerator transforms reference

This topic provides 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.

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>
  - ...
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. At this point of the chain, 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`. So 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 transform 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. Put another way, the 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.
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(<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. We've used a convention that mimics the actual behavior for clarity, but the following applies **T1** and **T2** on all `.yaml` files even though we VMware has placed 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 YAML text. It's therefore 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`.

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: ['*.yaml']      # Here comes a third Combo, used as the 2nd child inside
the first
    chain:
      - type: T3
      - type: T4
```

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.

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** o **T2** o **T1**.

An empty chain acts as function identity.

Syntax reference

```
type: Chain
transformations:
  - <transform>
  - <transform>
  - <transform>
  - ...
condition: <SpEL expression>
```

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.

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.

</a/>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

Let's start with 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 like so

```
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.

You can pretend, for some reason, that 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 such as this:

```
    type: InvokeFragment
    reference: my-fragment
    let:
      - name: indentationLevel
        value: '2 * #someOption'
```

Now for some other crazy example to better explain the interactions. If the invocation in the accelerator looked like this:

```
engine:
  merge:
    - include: ["..."]
    - ...
    - chain:
        - include: ["**/README.md"]
        - type: InvokeFragment
          reference: my-fragment
```

Then there is absolutely 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

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"
```

Doing the same 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"
```

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)"
```

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')"
```

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

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
```

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
```

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.

Currently, only Java-related recipes are supported. The engine leverages version `7.21.3` of Open Rewrite and parses Java files using the grammar for Java 11.

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 vs. 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

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?

Such 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.

Syntax reference

```
type: Combo      # often omitted
merge:
  - <transform>
chain:
  - <transform>
  - ...
onConflict: <conflict resolution>  # defaults to 'UseLast'
```

```
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.

See also

- Combo
- UniquePath

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` are 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:
        first: First Option
        second: Second Option
      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`.

Accelerator custom resource definition

This topic tells you about the Application Accelerator custom resource definition.

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.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 check for repository updates. If not provided it defaults to 10 min. There is an additional refresh interval (currently 10s) involved before accelerators may appear in the UI. There could be a 10s delay before changes are reflected in the UI.*	Optional (**)
git.ref	Git reference to checkout and monitor for changes, defaults to 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 (**)
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.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 check for repository updates. If not provided it defaults to 10 min.	Optional (**)
git.ref	Git reference to checkout and monitor for changes, defaults to 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 (**)
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

* Any optional fields marked with an asterisk (*) are populated from a field 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 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, you must interact with the `acc-server`. For more information, see How to expose this server follow the instructions.

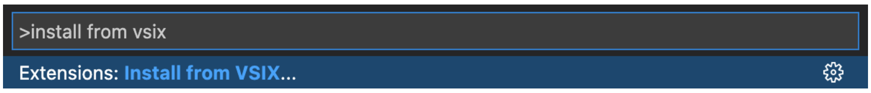
Installation

Use the following steps to install the Application Accelerator Visual Studio extension:

1. Sign in to VMware Tanzu Network and download the “Tanzu App Accelerator Extension for Visual Studio Code” file from the product page for VMware Tanzu Application Platform.
2. Open VS Code.

   Option 1:

   1. From the Command Palette (cmd + shift + P), run “Extensions: Install from VSIX…”.
   2. Select the extension file **tanzu-app-accelerator-0.1.2.vsix**.

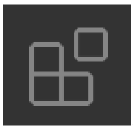
      

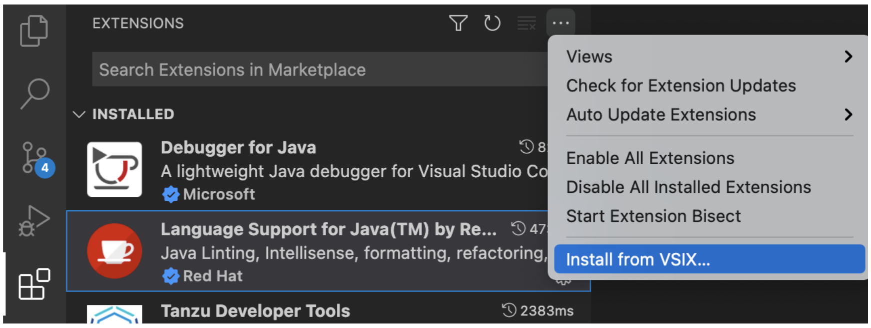
      >install from vsix

      Extensions: Install from VSIX...

      .

   Option 2:

   1. Select the **Extensions** tab: .
   2. Select `Install from VSIX…` from the overflow menu.



EXTENSIONS

Search Extensions in Marketplace

INSTALLED

Debugger for Java
A lightweight Java debugger for Visual Studio Co
Microsoft

Language Support for Java(TM) by Re...
Java Linting, Intellisense, formatting, refactoring,
Red Hat

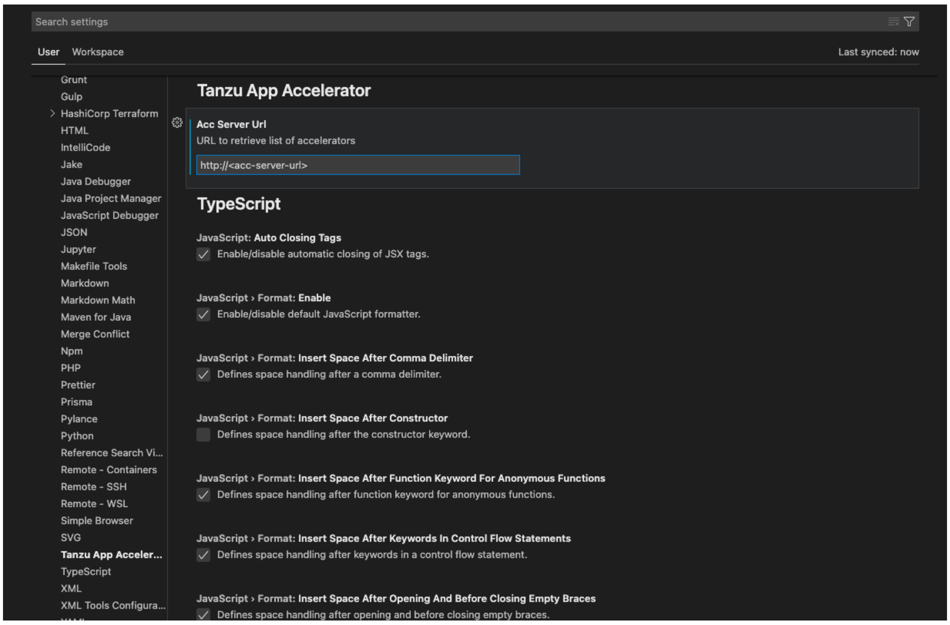
Tanzu Developer Tools 2383ms

.

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 `Acc Server Url`.
3. Add the `acc-server` URL.



Search settings

User Workspace Last synced: now

Tanzu App Accelerator

Acc Server Url
URL to retrieve list of accelerators
http://<acc-server-url>

TypeScript

JavaScript: **Auto Closing Tags**
☑ Enable/disable automatic closing of JSX tags.

JavaScript › Format: **Enable**
☑ Enable/disable default JavaScript formatter.

JavaScript › Format: **Insert Space After Comma Delimiter**
☑ Defines space handling after a comma delimiter.

JavaScript › Format: **Insert Space After Constructor**
□ Defines space handling after the constructor keyword.

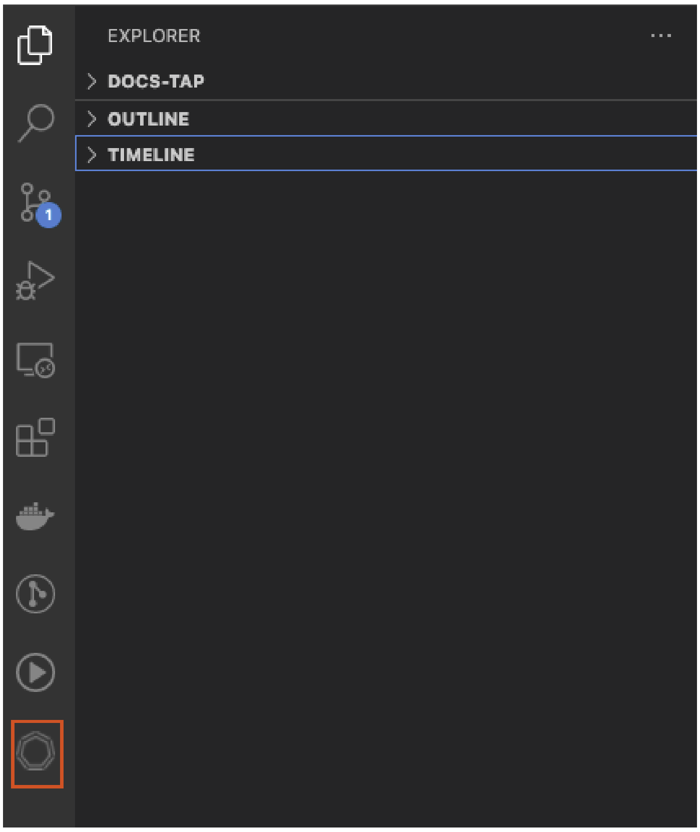
JavaScript › Format: **Insert Space After Function Keyword For Anonymous Functions**
☑ Defines space handling after function keyword for anonymous functions.

JavaScript › Format: **Insert Space After Keywords In Control Flow Statements**
☑ Defines space handling after keywords in a control flow statement.

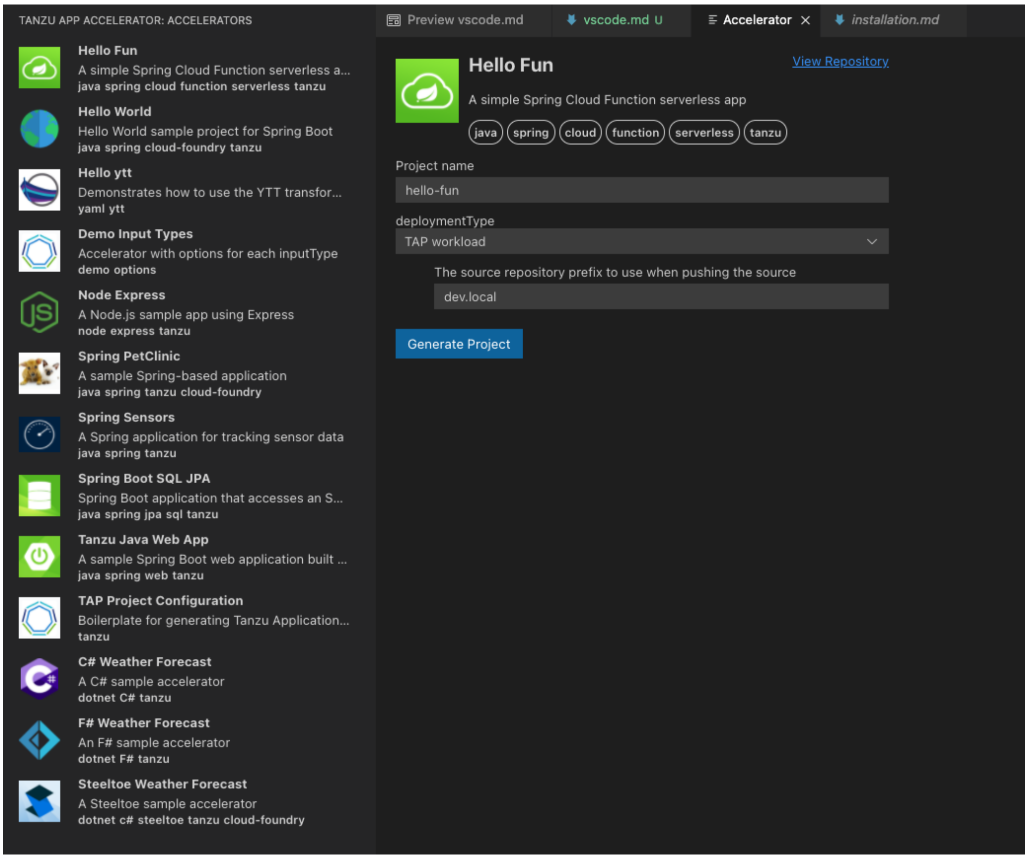
JavaScript › Format: **Insert Space After Opening And Before Closing Empty Braces**
☑ Defines space handling after opening and before closing empty braces.

Using the extension

After adding the `acc-server` URL you should can explore the defined accelerators accessing the new added icon:



Choose any of the defined accelerators, fill the options and click the `generate project`



TANZU APP ACCELERATOR: ACCELERATORS

Hello Fun
A simple Spring Cloud Function serverless a...
java spring cloud function serverless tanzu

Hello World
Hello World sample project for Spring Boot
java spring cloud-foundry tanzu

Hello ytt
Demonstrates how to use the YTT transfor...
yaml ytt

Demo Input Types
Accelerator with options for each inputType
demo options

Node Express
A Node.js sample app using Express
node express tanzu

Spring PetClinic
A sample Spring-based application
java spring tanzu cloud-foundry

Spring Sensors
A Spring application for tracking sensor data
java spring tanzu

Spring Boot SQL JPA
Spring Boot application that accesses an S...
java spring jpa sql tanzu

Tanzu Java Web App
A sample Spring Boot web application built ...
java spring web tanzu

TAP Project Configuration
Boilerplate for generating Tanzu Application...
tanzu

C# Weather Forecast
A C# sample accelerator
dotnet C# tanzu

F# Weather Forecast
An F# sample accelerator
dotnet F# tanzu

Steeltoe Weather Forecast
A Steeltoe sample accelerator
dotnet c# steeltoe tanzu cloud-foundry

Hello Fun

A simple Spring Cloud Function serverless app

java spring cloud function serverless tanzu

Project name
hello-fun

deploymentType
TAP workload

The source repository prefix to use when pushing the source
dev.local

Generate Project

Troubleshoot Application Accelerator

This topic provides troubleshooting steps for development, accelerator authorship, and operations issues in Application Accelerator.

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.SpelEvaluationException: EL1007E: Property or field 'test' cannot be found on null
  ```

  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

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-system
  ```

- 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/sample-accelerators/hello-fu
n'',
      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/sample-accelerators/hello-fun', erro
r: 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/sample-accelerators/hello-fu
nk'',
      error: authentication required'
    ready: false
    url: ""
  conditions:
  - lastTransitionTime: "2021-11-18T21:09:52Z"
    message: |-
      failed to resolve GitRepository
      unable to clone 'https://github.com/sample-accelerators/hello-funk', erro
r: 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/sample-accelerators/hello-funk', error: aut
hentication 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 accelerator-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 accelerator-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":"Accelerator","metadata":{"annotations":{},"name":"more-fun","namespace":"accelerator-system"},"spec":{"description":"Test-image","source":{"image":"trisberg/more-fun-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/invocations
     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 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. Developers can use plug-ins to integrate their existing polyglot apps.

Multi-cloud 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

There are two modes of deployment for registering apps with the Application Live View running on a Kubernetes cluster:

- **Connector**: A component responsible for discovering multiple apps running on a Kubernetes cluster.

- **Sidecar**: A proxy component that is started alongside a single app inside the same pod, running on a Kubernetes cluster.

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. Developers can use plug-ins to integrate their existing polyglot apps.

Multi-cloud 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

There are two modes of deployment for registering apps with the Application Live View running on a Kubernetes cluster:

- **Connector**: A component responsible for discovering multiple apps running on a Kubernetes cluster.
- **Sidecar**: A proxy component that is started alongside a single app inside the same pod, running on a Kubernetes cluster.

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.

Application Live View installs three packages for `full`, `light`, and `iterate` profiles:

- For the `view` profile, Application Live View installs Application Live View back-end package (`backend.appliveview.tanzu.vmware.com`). This installs the Application Live View back-end component with Tanzu Application Platform GUI in `app-live-view` namespace.

- For the `run` profile, Application Live View installs Application Live View connector package (`connector.appliveview.tanzu.vmware.com`). This installs the Application Live View connector component as DaemonSet in `app-live-view-connector` namespace.
- For the `build` profile, Application Live View installs Application Live View Conventions package (`conventions.appliveview.tanzu.vmware.com`). This installs the Application Live View Convention Service in `app-live-view-conventions` namespace.

> **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.

In addition, install Cartographer Conventions which is bundled with Supply Chain Choreographer as of the v0.4.0 release. 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.

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  1.2.0-build.2  2022-06-01T00:00:10Z
   ```

2. (Optional) Change the default installation settings 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. For example, `1.2.0-build.2`.

   For example:

```
$ tanzu package available get backend.appliview.tanzu.vmware.com/1.2.0-build.
2 --values-schema --namespace tap-install
```

For more information about values schema options, see the properties listed earlier.

3. Create `app-live-view-backend-values.yaml` with the following details:

   For a SINGLE-CLUSTER environment, the Application Live View back end is exposed through the Kubernetes cluster service.

   Note: If it is a Tanzu Application Platform profile installation and top-level key `shared.ingress_domain` is set in the `tap-values.yml`, the back end is automatically exposed through the ingress. The following configurations are applied by default:

   ```
   ingressEnabled: true
   ingress_domain: ${shared.ingress_domain}
   ```

   For a MULTI-CLUSTER environment, Tanzu Application Platform uses the `shared.ingress_domain` by default. You can override this setting with the following values:

   ```
   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 `appliview` subdomain is prepended to the value provided.

   To configure TLS certificate delegation information for the domain, add the following values to `app-live-view-backend-values.yaml`:

   ```
   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.

4. Install the Application Live View back-end package by running:

   ```
   tanzu package install appliview -p backend.appliview.tanzu.vmware.com -v VE
   RSION-NUMBER -n tap-install -f app-live-view-backend-values.yaml
   ```

   Where `VERSION-NUMBER` is the version of the package listed.

   For example:

   ```
   $ tanzu package install appliview -p backend.appliview.tanzu.vmware.com -v
   1.2.0-build.2 -n tap-install -f app-live-view-backend-values.yaml
   - Installing package 'backend.appliview.tanzu.vmware.com'
   | Getting namespace 'tap-install'
   | Getting package metadata for 'backend.appliview.tanzu.vmware.com'
   | Creating service account 'appliview-tap-install-sa'
   | Creating cluster admin role 'appliview-tap-install-cluster-role'
   | Creating cluster role binding 'appliview-tap-install-cluster-rolebinding'
   | Creating package resource
   | Package install status: Reconciling

   Added installed package 'appliview' in namespace 'tap-install'
   ```

   Note: The Application Live View back-end component is deployed in `app-live-view` namespace by default.

5. 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:         1.2.0-build.2
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  1.2.0-build.2  2022-06-01T00:00:10Z
   ```

2. (Optional) Change the default installation settings 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, `1.2.0-build.2`.

   For example:

   ```
   $ tanzu package available get connector.appliveview.tanzu.vmware.com/1.2.0-buil
   d.2 --values-schema --namespace tap-install
   ```

   For more information about values schema options, see the properties listed earlier.

3. Create `app-live-view-connector-values.yaml` with the following details:

   For SINGLE-CLUSTER environment, the Application Live View connector connects to the `cluster-local` Application Live View back end to register the applications.

   Note: If it is a Tanzu Application Platform profile installation and top-level key `shared.ingress_domain` is set in the `tap-values.yml`, the Application Live View connector and Application Live View back end are configured to communicate through ingress. The the Application Live View connector uses the `shared.ingress_domain` to reach the back end.

   When using `shared.ingress_domain`, unless you enable TLS in the appliveview (Application Live View back end), set:

   ```
   backend:
       sslDisabled: false
   ```

   For a MULTI-CLUSTER environment, use the following values:

   ```
   backend:
       sslDisabled: false
   ```

```
    host: appliview.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 `appliview` subdomain to the provided value.

Note: The `backend.sslDisabled` is set to `false` by default. If TLS is not enabled for the `INGRESS-DOMAIN` in the Application Live View back end, set the `backend.sslDisabled` to `true`.

Note: If it is a Tanzu Application Platform profile installation and top-level key `shared.ingress_domain` is set in the `tap-values.yml`, the Application Live View connector is automatically configured to use the `shared.ingress_domain` to reach the Application Live View back end.

You can edit the values to suit your project needs or leave the default values as is.

Note: 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. Install the Application Live View connector package by running:

   ```
   tanzu package install appliview-connector -p connector.appliview.tanzu.vmwa
   re.com -v VERSION-NUMBER -n tap-install -f app-live-view-connector-values.yaml
   ```

   Where `VERSION-NUMBER` is the version of the package listed. For example, `1.2.0-build.2`.

   For example:

   ```
   $ tanzu package install appliview-connector -p connector.appliview.tanzu.vm
   ware.com -v 1.2.0-build.2 -n tap-install -f app-live-view-connector-values.yaml
   | Installing package 'connector.appliview.tanzu.vmware.com'
   | Getting namespace 'tap-install'
   | Getting package metadata for 'connector.appliview.tanzu.vmware.com'
   | Creating service account 'appliview-connector-tap-install-sa'
   | Creating cluster admin role 'appliview-connector-tap-install-cluster-role'
   | Creating cluster role binding 'appliview-connector-tap-install-cluster-role
   binding'
   - Creating package resource
   / Package install status: Reconciling

   Added installed package 'appliview-connector' in namespace 'tap-install'
   ```

   Note: Each cluster installs the connector as a DaemonSet. 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.

5. Verify the `Application Live View connector` package installation by running:

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

   For example:

   ```
   tanzu package installed get appliview-connector -n tap-install
   5s
   | Retrieving installation details for appliview-connector...
   NAME:                    appliview-connector
   PACKAGE-NAME:            connector.appliview.tanzu.vmware.com
   PACKAGE-VERSION:         1.2.0-build.2
   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  1.2.0-build.2  2022-06-01T00:00:00Z
   ```

2. Install the Application Live View Conventions package by running:

   ```
   tanzu package install appliveview-conventions -p conventions.appliveview.tanzu.
   vmware.com -v VERSION-NUMBER -n tap-install
   ```

   Where `VERSION-NUMBER` is the version of the package listed. For example, `1.2.0-build.2`.

   For example:

   ```
   $ tanzu package install appliveview-conventions -p conventions.appliveview.tanz
   u.vmware.com -v 1.2.0-build.2 -n 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-rol
   e'
   | Creating cluster role binding 'appliveview-conventions-tap-install-cluster-ro
   lebinding'
   - Creating package resource
   \ Package install status: Reconciling

   Added installed package 'appliveview-conventions' in namespace 'tap-install'
   ```

3. 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:         1.2.0-build.2
   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 Application Platform GUI. To access the Application Live View UI, see Application Live View in Tanzu Application Platform GUI.

Enabling Spring Boot apps for Application Live View

This topic tells you how developers can 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>
```

The Application Live View Convention then sets the runtime environment properties `management.endpoints.web.exposure.include="*"` and `management.endpoint.health.show-details=true` onto 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`.

For more information on the labels automatically set by Application Live View Convention, see Convention server.

Important security advice

The Application Live View Convention automatically exposes all the actuators of an app so that Application Live View can access all those actuator endpoints and visualize all the details about the UI. This overrides configuration settings that your app itself might contain, for example, if you configured your app to expose only specific actuators.

Read about the Application Live View Convention and the Spring Boot Convention to understand the potential impact of this, and manually configure this to suit your security needs.

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>
```

Note: 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
```

Note: 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-server
```

Enabling Spring Boot apps for Application Live View

This topic tells you how developers can 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>
```

The Application Live View Convention then sets the runtime environment properties `management.endpoints.web.exposure.include="*"` and `management.endpoint.health.show-details=true` onto 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`.

For more information on the labels automatically set by Application Live View Convention, see Convention server.

Important security advice

The Application Live View Convention automatically exposes all the actuators of an app so that Application Live View can access all those actuator endpoints and visualize all the details about the UI. This overrides configuration settings that your app itself might contain, for example, if you configured your app to expose only specific actuators.

Read about the Application Live View Convention and the Spring Boot Convention to understand the potential impact of this, and manually configure this to suit your security needs.

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>
```

Note: 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
```

Note: 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-server
```

Enabling Steeltoe apps for Application Live View (beta)

This topic tells you how developers can extend .NET Core Apps to Steeltoe apps and enable Application Live View on Steeltoe workloads within Tanzu Application Platform (commonly known as TAP).

Caution: Enabling Steeltoe apps for Application Live View is currently in beta and is intended for evaluation and test purposes only. Do not use in a production environment.

Enable Steeltoe apps

You can enable Application Live View to interact with a Steeltoe app within Tanzu Application Platform.

To expose management actuator endpoints, add following configuration to your `appsettings.json` file:

```
{
  "Management": {
    "Endpoints": {
      "Actuator":{
        "Exposure": {
          "Include": [ "*" ]
        }
      }
    }
  }
}
```

To enable logging, add the following configuration to your `appsettings.json` file:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Steeltoe": "Warning",
      "Sample": "Information"
    }
  }
}
```

The thread metrics is available in SteeltoeVersion `3.2.0-rc1`. Therefore, to enable Threads page in Application Live View UI, add the following configuration to your `.csproj` file:

```
<PropertyGroup>
    <SteeltoeVersion>3.2.0-rc1</SteeltoeVersion>
</PropertyGroup>
```

To enable Application Live View on the Steeltoe TAP workload, you must manually add the label `tanzu.app.live.view.application.flavours: steeltoe` on your workload yaml:

```
metadata:
    labels:
        tanzu.app.live.view.application.flavours: steeltoe
```

Note: 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
```

Application Live View Convention server

This topic provides information about Application Live View Convention, which provides a Webhook handler for Convention Service for VMware Tanzu.

Role of Application Live View Convention

The Application Live View Convention works in conjunction with core Convention Service. 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.

For running Spring Boot apps, the 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: "8081"`: 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.

For running Spring Cloud Gateway apps, the 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: "8081"`: 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, the connector. The port is read from the `JAVA_TOOLS_OPTIONS` settings, especially from the `-Dmanagement.server.port=<..>` key-value pair.

The management port is either set automatically by the Spring Boot convention or you can set it as a key-value pair in the `JAVA_TOOLS_OPTIONS` environment variable from the `workload.yml` directly.

If there is no port specified, the connector uses the standard port `8080`.

In addition to that, the convention automatically adds the following app properties to the `JAVA_TOOLS_OPTIONS` environment variable:

- `-Dmanagement.endpoints.web.exposure.include="*"`: Exposes actuator endpoints of the app.
- `-Dmanagement.endpoint.health.show-details=true`: Shows the health details.

Important security advice

The Application Live View Convention automatically exposes all the actuators of an app so that Application Live View can access all those actuator endpoints and visualize all the details on the UI.

This overrides configuration settings that your app itself might contain, for example, if you configured your app to expose only specific actuators. To prevent the Application Live View convention from exposing all actuators, there are multiple options.

Uninstall the convention

One option is to uninstall the Application Live View convention. This results in no convention being applied automatically. You can still use Application Live View, but you must add the labels and environment settings yourself.

Deactivate the convention for specific workloads

Another option is to deactivate Application Live View for specific workloads. You can add the label `tanzu.app.live.view: "false"` manually, for example, by adding the label to the `workload.yml`. If the convention recognizes this label exists and is set to `false`, the convention does not apply any additional Application Live View configurations.

Manually configure the Application Live View settings for a workload

Another option is to keep Application Live View enabled for workloads, but set the corresponding labels and environment properties explicitly yourself. For example, using the `workload.yml`:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: tanzu-java-web-app
  namespace: default
  labels:
    tanzu.app.live.view: "true"
    tanzu.app.live.view.application.actuator.port: "7777"
    tanzu.app.live.view.application.flavours: spring-boot
    tanzu.app.live.view.application.name: tanzu-java-web-app
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/workload-type: web
spec:
  env:
  - name: JAVA_TOOL_OPTIONS
    value: >-
      -Dmanagement.server.port=7777
      -Dmanagement.endpoints.web.exposure.include=health,mappings,info
      -Dmanagement.endpoint.health.show-details=always
  source:
    git:
      ref:
        branch: main
      url: https://github.com/sample-accelerators/tanzu-java-web-app
```

Description of metadata labels

If a workload resource explicitly defines a label under `metadata.labels` in the `workload.yaml`, then 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`	`spring-boot-app`	Label	When deploying a workload in Tanzu Application Platform, this label is set to `spring-boot-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.

Metadata	Default	Type	Description
`tanzu.app.live.view.application.flavours`	`spring-boot,spring-cloud-gateway`	Label	When deploying a Spring Boot workload in Tanzu Application Platform, this label is set to `spring-boot` as default across the supply chain. For Spring Cloud Gateway app, it is set to `spring-boot,spring-cloud-gateway` as default.
`management.endpoints.web.exposure.include`	`*`	Environment Property	The user provided environment property takes precedence over the default value set by Application Live View Convention Server.
`management.endpoint.health.show-details`	always	Environment Property	The user provided environment property takes precedence over the default value set by Application Live View Convention Server.

Similarly, to override the default value for `management.endpoints.web.exposure.include` or `management.endpoint.health.show-details`, add it to the workload as follows:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  ...
spec:
  env:
  - name: JAVA_TOOL_OPTIONS
    value: >-
      -Dmanagement.endpoints.web.exposure.include=health,mappings,info
      -Dmanagement.endpoint.health.show-details=always
```

Application Live View Convention Server detects properties defined in the workload `env` section and respects those values.

Warning! You can also define properties such as `management.endpoints.web.exposure.include` and `management.endpoint.health.show-details` in `application.properties` or `application.yml` in the Spring Boot or Spring Cloud Gateway Application. Properties defined in this way have lower priority and will be overridden by the Application Live View Convention default values.

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` the name of the deployed workload, for example `tanzu-java-web-app`.

Expected output for Spring Boot Workload:

```
apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
  creationTimestamp: "2021-11-10T10:19:38Z"
  generation: 1
  labels:
    app.kubernetes.io/component: intent
    app.kubernetes.io/part-of: tanzu-java-web-appweb
    carto.run/cluster-supply-chain-name: source-to-url
    carto.run/cluster-template-name: convention-template
    carto.run/resource-name: config-provider
    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: 998ab107-c232-4dcf-a4b2-1d499b7709c6
  resourceVersion: "4502417"
  uid: 92c65a88-5beb-4405-b659-3b78834df125
spec:
  serviceAccountName: service-account
  template:
    metadata:
      annotations:
        developer.conventions/target-containers: workload
      labels:
        app.kubernetes.io/component: run
        app.kubernetes.io/part-of: tanzu-java-web-appweb
        carto.run/workload-name: tanzu-java-web-app
    spec:
      containers:
      - image: dev.registry.tanzu.vmware.com/app-live-view/test/tanzu-java-web-app@sha
256:db323d46a03e54948e844e7a7fced7d42b737c90b1c3a3a9bb775de9bce92c30
        name: workload
        resources: {}
        securityContext:
          runAsUser: 1000
      serviceAccountName: service-account
status:
  conditions:
  - lastTransitionTime: "2021-11-10T10:19:46Z"
    status: "True"
    type: ConventionsApplied
  - lastTransitionTime: "2021-11-10T10:19:46Z"
    status: "True"
    type: Ready
  observedGeneration: 1
  template:
    metadata:
      annotations:
        boot.spring.io/actuator: http://:8080/actuator
        boot.spring.io/version: 2.5.4
        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
        developer.conventions/target-containers: workload
      labels:
        app.kubernetes.io/component: run
        app.kubernetes.io/part-of: tanzu-java-web-appweb
        carto.run/workload-name: tanzu-java-web-app
        conventions.carto.run/framework: spring-boot
        tanzu.app.live.view: "true"
        tanzu.app.live.view.application.flavours: spring-boot
        tanzu.app.live.view.application.name: demo
    spec:
      containers:
      - env:
        - name: JAVA_TOOL_OPTIONS
          value: -Dmanagement.endpoint.health.show-details="always" -Dmanagement.endpo
ints.web.base-path="/actuator"
            -Dmanagement.endpoints.web.exposure.include="*" -Dmanagement.server.port
="8080"
            -Dserver.port="8080" -Dserver.shutdown.grace-period="24s"
        image: dev.registry.tanzu.vmware.com/app-live-view/test/tanzu-java-web-app@sha
256:db323d46a03e54948e844e7a7fced7d42b737c90b1c3a3a9bb775de9bce92c30
        name: workload
        ports:
        - containerPort: 8080
          protocol: TCP
        resources: {}
        securityContext:
          runAsUser: 1000
      serviceAccountName: service-account
```

Expected output for Spring Cloud Gateway workload:

```
apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
  creationTimestamp: "2021-11-10T10:19:38Z"
  generation: 1
  labels:
    app.kubernetes.io/component: intent
    app.kubernetes.io/part-of: tanzu-java-web-appweb
    carto.run/cluster-supply-chain-name: source-to-url
    carto.run/cluster-template-name: convention-template
    carto.run/resource-name: config-provider
    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: 998ab107-c232-4dcf-a4b2-1d499b7709c6
  resourceVersion: "4502417"
  uid: 92c65a88-5beb-4405-b659-3b78834df125
spec:
  serviceAccountName: service-account
  template:
    metadata:
      annotations:
        developer.conventions/target-containers: workload
      labels:
        app.kubernetes.io/component: run
        app.kubernetes.io/part-of: tanzu-java-web-appweb
        carto.run/workload-name: tanzu-java-web-app
    spec:
      containers:
      - image: dev.registry.tanzu.vmware.com/app-live-view/test/tanzu-java-web-app@sha
256:db323d46a03e54948e844e7a7fced7d42b737c90b1c3a3a9bb775de9bce92c30
        name: workload
        resources: {}
        securityContext:
          runAsUser: 1000
      serviceAccountName: service-account
status:
  conditions:
  - lastTransitionTime: "2021-11-10T10:19:46Z"
    status: "True"
    type: ConventionsApplied
  - lastTransitionTime: "2021-11-10T10:19:46Z"
    status: "True"
    type: Ready
  observedGeneration: 1
  template:
    metadata:
      annotations:
        boot.spring.io/actuator: http://:8080/actuator
        boot.spring.io/version: 2.5.4
        conventions.carto.run/applied-conventions: |-
          appliveview-sample/app-live-view-connector-boot
          appliveview-sample/app-live-view-connector-scg
          appliveview-sample/app-live-view-appflavours-boot
          appliveview-sample/app-live-view-appflavours-scg
          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
        developer.conventions/target-containers: workload
      labels:
        app.kubernetes.io/component: run
        app.kubernetes.io/part-of: tanzu-java-web-appweb
        carto.run/workload-name: tanzu-java-web-app
        conventions.carto.run/framework: spring-boot
        tanzu.app.live.view: "true"
        tanzu.app.live.view.application.flavours: spring-boot,spring-cloud-gateway
        tanzu.app.live.view.application.name: demo
    spec:
      containers:
      - env:
        - name: JAVA_TOOL_OPTIONS
          value: -Dmanagement.endpoint.health.show-details="always" -Dmanagement.endpo
ints.web.base-path="/actuator"
            -Dmanagement.endpoints.web.exposure.include="*" -Dmanagement.server.port
="8080"
            -Dserver.port="8080" -Dserver.shutdown.grace-period="24s"
        image: dev.registry.tanzu.vmware.com/app-live-view/test/tanzu-java-web-app@sha
256:db323d46a03e54948e844e7a7fced7d42b737c90b1c3a3a9bb775de9bce92c30
        name: workload
        ports:
        - containerPort: 8080
          protocol: TCP
        resources: {}
        securityContext:
          runAsUser: 1000
      serviceAccountName: service-account
```

In your output:

- `status.metadata.template.spec.containers.env.value` shows the list of applied environment properties by Application Live View convention server.
- `status.metadata.template.labels` shows the list of applied labels by Application Live View convention server.
- `status.metadata.template.annotations` shows the list of applied annotations by Application Live View convention server.

Custom configuration for the connector

This topic tells you how developers can 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, the Application Live View convention 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

Label Name	Mandatory	Type	Default	Significance
`tanzu.app.live.view.application.protocol`	false	http / https	`http`	Protocol scheme
`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 tells you how developers can 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	Sidecar 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`	`app.live.view.sidecar.application-actuator-path=actuator`, `app.live.view.sidecar.application-actuator-port=8080`
`/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`	`app.live.view.sidecar.application-actuator-path=path`, `app.live.view.sidecar.application-actuator-port=8082`

management.server.base-path	management.server.port	management.endpoints.web.base-path	server.servlet.context.path	Comments	Connector Configuration	Sidecar Configuration
`/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`	`app.live.view.sidecar.application-actuator-path=path/manage/actuator`, `app.live.view.sidecar.application-actuator-port=8082`
None	None	`/`	None	Actuators are deactivated to avoid conflicts	None	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`	`app.live.view.sidecar.application-actuator-path=manage`, `app.live.view.sidecar.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`	`app.live.view.sidecar.application-actuator-path=path/actuator`, `app.live.view.sidecar.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`	`app.live.view.sidecar.application-actuator-path=actuator`, `app.live.view.sidecar.application-actuator-port=8082`
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`	`app.live.view.sidecar.application-actuator-path=api/actuator`, `app.live.view.sidecar.application-actuator-port=8080`

management.server.base-path	management.server.port	management.endpoints.web.base-path	server.servlet.context.path	Comments	Connector Configuration	Sidecar Configuration
`/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`	`app.live.view.sidecar.application-actuator-path=path/actuator`, `app.live.view.sidecar.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`	`app.live.view.sidecar.application-actuator-path=path/manage`, `app.live.view.sidecar.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`	`app.live.view.sidecar.application-actuator-path=api/manage`, `app.live.view.sidecar.application-actuator-port=8080`
`/path`	None	`/`	`/api`	Actuators are deactivated to avoid conflicts	None	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`	`app.live.view.sidecar.application-actuator-path=path`, `app.live.view.sidecar.application-actuator-port=8082`
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`	`app.live.view.sidecar.application-actuator-path=api/manage`, `app.live.view.sidecar.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 App 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.

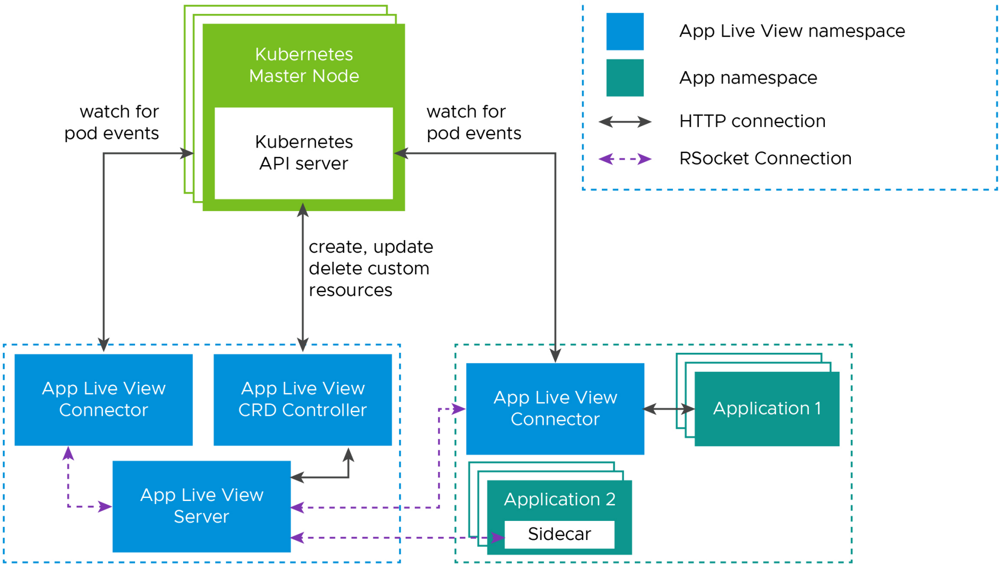
Support for polyglot apps with Application Live View

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 Enabling Spring Boot apps for Application Live View.
- To enable Application Live View on Steeltoe apps, see Enabling Steeltoe apps for Application Live View.

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 Server**

  Application Live View Server is the central server component 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**

  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 Server 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.

  You can deploy Application Live View Connector in two modes:

  - `Cluster access`: Deploy as a Kubernetes DaemonSet to discover apps across all the namespaces running in a worker node of a Kubernetes cluster. This is the default mode of Application Live View Connector.
  - `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 Convention Webhook component with Tanzu Application Platform.

Design flow

As illustrated in the architecture diagram, the **App Live View** namespace contains all the Application Live View components and the **Apps** namespace contains all the apps to be registered with Application Live View Server.

The apps run by the user are registered with Application Live View Server using Application Live View Connector.

Application Live View Connector, which is a lean model, uses specific labels to discover apps across cluster or namespace. 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 using labels. Then Application Live View Connector registers the filtered app instances with Application Live View server. Application Live View server proxies the call to the connector for querying actuator endpoint information.

The Application Live View Server fetches the actuator data of the app by proxying the request to Application Live View Connector by using an RSocket connection.

Troubleshooting

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 App 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 App Live View connector pod logs to see if the connector is sending updates to the back end.
2. Try deleting the connector pod so it is re-created by running:

   ```
   kubectl -n app-live-view-connector delete pods -l=name=application-live-view-co
   nnector
   ```

No live information for pod with ID

Symptom

In Tanzu Application Platform GUI, you receive the error `No live information for pod with id`.

Cause

This might happen because of stale information in App 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 `sslDisabled` flag in the values YAML file does not accept values without quotes.

Cause

The `sslDisabled` 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)
```

Configure the sidecar as follows:

```
app.live.view.sidecar.application-actuator-path=/manage  (manage is the custom actuato
r path set on the application)
app.live.view.sidecar.application-actuator-port=8085  (8085 is the custom management s
erver port set on the application)
```

Uninstalling Application Live View for VMware Tanzu

This topic tells you how to uninstall Application Live View from Tanzu Application Platform (commonly known as TAP).

To uninstall the Application Live View Backend, 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

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.

With AppSSO, Service Operators can configure and deploy authorization servers. Application Operators can then configure 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, and client restrictions.

AppSSO’s authorization server is based off of Spring Authorization Server.

Overview of Application Single Sign-On for VMware Tanzu

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.

With AppSSO, Service Operators can configure and deploy authorization servers. Application Operators can then configure 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, and client restrictions.

AppSSO’s authorization server is based off of Spring Authorization Server.

Install Application Single Sign-On

This document describes how to install Application Single Sign-On (AppSSO) from the Tanzu Application Platform package repository.

Note: Use the instructions on this page if you do not want to use a profile to install packages. Both the run, iterate and full profiles include AppSSO. For more information about profiles, see About Tanzu Application Platform components and profiles.

Installation

1. Learn more about the AppSSO package:

```
tanzu package available get sso.apps.tanzu.vmware.com --namespace tap-install
```

2. Install the AppSSO package:

```
tanzu package install appsso \
 --namespace tap-install \
 --package-name sso.apps.tanzu.vmware.com \
 --version 1.0.0
```

3. Confirm the package has reconciled successfully:

```
tanzu package installed get appsso --namespace tap-install
```

See also

- AppSSO for Platform Operators documentation.

Getting started with Application Single Sign-On

This document provides references for getting started with Application Single Sign-On (AppSSO).

Getting started

- If this is your first time using AppSSO, see the full Getting Started guide.
- If already have a basic understanding of AppSSO, and want to integrate with Tanzu Application Platform Workloads, see Register an app with AppSSO.

Overview of Convention Service

The Cartographer Conventions component must be installed to add conventions to your Pod. The v0.07.0 version of the convention controller is a passive system that translates the CRDs to the new group.

Caution: This component is being deprecated in favor of Cartographer Conventions.

Cartographer Conventions

This topic describes an overview of Cartographer Conventions and how you can use it with Tanzu Application Platform.

Overview

> **Note**
>
> This component is replacing the convention controller.

Cartographer Conventions provides a means for operators to express their knowledge about how applications should run on Kubernetes as a convention. Cartographer Conventions 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:

- **The convention controller:** The convention controller provides the metadata to the convention server and executes the updates to Pod Template Spec as per the convention server's requests.
- **The convention server:** The convention server receives and evaluates metadata associated with a workload and requests updates to the Pod Template Spec associated with that workload. You can have one or more convention servers for a single convention controller instance. Cartographer Conventions supports defining and applying conventions for Pods.

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. It is possible for a convention to 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 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 can not include the same metadata.

Applying conventions without using image metadata

Conventions can be defined to 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. Such conventions are a great way to ensure infrastructure uniformity 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.

Cartographer Conventions

This topic describes an overview of Cartographer Conventions and how you can use it with Tanzu Application Platform.

Overview

Note

This component is replacing the convention controller.

Cartographer Conventions provides a means for operators to express their knowledge about how applications should run on Kubernetes as a convention. Cartographer Conventions 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:

- **The convention controller:** The convention controller provides the metadata to the convention server and executes the updates to Pod Template Spec as per the convention server's requests.
- **The convention server:** The convention server receives and evaluates metadata associated with a workload and requests updates to the Pod Template Spec associated with that workload. You can have one or more convention servers for a single convention controller instance. Cartographer Conventions supports defining and applying conventions for Pods.

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. It is possible for a convention to 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 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 can not include the same metadata.

Applying conventions without using image metadata

Conventions can be defined to 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. Such conventions are a great way to ensure infrastructure uniformity 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

Cartographer Conventions is bundled with Supply Chain Choreographer as of the v0.4.0 release. See Installing Supply Chain Choreographer.

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. 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 been applied. So if needed, 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 controller.

Convention controller

The convention controller is the orchestrator of one or many convention servers deployed to the cluster. 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 document covers developing conventions using GOLANG, but this can be done using other languages by following the specifications.

Prerequisites

The following prerequisites must be met before a convention is developed and deployed:

- The Kubernetes command line 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
  ```

- The ko CLI is installed by using GitHub. 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.

1. 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)
    }
}
...
```

2. 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 is used to define the criteria for applying conventions. See PodTemplateSpec in the Kubernetes documentation. The following are a few examples.

Matching criteria by labels or annotations

When you use labels or annotations to define whether a convention must be applied, the server checks the PodTemplateSpec of workloads.

- PodTemplateSpec

```
...
template:
  metadata:
    labels:
      awesome-label: awesome-value
    annotations:
      awesome-annotation: awesome-value
...
```

- Handler

```
package convention
...
func conventionHandler(template *corev1.PodTemplateSpec, images []model.ImageConfig) ([]string, error) {
    c:= []string{}
    // This convention is applied if a specific label is present.
    if lv, le := template.Labels["awesome-label"]; le && lv == "awesome-value" {
        // DO COOL STUFF
        c = append(c, "awesome-label-convention")
    }
    // This convention is applied if a specific annotation is present.
    if av, ae := template.Annotations["awesome-annotation"]; ae && av == "awesome-value" {
        // DO COOL STUFF
        c = append(c, "awesome-annotation-convention")
    }

    return c, nil
}
...
```

Where:

- `conventionHandler` is the *handler*.
- `awesome-label` is the **label** that you want to validate.
- `awesome-annotation` is the **annotation** that you want to validate.
- `awesome-value` is the value that must have the **label/annotation**.

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.

   - If the convention must be built and deployed, use the [ko] tool on GitHub (https://github.com/google/ko). It compiles yout *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 be applied 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, check for 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, check 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 Convention Service

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 reports an error `failed to get CABundle` as follows:

```
{"level":"error","ts":1638222343.6839523,"logger":"controllers.PodIntent.PodInt
ent.ResolveConventions","msg":"failed to get CABundle","ClusterPodConventio
n":"base-convention","error":"unable to find valid certificaterequests for cert
ificate \"convention-template/webhook-certificate\"","stacktrace":"reflect.Valu
e.Call\n\treflect/value.go:339\ngithub.com/vmware-labs/reconciler-runtime/recon
cilers.(*SyncReconciler).sync\n\tgithub.com/vmware-labs/reconciler-runtime@v0.
3.0/reconcilers/reconcilers.go:287\ngithub.com/vmware-labs/reconciler-runtime/r
econcilers.(*SyncReconciler).Reconcile\n\tgithub.com/vmware-labs/reconciler-run
time@v0.3.0/reconcilers/reconcilers.go:276\ngithub.com/vmware-labs/reconciler-r
untime/reconcilers.Sequence.Reconcile\n\tgithub.com/vmware-labs/reconciler-runt
ime@v0.3.0/reconcilers/reconcilers.go:815\ngithub.com/vmware-labs/reconciler-ru
ntime/reconcilers.(*ParentReconciler).reconcile\n\tgithub.com/vmware-labs/recon
ciler-runtime@v0.3.0/reconcilers/reconcilers.go:146\ngithub.com/vmware-labs/rec
onciler-runtime/reconcilers.(*ParentReconciler).Reconcile\n\tgithub.com/vmware-
labs/reconciler-runtime@v0.3.0/reconcilers/reconcilers.go:120\nsigs.k8s.io/cont
roller-runtime/pkg/internal/controller.(*Controller).Reconcile\n\tsigs.k8s.io/c
ontroller-runtime@v0.10.3/pkg/internal/controller/controller.go:114\nsigs.k8s.i
o/controller-runtime/pkg/internal/controller.(*Controller).reconcileHandler\n\t
sigs.k8s.io/controller-runtime@v0.10.3/pkg/internal/controller/controller.go:31
1\nsigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).process
NextWorkItem\n\tsigs.k8s.io/controller-runtime@v0.10.3/pkg/internal/controller/
controller.go:266\nsigs.k8s.io/controller-runtime/pkg/internal/controller.(*Con
troller).Start.func2.2\n\tsigs.k8s.io/controller-runtime@v0.10.3/pkg/internal/c
ontroller/controller.go:227"}
```

Cause

`convention server` (ClusterPodConvention) is configured with 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.g
o: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. Check the `convention server` logs to identify the cause of the error:
    1. Use the following command 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`.
   - Be aware that 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 the following command:

```
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   Killing    13m (x2 over 13m)     kubelet            Container webhook
failed liveness probe, will be restarted
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
```

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.

API Structure

The `PodConventionContext` API object in the `webhooks.conventions.carto.run` API group is the structure used for both request and response from the convention server.

Template Status

The enriched `PodTemplateSpec` is reflected at `.status.template`. For more information about `PodTemplateSpec`, see the Kubernetes documentation.

Chaining Multiple Conventions

You can define multiple `ClusterPodConventions` and apply them to different types of workloads. You can also apply multiple conventions to a single workload.

The `PodIntent` reconciler lists all `ClusterPodConvention` resources and applies them serially. To ensure the consistency of enriched `PodTemplateSpec`, the list of `ClusterPodConventions`is sorted alphabetically by name before applying conventions. 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 successfully. A list of all applied conventions is stored under the annotation `conventions.carto.run/applied-conventions`.

Collecting Logs from the Controller

The convention controller is a Kubernetes operator and can be deployed in a cluster with other components. If you have trouble, you can retrieve and examine the logs from the controller to help identify issues.

To retrieve Pod logs from the `conventions-controller-manager` running in the `conventions-system` namespace:

```
kubectl -n conventions-system logs -l control-plane=controller-manager
```

For example:

```
...
{"level":"info","ts":1637073467.3334172,"logger":"controllers.PodIntent.PodIntent.Appl
yConventions","msg":"applied convention","diff":"  interface{}(\n- \ts\"&PodTemplateSp
ec{ObjectMeta:{      0 0001-01-01 00:00:00 +0000 UTC <nil> <nil> map[app.kubernetes.i
o/component:run app.kubernetes.io/part-of:spring-petclinic-app-db carto.run/workload-n
ame:spring-petclinic-app-db] map[developer.conventions/target-container\"...,\n+ \tv1.
PodTemplateSpec{\n+ \t\tObjectMeta: v1.ObjectMeta{\n+ \t\t\tLabels: map[string]string
{\n+ \t\t\t\t\"app.kubernetes.io/component\": \"run\",\n+ \t\t\t\t\"app.kubernetes.io/
part-of\":   \"spring-petclinic-app-db\",\n+ \t\t\t\t\"carto.run/workload-name\":
\"spring-petclinic-app-db\",\n+ \t\t\t\t\"tanzu.app.live.view\":         \"true\",\n+
\t\t\t\t...\n+ \t\t\t},\n+ \t\t\tAnnotations: map[string]string{\"developer.convention
s/target-containers\": \"workload\"},\n+ \t\t},\n+ \t\tSpec: v1.PodSpec{Containers: []
v1.Container{{...}}, ServiceAccountName: \"default\"},\n+ \t},\n  )\n","convention":"a
ppliveview-sample"}
...
```

References

- ImageConfig
- PodConventionContextSpec
- PodConventionContextStatus
- PodConventionContext
- ClusterPodConvention
- PodIntent
- BOM

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.

API Structure

The `PodConventionContext` API object in the `webhooks.conventions.carto.run` API group is the structure used for both request and response from the convention server.

Template Status

The enriched `PodTemplateSpec` is reflected at `.status.template`. For more information about `PodTemplateSpec`, see the Kubernetes documentation.

Chaining Multiple Conventions

You can define multiple `ClusterPodConventions` and apply them to different types of workloads. You can also apply multiple conventions to a single workload.

The `PodIntent` reconciler lists all `ClusterPodConvention` resources and applies them serially. To ensure the consistency of enriched `PodTemplateSpec`, the list of `ClusterPodConventions`is sorted alphabetically by name before applying conventions. 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 successfully. A list of all applied conventions is stored under the annotation `conventions.carto.run/applied-conventions`.

Collecting Logs from the Controller

The convention controller is a Kubernetes operator and can be deployed in a cluster with other components. If you have trouble, you can retrieve and examine the logs from the controller to help identify issues.

To retrieve Pod logs from the `conventions-controller-manager` running in the `conventions-system` namespace:

```
kubectl -n conventions-system logs -l control-plane=controller-manager
```

For example:

```
...
{"level":"info","ts":1637073467.3334172,"logger":"controllers.PodIntent.PodIntent.Appl
yConventions","msg":"applied convention","diff":"  interface{}(\n- \ts\"&PodTemplateSp
ec{ObjectMeta:{      0 0001-01-01 00:00:00 +0000 UTC <nil> <nil> map[app.kubernetes.i
o/component:run app.kubernetes.io/part-of:spring-petclinic-app-db carto.run/workload-n
ame:spring-petclinic-app-db] map[developer.conventions/target-container\"...,\n+ \tv1.
PodTemplateSpec{\n+ \t\tObjectMeta: v1.ObjectMeta{\n+ \t\t\tLabels: map[string]string
{\n+ \t\t\t\t\"app.kubernetes.io/component\": \"run\",\n+ \t\t\t\t\"app.kubernetes.io/
part-of\":   \"spring-petclinic-app-db\",\n+ \t\t\t\t\"carto.run/workload-name\":
\"spring-petclinic-app-db\",\n+ \t\t\t\t\"tanzu.app.live.view\":         \"true\",\n+
\t\t\t\t...\n+ \t\t\t},\n+ \t\t\tAnnotations: map[string]string{\"developer.convention
s/target-containers\": \"workload\"},\n+ \t\t},\n+ \t\tSpec: v1.PodSpec{Containers: []
v1.Container{{...}}, ServiceAccountName: \"default\"},\n+ \t},\n  )\n","convention":"a
ppliveview-sample"}
...
```

References

- ImageConfig
- PodConventionContextSpec
- PodConventionContextStatus
- PodConventionContext
- ClusterPodConvention
- PodIntent
- BOM

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

`ClusterPodConvention` defines a way to connect to convention servers. It provides a way to apply a set of conventions to a `PodTemplateSpec` and the artifact metadata. A convention will typically focus on a particular application framework, but may 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:
  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": "`some byte array`"
}
```

Where:

- `name`: For a cloud native buildpack SBOM, it starts with prefix `cnb-sbom:` and is followed by the location of the BOM definition in the *layer*. For example: `cnb-sbom:/layers/sbom/launch/paketo-buildpacks_executable-jar/sbom.cdx.json`. For any non CNB-SBOM, the `name` might change.
- `raw`: 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 will forward BOMs to the convention servers that it can discover from known sources, including:

- *CNB-SBOM*

cert-manager, Contour

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

Contour is an ingress controller for Kubernetes that supports dynamic configuration updates and multiteam ingress delegation. It provides the control plane for the Envoy edge and service proxy. For more information about Contour, see the Contour documentation.

cert-manager, Contour

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

Contour is an ingress controller for Kubernetes that supports dynamic configuration updates and multiteam ingress delegation. It provides the control plane for the Envoy edge and service proxy. For more information about Contour, see the Contour documentation.

Install cert-manager, Contour

This document tells you how to install cert-manager, Contour, and 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 cert-manager, contour, and Flux CD Source Controller. For more information about profiles, see Components and installation profiles.

Prerequisites

Before installing cert-manager, Contour, and Flux CD Source Controller:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.

Install cert-manager

To install cert-manager 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  1.5.3+tap.1  2021-08-23T17:22:51Z
   ```

2. Create a file named `cert-manager-rbac.yaml` using the following sample and apply the configuration.

   ```
   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
```

For example:

```
kubectl apply -f cert-manager-rbac.yaml
```

3. Create a file named `cert-manager-install.yaml` using the following sample and apply the configuration.

```
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: {}
```

Where:

- `VERSION-NUMBER` is the version of the package listed in step 1.

For example:

```
kubectl apply -f cert-manager-install.yaml
```

4. Verify the package install by running:

```
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:         1.5.3+tap.1
STATUS:                  Reconcile succeeded
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`

```
kubectl get deployment cert-manager -n cert-manager
```

For example:

```
$ kubectl get deploy cert-manager -n cert-manager
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
cert-manager   1/1     1            1           2m18s
```

Verify that `STATUS` is `Running`

Install Contour

To install Contour from the Tanzu Application Platform package repository:

1. 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
   - Retrieving package versions for contour.tanzu.vmware.com...
     NAME                       VERSION       RELEASED-AT
     contour.tanzu.vmware.com  1.18.2+tap.1  2021-10-05T00:00:00Z
   ```

2. Create a file named `contour-rbac.yaml` 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
   ```

3. Apply the configuration by running:

   ```
   kubectl apply -f contour-rbac.yaml
   ```

4. Create a file named `contour-install.yaml` using the following sample and apply the configuration. The following configuration installs the Contour package with default options. If you want 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 in step 1.

5. (Optional) Make changes to the default installation settings:

    1. Gather values schema by running:

        ```
        tanzu package available get contour.tanzu.vmware.com/1.18.2+tap.1 --value
        s-schema -n tap-install
        ```

        For example:

        ```
        $ tanzu package available get contour.tanzu.vmware.com/1.18.2+tap.1 --val
        ues-schema -n tap-install
        | Retrieving package details for contour.tanzu.vmware.com/1.18.2+tap.1...
          KEY                                DEFAULT               TYPE     DES
        CRIPTION

          certificates.duration              8760h                 string   If
        using cert-manager, how long the certificates should be valid for. If use
        CertManager is false, this field is ignored.
          certificates.renewBefore           360h                  string   If
        using cert-manager, how long before expiration the certificates should be
        renewed. If useCertManager is false, this field is ignored.
          contour.configFileContents         <nil>                 object   The
        YAML contents of the Contour config file. See https://projectcontour.io/d
        ocs/v1.18.2/configuration/#configuration-file for more information.
          contour.logLevel                   info                  string   The
        Contour log level. Valid options are info and debug.

          contour.replicas                   2                     integer  How
        many Contour pod replicas to have.

          contour.useProxyProtocol           false                 boolean  Whe
        ther to enable PROXY protocol for all Envoy listeners.

          envoy.hostPorts.enable             true                  boolean  Whe
        ther to enable host ports. If false, http and https are ignored.

          envoy.hostPorts.http               80                    integer  If
        enable == true, the host port number to expose Envoy's HTTP listener on.

          envoy.hostPorts.https              443                   integer  If
        enable == true, the host port number to expose Envoy's HTTPS listener on.

          envoy.logLevel                     info                  string   The
        Envoy log level.

          envoy.service.annotations          <nil>                 object   Ann
        otations to set on the Envoy service.

          envoy.service.aws.LBType           classic               string   AWS
        loadbalancer type.

          envoy.service.externalTrafficPolicy  Cluster             string   The
        external traffic policy for the Envoy service.

          envoy.service.nodePorts.http       <nil>                 integer  If
        type == NodePort, the node port number to expose Envoy's HTTP listener o
        n. If not specified, a node port will be auto-assigned by Kubernetes.
          envoy.service.nodePorts.https      <nil>                 integer  If
        type == NodePort, the node port number to expose Envoy's HTTPS listener o
        n. If not specified, a node port will be auto-assigned by Kubernetes.
          envoy.service.type                 NodePort              string   The
        type of Kubernetes service to provision for Envoy.

          envoy.terminationGracePeriodSeconds  300                 integer  The
        termination grace period, in seconds, for the Envoy pods.

          envoy.hostNetwork                  false                 boolean  Whe
        ther to enable host networking for the Envoy pods.

          infrastructure_provider            vsphere               string   The
        ```

```
infrastructure in which to deploy Contour and Envoy. example:- vsphere, a
ws
  namespace                            tanzu-system-ingress  string   The
namespace in which to deploy Contour and Envoy.
```

2. Create a `contour-install.yaml` file using the following sample as a guide. 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: 1.18.2+tap.1
         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` or `minikube` can fail to complete the package install if LoadBalancer services are not supported.

   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 CNR 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 CNR value `ingress.internal.namespace` to point to this namespace.

6. Install the package by running:

   ```
   kubectl apply -f contour-install.yaml
   ```

7. Verify the package install by running:

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

   For example:

   ```
   $ tanzu package installed get contour -n tap-install
   / Retrieving installation details for contour...
   NAME:                    contour
   PACKAGE-NAME:            contour.tanzu.vmware.com
   PACKAGE-VERSION:         1.18.2+tap.1
   STATUS:                  Reconcile succeeded
   ```

```
CONDITIONS:               [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`

8. 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-857d46c845-4r6c5   1/1     Running   1          18d
contour-857d46c845-p6bbq   1/1     Running   1          18d
envoy-mxkjk                2/2     Running   2          18d
envoy-qlg8l                2/2     Running   2          18d
```

Ensure that all pods are `Running` with all containers ready.

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.

To learn more about Cloud Native Runtimes, see Cloud Native Runtimes for VMware Tanzu.

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.

To learn more about Cloud Native Runtimes, see Cloud Native Runtimes for VMware Tanzu.

Install Cloud Native Runtimes

This topic describes how you can install Cloud Native Runtimes 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 Cloud Native Runtimes. For more information about profiles, see Components and installation profiles.

Prerequisites

Before installing Cloud Native Runtimes:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- Ensure you have an internal or external Contour namespace on a cluster with v1.19.1 of Contour installed when you configure `reuse_crds:true`.

Note: Cloud Native Runtimes fails to install when configured with `reuse_crds:true` and no internal or external Contour namespace provided on a cluster with Contour installed at a version other than v1.19.1.

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
- Retrieving package versions for cnrs.tanzu.vmware.com...
  NAME                   VERSION  RELEASED-AT
  cnrs.tanzu.vmware.com  1.0.3    2021-10-20T00:00:00Z
```

2. (Optional) Make changes to the default installation settings:

    1. Gather values schema.

        ```
        tanzu package available get cnrs.tanzu.vmware.com/1.0.3 --values-schema -
        n tap-install
        ```

        For example:

        ```
        $ tanzu package available get cnrs.tanzu.vmware.com/1.0.3 --values-schema
        -n tap-install
        | Retrieving package details for cnrs.tanzu.vmware.com/1.0.3...
          KEY                           DEFAULT  TYPE             DESCRIPTION
          ingress.external.namespace  <nil>    string   Optional: Only valid if a
        Contour instance is already present in the cluster. Specify a namespace w
        here an existing Contour is installed on your cluster (for external servi
        ces) if you want CNR to use your Contour instance.
          ingress.internal.namespace  <nil>    string   Optional: Only valid if a
        Contour instance is already present in the cluster. Specify a namespace w
        here an existing Contour is installed on your cluster (for internal servi
        ces) if you want CNR to use your Contour instance.
          ingress.reuse_crds          false    boolean  Optional: Only valid if a
        Contour instance is already present in the cluster. Set to "true" if you
        want CNR to re-use the cluster's existing Contour CRDs.
          local_dns.domain            <nil>    string   Optional: Set a custom do
        main for CoreDNS. Only applicable when "local_dns.enable" is set to "tru
        e" and "provider" is set to "local" and running on Kind.
          local_dns.enable            false    boolean  Optional: Only for when
        "provider" is set to "local" and running on Kind. Set to true to enable l
        ocal DNS.
          pdb.enable                  true     boolean  Optional: Set to true to
        enable Pod Disruption Budget. If provider local is set to "local", the PD
        B will be disabled automatically.
          provider                    <nil>    string   Optional: Kubernetes clus
        ter provider. To be specified if deploying CNR on a local Kubernetes clus
        ter provider.
        ```

    2. Create a `cnr-values.yaml` by using the following sample as a guide:

        Sample `cnr-values.yaml` for Cloud Native Runtimes:

        ```
        ---
        # if deploying on a local cluster such as Kind. Otherwise, you can remove
        this field
        provider: local
        ```

        Note: For most installations, you can leave the `cnr-values.yaml` empty, and use the default values.

        If you are running on a single-node cluster, such as kind or minikube, set the `provider: local` option. This option reduces resource requirements by using a HostPort service instead of a LoadBalancer and reduces the number of replicas.

        Cloud Native Runtimes reuses the existing `tanzu-system-ingress` Contour installation for external and internal access when installed in the `light` or `full` profile. If you want to use a separate Contour installation for system-internal traffic, set `cnrs.ingress.internal.namespace` to the empty string (`""`).

        For more information about using Cloud Native Runtimes with kind, see the Cloud Native Runtimes documentation. If you are running on a multinode cluster, do not set `provider`.

If your environment has Contour packages, Contour might conflict with the Cloud Native Runtimes installation.

For information about how to prevent conflicts, see Installing Cloud Native Runtimes for Tanzu with an Existing Contour Installation in the Cloud Native Runtimes documentation. Specify values for `ingress.reuse_crds`, `ingress.external.namespace`, and `ingress.internal.namespace` in the `cnr-values.yaml` file.

3. Install the package by running:

```
tanzu package install cloud-native-runtimes -p cnrs.tanzu.vmware.com -v 1.0.3 -
n tap-install -f cnr-values.yaml --poll-timeout 30m
```

For example:

```
$ tanzu package install cloud-native-runtimes -p cnrs.tanzu.vmware.com -v 1.0.3
-n tap-install -f 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'
```

Use an empty file for `cnr-values.yaml` if you want the default installation configuration. Otherwise, see the previous step to learn more about setting installation configuration values.

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 cc...
NAME:                    cloud-native-runtimes
PACKAGE-NAME:            cnrs.tanzu.vmware.com
PACKAGE-VERSION:         1.0.3
STATUS:                  Reconcile succeeded
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`

5. Configure a namespace to use Cloud Native Runtimes:

> **Important**
>
> 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 in Set up developer namespaces to use your installed packages. Otherwise, you must complete the following steps for each namespace where you create Knative services.

Service accounts that run workloads using Cloud Native Runtimes 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 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 current namespace 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=.dockerconfigjson={} --type=kubernetes.io/dockerconfigjson
   kubectl annotate secret pull-secret secretgen.carvel.dev/image-pull-secret=""
   ```

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"}]}'
   ```

3. 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.

Overview of Spring Boot conventions

This topic tells you about the Spring Boot convention server.

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

Run the `docker inspect` command to make the Spring Boot convention server look inside the image. Example command:

```
$ docker inspect springio/petclinic
```

Example output:

```
[
    {
        "Id": "sha256:...",
        "RepoTags": [
            "springio/petclinic:latest"
        ],
        "RepoDigests": [
            "springio/petclinic@sha256:..."
        ],
        "Parent": "",
        "Container": "",
        ...
        "ContainerConfig": {
            "Hostname": "",
            "Domainname": "",
            "User": "",
```

```
            ...
            "Labels": null
        },
        "DockerVersion": "",
        "Author": "",
        "Config": {
...
]
```

The convention server searches inside the image for `Config -> Labels -> io.buildpacks.build.metadata` to find the `bom` file. It looks inside the `bom` file for metadata to evaluate whether the convention is to be applied.

For the list of conventions, see Conventions.

Overview of Spring Boot conventions

This topic tells you about the Spring Boot convention server.

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

Run the `docker inspect` command to make the Spring Boot convention server look inside the image. Example command:

```
$ docker inspect springio/petclinic
```

Example output:

```
[
    {
        "Id": "sha256:...",
        "RepoTags": [
            "springio/petclinic:latest"
        ],
        "RepoDigests": [
            "springio/petclinic@sha256:..."
        ],
        "Parent": "",
        "Container": "",
        ...
        "ContainerConfig": {
            "Hostname": "",
            "Domainname": "",
            "User": "",
            ...
            "Labels": null
        },
        "DockerVersion": "",
        "Author": "",
        "Config": {
...
]
```

The convention server searches inside the image for `Config -> Labels -> io.buildpacks.build.metadata` to find the `bom` file. It looks inside the `bom` file for metadata to evaluate whether the convention is to be applied.

For the list of conventions, see Conventions.

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 --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package available list spring-boot-conventions.tanzu.vmware.com --namespace tap-install
   / Retrieving package versions for spring-boot-conventions.tanzu.vmware.com...
   NAME                                       VERSION   RELEASED-AT
   ...
   spring-boot-conventions.tanzu.vmware.com   0.1.2     2021-10-28T00:00:00Z
   ...
   ```

2. Install the package by running:

   ```
   tanzu package install spring-boot-conventions \
   --package-name spring-boot-conventions.tanzu.vmware.com \
   --version 0.1.2 \
   --namespace tap-install
   ```

3. 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:         0.1.2
   STATUS:                  Reconcile succeeded
   CONDITIONS:              [{ReconcileSucceeded True  }]
   USEFUL-ERROR-MESSAGE:
   ```

   Verify that `STATUS` is `Reconcile succeeded`

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.port=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 dependancies 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:

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.

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://: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
     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" -Dmanagement.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.apps.tanzu.vmware.com/v1alpha1
  kind: PodIntent
  metadata:
    annotations:
      kubectl.kubernetes.io/last-applied-configuration: |
        {"apiVersion":"conventions.apps.tanzu.vmware.com/v1alpha1","kind":"PodInten
t","metadata":{"annotations":{},"name":"spring-sample","namespace":"default"},"spec":
{"template":{"spec":{"containers":[{"image":"springio/petclinic","name":"workloa
d"}]}}}}
    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.apps.tanzu.vmware.com/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 Service Bindings

This topic tells you about using Service Bindings in Tanzu Application Platform (commonly know as TAP).

Service Bindings for Kubernetes implements the Service Binding Specification for Kubernetes. VMware is tracking changes to the specifications as it approaches a stable release, currently targeting pre-RC3 in GitHub. Backwards and forwards compatibility should not be expected for alpha versioned resources.

Supported service binding specifications

Service Bindings for Kubernetes is an open-source product. For more information, see the Service Binding for Kubernetes readme and the Service Binding for Kubernetes community website.

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).

Service Bindings for Kubernetes implements the Service Binding Specification for Kubernetes. VMware is tracking changes to the specifications as it approaches a stable release, currently targeting pre-RC3 in GitHub. Backwards and forwards compatibility should not be expected for alpha versioned resources.

Supported service binding specifications

Service Bindings for Kubernetes is an open-source product. For more information, see the Service Binding for Kubernetes readme and the Service Binding for Kubernetes community website.

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 service-bindings.labs.vmware.com --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package available list service-bindings.labs.vmware.com --namespace tap-install
   - Retrieving package versions for service-bindings.labs.vmware.com...
     NAME                              VERSION  RELEASED-AT
     service-bindings.labs.vmware.com  0.5.0    2021-09-15T00:00:00Z
   ```

2. Install the package by running:

   ```
   tanzu package install service-bindings -p service-bindings.labs.vmware.com -v 0.5.0 -n tap-install
   ```

   Example output:

   ```
   / Installing package 'service-bindings.labs.vmware.com'
   | Getting namespace 'tap-install'
   - Getting package metadata for 'service-bindings.labs.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-rolebinding'
   \ 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:            service-bindings.labs.vmware.com
   PACKAGE-VERSION:         0.5.0
   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
   manager-6d85fffbcd-j4gvs   1/1     Running   0          22s
   ```

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
```

Overview of Services Toolkit

The Services Toolkit comprises the following Kubernetes native components, which support the management, lifecycle, discoverability and connectivity of Service Resources (databases, message queues, DNS records, and so on) on Kubernetes:

- Service offering
- Service API projection
- Service resource replication
- Service resource claims

To learn more about Services Toolkit, see the Services Toolkit for VMware Tanzu documentation

Overview of Services Toolkit

The Services Toolkit comprises the following Kubernetes native components, which support the management, lifecycle, discoverability and connectivity of Service Resources (databases, message queues, DNS records, and so on) on Kubernetes:

- Service offering
- Service API projection
- Service resource replication
- Service resource claims

To learn more about Services Toolkit, see the Services Toolkit for VMware Tanzu documentation

Install Services Toolkit

This topic describes how to install Services Toolkit 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 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 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  0.7.1            2022-07-12T00:00:00Z
```

2. Install Services Toolkit by running:

   ```
   tanzu package install services-toolkit -n tap-install -p services-toolkit.tanz
   u.vmware.com -v VERSION-NUMBER
   ```

   Where `VERSION-NUMBER` is the Services Toolkit version you want to install. For example, `0.7.1`.

3. Verify that the package installed by running:

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

   and checking 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:         0.7.1
   STATUS:                  Reconcile succeeded
   CONDITIONS:              [{ReconcileSucceeded True  }]
   USEFUL-ERROR-MESSAGE:
   ```

Overview of Flux CD Source Controller

The fluxcd-source-controller is a Kubernetes Operator, specialised in artifacts acquisition from external sources such as Git, Helm repositories and S3 buckets. 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

The fluxcd-source-controller is a Kubernetes Operator, specialised in artifacts acquisition from external sources such as Git, Helm repositories and S3 buckets. 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 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 Source Controller package has no configurable properties.

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-install
   ```

   For example:

   ```
   $ tanzu package available list fluxcd.source.controller.tanzu.vmware.com -n tap-install
       \ Retrieving package versions for fluxcd.source.controller.tanzu.vmware.com...
         NAME                                       VERSION  RELEASED-AT
         fluxcd.source.controller.tanzu.vmware.com  0.16.0   2021-10-27 19:00:00 -0500 -05
   ```

2. Install the package by running:

   ```
   tanzu package install fluxcd-source-controller -p fluxcd.source.controller.tanzu.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.tanzu.vmware.com -v 0.16.0 -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-role'
   | Creating cluster role binding 'fluxcd-source-controller-tap-install-cluster-rolebinding'
   | 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. Follow these steps to consume a `GitRepository` object:

   1. Create the following `gitrepository-sample.yaml` file:

      ```
      apiVersion: source.toolkit.fluxcd.io/v1beta1
      kind: GitRepository
      metadata:
        name: gitrepository-sample
      spec:
        interval: 1m
        url: https://github.com/stefanprodan/podinfo
        ref:
          branch: master
      ```

   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/stefanprodan/podinfo   True
      Fetched revision: master/132f4e719209eb10b9485302f8593fc0e680f4fc   4s
      ```

For more examples, see the samples directory on fluxcd/source-controller/samples in GitHub.

Documentation

For documentation specific to fluxcd-source-controller, see the main repository fluxcd/source-controller in GitHub.

Overview of Source Controller

Tanzu Source Controller provides a standard interface for artifact acquisition. It supports two resource types:

- ImageRepository
- 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.

Tanzu Source Controller extends the functionality. For more information about Flux CD Source Controller, see the fluxcd/source-controller project on GitHub.

Overview of Source Controller

Tanzu Source Controller provides a standard interface for artifact acquisition. It supports two resource types:

- ImageRepository
- 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.

Tanzu Source Controller extends the functionality. For more information about Flux CD Source Controller, see the fluxcd/source-controller project on GitHub.

Install Source Controller

This document 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 --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package available list controller.source.apps.tanzu.vmware.com --namespace 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
   ```

2. (Optional) Gather values schema:

   ```
   tanzu package available get controller.source.apps.tanzu.vmware.com/VERSION-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 controller.source.apps.tanzu.vmware.com/0.4.1 --values-schema --namespace tap-install

   Retrieving package details for controller.source.apps.tanzu.vmware.com/0.4.1...
   KEY               DEFAULT  TYPE    DESCRIPTION
   aws_iam_role_arn           string  Optional: The AWS IAM Role ARN to attach to the Source Controller service account
   ca_cert_data               string  Optional: PEM Encoded certificate data for image registries with private CA.
   ```

3. (Optional) There are two optional fields that can override Source Controller's default installation setting:

   - `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, this option can be used to trust additional certificate authorities.
   - `aws_iam_role_arn` Annotates Source Controller service with an AWS IAM role. This allows Source Controller to pull images from ECR.

   To provide a custom certificate, create a file named `source-controller-values.yaml` that includes the PEM-encoded CA certificate data.

For example:

```
ca_cert_data: |
    -----BEGIN CERTIFICATE-----
    MIICpTCCAYUCBgkqhkiG9w0BBQ0wMzAbBgkqhkiG9w0BBQwwDgQIYg9x6gkCAggA
    ...
    9TlA7A4FFpQqbhAuAVH6KQ8WMZIrVxJSQ03c9lKVkI62wQ==
    -----END CERTIFICATE-----
```

To add AWS IAM role ARN in Source Controller Servc, create a file named `source-controller-values.yaml` that includes the following:

```
aws_iam_role_arn: "eks.amazonaws.com/role-arn: arn:aws:iam::112233445566:role/source-controller-manager"
```

4. Install the package:

```
tanzu package install source-controller -p controller.source.apps.tanzu.vmware.com -v VERSION-NUMBER -n tap-install -f 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 -p controller.source.apps.tanzu.vmware.com -v 0.4.1  -n tap-install -f source-controller-values.yaml
\ Installing package 'controller.source.apps.tanzu.vmware.com'
| Getting package metadata for 'controller.source.apps.tanzu.vmware.com'
| Creating service account 'source-controller-default-sa'
| Creating cluster admin role 'source-controller-default-cluster-role'
| Creating cluster role binding 'source-controller-default-cluster-rolebinding'
| Creating secret 'source-controller-default-values'
| Creating package resource
- Waiting for 'PackageInstall' reconciliation for 'source-controller'
- 'PackageInstall' resource install status: Reconciling



 Added installed package 'source-controller'
```

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
- Retrieving installation details for source-controller...
NAME:                    source-controller
PACKAGE-NAME:            controller.source.apps.tanzu.vmware.com
PACKAGE-VERSION:         0.4.1
STATUS:                  Reconcile succeeded
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

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

```
---
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. Support is planned for a future release.
4. `LATEST` - metaversion defined in `maven-metadata.xml (versioning/latest)`. Support is planned for a future release.
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 may change at any time. The `.spec.interval` defines how frequently to check for updated artifacts.

Developer Conventions overview

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 Tanzu apps workload apply.
- 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 Tanzu apps workload apply.
- 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

Developer Conventions overview

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 Tanzu apps workload apply.
- 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 Tanzu apps workload apply.
- 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.

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-name 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 Uninstalling Tanzu Application Platform packages. The package name for developer conventions is `developer-conventions`.

Overview of Learning Center for Tanzu Application Platform

Learning Center provides a platform for creating and self-hosting workshops. It allows you to create workshops from markdown files that are displayed to the learner in a terminal shell environment with an instructional wizard UI.

The UI can embed slide content, an integrated development environment (IDE), a web console for accessing the Kubernetes cluster, and other custom web applications.

Although Learning Center requires Kubernetes to run, and is used to teach users about Kubernetes, you can use it to host training for other purposes as well. For example, you can use it to help train users in web-based applications, use of databases, or programming languages, where the user has no interest or need for Kubernetes.

Use cases

Use case scenarios that Learning Center supports include:

- Supervised workshops. For example, a workshop run at a conference, at a customer site, or online. The workshop has a set time period and you know the maximum number of users to expect. After the training is complete, the Kubernetes cluster created for the workshop is destroyed.
- Temporary learning portal. This is for when you must provide access to a small set of workshops for a short duration for hands on demos at a conference vendor booth. Users select which topic they want to learn about and do that workshop. The workshop instance is created on demand. When they have finished the workshop, that workshop instance is destroyed to free up resources. After the conference has finished, the Kubernetes cluster is destroyed.
- Permanent learning portal. Similar to the temporary learning portal, but runs on an extended basis as a public website where anyone can come and learn at any time.
- Personal training or demos. This is where anyone who wants to run a workshop on their own Kubernetes cluster to learn that topic, or where a product demo was packaged up as a workshop and they want to use it to demonstrate the product to a customer. You can destroy the workshop environment when complete, but there is no need for the cluster to be destroyed.

When running workshops, wherever possible a shared Kubernetes cluster reduces the amount of setup required. This works for developer-focused workshops, becauses it is usually not necessary to provide elevated access to the Kubernetes cluster, and you can use role-based access controls (RBAC) to prevent users from interfering with each other. You can also set quotas so users are restricted as to how much resources they can use.

When you run workshops that deal with cluster operations, for which users need cluster admin access, create a separate cluster for each user. Learning Center doesn't deal with provisioning clusters, only with deploying a workshop environment in a cluster after it exists.

Use case requirements

In implementing to the preceding scenarios, the primary requirements related to creation of workshop content, and what you can do at runtime, are as follows:

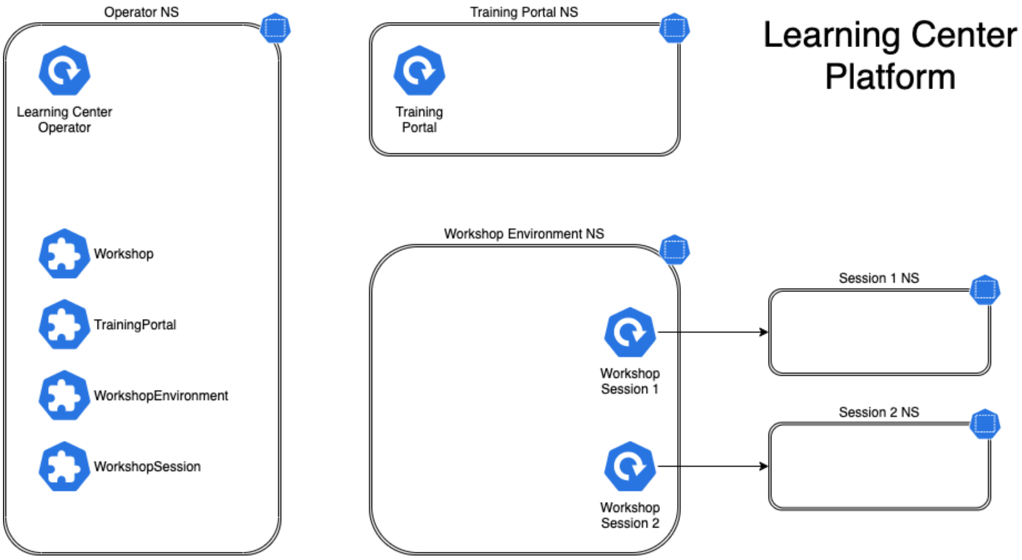
- You must store everything for the workshop in a Git repository, with no dependency on using a special web application or service to create a workshop.
- Use GitHub as a means to distribute workshop content. Alternatively, you can distribute the workshop as a container image. The latter is necessary if special tools must be installed for use in a workshop.
- Provide instructions to the user to complete the workshop as Markdown or AsciiDoc files.
- You can annotate instructions as executable commands so that when clicked in the workshop dashboard, they execute for the user in the appropriate terminal to avoid mistakes when commands are entered manually.
- You can annotate text as copyable so when clicked in the workshop dashboard, it is copied into the browser paste buffer ready for pasting into the terminal or other web application.
- Provide each user access to one or more namespaces in the Kubernetes cluster unique to their session. For Kubernetes based workshops, this is where applications are deployed as part of the workshop.

- You can create additional Kubernetes resources specific to a workshop session in advance of the session. This enables the deployment of applications for each user session.
- You can deploy additional Kubernetes resources common to all workshop sessions when the workshop environment is first created. This enables deployment of applications shared by all users.
- Apply resource quotas on each workshop session to control how much resources users can consume.
- Apply role-based access control (RBAC) on each workshop session to control what users can do.
- Provide access to an editor (IDE) in the workshop dashboard in the web browser for users to edit files during the workshop.
- Provide access to a web-based console for accessing the Kubernetes cluster. Use of the Kubernetes dashboard or Octant is supported.
- Ability to integrate additional web-based applications into the workshop dashboard specific to the topic of the workshop.
- Ability for the workshop dashboard to display slides used by an instructor in support of the workshop.

Platform architectural overview

The Learning Center relies on a Kubernetes Operator to perform the bulk of the work. The actions of the operator are controlled by using a set of custom resources specific to the Learning Center.

There are multiple ways of using the custom resources to deploy workshops. The primary way is to create a training portal, which in turn then triggers the setup of one or more workshop environments, one for each distinct workshop. When users access the training portal and select the workshop they want to do, the training portal allocates to that user a workshop session (creating one if necessary) against the appropriate workshop environment, and the user is redirected to that workshop session instance.



You can associate each workshop session with one or more Kubernetes namespaces specifically for use during that session. Role based access control (RBAC) applied to the unique Kubernetes service account for that session ensures that the user can only access the namespaces and other resources that they are allowed to for that workshop.

In this scenario, the custom resource types that come into play are:

- `Workshop` - Provides the definition of a workshop. Preloaded by an admin into the cluster, it defines where the workshop content is hosted, or the location of a container image which bundles the workshop content and any additional tools required for the workshop. The definition also lists additional resources that must be created which are to be shared between all workshop sessions, or for each session, with details of resources quotas and access roles required by the workshop.

- `TrainingPortal` - Created by an admin in the cluster to trigger the deployment of a training portal. The training portal can provide access to one or more distinct workshops defined by a `Workshop` resource. The training portal provides a web based interface for registering for workshops and accessing them. It also provides a REST API for requesting access to workshops, allowing custom front ends to be created which integrate with separate identity providers and which provide an alternate means for browsing and accessing workshops.
- `WorkshopEnvironment` - Used by the training portal to trigger the creation of a workshop environment for a workshop. This causes the operator to set up a namespace for the workshop into which shared resources are deployed, and where the workshop sessions are run.
- `WorkshopSession` - Used by the training portal to trigger the creation of a workshop session against a specific workshop environment. This causes the operator to set up any namespaces specific to the workshop session and pre-create additional resources required for a workshop session. Workshop sessions can either be created up front in reserve, to be handed out when requested, or created on demand.

Next steps

Learn more about:

- Workshops
- Getting started with Learning Center
- Installing Learning Center
- Local install guides
- Air-gapped environment requirements

Overview of Learning Center for Tanzu Application Platform

Learning Center provides a platform for creating and self-hosting workshops. It allows you to create workshops from markdown files that are displayed to the learner in a terminal shell environment with an instructional wizard UI.

The UI can embed slide content, an integrated development environment (IDE), a web console for accessing the Kubernetes cluster, and other custom web applications.

Although Learning Center requires Kubernetes to run, and is used to teach users about Kubernetes, you can use it to host training for other purposes as well. For example, you can use it to help train users in web-based applications, use of databases, or programming languages, where the user has no interest or need for Kubernetes.

Use cases

Use case scenarios that Learning Center supports include:

- Supervised workshops. For example, a workshop run at a conference, at a customer site, or online. The workshop has a set time period and you know the maximum number of users to expect. After the training is complete, the Kubernetes cluster created for the workshop is destroyed.
- Temporary learning portal. This is for when you must provide access to a small set of workshops for a short duration for hands on demos at a conference vendor booth. Users select which topic they want to learn about and do that workshop. The workshop instance is created on demand. When they have finished the workshop, that workshop instance is

destroyed to free up resources. After the conference has finished, the Kubernetes cluster is destroyed.

- Permanent learning portal. Similar to the temporary learning portal, but runs on an extended basis as a public website where anyone can come and learn at any time.
- Personal training or demos. This is where anyone who wants to run a workshop on their own Kubernetes cluster to learn that topic, or where a product demo was packaged up as a workshop and they want to use it to demonstrate the product to a customer. You can destroy the workshop environment when complete, but there is no need for the cluster to be destroyed.

When running workshops, wherever possible a shared Kubernetes cluster reduces the amount of setup required. This works for developer-focused workshops, becauses it is usually not necessary to provide elevated access to the Kubernetes cluster, and you can use role-based access controls (RBAC) to prevent users from interfering with each other. You can also set quotas so users are restricted as to how much resources they can use.

When you run workshops that deal with cluster operations, for which users need cluster admin access, create a separate cluster for each user. Learning Center doesn’t deal with provisioning clusters, only with deploying a workshop environment in a cluster after it exists.

Use case requirements

In implementing to the preceding scenarios, the primary requirements related to creation of workshop content, and what you can do at runtime, are as follows:

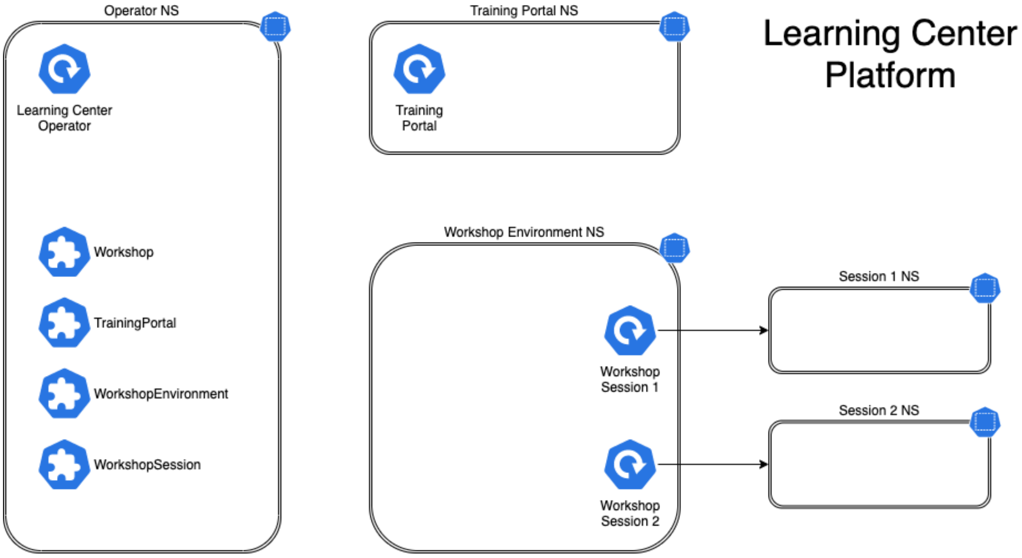
- You must store everything for the workshop in a Git repository, with no dependency on using a special web application or service to create a workshop.
- Use GitHub as a means to distribute workshop content. Alternatively, you can distribute the workshop as a container image. The latter is necessary if special tools must be installed for use in a workshop.
- Provide instructions to the user to complete the workshop as Markdown or AsciiDoc files.
- You can annotate instructions as executable commands so that when clicked in the workshop dashboard, they execute for the user in the appropriate terminal to avoid mistakes when commands are entered manually.
- You can annotate text as copyable so when clicked in the workshop dashboard, it is copied into the browser paste buffer ready for pasting into the terminal or other web application.
- Provide each user access to one or more namespaces in the Kubernetes cluster unique to their session. For Kubernetes based workshops, this is where applications are deployed as part of the workshop.
- You can create additional Kubernetes resources specific to a workshop session in advance of the session. This enables the deployment of applications for each user session.
- You can deploy additional Kubernetes resources common to all workshop sessions when the workshop environment is first created. This enables deployment of applications shared by all users.
- Apply resource quotas on each workshop session to control how much resources users can consume.
- Apply role-based access control (RBAC) on each workshop session to control what users can do.
- Provide access to an editor (IDE) in the workshop dashboard in the web browser for users to edit files during the workshop.
- Provide access to a web-based console for accessing the Kubernetes cluster. Use of the Kubernetes dashboard or Octant is supported.
- Ability to integrate additional web-based applications into the workshop dashboard specific to the topic of the workshop.

- Ability for the workshop dashboard to display slides used by an instructor in support of the workshop.

Platform architectural overview

The Learning Center relies on a Kubernetes Operator to perform the bulk of the work. The actions of the operator are controlled by using a set of custom resources specific to the Learning Center.

There are multiple ways of using the custom resources to deploy workshops. The primary way is to create a training portal, which in turn then triggers the setup of one or more workshop environments, one for each distinct workshop. When users access the training portal and select the workshop they want to do, the training portal allocates to that user a workshop session (creating one if necessary) against the appropriate workshop environment, and the user is redirected to that workshop session instance.



You can associate each workshop session with one or more Kubernetes namespaces specifically for use during that session. Role based access control (RBAC) applied to the unique Kubernetes service account for that session ensures that the user can only access the namespaces and other resources that they are allowed to for that workshop.

In this scenario, the custom resource types that come into play are:

- `Workshop` - Provides the definition of a workshop. Preloaded by an admin into the cluster, it defines where the workshop content is hosted, or the location of a container image which bundles the workshop content and any additional tools required for the workshop. The definition also lists additional resources that must be created which are to be shared between all workshop sessions, or for each session, with details of resources quotas and access roles required by the workshop.
- `TrainingPortal` - Created by an admin in the cluster to trigger the deployment of a training portal. The training portal can provide access to one or more distinct workshops defined by a `Workshop` resource. The training portal provides a web based interface for registering for workshops and accessing them. It also provides a REST API for requesting access to workshops, allowing custom front ends to be created which integrate with separate identity providers and which provide an alternate means for browsing and accessing workshops.
- `WorkshopEnvironment` - Used by the training portal to trigger the creation of a workshop environment for a workshop. This causes the operator to set up a namespace for the workshop into which shared resources are deployed, and where the workshop sessions are run.
- `WorkshopSession` - Used by the training portal to trigger the creation of a workshop session against a specific workshop environment. This causes the operator to set up any namespaces specific to the workshop session and pre-create additional resources required

for a workshop session. Workshop sessions can either be created up front in reserve, to be handed out when requested, or created on demand.

Next steps

Learn more about:

- Workshops
- Getting started with Learning Center
- Installing Learning Center
- Local install guides
- Air-gapped environment requirements

Install Learning Center

This topic describes how to install Learning Center 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 Learning Center. For more information about profiles, see Components and installation profiles.

To install Tanzu Learning Center, see the following sections.

For general information about Learning Center, see Learning Center. For information about deploying Learning Center operator, see Install and configure the Learning Center operator.

Prerequisites

Before installing Learning Center:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- The cluster must have an ingress router configured. If you have installed the Tanzu Application Platform package through the full profile or light profile, it already deploys a contour ingress controller.
- The operator, when deploying instances of the workshop environments, must be able to expose them through an external URL for access. For the custom domain you are using, DNS must have been configured with a wildcard domain to forward all requests for sub-domains of the custom domain to the ingress router of the Kubernetes cluster.
- By default, the workshop portal and workshop sessions are accessible over HTTP connections. If you wish to use secure HTTPS connections, you must have access to a wildcard SSL certificate for the domain under which you want to host the workshops. You cannot use a self-signed certificate.
- Any ingress routes created use the default ingress class if you have multiple ingress class types available and you must override which is used.

Install Learning Center

To install Learning Center:

1. List version information for the package by running:

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

Example output:

```
 NAME                             VERSION        RELEASED-AT
 learningcenter.tanzu.vmware.com  0.1.0          2021-12-01 08:18:48 -0500 EDT
```

2. (Optional) See all the configurable parameters on this package by running:

   Remember to change the 0.x.x version

   ```
   tanzu package available get learningcenter.tanzu.vmware.com/0.x.x --values-sche
   ma --namespace tap-install
   ```

3. Create a config file named `learning-center-config.yaml`.

4. To override the `shared.ingress_domain` in the values file of Tanzu Application Platform, add the parameter `ingressDomain` to `learning-center-config.yaml`. For example:

   ```
   ingressDomain: YOUR-INGRESS-DOMAIN
   ```

   Where `YOUR-INGRESS-DOMAIN` is the domain name for your Kubernetes cluster.

   When deploying workshop environment instances, the operator must be able to expose the instances through an external URL. You need this access to discover the domain name that can be used as a suffix to host names for instances.

   For the custom domain you are using, DNS must have been configured with a wildcard domain to forward all requests for sub-domains of the custom domain to the ingress router of the Kubernetes cluster.

   If you are running Kubernetes on your local machine using a system such as `minikube` and you don't have a custom domain name that maps to the IP address for the cluster, you can use a `nip.io` address. For example, if `minikube ip` returns `192.168.64.1`, you can use the `192.168.64.1.nip.io` domain. You cannot use an address of form `127.0.0.1.nip.io` or `subdomain.localhost`. This causes a failure. Internal services needing to connect to each other connect to themselves instead, because the address resolves to the host loopback address of `127.0.0.1`.

5. Add the `ingressSecret` to `learning-center-config.yaml`, as in this example:

   ```
     ingressSecret:
     certificate: |
       -----BEGIN CERTIFICATE-----
       MIIFLTCCBBWgAwIBAgaSAys/V2NCTG9uXa9aAiYt7WJ3MA0GCSqGaIb3DQEBCwUA
                                       ...
       dHa6Ly9yMy5vamxlbmNyLm9yZzAiBggrBgEFBQawAoYWaHR0cDoaL3IzLmkubGVu
       -----END CERTIFICATE-----
     privateKey: |
       -----BEGIN PRIVATE KEY-----
       MIIEvQIBADAaBgkqhkiG9waBAQEFAASCBKcwggSjAgEAAoIBAaCx4nyc2xwaVOzf
                                       ...
       IY/9SatMcJZivH3F1a7SXL98PawPIOSR7986P7rLFHzNjaQQ0DWTaXBRt+oUDxpN
       -----END PRIVATE KEY-----
   ```

   If you already have a TLS secret, follow these steps **before deploying any workshop**: - Create the `learningcenter` namespace manually or the one you defined - Copy the TLS secret to the `learningcenter` namespace or the one you defined and use the `secretName` property as in this example:

   ```
   ingressSecret:
    secretName: workshops.example.com-tls
   ```

   By default, the workshop portal and workshop sessions are accessible over HTTP connections.

   To use secure HTTPS connections, you must have access to a wildcard SSL certificate for the domain under which you want to host the workshops. You cannot use a self-signed certificate.

You can create wildcard certificates by using letsencrypt https://letsencrypt.org/_. After you have the certificate, you can define the `certificate` and `privateKey` properties under the `ingressSecret` property to specify the certificate on the configuration YAML.

6. Any ingress routes created use the default ingress class. If you have multiple ingress class types available, and you need to override which is used, define the `ingressClass` property in `learning-center-config.yaml` **before deploying any workshop**:

```
ingressClass: contour
```

7. Install Learning Center operator by running:

   Remember to change the 0.x.x version

```
tanzu package install learning-center --package-name learningcenter.tanzu.vmware.com --version 0.x.x -f learning-center-config.yaml
```

   The preceding command creates a default namespace in your Kubernetes cluster called `learningcenter`, and the operator, and any required namespaced resources, are created in it. A set of custom resource definitions and a global cluster role binding are also created.

   You can confirm that the operator deployed successfully by running:

```
kubectl get all -n learningcenter
```

   The pod for the operator should be marked as running.

Install the Self-Guided Tour Training Portal and Workshop

To install the Self-Guided Tour Training Portal and Workshop:

1. Confirm you have the workshop package installed by running:

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

2. Install the Learning Center Training Portal with the Self-Guided Tour Workshop by running:

   Remember to change the 0.x.x version

```
tanzu package install learning-center-workshop --package-name workshops.learningcenter.tanzu.vmware.com --version 0.x.x -n tap-install
```

3. Check for the Training Portals available in your environment by running:

```
kubectl get trainingportals
```

   Example output:

```
NAME                         URL                                                   ADMINU
SERNAME          ADMINPASSWORD                           STATUS
     learningcenter-tutorials   http://learningcenter-tutorials.example.com    le
arningcenter         QGBaM4CF01toPiZLW5NrXTcIYSpw2UJK    Running
```

Supported Learning Center Values Configuration

Admins are provided the following sample learning-center-config.yaml file to see the possible configurations supported by Learning Center. These configurations are additional ones that admins can provide to the operator resource but are by no means necessary for Learning Center to work. It is enough to follow the previous instructions on this page for Learning Center to run.

It is important to note that Learning Center has default values in place for the learning-center-config.yaml file. Admins only need to provide the values they want to override. As in the example above, overriding the ingressDomain property is enough to get Learning Center to work.

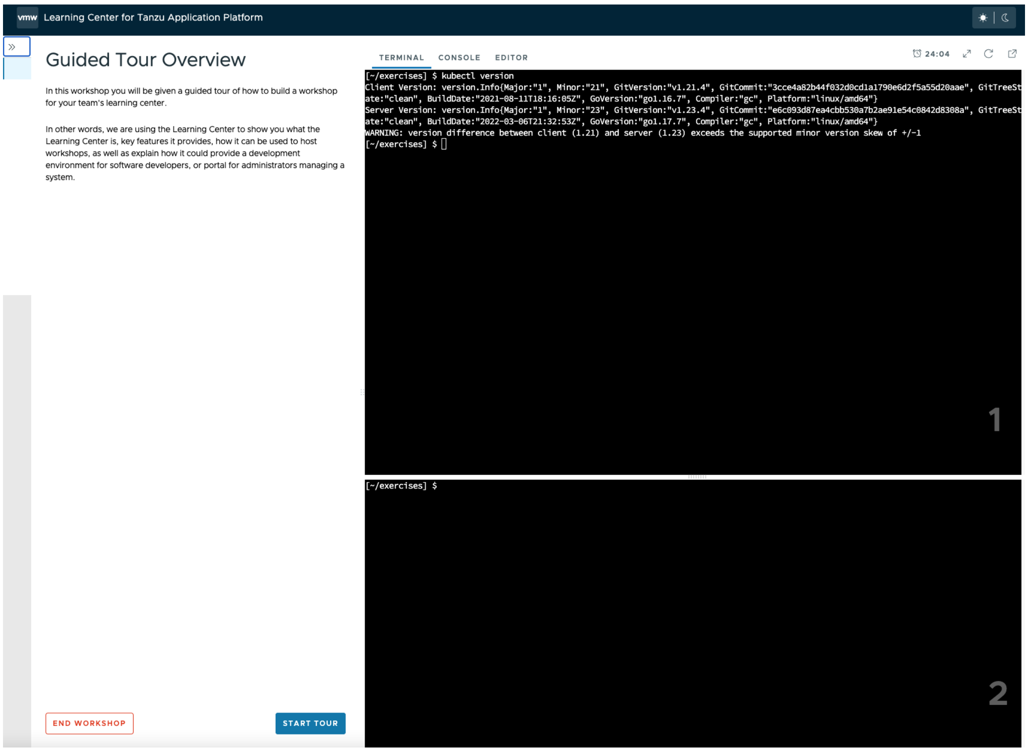
```
#! The namespace in which to deploy Learning Center. For now this must be "learningcen
ter" as
namespace: learningcenter
#! DNS parent subdomain used for training portal and workshop ingresses.
ingressDomain: workshops.example.com
#! Ingress class for where multiple ingress controllers exist and need to
#! use that which is not marked as the default.
ingressClass: null
#! SSL certificate for secure ingress. This must be a wildcard certificate for
#! children of DNS parent ingress subdomain.
ingressSecret:
  certificate: null
  privateKey: null
  secretName: null
#! Configuration for persistent volumes. The default storage class specified
#! by the cluster is used if not defined. You might need to set storage group
#! where a cluster has pod security policies enabled, usually
#! to one. Set storage user and storage group in exceptional cases
#! where storage class uses maps to NFS storage and storage server requires
#! that a specific user and group always be used.
storageClass: null
storageUser: null
storageGroup: null
#! Credentials for accessing training portal instances. If not specified,
#! random passwords are generated that you can obtain from the custom resource
#! for the training portal.
portalCredentials:
  systemAdmin:
    username: learningcenter
    password: null
  clientAccess:
    username: robot@learningcenter
    password: null
#! Container image versions for various components of Learning Center. The Learning Ce
nter
#! operator needs to be modified to read names of images for the registry
#! and docker-in-docker from config map to enable disconnected install.
#! Prepull images to nodes in cluster. Should be an empty list if no images
#! should be prepulled. Normally you would only want to prepull workshop
#! images. This is done to reduce start-up times for sessions.
prepullImages: ["base-environment"]
#! Docker daemon settings when building docker images in a workshop is
#! enabled. Proxy cache provides a way of partially getting around image
#! pull limits for Docker Hub image registry, with the remote URL being
#! set to "https://registry-1.docker.io".
dockerDaemon:
  networkMTU: 1500
  proxyCache:
    remoteURL: null
    username: null
    password: null
#! Used to restrict access to IP addresses or IP subnets. This must be a CIDR block ra
nge corresponding to the subnet or a portion of a
#! subnet you want to block. A Kubernetes `NetworkPolicy` is used to enforce the restr
iction. So the
#! Kubernetes cluster must use a network layer supporting network policies, and the ne
cessary Kubernetes
#! controllers supporting network policies must be enabled when the cluster is install
ed.
network:
  blockCIDRs:
  - 169.254.169.254/32
  - fd00:ec2::254/128
```

See Restricting Network Access for more information on blocking CIDRs.

Learning Center workshops

This topic gives you an overview of Learning Center workshops.

The Learning Center workshop dashboard comprises a set of workshop instructions on the left-hand side and a series of tabbed views on the right-hand side. For workshops that require users to run commands, one or more terminal shells are provided. For more information about workshops including creating your own, see Create workshops.



Guided Tour Overview

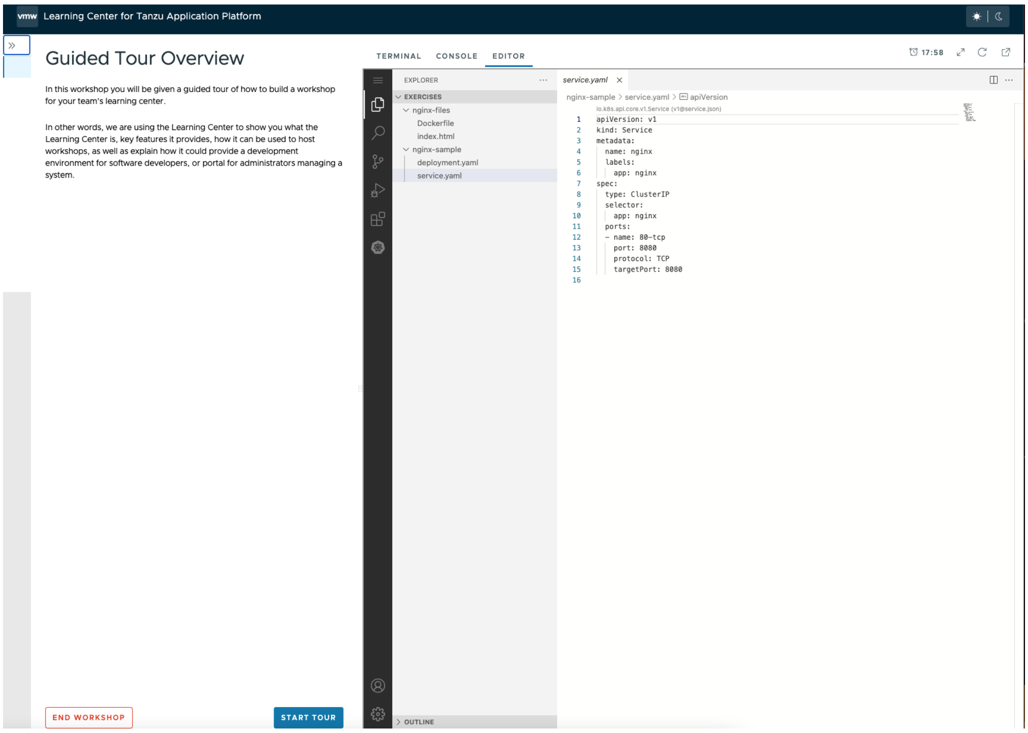
In this workshop you will be given a guided tour of how to build a workshop for your team's learning center.

In other words, we are using the Learning Center to show you what the Learning Center is, key features it provides, how it can be used to host workshops, as well as explain how it could provide a development environment for software developers, or portal for administrators managing a system.

```
[~/exercises] $ kubectl version
Client Version: version.Info{Major:"1", Minor:"21", GitVersion:"v1.21.4", GitCommit:"3cce4a82b44f032d0cd1a1790e6d2f5a55d20aae", GitTreeSt
ate:"clean", BuildDate:"2021-08-11T18:16:05Z", GoVersion:"go1.16.7", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"23", GitVersion:"v1.23.4", GitCommit:"e6c093d87ea4cbb530a7b2ae91e54c0842d8308a", GitTreeSt
ate:"clean", BuildDate:"2022-03-06T21:32:53Z", GoVersion:"go1.17.7", Compiler:"gc", Platform:"linux/amd64"}
WARNING: version difference between client (1.21) and server (1.23) exceeds the supported minor version skew of +/-1
[~/exercises] $
```

```
[~/exercises] $
```

The terminals provide access to the editors `vi` and `nano`. To provide a UI based editor, you can enable the embedded editor view and use the embedded IDE based on VS Code.



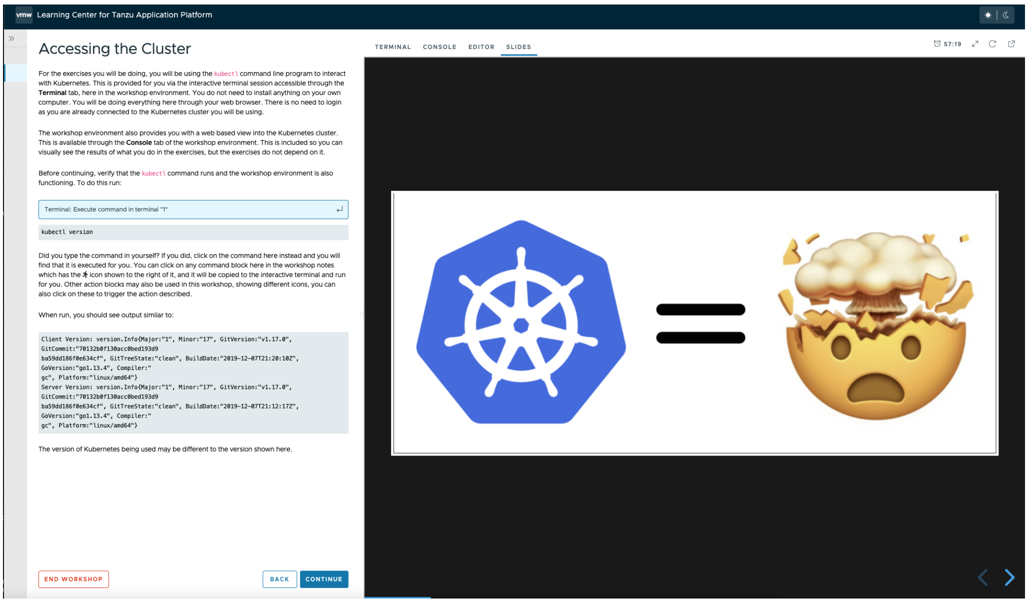
Guided Tour Overview

In this workshop you will be given a guided tour of how to build a workshop for your team's learning center.

In other words, we are using the Learning Center to show you what the Learning Center is, key features it provides, how it can be used to host workshops, as well as explain how it could provide a development environment for software developers, or portal for administrators managing a system.

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  type: ClusterIP
  selector:
    app: nginx
  ports:
  - name: 80-tcp
    port: 8080
    protocol: TCP
    targetPort: 8080
```

To complement the workshop instructions, or to be available for use by the instructor, you can include slides with a workshop. For slides you can use HTML based slide presentation tools such as `reveal.js`, or you can embed a PDF file.



Accessing the Cluster

For the exercises you will be doing, you will be using the `kubectl` command line program to interact with Kubernetes. This is provided for you via the interactive terminal session accessible through the **Terminal** tab, here in the workshop environment. You do not need to install anything on your own computer. You will be doing everything here through your web browser. There is no need to login as you are already connected to the Kubernetes cluster you will be using.

The workshop environment also provides you with a web based view into the Kubernetes cluster. This is available through the **Console** tab of the workshop environment. This is included so you can visually see the results of what you do in the exercises, but the exercises do not depend on it.

Before continuing, verify that the `kubectl` command runs and the workshop environment is also functioning. To do this run:

Terminal: Execute command in terminal "1"

```
kubectl version
```

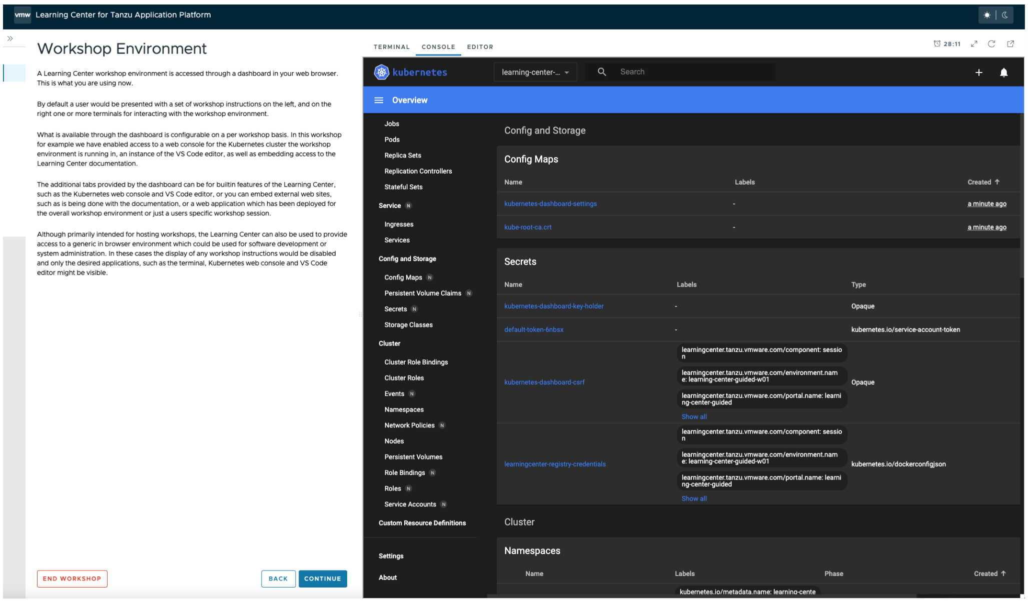
Did you type the command in yourself? If you did, click on the command here instead and you will find that it is executed for you. You can click on any command block here in the workshop notes which has the 🏃 icon shown to the right of it, and it will be copied to the interactive terminal and run for you. Other action blocks may also be used in this workshop, showing different icons, you can also click on these to trigger the action described.

When run, you should see output similar to:

```
Client Version: version.Info{Major:"1", Minor:"17", GitVersion:"v1.17.0",
GitCommit:"70132b0f130acc0bed193d9
ba59dd186f0e634cf", GitTreeState:"clean", BuildDate:"2019-12-07T21:20:10Z",
GoVersion:"go1.13.4", Compiler:"
gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"17", GitVersion:"v1.17.0",
GitCommit:"70132b0f130acc0bed193d9
ba59dd186f0e634cf", GitTreeState:"clean", BuildDate:"2019-12-07T21:12:17Z",
GoVersion:"go1.13.4", Compiler:"
gc", Platform:"linux/amd64"}
```

The version of Kubernetes being used may be different to the version shown here.

If the workshop involves working with Kubernetes, you can enable a web console for accessing the Kubernetes cluster. The default web console uses the Kubernetes dashboard.



Workshop Environment

A Learning Center workshop environment is accessed through a dashboard in your web browser. This is what you are using now.

By default a user would be presented with a set of workshop instructions on the left, and on the right one or more terminals for interacting with the workshop environment.

What is available through the dashboard is configurable on a per workshop basis. In this workshop for example we have enabled access to a web console for the Kubernetes cluster the workshop environment is running in, an instance of the VS Code editor, as well as embedding access to the Learning Center documentation.

The additional tabs provided by the dashboard can be for builtin features of the Learning Center, such as the Kubernetes web console and VS Code editor, or you can embed external web sites, such as is being done with the documentation, or a web application which has been deployed for the overall workshop environment or just a users specific workshop session.

Although primarily intended for hosting workshops, the Learning Center can also be used to provide access to a generic in browser environment which could be used for software development or system administration. In these cases the display of any workshop instructions would be disabled and only the desired applications, such as the terminal, Kubernetes web console and VS Code editor might be visible.

Config and Storage

Config Maps

Name	Labels	Created ↑
kubernetes-dashboard-settings	-	a minute ago
kube-root-ca.crt	-	a minute ago

Secrets

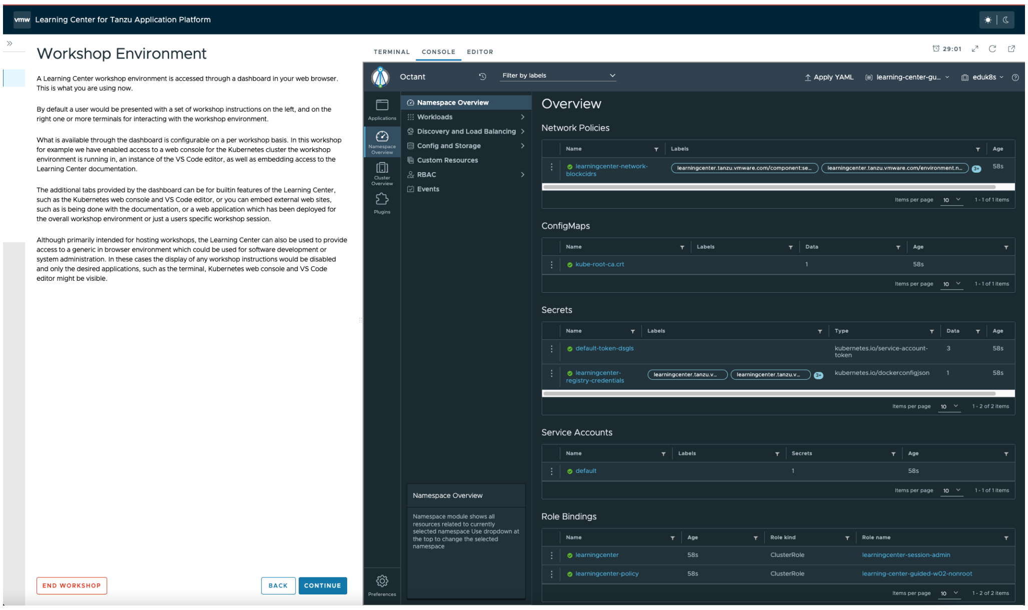
Name	Labels	Type
kubernetes-dashboard-key-holder	-	Opaque
default-token-6nbsx	-	kubernetes.io/service-account-token
kubernetes-dashboard-csrf	learningcenter.tanzu.vmware.com/component: session learningcenter.tanzu.vmware.com/environment.name: learning-center-guided-w01 learningcenter.tanzu.vmware.com/portal.name: learning-center-guided Show all	Opaque
learningcenter-registry-credentials	learningcenter.tanzu.vmware.com/component: session learningcenter.tanzu.vmware.com/environment.name: learning-center-guided-w01 learningcenter.tanzu.vmware.com/portal.name: learning-center-guided Show all	kubernetes.io/dockerconfigjson

Cluster

Namespaces

Name	Labels	Phase	Created ↑
	kubernetes.io/metadata.name: learning-cente		

Alternatively, you can enable Octant as the web console.



Workshop Environment

A Learning Center workshop environment is accessed through a dashboard in your web browser. This is what you are using now.

By default a user would be presented with a set of workshop instructions on the left, and on the right one or more terminals for interacting with the workshop environment.

What is available through the dashboard is configurable on a per workshop basis. In this workshop for example we have enabled access to a web console for the Kubernetes cluster the workshop environment is running in, an instance of the VS Code editor, as well as embedding access to the Learning Center documentation.

The additional tabs provided by the dashboard can be for builtin features of the Learning Center, such as the Kubernetes web console and VS Code editor, or you can embed external web sites, such as is being done with the documentation, or a web application which has been deployed for the overall workshop environment or just a users specific workshop session.

Although primarily intended for hosting workshops, the Learning Center can also be used to provide access to a generic in browser environment which could be used for software development or system administration. In these cases the display of any workshop instructions would be disabled and only the desired applications, such as the terminal, Kubernetes web console and VS Code editor might be visible.

Overview

Network Policies

Name	Labels	Age
learningcenter-network-blockcidrs	learningcenter.tanzu.vmware.com/component:se... learningcenter.tanzu.vmware.com/environment.n... 3+	58s

Items per page 10 1 - 1 of 1 items

ConfigMaps

Name	Labels	Data	Age
kube-root-ca.crt		1	58s

Items per page 10 1 - 1 of 1 items

Secrets

Name	Labels	Type	Data	Age
default-token-dsgls		kubernetes.io/service-account-token	3	58s
learningcenter-registry-credentials	learningcenter.tanzu.v... learningcenter.tanzu.v... 3+	kubernetes.io/dockerconfigjson	1	58s

Items per page 10 1 - 2 of 2 items

Service Accounts

Name	Labels	Secrets	Age
default		1	58s

Items per page 10 1 - 1 of 1 items

Role Bindings

Name	Age	Role kind	Role name
learningcenter	58s	ClusterRole	learningcenter-session-admin
learningcenter-policy	58s	ClusterRole	learning-center-guided-w02-nonroot

Items per page 10 1 - 2 of 2 items

Namespace Overview

Namespace module shows all resources related to currently selected namespace Use dropdown at the top to change the selected namespace

Getting started with Learning Center

This topic describes how you can get started with Learning Center for Tanzu Application Platform. For information about Learning Center and its use cases, see Learning Center for Tanzu Application Platform.

Installing Learning Center

Before deploying workshops, you must install a Kubernetes operator for Learning Center. The operator manages the setup of the environment for each workshop and deploys instances of a workshop for each person.

For basic information about installing the Learning Center operator, see Install Learning Center.

Get started

See the following useful information about getting started with Learning Center:

- Install and configure the Learning Center operator
- Get started with workshops
- Get started with training portals
- Delete an operator

Getting started with Learning Center

This topic describes how you can get started with Learning Center for Tanzu Application Platform. For information about Learning Center and its use cases, see Learning Center for Tanzu Application Platform.

Installing Learning Center

Before deploying workshops, you must install a Kubernetes operator for Learning Center. The operator manages the setup of the environment for each workshop and deploys instances of a workshop for each person.

For basic information about installing the Learning Center operator, see Install Learning Center.

Get started

See the following useful information about getting started with Learning Center:

- Install and configure the Learning Center operator
- Get started with workshops
- Get started with training portals
- Delete an operator

Configure the Learning Center operator

This topic gives you information about installing and configuring the Learning Center operator.

Before deploying workshops, you must install a Kubernetes operator for Learning Center. The operator manages the setup of the environment for each workshop and deploys instances of a workshop for each person.

For basic information about installing the operator, see Install Learning Center.

Installing and setting up Learning Center operator

You can deploy the Learning Center operator to any Kubernetes cluster supporting custom resource definitions and the concept of operators. The cluster must have an ingress router configured, though only a basic deployment of the ingress controller is usually required. You do not need to configure the ingress controller to handle cluster wide edge termination of secure HTTP connections. Learning Center creates Kubernetes Ingress resources and supplies any secret for use with secure HTTP connections for each ingress.

For the ingress controller, VMware recommends the use of Contour over alternatives such as nginx. An nginx-based ingress controller has a less than optimal design. Every time a new ingress is created or deleted, the nginx config is reloaded. This causes websocket connections to terminate after a period of time. Learning Center terminals reconnect automatically in the case of the websocket connection being lost. However, not all applications you might use with specific workshops can handle loss of websocket connections so gracefully, and they might be impacted due to the use of an nginx ingress controller. This problem is not specific to Learning Center. It can impact any application when an nginx ingress controller is used frequently and ingresses are created or deleted frequently.

You can use a hosted Kubernetes solution from an IaaS provider such as Google, AWS, or Azure. If you do, as needed, increase any HTTP request timeout specified on the inbound load balancer for the ingress controller so that you can use long-lived websocket connections. In some cases, load balancers of hosted Kubernetes solutions only have a 30-second timeout. If possible, configure the timeout applying to websockets to be 1 hour.

If you deploy the web-based training portal, the cluster must have available persistent volumes of type `ReadWriteOnce (RWO)`. A default storage class must be defined so that persistent volume claims do not need to specify a storage class. For some Kubernetes distributions, including from IBM, you must configure Learning Center as to what user and group must be used for persistent volumes. If no default storage class is specified, or a specified storage class is required, you can configure Learning Center with the name of the storage class.

To install the Learning Center operator, you must have cluster admin access.

Cluster pod security policies

The Learning Center operator defines pod security policies to limit what users can do from workshops when deploying workloads to the cluster. The default policy prohibits running of images as the `root` user or using a privileged pod. Specified workshops can relax these restrictions and apply a policy that enables additional privileges required by the workshop.

VMware recommends that the pod security policy admission controller be enabled for the cluster to ensure that the pod security policies are applied. If the admission controller is not enabled, users can deploy workloads that run as the `root` user in a container, or run privileged pods.

If you are unable to enable the pod security policy admission controller, you should only provide access to workshops deployed using the Learning Center operator to users you trust.

Whether the absence of the pod security policy admission controller causes issues with access to persistent volumes depends on the cluster. Although minikube does not enable the pod security policy admission controller, it works as persistent volumes when mounted to give write permissions to all users.

No matter whether pod security policies are enabled, individual workshops must be reviewed as to what added privileges they grant before allowing their use in a cluster.

Specifying the ingress domain

When deploying instances of workshop environments, the operator must expose the instances by using an external URL for access to define the domain name that is used as a suffix to host names for instances.

Note: For the custom domain you are using, configure your DNS with a wildcard domain to forward all requests for subdomains of the custom domain to the ingress router of the Kubernetes cluster.

VMware recommends that you avoid using a `.dev` or `.app` domain name, because such domain names require browsers to use HTTPS and not HTTP. Although you can provide a certificate for secure connections under the domain name for use by Learning Center, this doesn’t extend to what a workshop may do. If workshop instructions require that you create ingresses in Kubernetes using HTTP only, a `.dev` or `.app` domain name cannot work in the browser.

Note: If you are running Kubernetes on your local machine using a system such as `minikube` and you don’t have a custom domain name that maps to the IP address for the cluster, you can use a `nip.io` address. For example, if `minikube ip` returned `192.168.64.1`, you can use the 192.168.64.1.nip.io domain. You cannot use an address of form `127.0.0.1.nip.io` or `subdomain.localhost`. This causes a failure as internal services needing to connect to each other end up connecting to themselves instead, because the address resolves to the host loopback address of `127.0.0.1`.

If needed, you can override the `shared.ingress_domain` in the values file of Tanzu Application Platform with the `ingressDomain` parameter of learning center:

```
ingressDomain: learningcenter.my-domain.com
```

Set the environment variable manually

Set the `INGRESS_DOMAIN` environment variable on the operator deployment. To set the `INGRESS_DOMAIN` environment variable, run:

```
kubectl set env deployment/learningcenter-operator -n learningcenter INGRESS_DOMAIN=test
```

Where `test` is the domain name for your Kubernetes cluster.

Or if using a `nip.io` address:

```
kubectl set env deployment/learningcenter-operator -n learningcenter INGRESS_DOMAIN=192.168.64.1.nip.io
```

Use of environment variables to configure the operator is a shortcut for a simple use. VMware recommends using Tanzu CLI, or for more complicated scenarios, you can use the `SystemProfile` custom resource.

Enforcing secure connections

By default, the workshop portal and workshop sessions are accessible over HTTP connections. To use secure HTTPS connections, you must have access to a wildcard SSL certificate for the domain under which you want to host the workshops. You cannot use a self-signed certificate.

You can create wildcard certificates by using letsencrypt <https://letsencrypt.org/>. After you have the certificate, you can define it as follows.

Configuration YAML

The easiest way to define the certificate is with the configuration passed to Tanzu CLI. So define the certificate and privateKey properties under the ingressSecret property to specify the certificate on the configuration YAML passed to Tanzu CLI:

```
ingressSecret:
  certificate: |
    -----BEGIN CERTIFICATE-----
    MIIFLTCCBBWgAwIBAgaSAys/V2NCTG9uXa9aAiYt7WJ3MA0GCSqGaIb3DQEBCwUA
                                ...
    dHa6Ly9yMy5vamxlbmNyLm9yZzAiBggrBgEFBQawAoYWaHR0cDoaL3IzLmkubGVu
    -----END CERTIFICATE-----
  privateKey: |
    -----BEGIN PRIVATE KEY-----
    MIIEvQIBADAaBgkqhkiG9waBAQEFAASCBKcwggSjAgEAAoIBAaCx4nyc2xwaVOzf
                                ...
    IY/9SatMcJZivH3F1a7SXL98PawPIOSR7986P7rLFHzNjaQQ0DWTaXBRt+oUDxpN
    -----END PRIVATE KEY-----
```

If you already have a TLS secret, follow these steps **before deploying any workshops**:

1. Create the learningcenter namespace manually or the one you defined.
2. Copy the TLS secret to the learningcenter namespace or to the one you defined, and use the secretName property as in this example:

   ```
   ingressSecret:
     secretName: workshops.example.com-tls
   ```

Create the TLS secret manually

To add the certificate as a secret in the learningcenter namespace or in the one you defined, the secret must be of type tls. You can create it using the kubectl create secret tls command:

```
kubectl create secret tls -n learningcenter workshops.example.com-tls --cert=workshop
s.example.com/fullchain.pem --key=workshops.example.com/privkey.pem
```

Having created the secret, if it is the secret corresponding to the default ingress domain you specified earlier, set the INGRESS_SECRET environment variable. This way you do not use the configuration passed to Tanzu CLI on the operator deployment. This ensures the secret is applied automatically to any ingress created:

```
kubectl set env deployment/learningcenter-operator -n learningcenter INGRESS_SECRET=wo
rkshops.example.com-tls
```

If the certificate isn’t that of the default ingress domain, you can supply the domain name and name of the secret when creating a workshop environment or training portal. In either case, you must create secrets for the wildcard certificates in the learningcenter namespace or the one that you defined.

Specifying the ingress class

Any ingress routes created use the default ingress class. If you have multiple ingress class types available, and you must override which is used, you can define the ingressClass property on the configuration YAML as follows.

Configuration YAML

Define the ingressClass property on the configuration YAML passed to Tanzu CLI:

```
ingressClass: contour
```

Set the environment variable manually

Set the INGRESS_CLASS environment variable for the learningcenter operator:

```
kubectl set env deployment/learningcenter-operator -n learningcenter INGRESS_CLASS=con
tour
```

This applies only to the ingress created for the training portal and workshop sessions. It does not apply to any ingress created from a workshop as part of the workshop instructions.

This can be necessary when a specific ingress provider is not reliable in maintaining websocket connections. For example, in the case of the nginx ingress controller when there are frequent creation or deletions of ingresses occurring in the cluster. See the earlier section, Installing and setting up Learning Center operator.

Trusting unsecured registries

One of the options available for workshops is to automatically deploy a container image registry each workshop session. When the Learning Center operator is configured to use a secure ingress with a valid wildcard certificate, the image registry works out of the box.

If the Learning Center operator is not set up to use secure ingress, the image registry is accessed over HTTP and is regarded as not secure.

When using the optional support for building container images using docker, the docker daemon deployed for the workshop session is configured for the image registry being not secure yet pushing images to the image registry still works.

In this case of an image registry that is not secure, deploying images from the image registry to the Kubernetes cluster does not work unless the Kubernetes cluster is configured to trust the registry that is not secure.

How you configure a Kubernetes cluster to trust an unsecured registry varies based on how the Kubernetes cluster is deployed and what container runtime it uses.

If you are using minikube with dockerd, to ensure that the registry is trusted, you must set up the trust the first time you create the minikube instance.

To do this, first determine which IP subnet minikube uses for the inbound ingress router of the cluster. If you already have a minikube instance running, you can determine this by running minikube ip. If, for example, this reported 192.168.64.1, the subnet used is 129.168.64.0/24.

With this information, when you create a fresh minikube instance, you must supply the --insecure-registry option with the subnet:

```
minikube start --insecure-registry="129.168.64.0/24"
```

This option tells dockerd to regard as not secure any image registry deployed in the Kubernetes cluster and accessed through a URL exposed using an ingress route of the cluster itself.

Currently, there is no way to configure containerd to treat as not secure image registries that match a wildcard subdomain or reside in a subnet. It is therefore not possible to run workshops that must deploy images from the per session image registry when using containerd as the underlying Kubernetes cluster container runtime. This is a limitation of containerd, and there are no known plans for containerd to support this ability. This limits your ability to use Kubernetes clusters deployed with a tool such as kind, which relies on using containerd.

Get started with Learning Center workshops

This topic helps you to get started working with Learning Center workshops. Workshops are where you create your content. You can create a workshop for individual use or group multiple workshops together with a Training Portal.

For more detailed instructions, go to Working with Learning Center Workshops

Creating the workshop environment

With the definition of a workshop already in existence, the first step to deploying a workshop is to create the workshop environment.

To create the workshop environment run:

```
kubectl apply -f {YOUR-GIT-REPO-URL}/lab-k8s-fundamentals/main/resources/workshop-envi
ronment.yaml
```

This results in a custom resource being created called `WorkshopEnvironment`:

```
workshopenvironment.learningcenter.tanzu.vmware.com/lab-k8s-fundamentals created
```

The custom resource created is cluster-scoped, and the command needs to be run as a cluster admin or other appropriate user with permission to create the resource.

The Learning Center Operator reacts to the creation of this custom resource and initializes the workshop environment.

For each distinct workshop environment, a separate namespace is created. This namespace is used to hold the workshop instances. The namespace may also be used to provision any shared application services the workshop definition describes which would be used across all workshop instances. Such shared application services are automatically provisioned by the Learning Center Operator when the workshop environment is created.

You can list the workshop environments which have been created by running:

```
kubectl get workshopenvironments
```

This results in the output:

```
NAME                   NAMESPACE              WORKSHOP               IMAGE
URL
lab-k8s-fundamentals   lab-k8s-fundamentals   lab-k8s-fundamentals   {YOUR-REGISTRY-UR
L}/lab-k8s-fundamentals:main   {YOUR-GIT-REPO-URL}/lab-k8s-fundamentals
```

Additional fields give the name of the workshop environment, the namespace created for the workshop environment, and the name of the workshop the environment was created from.

Requesting a workshop instance

To request a workshop instance, a custom resource of type `WorkshopRequest` needs to be created.

This is a namespaced resource allowing who can create them to be delegated using role-based access controls. Further, in order to be able to request an instance of a specific workshop, you need to know the secret token specified in the description of the workshop environment. If necessary, raising requests against a specific workshop environment can also be constrained to a specific set of namespaces on top of any defined role-based access control (RBAC) rules.

In the context of an appropriate namespace, run:

```
kubectl apply -f {YOUR-GIT-REPO-URL}/lab-k8s-fundamentals/main/resources/workshop-requ
est.yaml
```

This should result in the output:

```
workshoprequest.learningcenter.tanzu.vmware.com/lab-k8s-fundamentals created
```

You can list the workshop requests in a namespace by running:

```
kubectl get workshoprequests
```

This displays output similar to:

```
NAME                   URL                                        USERNAME    PASSWORD
lab-k8s-fundamentals   http://lab-k8s-fundamentals-cvh51.test    learningcenter      buQ
OgZvfHM7m
```

The additional fields provide the URL where the workshop instance can be accessed and the username and password for you to provide when prompted by your web browser.

The user name and password only come into play when you use the lower-level resources to set up workshops. If you use the `TrainingPortal` custom resource, you will see that these fields are empty. This is because, for that case, the workshop instances are deployed so that they rely on user registration and access mediated by the web-based training portal. Visiting the URL for a workshop instance directly when using `TrainingPortal`, redirects you back to the web portal in order to log in if necessary.

You can monitor the progress of this workshop deployment by listing the deployments in the namespace created for the workshop environment:

```
kubectl get all -n lab-k8s-fundamentals
```

For each workshop instance a separate namespace is created for the session. This is linked to the workshop instance, and is where any applications are deployed as part of the workshop. If the definition of the workshop includes a set of resources that should be automatically created for each session namespace, they are created by the Learning Center Operator. It is therefore possible to pre-deploy applications for each session.

In this case, we used `WorkshopRequest`; whereas when using `TrainingPortal`, we created a `WorkshopSession`. The workshop request does result in creating a `WorkshopSession`, but `TrainingPortal` skips the workshop request and directly creates a `WorkshopSession`.

The purpose of having `WorkshopRequest` as a separate custom resource is to allow RBAC and other controls to be used to allow non-cluster administrators to create workshop instances.

Deleting the workshop instance

When you have finished with the workshop instance, you can delete it by deleting the custom resource for the workshop request:

```
kubectl delete workshoprequest/lab-k8s-fundamentals
```

Deleting the workshop environment

If you want to delete the whole workshop environment, it is recommended to first delete all workshop instances. Once this has been done, you can then delete the custom resource for the workshop environment:

```
kubectl delete workshopenvironment/lab-k8s-fundamentals
```

If you don’t delete the custom resources for the workshop requests, the workshop instances are still cleaned up and removed when the workshop environment is removed. The custom resources for the workshop requests still remain, however, and need to be deleted separately.

Get started with Learning Center training portals

This topic describes how you configure and use a `TrainingPortal`, which deploys a set of workshops for attendees.

Working with multiple workshops

The quickest way to deploy a set of workshops to use in a training session is to deploy a `TrainingPortal`. This deploys a set of workshops with one instance of each workshop for each attendee. A web-based portal is provided for registering attendees and allocating them to workshops.

The `TrainingPortal` custom resource provides a high-level mechanism for creating a set of workshop environments and populating it with workshop instances. When the Learning Center operator processes this custom resource, it creates other custom resources to trigger the creation of the workshop environment and the workshop instances. If you want more control, you can use these latter custom resources directly instead.

Loading the workshop definition

A custom resource of type `Workshop` describes each workshop. Before you can create a workshop environment, you must load the definition of the workshop.

Here is an example `Workshop` custom resource:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-k8s-fundamentals
spec:
  title: Kubernetes Fundamentals
  description: Workshop on getting started with Kubernetes
  url: {YOUR-GIT-REPO-URL}/lab-k8s-fundamentals
  vendor: learningcenter.io
  authors:
  - Graham Dumpleton
  difficulty: intermediate
  duration: 1h
  tags:
  - kubernetes
  content:
    image: projects.registry.vmware.com/learningcenter/lab-k8s-fundamentals:latest
  session:
    namespaces:
      budget: medium
    applications:
      terminal:
        enabled: true
        layout: split
      console:
        enabled: true
      editor:
        enabled: true
```

To load the definition of the workshop, run:

```
kubectl apply -f {YOUR-GIT-REPO-URL}/lab-k8s-fundamentals/main/resources/workshop.yaml
```

The custom resource created is cluster-scoped. The command must be run as a cluster admin or other appropriate user with permission to create the resource.

If successfully loaded, the command outputs:

```
workshop.learningcenter.tanzu.vmware.com/lab-k8s-fundamentals created
```

To list the workshop definitions that have been loaded and that can be deployed, run:

```
kubectl get workshops
```

For this workshop, this outputs:

```
NAME                  IMAGE                                                 FILES  URL
lab-k8s-fundamentals  {YOUR-REGISTRY-URL}/lab-k8s-fundamentals:main              {YOUR-GIT-
REPO-URL}/lab-k8s-fundamentals
```

The added fields in this case give:

- The name of the custom workshop container image deployed for the workshop.
- A URL for more information about the workshop.

The definition of a workshop is loaded as a step of its own, rather than referring to a remotely hosted definition. This allows a cluster admin to audit the workshop definition to ensure it isn't doing something the cluster admin doesn't want to allow. After the cluster admin approves the workshop definition, it can be used to create instances of the workshop.

Creating the workshop training portal

To deploy a workshop for one or more users, use the `TrainingPortal` custom resource. This custom resource specifies a set of workshops to be deployed and the number of people taking the workshops.

The `TrainingPortal` custom resource we use in this example is:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-k8s-fundamentals
spec:
  workshops:
  - name: lab-k8s-fundamentals
    capacity: 3
    reserved: 1
    expires: 1h
    orphaned: 5m
```

To create the custom resource, run:

```
kubectl apply -f {YOUR-GIT-REPO-URL}/lab-k8s-fundamentals/main/resources/training-port
al.yaml
```

The custom resource created is cluster-scoped. The command must be run as a cluster admin or other appropriate user with permission to create the resource.

This results in the output:

```
trainingportal.learningcenter.tanzu.vmware.com/lab-k8s-fundamentals created
```

There is actually much more going on than this. To see all the resources created, run:

```
kubectl get learningcenter-training -o name
```

You should see:

```
workshop.learningcenter.tanzu.vmware.com/lab-k8s-fundamentals
trainingportal.learningcenter.tanzu.vmware.com/lab-k8s-fundamentals
workshopenvironment.learningcenter.tanzu.vmware.comlab-k8s-fundamentals-w01
workshopsession.learningcenter.tanzu.vmware.com/lab-k8s-fundamentals-w01-s001
```

In addition to the original `Workshop` custom resource providing the definition of the workshop, and the `TrainingPortal` custom resource you just created, you've also created the `WorkshopEnvironment` and `WorkshopSession` custom resources.

The `WorkshopEnvironment` custom resource sets up the environment for a workshop, including deploying any application services that must exist and are shared by all workshop instances.

The `WorkshopSession` custom resource results in the creation of a single workshop instance.

To see a list of the workshop instances created and their details, run:

```
kubectl get workshopsessions
```

This yields output similar to:

```
NAME                            URL                                          USERNAME
PASSWORD
lab-k8s-fundamentals-w01-s001   http://lab-k8s-fundamentals-w01-s001.test
```

Only one workshop instance is created. Though the maximum capacity is set to three, the reserved number of instances (hot spares) is defined as one. Additional workshops instances are only created as workshop sessions are allocated to users. One reserved instance is always maintained until capacity is reached.

If you need a different number of workshop instances, set the `portal.capacity` field of the `TrainingPortal` custom resource YAML input file before creating the resource. Changing the values after the resource is created has no effect.

In this case, only one workshop is listed to be hosted by the training portal. You can deploy more than one workshop at the same time by adding the names of other workshops to `workshops`.

The first time you deploy the workshop, it can take a few moments to pull down the workshop image and start.

To access the workshops, attendees of a training session need to visit the web-based portal for the training session. Find the URL for the web portal by running:

```
kubectl get trainingportals
```

This should yield output similar to:

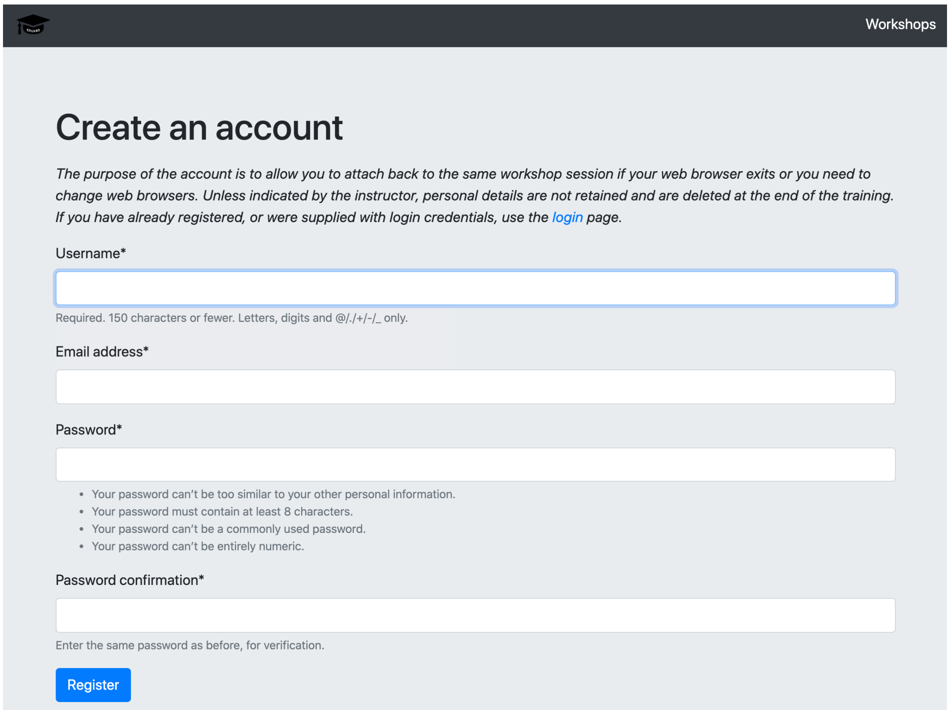
```
NAME                 URL                                   ADMINUSERNAME  ADMINPASSWO
RD
lab-k8s-fundamentals  https://lab-k8s-fundamentals-ui.test  learningcenter         mGI
2C1TkHEBoFgKiZetxMnwAldRU80aN
```

Attendees should only be given the URL. The password listed is only for use by the instructor of the training session if required.

Accessing workshops via the web portal

Attendees can access workshops through the web portal by following two steps:

1. The attendee visits the web-based portal for the training session and is presented with a login page. However, before logging in, the attendee must register for an account. The attendee clicks the link to the registration page and fills it in.



Create an account

The purpose of the account is to allow you to attach back to the same workshop session if your web browser exits or you need to change web browsers. Unless indicated by the instructor, personal details are not retained and are deleted at the end of the training. If you have already registered, or were supplied with login credentials, use the login page.

Username*

Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Email address*

Password*

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

Password confirmation*

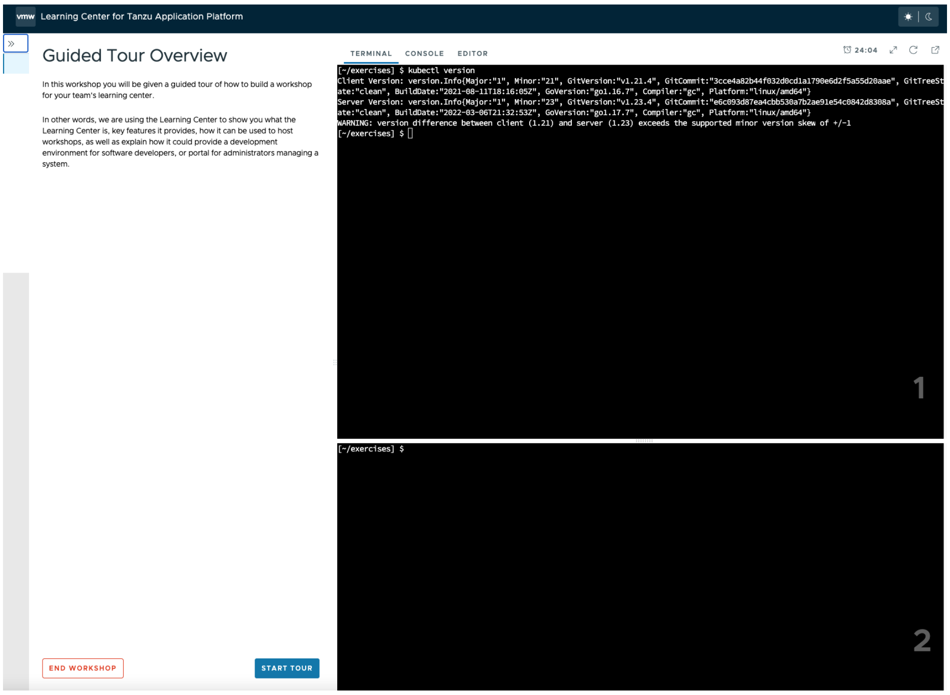
Enter the same password as before, for verification.

Registration is required so if the attendee's web browser exits or the attendee needs to switch web browsers, the attendee can log in again and access the same workshop instance.

2. Upon registering, the attendee is presented with a list of workshops available for the training session.

    - An orange dot beside a workshop means that no instance for that workshop has been allocated to the user as yet, but that some are available.
    - A red dot indicates there are no more workshop instances available.
    - A green dot indicates a workshop instance has already been reserved by the attendee.

    The attendee clicks the "Start workshop" button. This allocates a workshop instance if one hasn't yet been reserved and redirects the attendee to that workshop instance.



Guided Tour Overview

In this workshop you will be given a guided tour of how to build a workshop for your team's learning center.

In other words, we are using the Learning Center to show you what the Learning Center is, key features it provides, how it can be used to host workshops, as well as explain how it could provide a development environment for software developers, or portal for administrators managing a system.

```
[~/exercises] $ kubectl version
Client Version: version.Info{Major:"1", Minor:"21", GitVersion:"v1.21.4", GitCommit:"3cce4a82b44f032d0cd1a1790e6d2f5a55d20aae", GitTreeSt
ate:"clean", BuildDate:"2021-08-11T18:16:05Z", GoVersion:"go1.16.7", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"23", GitVersion:"v1.23.4", GitCommit:"e6c093d87ea4cbb530a7b2ae91e54c0842d8308a", GitTreeSt
ate:"clean", BuildDate:"2022-03-06T21:32:53Z", GoVersion:"go1.17.7", Compiler:"gc", Platform:"linux/amd64"}
WARNING: version difference between client (1.21) and server (1.23) exceeds the supported minor version skew of +/-1
[~/exercises] $
```

```
[~/exercises] $
```

Deleting the workshop training portal

The workshop training portal is intended for running workshops with a fixed time period where all workshop instances are deleted when complete.

To delete all workshop instances and the web-based portal, run:

```
kubectl delete trainingportal/lab-k8s-fundamentals
```

Deleting Learning Center

This topic describes how you can delete Learning Center.

1. Delete all current workshop environments by running:

    ```
    kubectl delete workshops,trainingportals,workshoprequests,workshopsessions,work
    shopenvironments --all
    ```

    Note: Ensure the Learning Center operator is still running when running this command.

2. Verify you have deleted all current workshop environments by running:

    ```
    kubectl get workshops,trainingportals,workshoprequests,workshopsessions,worksho
    ```

```
penvironments --all-namespaces
```

Note: This command does not delete the workshops in the `workshops.learningcenter.tanzu.vmware.com` package.

3. Uninstall the Learning Center package by running:

```
tanzu package installed delete {NAME_OF_THE_PACKAGE} -n tap-install
```

Note: This command also removes the added custom resource definitions and the `learningcenter` namespace.

Note: If you have installed the Tanzu Application Platform package, Learning Center will be recreated.

4. To remove the Learning Center package, add the following lines to your `tap-values` file.

```
excluded_packages:
- learningcenter.tanzu.vmware.com
- workshops.learningcenter.tanzu.vmware.com
```

Learning Center local install guides

The following topics describe how you install Learning Center on your local environment:

- Installing on Kind
- Installing on Minikube

Learning Center local install guides

The following topics describe how you install Learning Center on your local environment:

- Installing on Kind
- Installing on Minikube

Install Learning Center on Kind

This topic describes how you install Learning Center on your local machine with Kind.

Kind was developed as a means to support development and testing of Kubernetes. Though it exists primarily for that purpose, Kind clusters are often used for local development of user applications as well. For Learning Center, you can use a local Kind cluster to develop workshop content or self-learning when deploying other people's workshops.

Because you are deploying to a local machine, you are unlikely to have access to your own custom domain name and certificate you can use with the cluster. If you don't, you can be restricted as to the sorts of workshops you can develop or run using the Learning Center in Kind. Kind uses `containerd`, which lacks certain features that allow you to trust any image registries hosted within a subnet. This means you cannot readily run workshops that use a local container image registry for each workshop session. If you must run workshops on your local computer that uses an image registry for each session, VMware recommends you use minikube with `dockerd` instead. For more information, see Installing on Minikube.

Also, since Kind has limited memory resources available, you may be prohibited from running workshops that have large memory requirements. Workshops that demonstrate the use of third-party applications requiring a multinode cluster also do not work unless the Kind cluster is specifically configured to be multinode rather than single node.

Requirements and setup instructions specific to Kind are detailed in this document. Otherwise, follow normal installation instructions for the Learning Center operator.

Prerequisites

You must complete the following installation prerequisites as a user prior to installation:

- Create a tanzunet account and have access to your tanzunet credentials.
- Install Kind on your local machine.
- Install tanzuCLI on your local machine.
- Install kubectlCLI on your local machine.

Kind cluster creation

When initially creating the Kind cluster, you must configure it so that the ingress controller is exposed. The Kind documentation provides the following command to do this, but check the documentation in case the details have changed.

```
cat <<EOF | kind create cluster --config=-
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
  kubeadmConfigPatches:
  - |
    kind: InitConfiguration
    nodeRegistration:
      kubeletExtraArgs:
        node-labels: "ingress-ready=true"
  extraPortMappings:
  - containerPort: 80
    hostPort: 80
    protocol: TCP
  - containerPort: 443
    hostPort: 443
    protocol: TCP
EOF
```

Once you have the Kind cluster up and running, you must install an ingress controller.

Ingress controller with DNS

The Kind documentation provides instructions for installing Ambassador, Contour, and Nginx-based ingress controllers.

VMware recommends that you use Contour rather than Nginx, because Nginx drops websocket connections whenever new ingresses are created. The Learning Center workshop environments do include a workaround to re-establish websocket connections for the workshop terminals without losing terminal state, but other applications used with workshops might not, such as terminals available through Visual Studio Code.

Avoid using the Ambassador ingress controller, because it requires all ingresses created to be annotated explicitly with an ingress class of “ambassador.” The Learning Center operator can be configured to do this automatically for ingresses created for the training portal and workshop sessions. However, any workshops that create ingresses as part of the workshop instructions do not work unless they are written to have the user manually add the ingress class when required due to the use of Ambassador.

If you have created a contour ingress controller, verify all pods have a running status. Run:

```
kubectl get pods -n projectcontour -o wide
```

Install carvel tools

You must install the kapp controller and secret-gen controller carvel tools in order to properly install VMware tanzu packages.

To install kapp controller, run:

```
kapp deploy -a kc -f https://github.com/vmware-tanzu/carvel-kapp-controller/releases/l
atest/download/release.yml
```

To install secret-gen controller, run:

```
kapp deploy -a sg -f https://github.com/vmware-tanzu/carvel-secretgen-controller/relea
ses/latest/download/release.yml
```

Note: Type “y” and enter to continue when prompted during installation of both kapp and secret-gen controllers.

Install Tanzu package repository

Follow these steps to install the Tanzu package repository:

1. To create a namespace, run:

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

2. Create a registry secret:

   ```
   tanzu secret registry add tap-registry \
   --username "TANZU-NET-USER" --password "TANZU-NET-PASSWORD" \
   --server registry.tanzu.vmware.com \
   --export-to-all-namespaces --yes --namespace tap-install
   ```

   Where:

   - `TANZU-NET-USER` and `TANZU-NET-PASSWORD` are your credentials for Tanzu Network.

3. Add a vpackage repository to your cluster:

   ```
   tanzu package repository add tanzu-tap-repository \
   --url registry.tanzu.vmware.com/tanzu-application-platform/tap-packages:VERSION
   -NUMBER \
   --namespace tap-install
   ```

   Where `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.2.2`.

   Note: We are currently on build 7. If this changes, we need to update the command with the correct build version after the –url flag.

4. To check the package repository install status, run:

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

   Wait for a reconciled successful status before attempting to install any other packages.

Create a configuration YAML file for Learning Center package

To create a configuration YAML file:

See Supported yaml file configurations to see a list of configurations you can provide to Learning Center.

1. Create a file called learningcenter-value.yaml in your current directory with the following data:

   ```
   ingressDomain: workshops.example.com
   ```

Where:

- `ingressDomain` is `<your-local-ip>.nip.io` if you are using a `nip.io` DNS address. Details about this are provided in the following section.
- `workshops.example.com with` is `<your-local-ip>.nip.io`.

Using a `nip.io` DNS address

Before you can start deploying workshops, you must configure the operator to tell it what domain name can be used to access anything deployed by the operator.

Being a local cluster that isn't exposed to the Internet with its own custom domain name, you can use a nip.io. address.

To calculate the `nip.io` address to use, first work out the IP address for the ingress controller exposed by Kind. This is usually the IP address of the local machine itself, even when you use Docker for Mac.

How you get the IP address for your local machine depends on the operating system you are using.

For example on a Mac, you can find your IP address by searching for network using spotlight and selecting the network option under system preferences. Here you can see your IP address under status.

After you have the IP address, add this as a prefix to the domain name `nip.io`. For example, if the address was `192.168.1.1`, use the domain name of `192.168.1.1.nip.io`.

To configure the Learning Center operator with this cluster domain, run:

```
kubectl set env deployment/learningcenter-operator -n eduk8s INGRESS_DOMAIN=192.168.1.
1.nip.io
```

This causes the Learning Center operator to redeploy with the new configuration. You can now deploy workshops.

Note: Some home Internet gateways implement what is called rebind protection. These gateways do not allow DNS names from the public Internet bind to local IP address ranges inside the home network. If your home Internet gateway has such a feature and it is enabled, it blocks `nip.io` addresses from working. In this case, you must configure your home Internet gateway to allow `*.nip.io` names to be bound to local addresses. Also, you cannot use an address of form `127.0.0.1.nip.io` or `subdomain.localhost`. This causes a failure, because when internal services need to connect to each other, they connect to themselves instead. This happens because the address resolves to the host loopback address of `127.0.0.1`.

Install Learning Center package onto a Kubernetes cluster

To install Learning Center on a Kubernetes cluster:

```
tanzu package install learningcenter --package-name learningcenter.tanzu.vmware.com --
version 0.1.0 -f ./learningcenter-value.yaml --namespace tap-install
```

This package installation uses the installed Package repository with a configuration learningcenter-value.yaml to install our Learning Center package.

Install workshop tutorial package onto a Kubernetes cluster

To install a workshop tutorial on a Kubernetes cluster:

```
tanzu package install learningcenter-tutorials --package-name workshops.learningcente
r.tanzu.vmware.com --version 0.1.0 --namespace tap-install
```

Make sure you install the workshop package after the Learning Center package has reconciled and successfully installed onto your cluster. In case of new versioning, to obtain package version numbers, run:

```
kubectl get packages -n tap-install
```

Run the workshop

To get the training portal URL, run:

```
kubectl get trainingportals
```

You get a URL that you can paste into your browser.

Congratulations, you are now running our tutorial workshop using the Learning Center operator.

Trusting insecure registries

Workshops can optionally deploy a container image registry for a workshop session. This image registry is secured with a password specific to the workshop session and is exposed through a Kubernetes ingress so it can be accessed from the workshop session.

In a typical scenario, Kind uses insecure ingress routes. Even were you to generate a self-signed certificate to use for ingress, it is not trusted by `containerd` that runs within Kind. You must tell Kind to trust any insecure registry running inside of Kind.

You must configure Kind to trust insecure registries when you first create the cluster. Kind, however, is that it uses `containerd` and not `dockerd`. The `containerd` runtime doesn't provide a way to trust any insecure registry hosted within the IP subnet used by the Kubernetes cluster. Instead, `containerd` requires that you enumerate every single host name or IP address on which an insecure registry is hosted. Because each workshop session created by the Learning Center for a workshop uses a different host name, this becomes cumbersome.

If you must used Kind, find out the image registry host name for a workshop deployment and configure `containerd` to trust a set of host names corresponding to low-numbered sessions for that workshop. This allows Kind to work, but once the host names for sessions go beyond the range of host names you set up, you need to delete the training portal and recreate it, so you can use the same host names again.

For example, if the host name for the image registry were of the form:

```
lab-docker-testing-wMM-sNNN-registry.192.168.1.1.nip.io
```

where `NNN` changes per session, you must use a command to create the Kind cluster. For example:

```
cat <<EOF | kind create cluster --config=-
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
  kubeadmConfigPatches:
  - |
    kind: InitConfiguration
    nodeRegistration:
      kubeletExtraArgs:
        node-labels: "ingress-ready=true"
  extraPortMappings:
  - containerPort: 80
    hostPort: 80
    protocol: TCP
  - containerPort: 443
    hostPort: 443
    protocol: TCP
containerdConfigPatches:
- |
  [plugins."io.containerd.grpc.v1.cri".registry.mirrors."lab-docker-testing-w01-s001-r
egistry.192.168.1.1.nip.io"]
    endpoint = ["http://lab-docker-testing-w01-s001-registry.192.168.1.1.nip.io"]
  [plugins."io.containerd.grpc.v1.cri".registry.mirrors."lab-docker-testing-w01-s002-r
egistry.192.168.1.1.nip.io"]
    endpoint = ["http://lab-docker-testing-w01-s002-registry.192.168.1.1.nip.io"]
  [plugins."io.containerd.grpc.v1.cri".registry.mirrors."lab-docker-testing-w01-s003-r
egistry.192.168.1.1.nip.io"]
    endpoint = ["http://lab-docker-testing-w01-s003-registry.192.168.1.1.nip.io"]
  [plugins."io.containerd.grpc.v1.cri".registry.mirrors."lab-docker-testing-w01-s004-r
egistry.192.168.1.1.nip.io"]
    endpoint = ["http://lab-docker-testing-w01-s004-registry.192.168.1.1.nip.io"]
  [plugins."io.containerd.grpc.v1.cri".registry.mirrors."lab-docker-testing-w01-s005-r
egistry.192.168.1.1.nip.io"]
    endpoint = ["http://lab-docker-testing-w01-s005-registry.192.168.1.1.nip.io"]
EOF
```

This allows you to run five workshop sessions before you have to delete the training portal and recreate it.

If you use this, you can use the feature of the training portal to automatically update when a workshop definition is changed. This is because the `wMM` value identifying the workshop environment changes any time you update the workshop definition.

There is no other known workaround for this limitation of `containerd`. As such, VMware recommends you use minikube with `dockerd` instead. For more information, see Installing on Minikube.

Install Learning Center on Minikube

This topic describes how you install Learning Center on your local machine with Minikube.

Minikube enables local deployment of Kubernetes for developing workshop content or for self-learning when deploying other people's workshops.

Because you are deploying to a local machine, you are unlikely to have access to your own custom domain name and certificate you can use with the cluster. You must take extra steps over a standard install of Minikube to ensure you can run certain types of workshops.

Also, because Minikube generally has limited memory resources available and is only a single-node cluster, you might be restricted from running workshops that have large memory requirements or that demonstrate the use of third-party applications requiring a multinode cluster.

Requirements and setup instructions specific to Minikube are detailed in this document. Otherwise, you can follow normal installation instructions for the Learning Center operator.

Trusting insecure registries

Workshops can optionally deploy a container image registry for a workshop session. This image registry is secured with a password specific to the workshop session and is exposed through a Kubernetes ingress so it can be accessed from the workshop session.

In a typical scenario, Minikube uses insecure ingress routes. Even were you to generate a self-signed certificate to use for ingress, it is not trusted by `dockerd` that runs within Minikube. You must tell Minikube to trust any insecure registry running inside of Minikube.

You must configure Minikube to trust insecure registries the first time you start a new cluster with it. That is, you must supply the details to `minikube start`, which means you must know the IP subnet Minikube uses.

If you already have a cluster running using Minikube, run `minikube ip` to discover the IP address it uses. From that you can discover the trusted subnet. For example, if `minikube ip` returned `192.168.64.1`, the trusted subnet is `192.168.64.0/24`.

With this information, when you start a new cluster with Minikube, run:

```
minikube start --insecure-registry=192.168.64.0/24
```

If you already have a cluster started with Minikube, you cannot stop it and then provide this option when it is restarted. You can only use this option for a completely new cluster.

Note: You must be using `dockerd`, not `containerd`, in the Minikube cluster. `containerd` does not accept an IP subnet when defining insecure registries to be trusted. It allows only specific hosts or IP addresses. Because you don't know what IP address Minikube will use in advance, you can't provide the IP address on the command line when starting Minikube to create the cluster.

Prerequisites

You must complete the following installation prerequisites as a user prior to installation:

- Create a tanzunet account and have access to your tanzunet credentials.
- Install miniKube on your local machine.
- Install tanzuCLI on your local machine.
- Install kubectlCLI on your local machine.

Ingress controller with DNS

After the Minikube cluster is running, you must enable the `ingress` and `ingress-dns` add-ons for Minikube. These deploy the nginx ingress controller along with support for integrating into DNS.

To enable these after the cluster has been created, run:

```
minikube addons enable ingress
minikube addons enable ingress-dns
```

You are now ready to install the Learning Center package.

Note: The ingress add-ons for Minikube do not work when using Minikube on top of Docker for Mac or Docker for Windows. On macOS you must use the Hyperkit VM driver. On Windows you must use the Hyper-V VM driver.

Install carvel tools

You must install the kapp controller and secret-gen controller carvel tools in order to properly install VMware tanzu packages.

To install kapp controller, run:

```
kapp deploy -a kc -f https://github.com/vmware-tanzu/carvel-kapp-controller/releases/l
atest/download/release.yml
```

To install secret-gen controller, run:

```
kapp deploy -a sg -f https://github.com/vmware-tanzu/carvel-secretgen-controller/relea
ses/latest/download/release.yml
```

Note: Type “y” and enter to continue when prompted during installation of both kapp and secret-gen controllers.

Install Tanzu package repository

Follow these steps to install the Tanzu package repository:

1. To create a namespace, run:

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

2. Create a registry secret:

   ```
   tanzu secret registry add tap-registry \
     --username "TANZU-NET-USER" --password "TANZU-NET-PASSWORD" \
     --server registry.tanzu.vmware.com \
     --export-to-all-namespaces --yes --namespace tap-install
   ```

   Where:

   - `TANZU-NET-USER` and `TANZU-NET-PASSWORD` are your credentials for Tanzu Network.

3. Add a package repository to your cluster:

   ```
   tanzu package repository add tanzu-tap-repository \
     --url registry.tanzu.vmware.com/tanzu-application-platform/tap-packages:VERSI
   ON-NUMBER \
     --namespace tap-install
   ```

Where `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.2.2`.

Note: We are currently on build 7; if this changes, we need to update the command with the correct build version after the –url flag.

4. To check the package repository install status, run:

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

Wait for a reconciled sucessful status before attempting to install any other packages.

Create a configuration YAML file for the Learning Center package

Create a file called learningcenter-value.yaml in your current directory with the following data:

See Supported yaml file configurations to see a list of configurations you can provide to Learning Center.

```
ingressDomain: workshops.example.com
```

Where:

- `ingressDomain` is `<your-local-ip>.nip.io` if you are using a `nip.io` DNS address. Details about this are provided in the following section.
- `workshops.example.com` is `<your-local-ip>.nip.io`

Using a `nip.io` DNS address

After the Learning Center operator is installed, before you can start deploying workshops, you must configure the operator to tell it what domain name can be used to access anything deployed by the operator.

Being a local cluster that isn't exposed to the Internet with its own custom domain name, you can use a nip.io. address.

To calculate the `nip.io` address to use, first work out the IP address of the cluster created by Minikube by running `minikube ip`. Add this as a prefix to the domain name `nip.io`. For example, if `minikube ip` returns `192.168.64.1`, use the domain name of `192.168.64.1.nip.io`.

To configure the Learning Center operator with this cluster domain, run:

```
kubectl set env deployment/learningcenter-operator -n learningcenter INGRESS_DOMAIN=19
2.168.64.1.nip.io
```

This causes the Learning Center operator to redeploy with the new configuration. You should now be able to start deploying workshops.

Note: Some home Internet gateways implement what is called rebind protection. These gateways do not let DNS names from the public Internet bind to local IP address ranges inside the home network. If your home Internet gateway has such a feature and it is enabled, it blocks `nip.io` addresses from working. In this case, you must configure your home Internet gateway to allow `*.nip.io` names to be bound to local addresses.

Install Learning Center package onto a minikube cluster

To install the Learning Center package onto a minikube cluster, run:

```
tanzu package install learningcenter --package-name learningcenter.tanzu.vmware.com --
version 0.1.0 -f ./learningcenter-value.yaml --namespace tap-install
```

This package installation uses the installed Package repository with a configuration learningcenter-value.yaml to install the Learning Center package.

Install workshop tutorial package onto a minikube cluster

To install the workshop tutorial package onto a minikube cluster, run:

```
tanzu package install learningcenter-tutorials --package-name workshops.learningcenter.tanzu.vmware.com --version 0.1.0 --namespace tap-install
```

Make sure you install the workshop package after the Learning Center package has reconciled and successfully installed onto your cluster. In case of new versioning, to obtain package version numbers, run:

```
kubectl get packages -n tap-install
```

Run the workshop

To get the training portal URL, run:

```
kubectl get trainingportals
```

You get a URL that you can paste into your browser.

Congratulations, you are now running the tutorial workshop using the Learning Center operator.

Working with large images

If you create or run workshops that work with the image registry created for a workshop session, and you push images to that image registry that have large layers, you must configure the version of nginx deployed for the ingress controller and increase the allowed size of request data for a HTTP request.

To do this, run:

```
kubectl edit configmap nginx-load-balancer-conf -n kube-system
```

To the config map resource, add the following property under `data`:

```
proxy-body-size: 1g
```

If you don’t increase this, `docker push` fails when trying to push container images with large layers.

Limited resource availability

When deploying a cluster, by default Minikube only configures support for 2Gi of memory. This usually isn’t adequate.

To view how much memory is available when a custom amount has been set as a default, run:

```
minikube config get memory
```

VMware recommends you configure Minikube to use 4Gi or more. This must be specified when the cluster is first created. Do this by using the `--memory` option to `minikube start` or by specifying a default memory value beforehand by using `minikube config set memory`.

In addition to increasing the memory available, you can increase the disk size, because fat container images can quickly use disk space within the cluster.

Storage provisioner issue

v1.12.3 of Minikube introduced a bug in the storage provisioner that causes potential corruption of data in persistent volumes where the same persistent volume claim name is used in two different namespaces. This affects Learning Center when:

- You deploy multiple training portals at the same time.

- You run multiple workshops at the same time that have docker or image registry support enabled.
- The workshop session itself is backed by persistent storage and multiple sessions run at the same time.

This issue is supposed to be fixed in Minikube v1.13.0; however, you can still encounter issues when deleting a training portal instance and recreating it immediately with the same name. This occurs because reclaiming of the persistent volume by the Minikube storage provisioner can be slow, and the new instance can grab the same original directory on disk with old data in it. After deleting a training portal instance, wait before recreating one with the same name to allow the storage provisioner to delete the old persistent volume.

Create workshops for Learning Center

This section provides information about how you create Learning Center workshops.

- Workshop configuration
- Workshop images
- Workshop content
- Building an image
- Workshop instructions
- Workshop runtime
- Workshop slides
- Air-gapped environment requirements

Create workshops for Learning Center

This section provides information about how you create Learning Center workshops.

- Workshop configuration
- Workshop images
- Workshop content
- Building an image
- Workshop instructions
- Workshop runtime
- Workshop slides
- Air-gapped environment requirements

Configure your Learning Center workshop

This topic describes the two main steps required to configure your Learning Center workshop. The first specifies the structure of the workshop content and the second defines the runtime requirements for deploying the workshop.

Specifying structure of the content

There are multiple ways you can configure a workshop to specify the structure of the content. The sample workshops use YAML files.

The `workshop/modules.yaml` file provides details about the list of available modules that make up your workshop and data variables for use in content.

The list of available modules represents all of the modules available to you. You might not use all of them. You might want to run variations of your workshop, such as for different programming languages. As such, which modules are active and are used for a specific workshop are listed in the separate workshop/workshop.yaml file. The active modules are listed with the name to be given to that workshop.

By default the workshop.yaml file specifies what modules are used. When you want to deliver different variations of the workshop content, you can provide multiple workshop files with different names. For example, you can name the workshop files workshop-java.yaml and workshop-python.yaml.

Where you have multiple workshop files and don't have the default workshop.yaml file, you can specify the default workshop file by setting the WORKSHOP_FILE environment variable in the runtime configuration.

The format for listing the available modules in the workshop/modules.yaml file is:

```
modules:
  workshop-overview:
    name: Workshop Overview
    exit_sign: Setup Environment
  setup-environment:
    name: Setup Environment
    exit_sign: Start Workshop
  exercises/01-sample-content:
    name: Sample Content
  workshop-summary:
    name: Workshop Summary
    exit_sign: Finish Workshop
```

Each available module is listed under modules, where the name used corresponds to the path to the file containing the content for that module. Any extension identifying the content type is left off.

For each module, set the name field to the page title to be displayed for that module. If no fields are provided and name is not set, the title for the module is derived from the name of the module file.

The corresponding workshop/workshop.yaml file, where all available modules are used, would have the format:

```
name: Markdown Sample
modules:
  activate:
    - workshop-overview
    - setup-environment
    - exercises/01-sample-content
    - workshop-summary
```

The top-level name field in this file is the name of this variation of the workshop content.

The modules.activate field is a list of modules to be used for the workshop. The names in this list must match the names as they appear in the modules file.

The order in which modules are listed under the modules.activate field in the workshop configuration file dictates the order pages are traversed. The order in which modules appear in the modules configuration file is not relevant.

At the bottom of each page, a **Continue** button is displayed to allow the user to go to the next page in sequence. You can customize the label on this button by setting the exit_sign field in the entry for the module in the modules configuration file.

In the last module in the workshop, a button is displayed, but where the user goes after clicking it varies. If you want the user to go to a different website upon completion, you can set the exit_link field of the final module to an external URL. Alternatively, you can set the RESTART_URL environment variable in a workshop environment to control where the user goes. If a destination for the final page is not provided, the user is redirected back to the starting page of the workshop.

When the user uses the training portal, the training portal overrides this environment variable so, at the completion of a workshop, the user returns to the training portal.

VMware recommends that for the last page, the exit_sign be set to “Finish Workshop” and exit_link not be specified. This enables the destination to be controlled from the workshop environment or training portal.

Specifying the runtime configuration

You can deploy workshop images directly to a container runtime. The Learning Center Operator is provided to manage deployments into a Kubernetes cluster. You define the configuration for the Learning Center Operator with a `Workshop` CRD in the `resources/workshop.yaml` file:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  vendor: learningcenter.tanzu.vmware.com
  title: Markdown Sample
  description: A sample workshop using Markdown
  url: YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE
  content:
    image: {YOUR-REGISTRY-URL}/lab-markdown-sample:main
  duration: 15m
  session:
    namespaces:
      budget: small
    applications:
      console:
        enabled: true
      editor:
        enabled: true
```

Where:

- `YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE` is the Git repository URL for `lab-markdown-sample`. For example, `{YOUR-GIT-REPO-URL}/lab-markdown-sample`.

In this sample, a custom workshop image bundles the workshop content into its own container image. The `content.image` setting specifies this. To instead download workshop content from a GitHub repository at runtime, use:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  vendor: learningcenter.tanzu.vmware.com
  title: Markdown Sample
  description: A sample workshop using Markdown
  url: YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE
  content:
    files: YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE
  duration: 15m
  session:
    namespaces:
      budget: small
    applications:
      console:
        enabled: true
      editor:
        enabled: true
```

Where:

- `YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE` is the Git repository URL for `lab-markdown-sample`. For example, `{YOUR-GIT-REPO-URL}/lab-markdown-sample`.

The difference is the use of the `content.files` setting. Here, the workshop content is overlaid on top of the standard workshop base image. To use an alternate base image with additional applications or packages installed, specify the alternate image against the `content.image` setting at the same time you set `content.files`.

Next steps

- Learn about configuration options for the workshop.yaml custom resource definitions (CRD) in Workshop resource.

Create the image for your Learning Center workshop

The workshop environment for the Learning Center is packaged as a container image. This topic describes how you create the Learning Center workshop image.

You can execute the image with remote content pulled down from GitHub or a web server. Alternatively, you can bundle your workshop content, including any extra tools required, in a new container image derived from the workshop environment base image.

Templates for creating a workshop

To get you started with your own workshop content, VMware provides a number of sample workshops. Different templates in Markdown or AsciiDoc are available to use depending on the syntax you use to create the workshop. These templates are available in a zip file called `LEARNING-CENTER-WORKSHOP-SAMPLES.ZIP` on the Tanzu Network TAP Product Page. The zip file contains the following projects that you can upload to your own Git repository:

- lab-markdown-sample
- lab-asciidoc-sample

When creating your own workshops, a suggested convention is to prefix the directory name with the Git repository name where it is hosted. For example, you can make the prefix `lab-`. This way it stands out as a workshop or lab when you have a number of Git repositories on the same Git hosting service account or organization.

Note: Do not make the name you use for a workshop too long. The DNS host name used for applications deployed from the workshop, when using certain methods of deployment, might exceed the 63 character limit. This is because the workshop deployment name is used as part of the namespace for each workshop session. This is in turn used in the DNS host names generated for the ingress host name. VMware suggests keeping the workshop name, and so your repository name, to 25 characters or less.

Workshop content directory layout

After creating a copy of the sample workshop content, you can see a number of files located in the top-level directory and a number of subdirectories forming a hierarchy. The files in the top-level directory are:

- `README.md` - A file stating what the workshop in your Git repository is about and how to deploy it. Replace the current content provided in the sample workshop with your own.
- `LICENSE` - A license file so people are clear about how they can use your workshop content. Replace this with what license you want to apply to your workshop content.
- `Dockerfile` - Steps to build your workshop into an image ready for deployment. Leave this as is, unless you want to customize it to install additional system packages or tools.
- `kustomization.yaml` - A kustomize resource file for loading the workshop definition. The Learning Center operator must be deployed before using this file.
- `.dockerignore` - List of files to ignore when building the workshop content into an image.
- `.eduk8signore` - List of files to ignore when downloading workshop content into the workshop environment at runtime.

Key subdirectories and the files contained within them are:

- `workshop` - Directory under which your workshop files reside.

- `workshop/modules.yaml` - Configuration file with details of available modules that make up your workshop and data variables for use in content.
- `workshop/workshop.yaml` - Configuration file that gives the name of the workshop, the list of active modules for the workshop, and any overrides for data variables.
- `workshop/content` - Directory under which your workshop content resides, including images to be displayed in the content.
- `resources` - Directory under which Kubernetes custom resources are stored for deploying the workshop using the Learning Center.
- `resources/workshop.yaml` - The custom resources for the Learning Center, which describe your workshop and requirements for deployment.
- `resources/training-portal.yaml` - A sample custom resource for the Learning Center for creating a training portal for the workshop, encompassing the workshop environment and a workshop instance.

A workshop can include other configuration files and directories with other types of content, but this is the minimal set of files to get you started.

Directory for workshop exercises

The number of files and directories can quickly add up at the top level of your repository. The same is true of the home directory for the user when running the workshop environment. To help with this proliferation of files, you can push files required for exercises during the workshop into the `exercises` subdirectory under the root of the repository.

With an `exercises` subdirectory, the initial working directory for the embedded terminal when created is set to `$HOME/exercises` instead of `$HOME`. If the embedded editor is enabled, the subdirectory is opened as the workspace for the editor. Only directories and files in that subdirectory are visible through the default view of the editor.

However, the `exercises` directory isn't set as the home directory of the user. This means if a user inadvertently runs `cd` with no arguments from the terminal, they go back to the home directory.

To avoid confusion and help a user return to where they must be, VMware recommends that when you instruct users to change directories, provide a full path relative to the home directory. For example, use a path of the form `~/exercises/example-1` rather than `example-1` for the `cd` command when changing directories. By using a full path, users can execute the command and be assured of going to the required location.

Working on your Learning Center workshop content

This topic tells you about the best practices for speeding up the iterative loop of editing and testing a Learning Center workshop when developing the content.

Workshop content is either embedded in a custom workshop image or downloaded from a Git repository or web server when the workshop session is created.

Deactivating reserved sessions

Deactivate the reserved sessions by setting the `reserved` field to `0` in your training portal instance:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-sample-workshop
spec:
  portal:
    sessions:
      maximum: 1
  workshops:
  - name: lab-sample-workshop
    reserved: 0
    expires: 120m
    orphaned: 15m
```

If you do not deactivate reserved sessions, a new session is always created ready for the next workshop session when there is available capacity to do so. If you modify workshop content while testing the current workshop session, terminate the session and start a new one, the workshop picks up the reserved session. The reserved session has a copy of the old content.

By deactivating reserved sessions, a new workshop session is always created on demand. This ensures the latest workshop content is used.

Because you might have to wait to create a new workshop, shut down the existing workshop session first. The new workshop session might also take some time to start if an updated version of the workshop image also has to be pulled down.

Live updates to the content

If you download workshop content from a Git repository or web server, and you are only doing simple updates to workshop instructions, scripts, or files bundled with the workshop, you can update the content in place without needing to restart the workshop session. To perform an update, download the workshop content after you have pushed back any changes to the hosted Git repository or updated the content available through the web server. From the workshop session terminal, run:

```
update-workshop
```

This command downloads any workshop content from the Git repository or web server, unpacks it into the live workshop session, and re-runs any script files found in the `workshop/setup.d` directory.

Find the location where the workshop content is downloading by viewing the file:

```
cat ~/.eduk8s/workshop-files.txt
```

You can change the location saved in this file if, for example, it references a specific version of the workshop content and you want to test with a different version.

Once the workshop content has been updated, reload the current page of the workshop instructions by clicking the reload icon on the dashboard while holding down the shift key.

If additional pages are added to the workshop instructions or pages are renamed, you must restart the workshop renderer process by running:

```
restart-workshop
```

If you didn’t rename the current pager or if the name changed, you can trigger a reload of the current page. Click the home icon or refresh the webpage if the name of the first page didn’t change.

If action blocks within the workshop instructions are broken, to change and test the workshop instructions within the live workshop session, you can edit the appropriate page under `/opt/workshop/content`. Navigate to the modified page or reload it to verify the change.

To change set up scripts that create files specific to a workshop session, edit the script under `/opt/workshop/setup.d` directory.

To trigger running of any setup scripts, run:

```
rebuild-workshop
```

If local changes to the workshop session take effect, you can restore the file in the original Git repository.

Updating workshop content in a live session in this way does not undo any deployments or changes you make in the Kubernetes cluster for that session. To retest parts of the workshop instructions, you might have to manually undo the changes in the cluster to replay them. This depends on your specific workshop content.

Custom workshop image changes

If your workshop uses a custom workshop image to provide additional tools and you have included the workshop instructions as part of the workshop image, you must use an image tag of `main`, `develop`, or `latest` during the development of workshop content. Do not use a version image reference.

For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-sample-workshop
spec:
  title: Sample Workshop
  description: A sample workshop
  content:
    image: {YOUR-GIT-REPO-URL}/lab-sample-workshop:main
```

When you use an image tag of `main`, `develop`, or `latest`, the image pull policy is set to `Always` to ensure that the custom workshop image is pulled down again for a new workshop session if the remote image changes. If the image tag is for a specific version, you must change the workshop definition every time when the workshop image changes.

Custom workshop image overlay

For a custom workshop image, you can set up the workshop definition to pull down the workshop content from the hosted Git repository or web server as the follows:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-sample-workshop
spec:
  title: Sample Workshop
  description: A sample workshop
  content:
    image: {YOUR-REGISTRY-URL}/lab-sample-workshop:main
    files: {YOUR-GIT-REPO-URL}/lab-sample-workshop
```

By pulling down the workshop content as an overlay of the custom workshop image when the workshop session starts, you only need to rebuild the custom workshop image when you need to make changes such as to include additional tools or to ensure the latest workshop instructions are included in the final custom workshop image.

Because the location of the workshop files is known, you can live update the workshop content in the session by following Live updates to the content.

If the additional set of tools required for a workshop is not specific to a workshop, VMware recommends that you create a standalone workshop base image where you can add the tools. You can always pull down content for a specific workshop from a Git repository or web server when the workshop session starts.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-sample-workshop
spec:
  title: Sample Workshop
  description: A sample workshop
  content:
    image: {YOUR-REGISTRY-URL}/custom-environment:main
    files: {YOUR-GIT-REPO-URL}/lab-sample-workshop
```

This separates generic tooling from specific workshops and so you can use the custom workshop base image for multiple workshops on different, but related topics that require the same tooling.

Changes to workshop definition

By default, to modify the definition for a workshop, you need to delete the training portal instance, update the workshop definition in the cluster, and recreate the training portal.

During the workshop content development, to change resource allocations, role access, or to specify what resource objects to be automatically created for the workshop environment or a specific workshop session, you can enable automatic updates in the training portal definition by setting `updates.workshop` field as `true`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-sample-workshop
spec:
  portal:
    sessions:
      maximum: 1
    updates:
      workshop: true
  workshops:
  - name: lab-sample-workshop
    expires: 120m
    orphaned: 15m
```

With automatic updates enabled, if the workshop definition in the cluster is modified, the existing workshop environment managed by the training portal for that workshop is shut down and replaced with a new workshop environment by using the updated workshop definition.

When an active workshop session is running, the actual deletion of the old workshop environment is delayed until that workshop session is terminated.

Local build of workshop image

If you do not package a workshop into a custom workshop image, VMware recommends to build a custom workshop image locally on your own machine by using `docker` to avoid keeping pushing changes to a hosted Git repository and using a Kubernetes cluster for local workshop content development.

Furthermore, to avoid pushing the image to a public image registry on the Internet, you must deploy an image registry to your local Kubernetes cluster where you run the Learning Center. In most cases, a basic deployment of an image registry in a local cluster access is not secure. As a result, you have to configure the Kubernetes cluster to trust the registry that is not secure. This can be difficult to do depending on the Kubernetes cluster you use, but it can enable quicker turnaround because you do not have to push or pull the custom workshop image across the public Internet.

After pushing the custom workshop image built locally to the local image registry, you can set the image reference in the workshop definition to pull the custom workshop from the local registry in the same cluster. To ensure that the custom workshop image is always pulled for a new workshop session after update, use the `latest` tag when tagging and pushing the image to the local registry.

Build an image for your Learning Center workshop

This topic describes how you include an extra system, third-party tool, or configuration in your image by bundling workshop content from the Learning Center workshop base image.

The following sample workshop template provides a `Dockerfile`.

Structure of the Dockerfile

The structure of the `Dockerfile` in the sample workshop template is:

```
FROM registry.tanzu.vmware.com/tanzu-application-platform/tap-packages@sha256:a8870aa6
0b45495d298df5b65c69b3d7972608da4367bd6e69d6e392ac969dd4

COPY --chown=1001:0 . /home/eduk8s/

RUN mv /home/eduk8s/workshop /opt/workshop

RUN fix-permissions /home/eduk8s
```

The default `Dockerfile` action is to:

- Copy all files from a registry to the `/home/eduk8s` directory. You must build the custom workshop images on the `registry.tanzu.vmware.com/tanzu-application-platform/tap-packages@sha256:a8870aa60b45495d298df5b65c69b3d7972608da4367bd6e69d6e392ac969dd4` workshop image. You can do this directly or you can also create an intermediate base image to install extra packages required by a number of different workshops. The `--chown=1001:0` option ensures that files are owned by the appropriate user and group.
- The `workshop` subdirectory is moved to `/opt/workshop` so that it is not visible to the user. This subdirectory is in an area searchable for workshop content, in addition to `/home/eduk8s/workshop`.

To customize your `Dockerfile`:

- You can ignore other files or directories from the repository, by listing them in the `.dockerignore` file.
- You can include `RUN` statements in the `Dockerfile` to run custom-build steps, but the `USER` inherited from the base image has user ID `1001` and is not the `root` user.

Base images and version tags

The sample `Dockerfile` provided above and the GitHub repository workshop templates reference the workshop base image as follows:

```
registry.tanzu.vmware.com/tanzu-application-platform/tap-packages@sha256:a8870aa60b454
95d298df5b65c69b3d7972608da4367bd6e69d6e392ac969dd4
```

Custom workshop base images

The `base-environment` workshop images include language run times for Node.js and Python. If you need a different language runtime or a different version of a language runtime, you must create a custom workshop base image which includes the environment you need. This custom workshop image is derived from `base-environment` but includes extra runtime components.

The following Dockerfile example creates a Java JDK11-customized image:

```
ARG IMAGE_REPOSITORY=dev.registry.tanzu.vmware.com/learning-center
FROM ${IMAGE_REPOSITORY}/pkgs-java-tools as java-tools
FROM registry.tanzu.vmware.com/tanzu-application-platform/tap-packages@sha256:a8870aa6
0b45495d298df5b65c69b3d7972608da4367bd6e69d6e392ac969dd4
COPY --from=java-tools --chown=1001:0 /opt/jdk11 /opt/java
COPY --from=java-tools --chown=1001:0 /opt/gradle /opt/gradle
COPY --from=java-tools --chown=1001:0 /opt/maven /opt/maven
COPY --from=java-tools --chown=1001:0 /opt/code-server/extensions/.  /opt/code-server/
extensions/
COPY --from=java-tools --chown=1001:0 /home/eduk8s/. /home/eduk8s/
COPY --from=java-tools --chown=1001:0 /opt/eduk8s/. /opt/eduk8s/
ENV PATH=/opt/java/bin:/opt/gradle/bin:/opt/maven/bin:$PATH \
    JAVA_HOME=/opt/java \
    M2_HOME=/opt/maven
```

Installing extra system packages

Installing extra system packages requires that you run the installation as `root`. You must switch the user commands before running the command, and then switch the user back to user ID of `1001`.

```
USER root

RUN ... commands to install system packages

USER 1001
```

VMware recommends that you only use the `root` user to install extra system packages. Don't use the `root` user when adding anything under `/home/eduk8s`. Otherwise, you must ensure the user ID and group for directories and files are set to `1001:0` and then run the `fix-permissions` command if necessary.

When you run any command as `root`, you must temporarily override the value of the `HOME` environment variable and set it to `/root`.

If you don't do this the `root` user drops configuration files in `/home/eduk8s`, thinking it is the `root` home directory, because the `HOME` environment variable is by default set to `/home/eduk8s`. This can cause commands run later during the workshop to fail if they try to update the configuration files as they have wrong permissions.

Fixing the file and group ownership and running `fix-permissions` can help with this problem, but not in every case, because of permissions the `root` user may apply and how container image layers work. VMware recommends that you use the following:

```
USER root

RUN HOME=/root && \
    ... commands to install system packages

USER 1001
```

Installing third-party packages

If you are not using system packaging tools to install extra packages, but are manually downloading packages and optionally compiling them to binaries, it is better to do this as the default user and not `root`.

If compiling packages, VMware recommends working in a temporary directory under `/tmp` and removing the directory as part of the same `RUN` statement when done.

If you are installing a binary, you can install it in `/home/eduk8s/bin`. This directory is in the application search path defined by the `PATH` environment variable for the image.

To install a directory hierarchy of files, create a separate directory under `/opt` to install everything. You can override the `PATH` environment variable in the `Dockerfile` to add an extra directory for application binaries and scripts. You can override the `LD_LIBRARY_PATH` environment variable for the location of shared libraries.

If installing any files from a `RUN` instruction into `/home/eduk8s`, VMware recommends that you run `fix-permissions` as part of the same instruction to avoid copies of files being made into a new layer, which applies to the case where `fix-permissions` is only run in a later `RUN` instruction. You can still leave the final `RUN` instruction for `fix-permissions` as it is smart enough not to apply changes if the file permissions are already set correctly and so it does not trigger a copy of a file when run more than once.

Writing instructions for your Learning Center workshop

This topic describes how you write and format the instructions for a Learning Center workshop. You can use either Markdown with file extension `.md` or AsciiDoc with file extension `.adoc` as the markup format for the individual module files that comprise the workshop instructions.

Annotation of executable commands

In conjunction with the standard Markdown and AsciiDoc, you can apply additional annotations to code blocks. The annotations indicate that a user can click the code block and have it copied to the terminal and executed.

If using Markdown, to annotate a code block so it is copied to the terminal and executed, use:

~~~
```execute
echo "Execute command."
```
~~~

When the user clicks the code block, the command is executed in the first terminal of the workshop dashboard.

If using AsciiDoc, you can instead use the `role` annotation in an existing code block:

```
[source,bash,role=execute]
----
echo "Execute command."
----
```

When the workshop dashboard is configured to display multiple terminals, you can qualify which terminal the command must be executed in by adding a suffix to the `execute` annotation. For the first terminal, use `execute-1`, for the second terminal `execute-2`, and so on:

~~~
```execute-1
echo "Execute command."
```

```execute-2
echo "Execute command."
```
~~~

To execute a command in all terminal sessions on the terminals tab of the dashboard, you can use `execute-all`:

~~~
```execute-all
clear
```
~~~

In most cases, a command the user executes completes immediately. To run a command that never returns, with the user needing to interrupt it to stop it, you can use the special string `<ctrl+c>` in a subsequent code block.

~~~
```execute
<ctrl+c>
```
~~~

When the user clicks on this code block, the command running in the corresponding terminal is interrupted.

Note: Using the special string `<ctrl+c>` is deprecated, and you must use the `terminal:interrupt` clickable action instead.

Annotation of text to be copied

To copy the content of the code block into the paste buffer instead of running the command, you can use:

~~~
```copy
echo "Text to copy."
```
~~~

After the user clicks this code block, they can then paste the content into another window.

If you have a situation where the text being copied must be modified before use, you can denote this special case by using `copy-and-edit` instead of `copy`. The text is still copied to the paste buffer, but is displayed in the browser in a way to highlight that it must be changed before use.

~~~
```copy-and-edit
echo "Text to copy and edit."
```
~~~

For AsciiDoc, similar to `execute`, you can add the `role` of `copy` or `copy-and-edit`:

```
[source,bash,role=copy]
----
echo "Text to copy."
----

[source,bash,role=copy-and-edit]
----
echo "Text to copy and edit."
----
```

For `copy` only, to mark an inline code section within a paragraph of text as copyable when clicked, you can append the special data variable reference `{{copy}}` immediately after the inline code block:

```
Text to `copy`{{copy}}.
```

Extensible clickable actions

The preceding means to annotate code blocks were the original methods used to indicate code blocks to be executed or copied when clicked. To support a growing number of clickable actions with different customizable purposes, annotation names are now name-spaced. The preceding annotations are still supported, but the following are now recommended, with additional options available to customize the way the actions are presented.

For code execution, instead of:

~~~
```execute
echo "Execute command."
```
~~~

you can use:

~~~
```terminal:execute
command: echo "Execute command."
```
~~~

The contents of the code block is YAML. The executable command must be set as the `command` property. By default when the user clicks the command, it is executed in terminal session 1. To select a different terminal session, you can set the `session` property.

~~~
```terminal:execute
command: echo "Execute command."
session: 1
```
~~~

To define a command the user clicks that executes in all terminal sessions on the terminals tab of the dashboard, you can also use:

~~~
```terminal:execute-all
command: echo "Execute command."
```
~~~

For `terminal:execute` or `terminal:execute-all`, to clear the terminal before the command is executed, set the `clear` property to `true`:

~~~
```terminal:execute
command: echo "Execute command."
~~~

~~~
clear: true
```
~~~

This clears the full terminal buffer and not just the displayed portion of the buffer.

With the new clickable actions, to indicate that a running command in a terminal session must be interrupted, use:

~~~
```terminal:interrupt
session: 1
```
~~~

(Optional) Set the `session` property within the code block to indicate an alternate terminal session to session 1.

To allow the user to send an interrupt to all terminals sessions on the terminals tab of the dashboard, use:

~~~
```terminal:interrupt-all
```
~~~

Where you want the user to enter input into a terminal rather than a command, such as when a running command prompts for a password, use:

~~~
```terminal:input
text: password
```
~~~

To allow the user to run commands or interrupt a command, set the `session` property to indicate a specific terminal to send it to if you don't want to send it to terminal session 1:

~~~
```terminal:input
text: password
session: 1
```
~~~

When providing terminal input in this way, the text by default still has a newline appended to the end, making it behave the same as using `terminal:execute`. If you do not want a newline appended, set the `endl` property to `false`.

~~~
```terminal:input
text: input
endl: false
```
~~~

To allow the user to clear all terminal sessions on the terminals tab of the dashboard, use:

~~~
```terminal:clear-all
```
~~~

This clears the full terminal buffer and not just the displayed portion of the terminal buffer. It does not have any effect when an application is running in the terminal using visual mode. To clear only the displayed portion of the terminal buffer when a command dialog box is displayed, use `terminal:execute` and run the `clear` command.

To allow the user to copy content to the paste buffer, use:

~~~
```workshop:copy
text: echo "Text to copy."
```
~~~

or:

~~~
```workshop:copy-and-edit
text: echo "Text to copy and edit."
```
~~~

A benefit of using these over the original methods is that by using the appropriate YAML syntax, you can control whether:

- A multiline string value is concatenated into one line.
- Line breaks are preserved.
- Initial or terminating new lines are included.

In the original methods, the string was always trimmed before use. By using the different forms as appropriate, you can annotate the displayed code block with a different message letting the user know what will happen.

The method for using AsciiDoc is similar, using the `role` for the name of the annotation and YAML as the content:

```
[source,bash,role=terminal:execute]
----
command: echo "Execute command."
----
```

Supported workshop editor

Learning Center currently **only** supports the code-server v4.4.0 of VS Code as an editor in workshops.

Clickable actions for the dashboard

In addition to the clickable actions related to the terminal and copying of text to the paste buffer, other actions are available for controlling the dashboard and opening URL links.

To allow the user to click in the workshop content to open a URL in a new browser, use:

~~~
```dashboard:open-url
url: https://www.example.com/
```
~~~

To allow the user to click in the workshop content to display a specific dashboard tab if hidden, use:

~~~
```dashboard:open-dashboard
name: Terminal
```
~~~

To allow the user to click in the workshop content to display the console tab, use:

~~~
```dashboard:open-dashboard
name: Console
```
~~~

To allow the user to click in the workshop content to display a specific view within the Kubernetes web console by using a clickable action block, rather than requiring the user to find the correct view, use:

~~~
```dashboard:reload-dashboard
name: Console
prefix: Console
title: List pods in namespace {{session_namespace}}
url: {{ingress_protocol}}://{{session_namespace}}-console.{{ingress_domain}}/#/pod?namespace={{session_namespace}}
description: ""
```
~~~

To allow the user to create a new dashboard tab with a specific URL, use:

~~~
```dashboard:create-dashboard
name: Example
~~~

~~~
url: https://www.example.com/
```
~~~

To allow the user to create a new dashboard tab with a new terminal session, use:

~~~
```dashboard:create-dashboard
name: Example
url: terminal:example
```
~~~

The value must be of the form `terminal:<session>`, where `<session>` is replaced with the name you want to give the terminal session. The terminal session name must be restricted to lowercase letters, numbers, and '-'. You must avoid using numeric terminal session names such as "1", "2", and "3", because these are used for the default terminal sessions.

To allow the user to reload an existing dashboard, using the URL it is currently targeting, use:

~~~
```dashboard:reload-dashboard
name: Example
```
~~~

If the dashboard is for a terminal session, there is no effect unless the terminal session was disconnected, in which case it is reconnected.

To allow the user to change the URL target of an existing dashboard by entering the new URL when reloading a dashboard, use:

~~~
```dashboard:reload-dashboard
name: Example
url: https://www.example.com/
```
~~~

The user cannot change the target of a dashboard that includes a terminal session.

To allow the user to delete a dashboard, use:

~~~
```dashboard:delete-dashboard
name: Example
```
~~~

The user cannot delete dashboards corresponding to builtin applications provided by the workshop environment, such as the default terminals, console, editor, or slides.

Deleting a custom dashboard including a terminal session does not destroy the underlying terminal session, and the user can reconnect it by creating a new custom dashboard for the same terminal session name.

Clickable actions for the editor

If the embedded editor is enabled, special actions are available that control the editor.

To allow the user to open an existing file you can use:

~~~
```editor:open-file
file: ~/exercises/sample.txt
```
~~~

You can use `~/` prefix to indicate the path relative to the home directory of the session. When the user opens the file, if you want the insertion point left on a specific line, provide the `line` property. Lines numbers start at `1`.

~~~
```editor:open-file
file: ~/exercises/sample.txt
line: 1
```
~~~

To allow the user to highlight certain lines of a file based on an exact string match, use:

~~~
```editor:select-matching-text
file: ~/exercises/sample.txt
text: "int main()"
```
~~~

The region of the match is highlighted by default. To allow the user to highlight any number of lines before or after the line with the match, you can set the `before` and `after` properties:

~~~
```editor:select-matching-text
file: ~/exercises/sample.txt
text: "int main()"
before: 1
after: 1
```
~~~

Setting both `before` and `after` to `0` causes the complete line that matched to be highlighted instead of a region within the line.

To match based on a regular expression, rather than an exact match, set `isRegex` to `true`:

~~~
```editor:select-matching-text
file: ~/exercises/sample.txt
text: "image: (.*)"
isRegex: true
```
~~~

When a regular expression is used, and subgroups are specified within the pattern, you can indicate which subgroup is selected:

~~~
```editor:select-matching-text
file: ~/exercises/sample.txt
text: "image: (.*)"
isRegex: true
group: 1
```
~~~

Where there are multiple possible matches in a file, and the one you want to match is not the first, you can set a range of lines to search:

~~~
```editor:select-matching-text
file: ~/exercises/sample.txt
text: "image: (.*)"
isRegex: true
start: 8
stop: 12
```
~~~

Absence of `start` means start at the beginning of the file. Absence of `stop` means stop at the end of the file. The line number given by `stop` is not included in the search.

For both an exact match and regular expression, the text to be matched must all be on one line. It is not possible to match text that spans across lines.

To allow the user to replace text within the file, first match it exactly or use a regular expression so it is marked as selected, then use:

~~~
```editor:replace-text-selection
file: ~/exercises/sample.txt
text: nginx:latest
```
~~~

To allow the user to append lines to the end of a file, use:

~~~
```editor:append-lines-to-file
file: ~/exercises/sample.txt
text: |
    Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed
~~~

~~~
    do eiusmod tempor incididunt ut labore et dolore magna aliqua.
```
~~~

If the user runs the action `editor:append-lines-to-file` and the file doesn't exist, it is created. You can use this to create new files for the user.

To allow the user to insert lines before a specified line in the file, use:

~~~
```editor:insert-lines-before-line
file: ~/exercises/sample.txt
line: 8
text: |
    Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed
    do eiusmod tempor incididunt ut labore et dolore magna aliqua.
```
~~~

To allow the user to insert lines after matching a line containing a specified string, use:

~~~
```editor:append-lines-after-match
file: ~/exercises/sample.txt
match: Lorem ipsum
text: |
    Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed
    do eiusmod tempor incididunt ut labore et dolore magna aliqua.
```
~~~

Where the file contains YAML, to allow the user to insert a new YAML value into an existing structure, use:

~~~
```editor:insert-value-into-yaml
file: ~/exercises/deployment.yaml
path: spec.template.spec.containers
value:
- name: nginx
  image: nginx:latest
```
~~~

To allow the user to execute a registered VS code command, use:

~~~
```editor:execute-command
command: spring.initializr.maven-project
args:
- language: Java
  dependencies: [ "actuator", "webflux" ]
  artifactId: demo
  groupId: com.example
```
~~~

Clickable actions for file download

If file downloads are enabled for the workshop, you can use the `files:download-file` clickable action:

~~~
```files:download-file
path: .kube/config
```
~~~

The action triggers saving the file to the user's local computer, and the file is not displayed in the user's web browser.

Clickable actions for the examiner

If the test examiner is enabled, special actions are available to run verification checks to verify whether a workshop user has performed a required step. You can trigger these verification checks by clicking on the action, or you can configure them to start running when the page loads.

For a single verification check that the user must click to run, use:

~~~
```examiner:execute-test
name: test-that-pod-exists
title: Verify that pod named "one" exists.
args:
- one
```
~~~

The `title` field is displayed as the title of the clickable action and must describe the nature of the test. If required, you can provide a `description` field for a longer explanation of the test. This is displayed in the body of the clickable action but is shown as preformatted text.

There must be an executable program (script or compiled application) in the `workshop/examiner/tests` directory with name matching the value of the `name` field.

The list of program arguments against the `args` field is passed to the test program.

The executable program for the test must exit with a status of 0 if the test is successful and nonzero if the test is a failure. The test should aim to return as quickly as possible and should not be a persistent program.

```
#!/bin/bash

kubectl get pods --field-selector=status.phase=Running -o name | egrep -e "^pod/$1$"

if [ "$?" != "0" ]; then
    exit 1
fi

exit 0
```

By default, the program for a test is stopped after a timeout of 15 seconds, and the test is deemed to have failed. To adjust the timeout, you can set the `timeout` value, which is in seconds. A value of 0 causes the default 15 seconds timeout to be applied. It is not possible to deactivate stopping the test program after running for the default or a specified `timeout` value.

~~~
```examiner:execute-test
name: test-that-pod-exists
title: Verify that pod named "one" exists
args:
- one
timeout: 5
```
~~~

To apply the test multiple times, you can enable the retry when a failure occurs. For this you must set the number of times to retry and the delay between retries. The value for the delay is in seconds.

~~~
```examiner:execute-test
name: test-that-pod-exists
title: Verify that pod named "one" exists
args:
- one
timeout: 5
retries: 10
delay: 1
```
~~~

When you use retries, the testing stops as soon as the test program returns that it was successful.

To have retries continue for as long as the page of the workshop instructions displays, set `retries` to the special YAML value of `.INF`:

~~~
```examiner:execute-test
name: test-that-pod-exists
title: Verify that pod named "one" exists
args:
~~~

~~~
- one
timeout: 5
retries: .INF
delay: 1
```
~~~

Rather than require a workshop user to click the action to run the test, you can have the test start as soon as the page is loaded, or when a section the page is contained in is expanded. Do this by setting `autostart` to `true`:

~~~
```examiner:execute-test
name: test-that-pod-exists
title: Verify that pod named "one" exists
args:
- one
timeout: 5
retries: .INF
delay: 1
autostart: true
```
~~~

When a test succeeds, to immediately start the next test in the same page, set `cascade` to `true`.

~~~
```examiner:execute-test
name: test-that-pod-exists
title: Verify that pod named "one" exists
args:
- one
timeout: 5
retries: .INF
delay: 1
autostart: true
cascade: true
```

```examiner:execute-test
name: test-that-pod-does-not-exist
title: Verify that pod named "one" does not exist
args:
- one
retries: .INF
delay: 1
```
~~~

Clickable actions for sections

For optional instructions, or instructions you want to hide until the workshop user is ready for them, you can designate sections to be hidden. When the user clicks the appropriate action, the section expands to show its content. You can use this for examples that initially hide a set of questions or a test at the end of each workshop page.

In order to designate a section of content as hidden, you must use two separate action code blocks marking the beginning and end of the section:

~~~
```section:begin
title: Questions
```

To show you understand ...

```section:end
```
~~~

The `title` must be set to the text you want to include in the banner for the clickable action.

A clickable action is only shown for the beginning of the section, and the action for the end is always hidden. Clicking the action for the beginning expands the section. The user can collapse the section again by clicking the action.

To create nested sections, you must name the action blocks for the beginning and end so they are correctly matched:

~~~
```section:begin
name: questions
title: Questions
```

To show you understand ...

```section:begin
name: question-1
prefix: Question
title: 1
```

...

```section:end
name: question-1
```

```section:end
name: questions
```
~~~

The `prefix` attribute allows you to override the default `Section` prefix used on the title for the action.

If a collapsible section includes an examiner action block set to automatically run, it only starts when the user expands the collapsible section.

In case you want a section header showing in the same style as other clickable actions, you can use:

~~~
```section:heading
title: Questions
```
~~~

When the user clicks on this, the action is still marked as completed, but it does not trigger any other action.

Overriding title and description

Clickable action blocks default to use a title with the prefix dictated by what the action block does. The body of the action block also defaults to use a value commensurate with the action.

Especially for complicated scenarios involving editing of files, the defaults might not be the most appropriate and be confusing, so you can override them. To override these defaults, set the `prefix`, `title`, and `description` fields of a clickable action block:

~~~
```action:name
prefix: Prefix
title: Title
description: Description
```
~~~

The banner of the action block in this example displays “Prefix: Title”, with the body showing “Description”.

Note: The description is always displayed as pre-formatted text within the rendered page.

Escaping of code block content

Because the Liquid template engine is applied to workshop content, you must escape content in code blocks that conflict with the syntactic elements of the Liquid template engine. To escape such elements, you can suspend processing by the template engine for that section of workshop content to ensure it is rendered correctly. Do this by using a Liquid `{% raw %}...{% endraw %}` block.

~~~
{% raw %}
```execute
echo "Execute command."
```
{% endraw %}
~~~

This has the side effect of preventing interpolation of data variables, so restrict it to only the required scope.

Interpolation of data variables

When creating page content, you can reference a number of predefined data variables. The values of the data variables are substituted into the page when rendered in the user's browser.

The workshop environment provides the following built-in data variables:

- `workshop_name`: The name of the workshop.
- `workshop_namespace`: The name of the namespace used for the workshop environment.
- `session_namespace`: The name of the namespace the workshop instance is linked to and into which any deployed applications run.
- `training_portal`: The name of the training portal the workshop is hosted by.
- `ingress_domain`: The host domain must be used in the any generated host name of ingress routes for exposing applications.
- `ingress_protocol`: The protocol (http/https) used for ingress routes created for workshops.

To use a data variable within the page content, surround it by matching pairs of brackets:

```
{{ session_namespace }}
```

Do this inside of code blocks, including clickable actions, as well as in URLs:

```
http://myapp-{{ session_namespace }}.{{ ingress_domain }}
```

When the workshop environment is hosted in Kubernetes and provides access to the underlying cluster, the following data variables are also available.

- `kubernetes_token`: The Kubernetes access token of the service account the workshop session is running as.
- `kubernetes_ca_crt`: The contents of the public certificate required when accessing the Kubernetes API URL.
- `kubernetes_api_url`: The URL for accessing the Kubernetes API. This is only valid when used from the workshop terminal.

Note: An older version of the rendering engine required that data variables be surrounded on each side with the character `%`. This is still supported for backwards compatibility, but VMware recommends you use matched pairs of brackets instead.

Adding custom data variables

You can introduce your own data variables by listing them in the `workshop/modules.yaml` file. A data variable is defined as having a default value, but the value is overridden if an environment variable of the same name is defined.

The field under which the data variables must be specified is `config.vars`:

```
config:
    vars:
```

```
    - name: LANGUAGE
      value: undefined
```

To use a name for a data variable that is different from the environment variable name, add a list of `aliases`:

```
config:
    vars:
    - name: LANGUAGE
      value: undefined
      aliases:
      - PROGRAMMING_LANGUAGE
```

The environment variables with names in the list of aliases are checked first, then the environment variable with the same name as the data variable. If no environment variables with those names are set, the default value is used.

You can override the default value for a data variable for a specific workshop by setting it in the corresponding workshop file. For example, `workshop/workshop-python.yaml` might contain:

```
vars:
  LANGUAGE: python
```

For more control over setting the values of data variables, you can provide the file `workshop/config.js`. The form of this file is:

```
function initialize(workshop) {
    workshop.load_workshop();

    if (process.env['WORKSHOP_FILE'] == 'workshop-python.yaml') {
        workshop.data_variable('LANGUAGE', 'python');
    }
}

exports.default = initialize;

module.exports = exports.default;
```

This JavaScript code is loaded and the `initialize()` function called to set up the workshop configuration. You can then use the `workshop.data_variable()` function to set up any data variables.

Because it is JavaScript, you can write any code to query process environment variables and set data variables based on those. This might include creating composite values constructed from multiple environment variables. You can even download data variables from a remote host.

Passing environment variables

You can pass environment variables, including remapping of variable names, by setting your own custom data variables. If you don’t need to set default values or remap the name of an environment variable, you can instead reference the name of the environment variable directly. You must prefix the name with `ENV_` when using it.

For example, to display the value of the `KUBECTL_VERSION` environment variable in the workshop content, use `ENV_KUBECTL_VERSION`, as in:

```
{{ ENV_KUBECTL_VERSION }}
```

Handling embedded URL links

You can include URLs in workshop content. This can be the literal URL, or the Markdown or AsciiDoc syntax for including and labelling a URL. What happens when a user clicks on a URL depends on the specific URL.

In the case of the URL being an external website, when the URL is clicked, the URL opens in a new browser tab or window. When the URL is a relative page referring to another page that is part of the workshop content, the page replaces the current workshop page.

You can define a URL where components of the URL are provided by data variables. Data variables useful for this are `session_namespace` and `ingress_domain`, because they can be used to create a URL to an application deployed from a workshop:

```
https://myapp-{{ session_namespace }}.{{ ingress_domain }}
```

Conditional rendering of content

Rendering pages is in part handled by the Liquid template engine. So you can use any constructs the template engine supports for conditional content:

```
{% if LANGUAGE == 'java' %}
....
{% endif %}
{% if LANGUAGE == 'python' %}
....
{% endif %}
```

Embedding custom HTML content

Custom HTML can be embedded in the workshop content by using the appropriate mechanism provided by the content rendering engine used.

If using Markdown, HTML can be embedded directly without being marked as HTML:

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin justo.

<div>
<table style="width:100%">
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>50</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
</table>
</div>

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin justo.
```

If using AsciiDoc, HTML can be embedded by using a passthrough block:

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin justo.

++++
<div>
<table style="width:100%">
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>50</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
</table>
</div>
++++

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin justo.
```

In both cases, VMware recommends that the HTML consist of only a single HTML element. If you have more than one, include them all in a `div` element. The latter is necessary if any of the HTML elements are marked as hidden and the embedded HTML is a part of a collapsible section. If you don't ensure the hidden HTML element is placed under the single top-level `div` element, the hidden HTML element is visible when the collapsible section is expanded.

In addition to visual HTML elements, you can also include elements for embedded scripts or style sheets.

If you have HTML markup that must be added to multiple pages, extract it into a separate file and use the include file mechanism of the Liquid template engine. You can also use the partial render mechanism of Liquid as a macro mechanism for expanding HTML content with supplied values.

Automate your Learning Center workshop runtime

Your workshop content can script the steps a user must run for a workshop. This topic tells you how to set this up.

In some cases, you must parameterize the content with information from the runtime environment. Data variables in workshop content allow this to a degree, but you can automate this by using scripts executed in the workshop container to set up configuration files.

Do this by supplying setup scripts that run when the container is started. You can also run persistent background processes in the container that perform extra work for you while a workshop is being run.

Predefined environment variables

When you create the workshop content, you can use data variables to automatically insert values corresponding to the specific workshop session or environment. For example: the name of the namespace used for the session and the ingress domain when creating an ingress route.

These data variables can display a YAML or JSON resource file in the workshop content with values already filled out. You can have executable commands that have the data variables substituted with values given as arguments to the commands.

For commands run in the shell environment, you can also reference a number of predefined environment variables.

Key environment variables are:

- `WORKSHOP_NAMESPACE` - The name of the namespace used for the workshop environment.
- `SESSION_NAMESPACE` - The name of the namespace the workshop instance is linked to and into which any deployed applications run.
- `INGRESS_DOMAIN` - The host domain that must be used in any generated host name of ingress routes for exposing applications.
- `INGRESS_PROTOCOL` - The protocol (HTTP or HTTPS) used for ingress routes created for workshops.

Instead of having an executable command in the workshop content, use:

~~~
```execute
kubectl get all -n %session_namespace%
```
~~~

With the value of the session namespace filled out when the page is rendered, you can use:

~~~
```execute
kubectl get all -n $SESSION_NAMESPACE
```
~~~

The shell inserts the value of the environment variable.

Running steps on container start

To run a script that makes use of the earlier environment variables when the container is started, and to perform tasks such as pre-create YAML or JSON resource definitions with values filled out, you can add an executable shell script to the `workshop/setup.d` directory. The name of the executable shell script must have a `.sh` suffix to be recognized and run.

If the container is restarted, the setup script runs again in the new container. If the shell script is performing actions against the Kubernetes REST API using kubectl or by using another means, the actions it performs must be tolerant of running more than once.

When using a setup script to fill out values in resource files, a useful utility is `envsubst`. You can use this in a setup script as follows:

```
#!/bin/bash

envsubst < frontend/ingress.yaml.in > frontend/ingress.yaml
```

A reference of the form `${INGRESS_DOMAIN}` in the input file is replaced with the value of the `INGRESS_DOMAIN` environment variable.

Setup scripts have the `/home/eduk8s` directory as the current working directory.

If you are creating or updating files in the file system and using a custom workshop image, ensure that the workshop image is created with correct file permissions to allow updates.

Running background applications

The setup scripts run once on container startup. You can use the script to start a background application needed to run in the container for the life of the workshop, but if that application stops, it does not restart.

If you must run a background application, you can integrate the management of the background application with the supervisor daemon run within the container. To have the supervisor daemon manage the application for you, add a configuration file snippet for the supervisor daemon in the `workshop/supervisor` directory. This configuration file must have a `.conf` extension.

The form of the configuration file snippet must be:

```
[program:myapplication]
process_name=myapplication
command=/opt/myapplication/sbin/start-myapplication
stdout_logfile=/proc/1/fd/1
stdout_logfile_maxbytes=0
redirect_stderr=true
```

The application must send any logging output to `stdout` or `stderr`, and the configuration snippet must direct log output to `/proc/1/fd/1` so it is captured in the container log file. If you must restart or shut down the application within the workshop interactive terminal, you can use the `supervisorctl` control script.

Terminal user shell environment

Neither the setup scripts that run when the container starts nor background applications affect the user environment of the terminal shell. The shell environment makes use of `bash` and the `$HOME/.bash_profile` script is read to perform added setup for the user environment. Because some default setup is included in `$HOME/.bash_profile`, you must not replace it, because you can loose that configuration.

To provide commands to initialize each shell environment, you can provide the file `workshop/profile`. When this file exists, it is sourced at the end of the `$HOME/.bash_profile` file when it is processed.

Overriding terminal shell command

The user starts each terminal session by using the `bash` terminal shell. A terminal prompt dialog box displays, allowing the user to manually enter commands or perform clickable actions targetting the terminal session.

To specify the command to run for a terminal session, you can supply an executable shell script file in the `workshop/terminal` directory.

The name of the shell script file for a terminal session must be of the form `SESSION.sh`, where `SESSION` is replaced with the name of the terminal session. The session names of the default terminals configured to be displayed with the dashboard are `1`, `2`, and `3`.

The shell script file might be used to run a terminal-based application such as `k9s`, or to create an SSH session to a remote system.

```
#!/bin/bash

exec k9s
```

If the command that is run exits, the terminal session is marked as exited and you need to reload that terminal session to start over again. Alternatively, you could write the shell script file as a loop so it restarts the command you want to run if it ever exits.

```
#!/bin/bash

while true; do
    k9s
    sleep 1
done
```

If you want to run an interactive shell and output a banner at the start of the session with special information for the user, use a script file to output the banner and then run the interactive shell:

```
#!/bin/bash

echo
echo "Your session namespace is "$SESSION_NAMESPACE".
echo

exec bash
```

Add presenter slides to your Learning Center workshop

If your workshop includes a presentation, include slides by placing them in the `workshop/slides` directory. Anything in this directory is served up as static files through a HTTP web server. The default webpage must be provided as `index.html`.

Use reveal.js presentation tool

To support the use of reveal.js, static media assets for that package are already bundled and available at the standard URL paths that the package expects. You can drop your slide presentation using reveal.js into the `workshop/slides` directory and it will work with no additional setup.

If you are using reveal.js for the slides and you have history enabled or are using section IDs to support named links, you can use an anchor to a specific slide and that slide will be opened when clicked on:

```
%slides_url%#/questions
```

When using embedded links to the slides in workshop content, if the workshop content is displayed as part of the dashboard, the slides open in the tab to the right rather than as a separate browser window or tab.

Use a PDF file for presenter slides

For slides bundled as a PDF file, add the PDF file to `workshop/slides` and then add an `index.html` which displays the PDF embedded in the page.

Requirements for Learning Center in an air-gapped environment

This topic gives you the list of configurations required for Learning Center to properly function in an air-gapped environment.

Learning Center can run in an air-gapped environment but workshops do not have this capability by default. Users must therefore take the following steps to ensure Learning Center functions as expected.

Workshop yaml changes

In an air-gapped environment a user has no Internet access, so workshop yamls should be modified to use:

1. Private container registries.
2. Private Maven, NPM, Python, Go, or any other language repository.

For example, in NPM you can modify the npmrc file to use:

```
// .npmrc
registry=https://myregistry-url
```

Self-signed certificates

Air-gapped environments normally use private Certificate Authorities (CA) that may require the use of self-signed certificates. You can allow the injection of CAs by:

1. Setting the env variable NODE_EXTRA_CA_CERTS to the path of the file that contains one or more trusted certificates in PEM format.
2. Add the following to your workshop definition:

```
spec:
  session:
    env:
    - name: NODE_EXTRA_CA_CERTS
      value: "$my-cert-pathway"
```

Internet dependencies

If the workshop requires the installation of any Internet dependency, such as a Linux Tool or any other tool, it must be done in the workshop image. See building an image

Define custom resources for Learning Center

This topic describes how you define custom resources for Learning Center workshops and training portals.

You can deploy workshop images directly to a container runtime. The Learning Center Operator enables managing the deployments into a Kubernetes cluster. A set of Kubernetes custom resource definitions (CRDs) controls the operation of the Learning Center Operator.

Note: The examples do not show all the possible fields of each custom resource type. Later documentation will go in-depth on all the possible fields and their definitions.

Workshop definition resource

The `Workshop` custom resource defines a workshop. It specifies the title and description of the workshop, the location of the workshop content or container image that you deploy, any resources that you pre-create in the workshop environment or for each instance of the workshop.

You can also define environment variables for the workshop image, the amount of CPU and memory resources for the workshop instance, any overall quota you will apply to the created namespaces and what the workshop uses.

A minimal example of the `Workshop` custom resource looks like this:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  title: Markdown Sample
  description: A sample workshop using Markdown
  content:
    files: {YOUR-GIT-REPO-URL}/lab-markdown-sample
  session:
    namespaces:
      budget: small
    applications:
      console:
        enabled: true
      editor:
        enabled: true
```

When you create an instance of the `Workshop` custom resource, the Learning Center Operator does not take any immediate action. This custom resource exists only to define the workshop.

Note: You create the `Workshop` custom resource at the cluster scope.

Workshop environment resource

You must create a workshop environment first to deploy the instances of a workshop. The `WorkshopEnvironment` custom resource defines the configuration of the workshop environment and the details of the workshop that you deploy.

A minimal example of the `WorkshopEnvironment` custom resource looks like this:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  request:
    token: lab-markdown-sample
  session:
    username: learningcenter
```

When you create an instance of the `WorkshopEnvironment` custom resource, the Learning Center Operator responds by creating a namespace to host the workshop instances. The `Workshop` resource defines the workshop instance and the `spec.workshop.name` field specifies the name of the

`Workshop` resource. The namespace you create uses the same name as that of the `metadata.name` field in the `WorkshopEnvironment` resource.

The `spec.request.token` field defines a token with which you must supply a request to create an instance of a workshop in this workshop environment. If necessary, you can also specify the namespaces from which a request for a workshop instance to initiate.

The `Workshop` defines a set of common resources that must exist for the workshop. Learning Center Operator creates these common resources after you created the namespace for the workshop environment. If necessary, these resources can include creation of separate namespaces with specific resources that you create in those namespaces instead.

Note: You create the `WorkshopEnvironment` custom resource at the cluster scope.

Workshop request resource

To create an instance of the workshop under the workshop environment, the typical path is to create an instance of the `WorkshopRequest` custom resource.

The `WorkshopRequest` custom resource is namespaced to allow who can create it. Role-based access control (RBAC) controls the request to create a workshop instance. This means you can allow non-privileged users to create workshops, although the deployment of the workshop instance might require elevated privileges.

A minimal example of the `WorkshopRequest` custom resource looks like this:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopRequest
metadata:
  name: lab-markdown-sample
spec:
  environment:
    name: lab-markdown-sample
    token: lab-markdown-sample
```

Apart from appropriate access from RBAC, the user requesting a workshop instance must know the name of the workshop environment and the secret token that permits workshop requests against that specific workshop environment.

You do not need to create the `WorkshopRequest` resource when you use the `TrainingPortal` resource to provide a web interface for accessing workshops. You only need to create the `WorkshopRequest` resource when you create the `WorkshopEnvironment` resource manually and do not use the training portal.

Workshop session resource

Although `WorkshopRequest` is the typical way to request workshop instances, the Learning Center Operator itself creates an instance of a `WorkshopSession` custom resource when the request is granted.

The `WorkshopSession` custom resource is the expanded definition of what the workshop instance is. It combines details from `Workshop` and `WorkshopEnvironment`, and also links back to the `WorkshopRequest` resource object that triggered the request. The Learning Center Operator reacts to an instance of `WorkshopSession` and creates the workshop instance based on that definition.

Note: You create the `WorkshopSession` custom resource at the cluster scope.

Training portal resource

The `TrainingPortal` custom resource provides a high-level mechanism for creating a set of workshop environments and populating them with workshop instances.

A minimal example of the `TrainingPortal` custom resource looks like this:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
```

```
metadata:
  name: lab-markdown-sample
spec:
  workshops:
  - name: lab-markdown-sample
    capacity: 1
```

You can set the capacity of the training room, which dictates how many workshop instances are created for each workshop.

Note: You create the `TrainingPortal` custom resource at the cluster scope.

System profile resource

The `SystemProfile` custom resource provides a mechanism for configuring the Learning Center Operator. This provides additional features that use environment variables to configure the operator.

A minimal example of the `SystemProfile` custom resource looks like this:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  ingress:
    domain: learningcenter.tanzu.vmware.com
    secret: learningcenter-tanzu-vmware-com-tls
    class: nginx
  environment:
    secrets:
      pull:
      - cluster-image-registry-pull
```

The operator, by default, looks for a default system profile called `default-system-profile`. Setting the `SYSTEM_PROFILE` environment variable on the deployment for the operator or using the `system.profile` setting on `TrainingPortal`, `WorkshopEnvironment`, or `WorkshopSession` custom resources for specific deployments can override the default name globally.

As only a global deployment of the operator is supported, the `SystemProfile` custom resource is created at cluster scope.

You can make changes to instances of the `SystemProfile` custom resource. The Learning Center Operator uses these changes without needing to redeploy the custom resource.

Note: You create the `SystemProfile` custom resource at the cluster scope.

Loading the workshop CRDs

The custom resource definitions for the custom resource described earlier are created in the Kubernetes cluster when you deploy the Learning Center operator by using the Tanzu CLI.

This is because `v1` versions of CRDs are only supported from Kubernetes v1.17. If you want to use the `v1` versions of the CRDs, you must create a copy of the Learning Center operator deployment resources and override the configuration.

Define custom resources for Learning Center

This topic describes how you define custom resources for Learning Center workshops and training portals.

You can deploy workshop images directly to a container runtime. The Learning Center Operator enables managing the deployments into a Kubernetes cluster. A set of Kubernetes custom resource definitions (CRDs) controls the operation of the Learning Center Operator.

Note: The examples do not show all the possible fields of each custom resource type. Later documentation will go in-depth on all the possible fields and their definitions.

Workshop definition resource

The `Workshop` custom resource defines a workshop. It specifies the title and description of the workshop, the location of the workshop content or container image that you deploy, any resources that you pre-create in the workshop environment or for each instance of the workshop.

You can also define environment variables for the workshop image, the amount of CPU and memory resources for the workshop instance, any overall quota you will apply to the created namespaces and what the workshop uses.

A minimal example of the `Workshop` custom resource looks like this:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  title: Markdown Sample
  description: A sample workshop using Markdown
  content:
    files: {YOUR-GIT-REPO-URL}/lab-markdown-sample
  session:
    namespaces:
      budget: small
    applications:
      console:
        enabled: true
      editor:
        enabled: true
```

When you create an instance of the `Workshop` custom resource, the Learning Center Operator does not take any immediate action. This custom resource exists only to define the workshop.

Note: You create the `Workshop` custom resource at the cluster scope.

Workshop environment resource

You must create a workshop environment first to deploy the instances of a workshop. The `WorkshopEnvironment` custom resource defines the configuration of the workshop environment and the details of the workshop that you deploy.

A minimal example of the `WorkshopEnvironment` custom resource looks like this:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  request:
    token: lab-markdown-sample
  session:
    username: learningcenter
```

When you create an instance of the `WorkshopEnvironment` custom resource, the Learning Center Operator responds by creating a namespace to host the workshop instances. The `Workshop` resource defines the workshop instance and the `spec.workshop.name` field specifies the name of the `Workshop` resource. The namespace you create uses the same name as that of the `metadata.name` field in the `WorkshopEnvironment` resource.

The `spec.request.token` field defines a token with which you must supply a request to create an instance of a workshop in this workshop environment. If necessary, you can also specify the namespaces from which a request for a workshop instance to initiate.

The `Workshop` defines a set of common resources that must exist for the workshop. Learning Center Operator creates these common resources after you created the namespace for the workshop environment. If necessary, these resources can include creation of separate namespaces with specific resources that you create in those namespaces instead.

Note: You create the `WorkshopEnvironment` custom resource at the cluster scope.

Workshop request resource

To create an instance of the workshop under the workshop environment, the typical path is to create an instance of the `WorkshopRequest` custom resource.

The `WorkshopRequest` custom resource is namespaced to allow who can create it. Role-based access control (RBAC) controls the request to create a workshop instance. This means you can allow non-privileged users to create workshops, although the deployment of the workshop instance might require elevated privileges.

A minimal example of the `WorkshopRequest` custom resource looks like this:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopRequest
metadata:
  name: lab-markdown-sample
spec:
  environment:
    name: lab-markdown-sample
    token: lab-markdown-sample
```

Apart from appropriate access from RBAC, the user requesting a workshop instance must know the name of the workshop environment and the secret token that permits workshop requests against that specific workshop environment.

You do not need to create the `WorkshopRequest` resource when you use the `TrainingPortal` resource to provide a web interface for accessing workshops. You only need to create the `WorkshopRequest` resource when you create the `WorkshopEnvironment` resource manually and do not use the training portal.

Workshop session resource

Although `WorkshopRequest` is the typical way to request workshop instances, the Learning Center Operator itself creates an instance of a `WorkshopSession` custom resource when the request is granted.

The `WorkshopSession` custom resource is the expanded definition of what the workshop instance is. It combines details from `Workshop` and `WorkshopEnvironment`, and also links back to the `WorkshopRequest` resource object that triggered the request. The Learning Center Operator reacts to an instance of `WorkshopSession` and creates the workshop instance based on that definition.

Note: You create the `WorkshopSession` custom resource at the cluster scope.

Training portal resource

The `TrainingPortal` custom resource provides a high-level mechanism for creating a set of workshop environments and populating them with workshop instances.

A minimal example of the `TrainingPortal` custom resource looks like this:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  workshops:
  - name: lab-markdown-sample
    capacity: 1
```

You can set the capacity of the training room, which dictates how many workshop instances are created for each workshop.

Note: You create the `TrainingPortal` custom resource at the cluster scope.

System profile resource

The `SystemProfile` custom resource provides a mechanism for configuring the Learning Center Operator. This provides additional features that use environment variables to configure the operator.

A minimal example of the `SystemProfile` custom resource looks like this:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  ingress:
    domain: learningcenter.tanzu.vmware.com
    secret: learningcenter-tanzu-vmware-com-tls
    class: nginx
  environment:
    secrets:
      pull:
      - cluster-image-registry-pull
```

The operator, by default, looks for a default system profile called `default-system-profile`. Setting the `SYSTEM_PROFILE` environment variable on the deployment for the operator or using the `system.profile` setting on `TrainingPortal`, `WorkshopEnvironment`, or `WorkshopSession` custom resources for specific deployments can override the default name globally.

As only a global deployment of the operator is supported, the `SystemProfile` custom resource is created at cluster scope.

You can make changes to instances of the `SystemProfile` custom resource. The Learning Center Operator uses these changes without needing to redeploy the custom resource.

Note: You create the `SystemProfile` custom resource at the cluster scope.

Loading the workshop CRDs

The custom resource definitions for the custom resource described earlier are created in the Kubernetes cluster when you deploy the Learning Center operator by using the Tanzu CLI.

This is because `v1` versions of CRDs are only supported from Kubernetes v1.17. If you want to use the `v1` versions of the CRDs, you must create a copy of the Learning Center operator deployment resources and override the configuration.

Configure the Workshop resource

This topic describes how you configure the `Workshop` custom resource, which defines a Learning Center workshop.

Workshop title and description

Each workshop must have the `title` and `description` fields. If you do not supply these fields, the `Workshop` resource is rejected when you attempt to load it into the Kubernetes cluster.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  title: Markdown Sample
  description: A sample workshop using Markdown
  content:
    files: {YOUR-GIT-REPO-URL}/lab-markdown-sample
```

Where:

- The `title` field has a single-line value specifying the subject of the workshop.
- The `description` field has a longer description of the workshop.

You can also supply the following optional information for the workshop:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  title: Markdown Sample
  description: A sample workshop using Markdown
  url: YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE
  difficulty: beginner
  duration: 15m
  vendor: learningcenter.tanzu.vmware.com
  authors:
  - John Smith
  tags:
  - template
  logo: data:image/png;base64,....
  content:
    files: YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE
```

Where:

- The `url` field is the Git repository URL for `lab-markdown-sample`. For example, `{YOUR-GIT-REPO-URL}/lab-markdown-sample`. It must be a URL you can use to get more information about the workshop.
- The `difficulty` field indicates the target audiences of the workshop. The value can be `beginner`, `intermediate`, `advanced`, or `extreme`.
- The `duration` field gives the maximum amount of time the workshop takes to complete. This field provides informational value and does not guarantee how long a workshop instance lasts. The field format is an integer number with `s`, `m`, or `h` suffix.
- The `vendor` field must be a value that identifies the company or organization with which the authors are affiliated. This is a company or organization name or a DNS host name under the control of whoever has created the workshop.
- The `authors` field must list the people who create the workshop.
- The `tags` field must list labels identifying what the workshop is about. This is used in a searchable catalog of workshops.
- The `logo` field must be an image provided in embedded data URI format that depicts the topic of the workshop. The image must be 400 by 400 pixels. You can use it in a searchable catalog of workshops.
- The `files` field is the Git repository URL for `lab-markdown-sample`. For example, `{YOUR-GIT-REPO-URL}/lab-markdown-sample`.

When referring to a workshop definition after you load it into a Kubernetes cluster, use the value of the `name` field given in the metadata. To experiment with different variations of a workshop, copy the original workshop definition YAML file and change the value of `name`. Make your changes and load it into the Kubernetes cluster.

Downloading workshop content

You can download workshop content when you create the workshop instance. If the amount of content is moderate, the download doesn’t increase startup time for the workshop instance. The alternative is to bundle the workshop content in a container image you build from the Learning Center workshop base image.

To download workshop content at the time the workshop instance starts, set the `content.files` field to the location of the workshop content:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  title: Markdown Sample
  description: A sample workshop using Markdown
  content:
    files: {YOUR-GIT-REPO-URL}/lab-markdown-sample
```

The location is a GitHub or GitLab repository, a URL to a tarball hosted on a HTTP server, or a reference to an OCI image artifact on a registry.

For a GitHub or GitLab repository, do not prefix the location with `https://` as it uses symbolic reference and is not a URL.

The format of the reference to a GitHub or GitLab repository is similar to what you use with Kustomize when referencing remote repositories. For example:

- `github.com/organisation/project?ref=develop` or `github.com/organisation/project?ref=main`: Use the workshop content you host at the root of the GitHub repository. Use the `main` branch. Be sure to specify the ref branch, because not specifying the branch may lead to content download errors.
- `github.com/organisation/project/subdir?ref=develop`: Use the workshop content you host at `subdir` of the GitHub repository. Use the `develop` branch.
- `gitlab.com/organisation/project`: Use the workshop content you host at the root of the GitLab repository. Use the `main` branch.
- `gitlab.com/organisation/project/subdir?ref=develop`: Use the workshop content you host at `subdir` of the GitLab repository. Use the `develop` branch.

For a URL to a tarball hosted on a HTTP server, the URL is in the following formats:

- `https://example.com/workshop.tar` - Use the workshop content from the top-level directory of the unpacked tarball.
- `https://example.com/workshop.tar.gz` - Use the workshop content from the top-level directory of the unpacked tarball.
- `https://example.com/workshop.tar?path=subdir` - Use the workshop content from the subdirectory path of the unpacked tarball.
- `https://example.com/workshop.tar.gz?path=subdir` - Use the workshop content from the subdirectory path of the unpacked tarball.

The tarball referenced by the URL is either uncompressed or compressed.

For GitHub, instead of referencing the Git repository containing the workshop content, use a URL to refer directly to the downloadable tarball for a specific version of the Git repository:

- `https://github.com/organization/project/archive/develop.tar.gz?path=project-develop`

You must reference the `.tar.gz` download and cannot use the `.zip` file. The base name of the tarball file is the branch or commit name. You must enter the `path` query string parameter where the argument is the name of the project and branch or project and commit. You must supply the path because the contents of the repository are not returned at the root of the archive.

GitLab also provides a means of downloading a package as a tarball:

- `https://gitlab.com/organization/project/-/archive/develop/project-develop.tar.gz?path=project-develop`

If the GitHub or GitLab repository is private, you can generate a personal access token providing read-only access to the repository and include the credentials in the URL:

- `https://username@token:github.com/organization/project/archive/develop.tar.gz?path=project-develop`

With this method, you supply a full URL to request a tarball of the repository and it does not refer to the repository itself. You can also reference private enterprise versions of GitHub or GitLab and the repository doesn't need to be on the public `github.com` or `gitlab.com` sites.

The last case is a reference to an OCI image artifact stored on a registry. This is not a full container image with the operating system, but an image containing only the files making up the workshop content. The URI formats for this are:

- `imgpkg+https://harbor.example.com/organisation/project:version` - Use the workshop content from the top-level directory of the unpacked OCI artifact. The registry in this case must support `https`.
- `imgpkg+https://harbor.example.com/organisation/project:version?path=subdir` - Use the workshop content from the subdirectory path of the unpacked OCI artifact you specify. The registry in this case must support `https`.
- `imgpkg+http://harbor.example.com/organisation/project:version` - Use the workshop content from the top-level directory of the unpacked OCI artifact. The registry in this case can only support `http`.
- `imgpkg+http://harbor.example.com/organisation/project:version?path=subdir` - Use the workshop content from the subdirectory path of the unpacked OCI artifact you specify. The registry in this case can only support `http`.

You can use `imgpkg://` instead of the prefix `imgpkg+https://`. The registry in this case must still support `https`.

For any of the formats, you can supply credentials as part of the URI:

- `imgpkg+https://username:password@harbor.example.com/organisation/project:version`

Access to the registry using a secure connection of `https` must have a valid certificate.

You can create the OCI image artifact by using `imgpkg` from the Carvel tool set. For example, from the top-level directory of the Git repository containing the workshop content, run:

```
imgpkg push -i harbor.example.com/organisation/project:version -f .
```

In all cases for downloading workshop content, the `workshop` subdirectory holding the actual workshop content is relocated to `/opt/workshop` so that it is not visible to a user. If you want to ignore other files so the user can not see them, you can supply a `.eduk8signore` file in your repository or tarball and list patterns for the files in it.

The contents of the `.eduk8signore` file are processed as a list of patterns and each is applied recursively to subdirectories. To ensure that a file is only ignored if it resides in the root directory, prefix it with `./`:

```
./.dockerignore
./.gitignore
./Dockerfile
./LICENSE
./README.md
./kustomization.yaml
./resources
```

Container image for the workshop

When you bundle the workshop content into a container image, the `content.image` field must specify the image reference identifying the location of the container image that you will deploy for the workshop instance:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  title: Markdown Sample
  description: A sample workshop using Markdown
  content:
    image: {YOUR-REGISTRY-URL}/lab-markdown-sample:main
```

Even though you can download workshop content when the workshop environment starts, you might still want to override the workshop image that is used as a base. You can do this when you have a custom workshop base image that includes added language runtimes or tools that the specialized workshops require.

For example, if running a Java workshop, you can enter the `jdk11-environment` for the workshop image. The workshop content is still downloaded from GitHub:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-spring-testing
spec:
  title: Spring Testing
  description: Playground for testing Spring development
  content:
    image: registry.tanzu.vmware.com/learning-center/jdk11-environment:latest
    files: {YOUR-GIT-REPO-URL}/lab-spring-testing
```

If you want to use the latest version of an image, always include the `:latest` tag. This is important because the Learning Center Operator looks for version tags `:main`, `:main`, `:develop` and `:latest`. When using these tags, the Operator sets the image pull policy to `Always` to ensure that a newer version is always pulled if available. Otherwise, the image is cached on the Kubernetes nodes and only pulled when it is initially absent. Any other version tags are always assumed to be unique and are never updated. Be aware of image registries that use a content delivery network (CDN) as front end. When using these image tags, the CDN can still regard them as unique and not do pull through requests to update an image even if it uses a tag of `:latest`.

When special custom workshop base images are available as part of the Learning Center project, instead of specifying the full location for the image, including the image registry, you can specify a short name. The Learning Center Operator then fills in the rest of the details:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-spring-testing
spec:
  title: Spring Testing
  description: Playground for testing Spring development
  content:
    image: jdk11-environment:latest
    files: github.com/eduk8s-tests/lab-spring-testing
```

The supported short versions of the names are:

- `base-environment:*`: A tagged version of the `base-environment` workshop image matched with the current version of the Learning Center Operator.

The `*` variants of the short names map to the most up-to-date version of the image available when the version of the Learning Center Operator was released. That version is guaranteed to work with that version of the Learning Center Operator. The `latest` version can be newer, with possible incompatibilities.

If required, you can remap the short names in the `SystemProfile` configuration of the Learning Center Operator. You can map additional short names to your own custom workshop base images for your own deployment of the Learning Center Operator, and with any of your own workshops.

Setting environment variables

To set or override environment variables for the workshop instance, you can supply the `session.env` field:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  title: Markdown Sample
  description: A sample workshop using Markdown
  content:
    files: {YOUR-GIT-REPO-URL}/lab-markdown-sample
  session:
    env:
    - name: REPOSITORY-URL
      value: YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE
```

Where:

- The `session.env` field is a list of dictionaries with the `name` and `value` fields.
- The `value` field is the Git repository for `lab-markdown-sample`. For example, `{YOUR-GIT-REPO-URL}/lab-markdown-sample`.

Values of fields in the list of resource objects can reference a number of predefined parameters. The available parameters are:

- `session_id`: A unique ID for the workshop instance within the workshop environment.
- `session_namespace`: The namespace you create for and bind to the workshop instance. This is the namespace unique to the session. A workshop can create its own resources.
- `environment_name`: The name of the workshop environment. Its current value is the name of the namespace for the workshop environment and subject to change.
- `workshop_namespace`: The namespace for the workshop environment. This is the namespace where you create all deployments of the workshop instances. It is also the namespace where the service account that the workshop instance runs.
- `service_account`: The name of the service account that the workshop instance runs as. It has access to the namespace you create for that workshop instance.
- `ingress_domain`: The host domain under which you can create host names when creating ingress routes.
- `ingress_protocol`: The protocol (http/https) you use for ingress routes and create for workshops.

The syntax for referencing the parameters is `$(parameter_name)`.

Use the `session.env` field to override environment variables only when they are required for the workshop. To set or override an environment for a specific workshop environment, set environment variables in the `WorkshopEnvironment` custom resource for the workshop environment instead.

Overriding the memory available

By default the container the workshop environment runs in is allocated 512Mi. If the editor is enabled, a total of 1Gi is allocated.

The memory allocation is sufficient for the workshop that is mainly aimed at deploying workloads into the Kubernetes cluster. If you run workloads in the workshop environment container and need more memory, you can override the default by setting `memory` under `session.resources`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  title: Markdown Sample
  description: A sample workshop using Markdown
  content:
    image: {YOUR-REGISTRY-URL}/lab-markdown-sample:main
  session:
    resources:
      memory: 2Gi
```

Mounting a persistent volume

In circumstances where a workshop needs persistent storage to ensure no loss of work, you can request a persistent volume be mounted into the workshop container after the workshop environment container is stopped and restarted:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  title: Markdown Sample
  description: A sample workshop using Markdown
  content:
    image: {YOUR-REGISTRY-URL}/lab-markdown-sample:main
  session:
    resources:
      storage: 5Gi
```

The persistent volume is mounted on top of the `/home/eduk8s` directory. Because this hides any workshop content bundled with the image, an init container is automatically configured and run, which copies the contents of the home directory to the persistent volume before the persistent volume is mounted on top of the home directory.

Resource budget for namespaces

In conjunction with each workshop instance, a namespace is created during the workshop. From the terminal of the workshop, you can deploy dashboard applications into the namespace through the Kubernetes REST API by using tools such as kubectl.

By default, this namespace has all the limit ranges and resource quotas the Kubernetes cluster can enforce. In most cases, this means there are no limits or quotas.

To control how much resources you can use when you set no limit ranges and resource quotas, or override any default limit ranges and resource quotas, you can set a resource budget for any namespace of the workshop instance in the `session.namespaces.budget` field:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  title: Markdown Sample
  description: A sample workshop using Markdown
  content:
    image: {YOUR-REGISTRY-URL}/lab-markdown-sample:main
  session:
    namespaces:
      budget: small
```

The resource budget sizings and quotas for CPU and memory are:

Budget	CPU	Memory
small	1000m	1Gi
medium	2000m	2Gi
large	4000m	4Gi
x-large	8000m	8Gi
xx-large	8000m	12Gi

Budget	CPU	Memory
xxx-large	8000m	16Gi

A value of 1000m is equivalent to 1 CPU.

Separate resource quotas for CPU and memory are applied for terminating and non-terminating workloads.

Only the CPU and memory quotas are listed in the preceding table, but limits also apply to the number of resource objects of certain types you can create, such as:

- persistent volume claims
- replication controllers
- services
- secrets

For each budget type, a limit range is created with fixed defaults. The limit ranges for CPU usage on a container are as follows:

Budget	Minimum	Maximum	Request	Limit
small	50m	1000m	50m	250m
medium	50m	2000m	50m	500m
large	50m	4000m	50m	500m
x-large	50m	8000m	50m	500m
xx-large	50m	8000m	50m	500m
xxx-large	50m	8000m	50m	500m

The limit ranges for memory are as follows:

Budget	Minimum	Maximum	Request	Limit
small	32Mi	1Gi	128Mi	256Mi
medium	32Mi	2Gi	128Mi	512Mi
large	32Mi	4Gi	128Mi	1Gi
x-large	32Mi	8Gi	128Mi	2Gi
xx-large	32Mi	12Gi	128Mi	2Gi
xxx-large	32Mi	16Gi	128Mi	2Gi

The request and limit values are the defaults of a container when there is no resources specification in a pod specification.

You can supply overrides in `session.namespaces.limits` to override the limit ranges and defaults for request and limit values when a budget sizing for CPU and memory is sufficient and there is no resources specification in a pod specification:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  title: Markdown Sample
  description: A sample workshop using Markdown
  content:
    image: {YOUR-REGISTRY-URL}/lab-markdown-sample:main
  session:
    namespaces:
      budget: medium
      limits:
        min:
          cpu: 50m
          memory: 32Mi
        max:
          cpu: 1
          memory: 1Gi
        defaultRequest:
          cpu: 50m
          memory: 128Mi
        default:
          cpu: 500m
          memory: 1Gi
```

Although all the configurable properties are listed in this example, you only need to supply the property for the value that you want to override.

If you need more control over the limit ranges and resource quotas, you can set the resource budget to `custom`. This removes any default limit ranges and resource quota that might be applied to the namespace. You can enter your own `LimitRange` and `ResourceQuota` resources as part of the list of resources created for each session.

Before disabling the quota and limit ranges or contemplating any switch to using a custom set of `LimitRange` and `ResourceQuota` resources, consider if that is what is really required.

The default requests defined by these for memory and CPU are fallbacks only. In most cases, instead of changing the defaults, you can enter the memory and CPU resources in the pod template specification of your deployment resources used in the workshop to indicate what the application requires. This allows you to control exactly what the application can use and so fit into the minimum quota required for the task.

This budget setting and the memory values are distinct from the amount of memory the container the workshop environment runs in. To change how much memory is available to the workshop container, set the `memory` setting under `session.resources`.

Patching workshop deployment

In order to set or override environment variables, you can provide `session.env`. To make other changes to the Pod template for the deployment used to create the workshop instance, provide an overlay patch. You can use this patch to override the default CPU and memory limit applied to the workshop instance or to mount a volume.

The patches are provided by setting `session.patches`. The patch is applied to the `spec` field of the pod template:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-resource-testing
spec:
  title: Resource testing
  description: Play area for testing memory resources
  content:
    files: github.com/eduk8s-tests/lab-resource-testing
  session:
    patches:
      containers:
      - name: workshop
        resources:
          requests:
            memory: "1Gi"
          limits:
            memory: "1Gi"
```

In this example, the default memory limit of “512Mi” is increased to “1Gi”. Although memory is set using a patch in this example, the `session.resources.memory` field is the preferred way to override the memory allocated to the container the workshop environment is running in.

The patch works differently than overlay patches that you can find elsewhere in Kubernetes. Specifically, when patching an array and the array contains a list of objects, a search is performed on the destination array. If an object already exists with the same value for the `name` field, the item in the source array is overlaid on top of the existing item in the destination array.

If there is no matching item in the destination array, the item in the source array is added to the end of the destination array.

This means an array doesn't outright replace an existing array, but a more intelligent merge is performed of elements in the array.

Creation of session resources

When a workshop instance is created, the deployment running the workshop dashboard is created in the namespace for the workshop environment. When more than one workshop instance is created under that workshop environment, all those deployments are in the same namespace.

For each workshop instance, a separate empty namespace is created with name corresponding to the workshop session. The workshop instance is configured so that the service account that the workshop instance runs under can access and create resources in the namespace created for that workshop instance. Each separate workshop instance has its own corresponding namespace and cannot see the namespace for another instance.

To pre-create additional resources within the namespace for a workshop instance, you can supply a list of the resources against the `session.objects` field within the workshop definition. You might use this to add additional custom roles to the service account for the workshop instance when working in that namespace or to deploy a distinct instance of an application for just that workshop instance, such as a private image registry:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-registry-testing
spec:
  title: Registry Testing
  description: Play area for testing image registry
  content:
    files: github.com/eduk8s-tests/lab-registry-testing
  session:
    objects:
    - apiVersion: apps/v1
      kind: Deployment
      metadata:
        name: registry
      spec:
        replicas: 1
        selector:
          matchLabels:
            deployment: registry
        strategy:
          type: Recreate
        template:
          metadata:
            labels:
              deployment: registry
          spec:
            containers:
            - name: registry
              image: registry.hub.docker.com/library/registry:2.6.1
              imagePullPolicy: IfNotPresent
              ports:
              - containerPort: 5000
                protocol: TCP
              env:
              - name: REGISTRY_STORAGE_DELETE_ENABLED
                value: "true"
    - apiVersion: v1
      kind: Service
      metadata:
        name: registry
      spec:
        type: ClusterIP
        ports:
        - port: 80
          targetPort: 5000
        selector:
          deployment: registry
```

For namespaced resources, it is not necessary to enter the `namespace` field of the resource `metadata`. When the `namespace` field is not present, the resource is created within the session namespace for that workshop instance.

When resources are created, owner references are added, making the `WorkshopSession` custom resource corresponding to the workshop instance the owner. This means that when the workshop instance is deleted, any resources are deleted.

Values of fields in the list of resource objects can reference a number of predefined parameters. The available parameters are:

- `session_id`: A unique ID for the workshop instance within the workshop environment.
- `session_namespace`: The namespace you create for and bound to the workshop instance. This is the namespace unique to the session and where a workshop can create its own resources.
- `environment_name`: The name of the workshop environment. Its current value is the name of the namespace for the workshop environment and subject to change.
- `workshop_namespace`: The namespace for the workshop environment. This is the namespace where you create all deployments of the workshop instances. It is also the namespace where the service account that the workshop instance runs.
- `service_account`: The name of the service account the workshop instance runs as and which has access to the namespace you create for that workshop instance.
- `ingress_domain`: The host domain under which you can create host names when creating ingress routes.
- `ingress_protocol`: The protocol (http/https) you use for ingress routes and create for workshops.

The syntax for referencing the parameter is `$(parameter_name)`.

For cluster-scoped resources, you must set the name of the created resource so that it embeds the value of `$(session_namespace)`. This way the resource name is unique to the workshop instance, and you do not get a clash with a resource for a different workshop instance.

For examples of making use of the available parameters, see the following sections.

Overriding default role-based access control (RBAC) rules

By default the service account created for the workshop instance has `admin` role access to the session namespace created for that workshop instance. This enables the service account to be used to deploy applications to the session namespace and manage secrets and service accounts.

Where a workshop doesn't require `admin` access for the namespace, you can reduce the level of access it has to `edit` or `view` by setting the `session.namespaces.role` field:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-role-testing
spec:
  title: Role Testing
  description: Play area for testing roles
  content:
    files: github.com/eduk8s-tests/lab-role-testing
  session:
    namespaces:
      role: view
```

To add additional roles to the service account, such as working with custom resource types added to the cluster, you can add the appropriate `Role` and `RoleBinding` definitions to the `session.objects` field described previously:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-kpack-testing
spec:
  title: Kpack Testing
  description: Play area for testing kpack
  content:
    files: github.com/eduk8s-tests/lab-kpack-testing
  session:
    objects:
    - apiVersion: rbac.authorization.k8s.io/v1
      kind: Role
      metadata:
        name: kpack-user
      rules:
      - apiGroups:
        - build.pivotal.io
        resources:
        - builds
        - builders
        - images
        - sourceresolvers
        verbs:
        - get
        - list
        - watch
        - create
        - delete
        - patch
        - update
    - apiVersion: rbac.authorization.k8s.io/v1
      kind: RoleBinding
      metadata:
        name: kpack-user
      roleRef:
        apiGroup: rbac.authorization.k8s.io
        kind: Role
        name: kpack-user
      subjects:
      - kind: ServiceAccount
        namespace: $(workshop_namespace)
        name: $(service_account)
```

Because the subject of a `RoleBinding` must specify the service account name and namespace it is contained within, both of which are unknown in advance, references to parameters for the workshop namespace and service account for the workshop instance are used when defining the subject.

You can add additional resources with `session.objects` to grant cluster-level roles and the service account `cluster-admin` role:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-admin-testing
spec:
  title: Admin Testing
  description: Play area for testing cluster admin
  content:
    files: github.com/eduk8s-tests/lab-admin-testing
  session:
    objects:
    - apiVersion: rbac.authorization.k8s.io/v1
      kind: ClusterRoleBinding
      metadata:
```

```
      name: $(session_namespace)-cluster-admin
    roleRef:
      apiGroup: rbac.authorization.k8s.io
      kind: ClusterRole
      name: cluster-admin
    subjects:
    - kind: ServiceAccount
      namespace: $(workshop_namespace)
      name: $(service_account)
```

In this case, the name of the cluster role binding resource embeds `$(session_namespace)` so that its name is unique to the workshop instance and doesn't overlap with a binding for a different workshop instance.

Running user containers as root

In addition to RBAC, which controls what resources a user can create and work with, Pod security policies are applied to restrict what Pods/containers a user deploys can do.

By default the deployments that a workshop user can create are allowed only to run containers as a non-root user. This means that many container images available on registries such as Docker Hub cannot be used.

If you are creating a workshop where a user must run containers as the root user, you must override the default `nonroot` security policy and select the `anyuid` security policy by using the `session.namespaces.security.policy` setting:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-policy-testing
spec:
  title: Policy Testing
  description: Play area for testing security policies
  content:
    files: github.com/eduk8s-tests/lab-policy-testing
  session:
    namespaces:
      security:
        policy: anyuid
```

This setting applies to the primary session namespace and any secondary namespaces created.

Creating additional namespaces

For each workshop instance, a primary session namespace is created. You can deploy or pre-deploy applications into this namespace as part of the workshop.

If you need more than one namespace per workshop instance, you can create secondary namespaces in a couple of ways.

If the secondary namespaces are to be created empty, you can list the details of the namespaces under the property `session.namespaces.secondary`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-namespace-testing
spec:
  title: Namespace Testing
  description: Play area for testing namespaces
  content:
    files: github.com/eduk8s-tests/lab-namespace-testing
  session:
    namespaces:
      role: admin
      budget: medium
      secondary:
```

```
      - name: $(session_namespace)-apps
        role: edit
        budget: large
        limits:
          default:
            memory: 512mi
```

When secondary namespaces are created, by default, the role, resource quotas, and limit ranges are set the same as the primary session namespace. Each namespace has a separate resource budget and it is not shared.

If required, you can override what `role`, `budget`, and `limits` are applied within the entry for the namespace.

Similarly, you can override the security policy for secondary namespaces on a case-by-case basis by adding the `security.policy` setting under the entry for the secondary namespace.

To create resources in the namespaces you create, create the namespaces by adding an appropriate `Namespace` resource to `session.objects` with the definitions of the resources you want to create in the namespaces:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-namespace-testing
spec:
  title: Namespace Testing
  description: Play area for testing namespaces
  content:
    files: github.com/eduk8s-tests/lab-namespace-testing
  session:
    objects:
    - apiVersion: v1
      kind: Namespace
      metadata:
        name: $(session_namespace)-apps
```

When listing any other resources to be created within the added namespace, such as deployments, ensure that the `namespace` is set in the `metadata` of the resource. For example, `$(session_namespace)-apps`.

To override what role the service account for the workshop instance has in the added namespace, you can set the `learningcenter.tanzu.vmware.com/session.role` annotation on the `Namespace` resource:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-namespace-testing
spec:
  title: Namespace Testing
  description: Play area for testing namespaces
  content:
    files: github.com/eduk8s-tests/lab-namespace-testing
  session:
    objects:
    - apiVersion: v1
      kind: Namespace
      metadata:
        name: $(session_namespace)-apps
        annotations:
          learningcenter.tanzu.vmware.com/session.role: view
```

To have a different resource budget set for the additional namespace, you can add the annotation `learningcenter.tanzu.vmware.com/session.budget` in the `Namespace` resource metadata and set the value to the required resource budget:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
```

```
metadata:
  name: lab-namespace-testing
spec:
  title: Namespace Testing
  description: Play area for testing namespaces
  content:
    files: github.com/eduk8s-tests/lab-namespace-testing
  session:
    objects:
    - apiVersion: v1
      kind: Namespace
      metadata:
        name: $(session_namespace)-apps
        annotations:
          learningcenter.tanzu.vmware.com/session.budget: large
```

To override the limit range values applied corresponding to the budget applied, you can add annotations starting with `learningcenter.tanzu.vmware.com/session.limits.` for each entry:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-namespace-testing
spec:
  title: Namespace Testing
  description: Play area for testing namespaces
  content:
    files: github.com/eduk8s-tests/lab-namespace-testing
  session:
    objects:
    - apiVersion: v1
      kind: Namespace
      metadata:
        name: $(session_namespace)-apps
        annotations:
          learningcenter.tanzu.vmware.com/session.limits.min.cpu: 50m
          learningcenter.tanzu.vmware.com/session.limits.min.memory: 32Mi
          learningcenter.tanzu.vmware.com/session.limits.max.cpu: 1
          learningcenter.tanzu.vmware.com/session.limits.max.memory: 1Gi
          learningcenter.tanzu.vmware.com/session.limits.defaultrequest.cpu: 50m
          learningcenter.tanzu.vmware.com/session.limits.defaultrequest.memory: 128Mi
          learningcenter.tanzu.vmware.com/session.limits.request.cpu: 500m
          learningcenter.tanzu.vmware.com/session.limits.request.memory: 1Gi
```

You only must supply annotations for the values you want to override.

If you need more fine-grained control over the limit ranges and resource quotas, set the value of the annotation for the budget to `custom` and add the `LimitRange` and `ResourceQuota` definitions to `session.objects`.

In this case you must set the `namespace` for the `LimitRange` and `ResourceQuota` resource to the name of the namespace, e.g., `$(session_namespace)-apps` so they are only applied to that namespace.

To set the security policy for a specific namespace other than the primary session namespace, you can add the annotation `learningcenter.tanzu.vmware.com/session.security.policy` in the `Namespace` resource metadata and set the value to `nonroot`, `anyuid`, or `custom` as necessary.

Shared workshop resources

Adding a list of resources to `session.objects` causes the given resources to be created for each workshop instance, whereas namespaced resources default to being created in the session namespace for a workshop instance.

If instead you want to have one common shared set of resources created once for the whole workshop environment, that is, used by all workshop instances, you can list them in the `environment.objects` field.

This might, for example, be used to deploy a single container image registry used by all workshop instances, with a Kubernetes job used to import a set of images into the container image registry, which are then referenced by the workshop instances.

For namespaced resources, it is not necessary to enter the `namespace` field of the resource `metadata`. When the `namespace` field is not present, the resource is created within the workshop namespace for that workshop environment.

When resources are created, owner references are added, making the `WorkshopEnvironment` custom resource correspond to the workshop environment of the owner. This means that when the workshop environment is deleted, any resources are also deleted.

Values of fields in the list of resource objects can reference a number of predefined parameters. The available parameters are:

- `workshop_name`: The name of the workshop. This is the name of the `Workshop` definition the workshop environment was created against.
- `environment_name`: The name of the workshop environment. Its current value is the name of the namespace for the workshop environment and subject to change.
- `environment_token`: The value of the token that must be used in workshop requests against the workshop environment.
- `workshop_namespace`: The namespace for the workshop environment. This is the namespace where all deployments of the workshop instances, and their service accounts, are created. It is the same namespace that shared workshop resources are created.
- `service_account`: The name of a service account you can use when creating deployments in the workshop namespace.
- `ingress_domain`: The host domain under which you can create host names when creating ingress routes.
- `ingress_protocol`: The protocol (http/https) used for ingress routes created for workshops.
- `ingress_secret`: The name of the ingress secret stored in the workshop namespace when secure ingress is used.

To create additional namespaces associated with the workshop environment, embed a reference to `$(workshop_namespace)` in the name of the additional namespaces with an appropriate suffix. Be careful that the suffix doesn't overlap with the range of session IDs for workshop instances.

When creating deployments in the workshop namespace, set the `serviceAccountName` of the `Deployment` resource to `$(service_account)`. This ensures the deployment makes use of a special Pod security policy set up by the Learning Center. If this isn't used and the cluster imposes a more strict default Pod security policy, your deployment might not work, especially if any image runs as `root`.

Workshop pod security policy

The pod for the workshop session is set up with a pod security policy that restricts what you can do from containers in the pod. The nature of the applied pod security policy is adjusted when enabling support for doing Docker builds. This in turn enables Docker builds inside the sidecar container attached to the workshop container.

If you are customizing the workshop by patching the pod specification using `session.patches` to add your own sidecar container, and that sidecar container must run as the root user or needs a custom pod security policy, you must override the default security policy for the workshop container.

To allow a sidecar container to run as the root user with no extra privileges required, you can override the default `nonroot` security policy and set it to `anyuid`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-policy-testing
```

```
spec:
  title: Policy Testing
  description: Play area for testing security policies
  content:
    files: github.com/eduk8s-tests/lab-policy-testing
  session:
    security:
      policy: anyuid
```

This is a different setting than described previously for changing the security policy for deployments made by a workshop user to the session namespaces. This setting applies only to the workshop container itself.

If you need more fine-grained control of the security policy, you must provide your own resources for defining the Pod security policy and map it so it is used. The details of the pod security policy must be in `environment.objects` and mapped by definitions added to `session.objects`. For this to be used, you must deactivate the application of the inbuilt pod security policies. You can do this by setting `session.security.policy` to `custom`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-policy-testing
spec:
  title: Policy Testing
  description: Play area for testing policy override
  content:
    files: github.com/eduk8s-tests/lab-policy-testing
  session:
    security:
      policy: custom
    objects:
    - apiVersion: rbac.authorization.k8s.io/v1
      kind: RoleBinding
      metadata:
        namespace: $(workshop_namespace)
        name: $(session_namespace)-podman
      roleRef:
        apiGroup: rbac.authorization.k8s.io
        kind: ClusterRole
        name: $(workshop_namespace)-podman
      subjects:
      - kind: ServiceAccount
        namespace: $(workshop_namespace)
        name: $(service_account)
  environment:
    objects:
    - apiVersion: policy/v1beta1
      kind: PodSecurityPolicy
      metadata:
        name: aa-$(workshop_namespace)-podman
      spec:
        privileged: true
        allowPrivilegeEscalation: true
        requiredDropCapabilities:
        - KILL
        - MKNOD
        hostIPC: false
        hostNetwork: false
        hostPID: false
        hostPorts: []
        runAsUser:
          rule: MustRunAsNonRoot
        seLinux:
          rule: RunAsAny
        fsGroup:
          rule: RunAsAny
        supplementalGroups:
          rule: RunAsAny
        volumes:
        - configMap
        - downwardAPI
        - emptyDir
        - persistentVolumeClaim
        - projected
        - secret
    - apiVersion: rbac.authorization.k8s.io/v1
      kind: ClusterRole
      metadata:
        name: $(workshop_namespace)-podman
      rules:
      - apiGroups:
        - policy
        resources:
        - podsecuritypolicies
        verbs:
        - use
        resourceNames:
        - aa-$(workshop_namespace)-podman
```

By overriding the pod security policy, you are responsible for limiting what you can do from the workshop pod. In other words, add only the extra capabilities you need. The pod security policy is applied only to the pod the workshop session runs in. It does not change any pod security policy applied to service accounts that exist in the session namespace or other namespaces you have created.

There is a better way to set the priority of applied Pod security policies when a default Pod security policy is applied globally by mapping it to the `system:authenticated` group. This causes priority falling back to the order of the names of the Pod security policies. VMware recommends you use `aa-` as a prefix to the custom Pod security name you create. This ensures it takes precedence over any global default Pod security policy such as `restricted`, `pks-restricted` or `vmware-system-tmc-restricted`, no matter what the name of the global policy default.

Custom security policies for user containers

You can also set the value of the `session.namespaces.security.policy` setting as `custom`. This gives you more fine-grained control of the security policy applied to the pods and containers that a user deploys during a session. In this case you must provide your own resources that define and map the pod security policy.

For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-policy-testing
spec:
  title: Policy Testing
  description: Play area for testing policy override
  content:
    files: github.com/eduk8s-tests/lab-policy-testing
  session:
    namespaes:
      security:
        policy: custom
    objects:
    - apiVersion: rbac.authorization.k8s.io/v1
      kind: RoleBinding
      metadata:
        namespace: $(workshop_namespace)
        name: $(session_namespace)-security-policy
      roleRef:
        apiGroup: rbac.authorization.k8s.io
        kind: ClusterRole
        name: $(workshop_namespace)-security-policy
      subjects:
      - kind: Group
        namespace: $(workshop_namespace)
        name: system:serviceaccounts:$(workshop_namespace)
  environment:
    objects:
    - apiVersion: policy/v1beta1
      kind: PodSecurityPolicy
      metadata:
        name: aa-$(workshop_namespace)-security-policy
      spec:
        privileged: true
        allowPrivilegeEscalation: true
        requiredDropCapabilities:
        - KILL
        - MKNOD
        hostIPC: false
        hostNetwork: false
        hostPID: false
        hostPorts: []
        runAsUser:
          rule: MustRunAsNonRoot
        seLinux:
          rule: RunAsAny
        fsGroup:
          rule: RunAsAny
        supplementalGroups:
          rule: RunAsAny
        volumes:
        - configMap
        - downwardAPI
        - emptyDir
        - persistentVolumeClaim
        - projected
        - secret
    - apiVersion: rbac.authorization.k8s.io/v1
      kind: ClusterRole
      metadata:
        name: $(workshop_namespace)-security-policy
      rules:
      - apiGroups:
        - policy
        resources:
        - podsecuritypolicies
        verbs:
        - use
        resourceNames:
        - aa-$(workshop_namespace)-security-policy
```

You can also do this on secondary namespaces by either changing the `session.namespaces.secondary.security.policy` setting to `custom` or using the `learningcenter.tanzu.vmware.com/session.security.policy: custom` annotation.

Defining additional ingress points

If running additional background applications, by default they are only accessible to other processes within the same container. For an application to be accessible to a user through their web browser, an ingress must be created mapping to the port for the application.

You can do this by supplying a list of the ingress points and the internal container port they map to by setting the `session.ingresses` field in the workshop definition:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    ingresses:
    - name: application
      port: 8080
```

The form of the host name used in the URL to access the service is:

```
$(session_namespace)-application.$(ingress_domain)
```

This name cannot be `terminal`, `console`, `slides`, `editor`, or the name of any built-in dashboard. These values are reserved for the corresponding built-in capabilities providing those features.

In addition to specifying ingresses for proxying to internal ports within the same Pod, you can enter a `host`, `protocol` and `port` corresponding to a separate service running in the Kubernetes cluster:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    ingresses:
    - name: application
      protocol: http
      host: service.namespace.svc.cluster.local
      port: 8080
```

You can use variables providing information about the current session within the `host` property if required:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    ingresses:
    - name: application
      protocol: http
      host: service.$(session_namespace).svc.cluster.local
      port: 8080
```

Available variables are:

- `session_namespace`: The namespace you create for and bind to the workshop instance. This is the namespace unique to the session and where a workshop can create its own resources.
- `environment_name`: The name of the workshop environment. Its current value is the name of the namespace for the workshop environment and subject to change.
- `workshop_namespace`: The namespace for the workshop environment. This is the namespace where you create all deployments of the workshop instances and where the service account that the workshop instance runs.
- `ingress_domain`: The host domain under which you can create host names when creating ingress routes.

If the service uses standard `http` or `https` ports, you can leave out the `port` property, and the port is set based on the value of `protocol`.

When a request is proxied, you can specify additional request headers that must be passed to the service:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    ingresses:
    - name: application
      protocol: http
      host: service.$(session_namespace).svc.cluster.local
      port: 8080
      headers:
      - name: Authorization
        value: "Bearer $(kubernetes_token)"
```

The value of a header can reference the following variable:

- `kubernetes_token`: The access token of the service account for the current workshop session, used for accessing the Kubernetes REST API.

Access controls enforced by the workshop environment or training portal protect accessing any service through the ingress. If you use the training portal, this must be transparent. Otherwise, supply any login credentials for the workshop again when prompted by your web browser.

External workshop instructions

In place of using workshop instructions provided with the workshop content, you can use externally hosted instructions instead. To do this set `sessions.applications.workshop.url` to the URL of an external web site:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      workshop:
        url: https://www.example.com/instructions
```

The external web site must displayed in an HTML iframe, is shown as is and must provide its own page navigation and table of contents if required.

The URL value can reference a number of predefined parameters. The available parameters are:

- `session_namespace`: The namespace you create for and bind to the workshop instance. This is the namespace unique to the session and where a workshop can create its own resources.
- `environment_name`: The name of the workshop environment. Its current value is the name of the namespace for the workshop environment and subject to change.
- `workshop_namespace`: The namespace for the workshop environment. This is the namespace where you create all deployments of the workshop instances and where the service account that the workshop instance runs.
- `ingress_domain`: The host domain under which you can create host names when creating ingress routes.
- `ingress_protocol`: The protocol (http/https) used for ingress routes that you create for workshops.

These could be used, for example, to reference workshops instructions hosted as part of the workshop environment:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      workshop:
        url: $(ingress_protocol)://$(workshop_namespace)-instructions.$(ingress_domai
n)
  environment:
    objects:
    - ...
```

In this case `environment.objects` of the workshop `spec` must include resources to deploy the application hosting the instructions and expose it through an appropriate ingress.

Deactivating workshop instructions

The aim of the workshop environment is to provide instructions for a workshop that users can follow. If you want instead to use the workshop environment as a development environment or as an administration console that provides access to a Kubernetes cluster, you can deactivate the display of workshop instructions provided with the workshop content. In this case, only the work area with the terminals, console, and so on, is displayed. To deactivate display of workshop instructions, add a `session.applications.workshop` section and set the `enabled` property to `false`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      workshop:
        enabled: false
```

Enabling the Kubernetes console

By default the Kubernetes console is not enabled. To enable it and make it available through the web browser when accessing a workshop, add a `session.applications.console` section to the workshop definition, and set the `enabled` property to `true`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      console:
        enabled: true
```

The Kubernetes dashboard provided by the Kubernetes project is used. To use Octant as the console, you can set the `vendor` property to `octant`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      console:
        enabled: true
        vendor: octant
```

When `vendor` is not set, `kubernetes` is assumed.

Enabling the integrated editor

By default the integrated web based editor is not enabled. To enable it and make it available through the web browser when accessing a workshop, add a `session.applications.editor` section to the workshop definition, and set the `enabled` property to `true`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      editor:
        enabled: true
```

The integrated editor used is based on Visual Studio Code. For more information about the editor, see https://github.com/cdr/code-server in GitHub.

To install additional VS Code extensions, do this from the editor. Alternatively, if building a custom workshop, you can install them from your `Dockerfile` into your workshop image by running:

```
code-server --install-extension vendor.extension
```

Replace `vendor.extension` with the name of the extension, where the name identifies the extension on the VS Code extensions marketplace used by the editor or provide a path name to a local `.vsix` file.

This installs the extensions into `$HOME/.config/code-server/extensions`.

If downloading extensions yourself and unpacking them or extensions are part of your Git repository, you can instead locate them in the `workshop/code-server/extensions` directory.

Enabling workshop downloads

You can provide a way for a workshop user to download files as part of the workshop content. Enable this by adding the `session.applications.files` section to the workshop definition and setting the `enabled` property to `true`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      files:
        enabled: true
```

The recommended way of providing access to files from workshop instructions is using the `files:download-file` clickable action block. This action ensures any file is downloaded to the local machine and is not displayed in the browser in place of the workshop instructions.

By default the user can access any files located under the home directory of the workshop user account. To restrict where the user can download files from, set the `directory` setting:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      files:
        enabled: true
        directory: exercises
```

When the specified directory is a relative path, it is evaluated relative to the home directory of the workshop user.

Enabling the test examiner

The test examiner is a feature that allows a workshop to have verification checks that the workshop instructions can trigger. The test examiner is deactivated by default. To enable it, add a `session.applications.examiner` section to the workshop definition and set the `enabled` property to `true`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      examiner:
        enabled: true
```

You must provide any executable test programs for verification checks in the `workshop/examiner/tests` directory.

The test programs must return an exit status of 0 if the test is successful and nonzero if it fails. Test programs must not be persistent programs that can run forever.

Clickable actions for the test examiner are used within the workshop instructions to trigger the verification checks. You can configure them to start when the page of the workshop instructions is loaded.

Enabling session image registry

Workshops using tools such as `kpack` or `tekton` and which need a place to push container images when built can enable a container image registry. A separate registry is deployed for each workshop session.

The container image registry is currently fully usable only if workshops are deployed under a Learning Center Operator configuration that uses secure ingress. This is because a registry that is not secure is not trusted by the Kubernetes cluster as the source of container images when doing deployments.

To enable the deployment of a registry per workshop session, add a `session.applications.registry` section to the workshop definition and set the `enabled` property to `true`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      registry:
        enabled: true
```

The registry mounts a persistent volume for storing of images. By default the size of that persistent volume is 5Gi. To override the size of the persistent volume, add the `storage` property under the `registry` section:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      registry:
        enabled: true
        storage: 20Gi
```

The amount of memory provided to the registry defaults to 768Mi. To increase this, add the `memory` property under the `registry` section.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      registry:
        enabled: true
        memory: 1Gi
```

The registry is secured with a user name and password unique to the workshop session, and must be accessed over a secure connection.

To allow access from the workshop session, the file `$HOME/.docker/config.json` containing the registry credentials are injected into the workshop session. This is used by tools such as `docker`.

For deployments in Kubernetes, a secret of type `kubernetes.io/dockerconfigjson` is created in the namespace and applied to the `default` service account in the namespace. This means deployments made using the default service account can pull images from the registry without additional configuration. If creating deployments using other service accounts, add configuration to the service account or deployment to add the registry secret for pulling images.

If you need access to the raw registry host details and credentials, they are provided as environment variables in the workshop session. The environment variables are:

- `REGISTRY_HOST`: Contains the host name for the registry for the workshop session.
- `REGISTRY_AUTH_FILE`: Contains the location of the `docker` configuration file. Must be the equivalent of `$HOME/.docker/config.json`.
- `REGISTRY_USERNAME`: Contains the user name for accessing the registry.
- `REGISTRY_PASSWORD`: Contains the password for accessing the registry. This is different for each workshop session.
- `REGISTRY_SECRET`: Contains the name of a Kubernetes secret of type `kubernetes.io/dockerconfigjson` added to the session namespace, which contains the registry credentials.

The URL for accessing the registry adopts the HTTP protocol scheme inherited from the environment variable `INGRESS_PROTOCOL`. This is the same HTTP protocol scheme the workshop sessions use.

To use any of the variables as data variables in workshop content, use the same variable name but in lowercase: `registry_host`, `registry_auth_file`, `registry_username`, `registry_password` and `registry_secret`.

Enabling ability to use Docker

To build container images in a workshop using `docker`, first enable it. Each workshop session is provided with its own separate Docker daemon instance running in a container.

Enabling support for running `docker` requires the use of a privileged container for running the Docker daemon. Because of the security implications of providing access to Docker with this configuration, VMware recommends that if you don't trust the people taking the workshop, any workshops that require Docker only be hosted in a disposable Kubernetes cluster that is destroyed at the completion of the workshop. You must not enable Docker for workshops hosted on a public service that is always kept running and where arbitrary users can access the workshops.

To enable support for using `docker` add a `session.applications.docker` section to the workshop definition and set the `enabled` property to `true`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      docker:
        enabled: true
```

The container that runs the Docker daemon mounts a persistent volume for storing of images which are pulled down or built locally. By default the size of that persistent volume is 5Gi. To override the size of the persistent volume, add the `storage` property under the `docker` section:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
```

```
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      docker:
        enabled: true
        storage: 20Gi
```

The amount of memory provided to the container running the Docker daemon defaults to 768Mi. To increase this, add the `memory` property under the `registry` section:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      docker:
        enabled: true
        memory: 1Gi
```

Access to the Docker daemon from the workshop session uses a local UNIX socket shared with the container running the Docker daemon. If it uses a local tool to access the socket connection for the Docker daemon directly rather than by running `docker`, it must use the `DOCKER_HOST` environment variable to set the location of the socket.

The Docker daemon is only available from within the workshop session and cannot be accessed outside of the pod by any tools deployed separately to Kubernetes.

Enabling WebDAV access to files

You can access or update local files within the workshop session from the terminal command line or editor of the workshop dashboard. The local files reside in the file system of the container the workshop session is running in.

To access the files remotely, you can enable WebDAV support for the workshop session.

To enable support for accessing files over WebDAV, add a `session.applications.webdav` section to the workshop definition, and set the `enabled` property to `true`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      webdav:
        enabled: true
```

This causes a WebDAV server running within the workshop session environment. A set of credentials is also generated and are available as environment variables. The environment variables are:

- `WEBDAV_USERNAME`: Contains the user name that must be used when authenticating over WebDAV.

- WEBDAV_PASSWORD: Contains the password that must be used when authenticating over WebDAV.

To use any of the environment variables related to the container image registry as data variables in workshop content, declare this in the workshop/modules.yaml file in the config.vars section:

```
config:
  vars:
  - name: WEBDAV_USERNAME
  - name: WEBDAV_PASSWORD
```

The URL endpoint for accessing the WebDAV server is the same as the workshop session, with /webdav/ path added. This can be constructed from the terminal using:

```
$INGRESS_PROTOCOL://$SESSION_NAMESPACE.$INGRESS_DOMAIN/webdav/
```

In workshop content it can be constructed using:

```
{{ingress_protocol}}://{{session_namespace}}.{{ingress_domain}}/webdav/
```

You can use WebDAV client support provided by your operating system or by using a standalone WebDAV client, such as CyberDuck.

Using WebDAV can make it easier to transfer files to or from the workshop session.

Customizing the terminal layout

By default a single terminal is provided in the web browser when accessing the workshop. If required, you can enable alternate layouts which provide additional terminals. To set the layout, add the session.applications.terminal section and include the layout property with the desired layout:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      terminal:
        enabled: true
        layout: split
```

The options for the layout property are:

- default: Single terminal.
- split: Two terminals stacked above each other in ratio 60/40.
- split/2: Three terminals stacked above each other in ratio 50/25/25.
- lower: A single terminal is placed below any dashboard tabs, rather than being a tab of its own. The ratio of dashboard tab to terminal is 70/30.
- none: No terminal is displayed but can still be created from the drop down menu.

When adding the terminal section, you must include the enabled property and set it to true as it is a required field when including the section.

If you don't want a terminal displayed and also want to deactivate the ability to create terminals from the drop-down menu, set enabled to false.

Adding custom dashboard tabs

Exposed applications, external sites and additional terminals, can be given their own custom dashboard tab. This is done by specifying the list of dashboard panels and the target URL:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    ingresses:
    - name: application
      port: 8080
    dashboards:
    - name: Internal
      url: "$(ingress_protocol)://$(session_namespace)-application.$(ingress_domain)/"
    - name: External
      url: http://www.example.com
```

The URL values can reference a number of predefined parameters. The available parameters are:

- `session_namespace`: The namespace you create for and bind to the workshop instance. This is the namespace unique to the session and where a workshop can create its own resources.
- `environment_name`: The name of the workshop environment. Its current value is the name of the namespace for the workshop environment and subject to change.
- `workshop_namespace`: The namespace for the workshop environment. This is the namespace where all deployments of the workshop instances you create and where the service account that the workshop instance runs.
- `ingress_domain`: The host domain under which you can create host names when creating ingress routes.
- `ingress_protocol`: The protocol (http/https) used for ingress routes that you create for workshops.

The URL can reference an external web site, however, that web site must not prohibit being embedded in an HTML iframe.

In the case of wanting to have a custom dashboard tab provide an additional terminal, the `url` property must use the form `terminal:<session>`, where `<session>` is replaced with the name of the terminal session. The name of the terminal session can be any name you choose, but must be restricted to lowercase letters, numbers, and dashes. You should avoid using numeric terminal session names such as “1”, “2”, and “3” as these are used for the default terminal sessions.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    dashboards:
    - name: Example
      url: terminal:example
```

Configure the WorkshopEnvironment resource

This topic describes how you configure the `WorkshopEnvironment` custom resource, which defines a Learning Center workshop environment.

Specifying the workshop definition

Creating a workshop environment is performed as a separate step to loading the workshop definition. This allows multiple distinct workshop environments using the same workshop definition to be created if necessary.

To specify which workshop definition is to be used for a workshop environment, set the `workshop.name` field of the specification for the workshop environment.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
```

The workshop environment name specified in the workshop environment metadata does not need to be the same. It has to be different if you create multiple workshop environments from the same workshop definition.

When the workshop environment is created, the namespace created for the workshop environment uses the `name` specified in the `metadata`. This name is also used in the unique names of each workshop instance created under the workshop environment.

Overriding environment variables

A workshop definition can set a list of environment variables that must be set for all workshop instances. To override an environment variable specified in the workshop definition. or one defined in the container image, you can supply a list of environment variables as `session.env`.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  session:
    env:
    - name: REPOSITORY-URL
      value: YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE
```

Where `YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE` is the Git repository URL for `lab-markdown-sample`. For example, `{YOUR-GIT-REPO-URL}/lab-markdown-sample`.

You can use this to set the location of a back-end service, such as an image registry, used by the workshop.

Values of fields in the list of resource objects can reference several predefined parameters. The available parameters are:

- `session_` - A unique ID for the workshop instance within the workshop environment.
- `session_` - The namespace created for and bound to the workshop instance. This is the namespace unique to the session and where a workshop can create its own resources.
- `environment_` - The name of the workshop environment. Currently, this is the same as the name of the namespace for the workshop environment. It is suggested that you do not rely on workshop environment name and namespace being the same, and use the most appropriate to cope with any future change.
- `workshop_` - The namespace for the workshop environment. This is the namespace where all deployments of the workshop instances are created and where the workshop instance runs the service account exists.

- `service_` - The workshop instance service account's name and access to the namespace created for that workshop instance.
- `ingress_` - The host domain under which host names are created when creating ingress routes.
- `ingress_` - The protocol (http/https) used for ingress routes created for workshops.

The syntax for referencing one of the parameters is `$(parameter_name)`.

Overriding the ingress domain

To access a workshop instance using a public URL, you must specify an ingress domain. If an ingress domain is not specified, the default ingress domain that the Learning Center operator configured with is used.

When setting a custom domain, DNS must be configured with a wildcard domain to forward all requests for subdomains of the custom domain to the ingress router of the Kubernetes cluster.

To provide the ingress domain, you can set the `session.ingress.domain` field.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  session:
    ingress:
      domain: training.learningcenter.tanzu.vmware.com
```

By default, the workshop session is exposed using an HTTP connection if overriding the domain. If you require a secure HTTPS connection, you must have access to a wildcard SSL certificate for the domain. A secret of type `tls` must be created for the certificate in the `learningcenter` namespace or the namespace where the Learning Center Operator is deployed. The name of that secret must then be set in the `session.ingress.secret` field.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  session:
    ingress:
      domain: training.learningcenter.tanzu.vmware.com
      secret: training.learningcenter.tanzu.vmware.com-tls
```

If HTTPS connections are terminated using an external load balancer and not by specifying a secret for ingresses managed by the Kubernetes ingress controller, then routing traffic into the Kubernetes cluster as HTTP connections, you can override the ingress protocol without specifying an ingress secret by setting the `session.ingress.protocol` field.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  session:
    ingress:
      domain: training.learningcenter.tanzu.vmware.com
      protocol: https
```

To override or set the ingress class, which dictates which ingress router is used when more than one option is available, you can add `session.ingress.class`.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  session:
    ingress:
      domain: training.learningcenter.tanzu.vmware.com
      secret: training.learningcenter.tanzu.vmware.com-tls
      class: nginx
```

Controlling access to the workshop

By default, requesting a workshop using the `WorkshopRequest` custom resource is deactivated and must be enabled for a workshop environment by setting `request.enabled` to `true`.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  request:
    enabled: true
```

With this enabled, anyone who can create a `WorkshopRequest` custom resource can request the creation of a workshop instance for the workshop environment.

To further control who can request a workshop instance in the workshop environment, you can first set an access token, which a user must know and supply with the workshop request. This is done by setting the `request.token` field.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  request:
    enabled: true
    token: lab-markdown-sample
```

The same name as the workshop environment is used in this example, which is probably not a good practice. Use a random value instead. The token value may be multiline.

As a second control measure, you can specify what namespaces the `WorkshopRequest` must be created. This means a user must have the specific ability to create `WorkshopRequest` resources in one of those namespaces.

You can specify the list of namespaces from which workshop requests for the workshop environment by setting `request.namespaces`.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  request:
    enabled: true
    token: lab-markdown-sample
    namespaces:
    - default
```

To add the workshop namespace in the list, rather than list the literal name, you can reference a predefined parameter specifying the workshop namespace by including `$(workshop_namespace)`.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  request:
    enabled: true
    token: lab-markdown-sample
    namespaces:
    - $(workshop_namespace)
```

Overriding the login credentials

When requesting a workshop using `WorkshopRequest`, a login dialog box is presented to the user when accessing the workshop instance URL. By default, the user name is `learningcenter`. The password is a random value the user must query from the `WorkshopRequest` status after creating the custom resource.

To override the user name, you can set the `session.username` field. To set the same fixed password for all workshop instances, you can set the `session.password` field.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  session:
    username: workshop
    password: lab-markdown-sample
```

Additional workshop resources

The workshop definition defined by the `Workshop` custom resource already declares a set of resources to be created with the workshop environment. You can use this when you have shared service applications the workshop needs, such as an container image registry or a Git repository server.

To deploy additional applications related to a specific workshop environment, you can declare them by adding them into the `environment.objects` field of the `WorkshopEnvironment` custom resource. You might use this deploy a web application used by attendees of a workshop to access their workshop instances.

For namespaced resources, it is not necessary to set the `namespace` field of the resource `metadata`. When the `namespace` field is not present, the resource is created within the workshop namespace for that workshop environment.

When resources are created, owner references are added, making the `WorkshopEnvironment` custom resource correspond to the owner of the workshop environment. This means that any resources are also deleted when the workshop environment is deleted.

Values of fields in the list of resource objects can reference several predefined parameters. The available parameters are:

- `workshop_` - The name of the workshop. This is the name of the `Workshop` definition the workshop environment was created against.

- `environment_` - The name of the workshop environment. Currently, this is the same as the name of the namespace for the workshop environment. Do not rely on the name and the workshop environment being the same, and use the most appropriate to cope with any future change.
- `environment_` - The token value must be used against the workshop environment in workshop requests.
- `workshop_` - The namespace for the workshop environment. This is the namespace where all deployments of the workshop instances and their service accounts are created. It is the same namespace that shared workshop resources are created.
- `service_` - The service account name is used when creating deployments in the workshop namespace.
- `ingress_` - The host domain under which host names are created when creating ingress routes.
- `ingress_` - The protocol (http/https) used for ingress routes created for workshops.
- `ingress_` - The name of the ingress secret stored in the workshop namespace when secure ingress is being used.

To create additional namespaces associated with the workshop environment, embed a reference to `$(workshop_namespace)` in the name of the additional namespaces, with an appropriate suffix. Be mindful that the suffix doesn't overlap with the range of session IDs for workshop instances.

When creating deployments in the workshop namespace, set the `serviceAccountName` of the `Deployment` resource to `$(service_account)`. This ensures the deployment uses a special Pod security policy set up by the Learning Center. If this isn't used and the cluster imposes a more strict default Pod security policy, your deployment might not work, especially if any image expects to run as `root`.

Creation of workshop instances

After a workshop environment is created, you can create the workshop instances. You can request a workshop instance by using the `WorkshopRequest` custom resource. This can be a separate step, or you can add them as resources under `environment.objects`.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  request:
    token: lab-markdown-sample
    namespaces:
    - $(workshop_namespace)
  session:
    username: learningcenter
    password: lab-markdown-sample
  environment:
    objects:
    - apiVersion: learningcenter.tanzu.vmware.com/v1beta1
      kind: WorkshopRequest
      metadata:
        name: user1
      spec:
        environment:
          name: $(environment_name)
          token: $(environment_token)
    - apiVersion: learningcenter.tanzu.vmware.com/v1beta1
      kind: WorkshopRequest
      metadata:
        name: user2
      spec:
        environment:
          name: $(environment_name)
          token: $(environment_token)
```

Using this method, the workshop environment is populated with workshop instances. You can query the workshop requests from the workshop namespace to discover the URLs for accessing each and the password if you didn't set one and a random password was assigned.

If you need more control over how the workshop instances were created using this method, you can use the `WorkshopSession` custom resource instead.

Configure the WorkshopRequest resource

This topic describes how you configure the `WorkshopRequest` custom resource, which defines a Learning Center workshop request.

Specifying workshop environment

The `WorkshopRequest` custom resource is used to request a workshop instance. It does not provide details needed to perform the deployment of the workshop instance. That information is sourced by the Learning Center Operator from the `WorkshopEnvironment` and `Workshop` custom resources.

The minimum required information in the workshop request is the name of the workshop environment. You supply this by setting the `environment.name` field.

For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopRequest
metadata:
  name: lab-markdown-sample
spec:
  environment:
    name: lab-markdown-sample
```

A request is successful only if requesting a workshop instance for a workshop environment is enabled for that workshop. You can enable requests in the `WorkshopEnvironment` custom resource for the workshop environment.

If multiple workshop requests, for the same workshop environment or different ones, are created in the same namespace, the `name` defined in the `metadata` for the workshop request must be different for each. The value of this name is not used to name workshop instances. You need the `name` value to delete the workshop instance, which is done by deleting the workshop request.

Specifying required access token

If a workshop environment is configured to require an access token when making a workshop request against that environment, you can specify decide the token by setting the `environment.token` field.

For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopRequest
metadata:
  name: lab-markdown-sample
spec:
  environment:
    name: lab-markdown-sample
    token: lab-markdown-sample
```

Even with the token, the request fails if the following is true:

- The workshop environment has restricted the namespaces from which a workshop request was made
- The workshop request was not created in one of the permitted namespaces

Configure the TrainingPortal resource

This topic describes how you configure the `TrainingPortal` custom resource, which triggers the deployment of a set of Learning Center workshop environments and a set number of workshop instances.

Specifying the workshop definitions

You run multiple workshop instances to perform training to a group of people by creating the workshop environment and then creating each workshop instance. The `TrainingPortal` workshop resource bundles that up as one step.

Before creating the training environment, you must load the workshop definitions as a separate step.

To specify the names of the workshops to be used for the training, list them under the `workshops` field of the training portal specification. Each entry needs to define a `name` property, matching the name of the `Workshop` resource you created.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: sample-workshops
spec:
  portal:
    sessions:
      maximum: 8
  workshops:
  - name: lab-asciidoc-sample
  - name: lab-markdown-sample
```

When the training portal is created, it:

- Sets up the underlying workshop environments.
- Creates any workshop instances required to be created initially for each workshop.
- Deploys a web portal for attendees of the training to access their workshop instances.

Limit the number of sessions

When defining the training portal, you can set a limit on the workshop sessions that can be run concurrently. Set this limit by using the `portal.sessions.maximum` property:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: sample-workshops
spec:
  portal:
    sessions:
      maximum: 8
  workshops:
  - name: lab-asciidoc-sample
  - name: lab-markdown-sample
```

When you specify this, the maximum capacity of each workshop is set to the maximum value for the training portal as a whole. This means that any one workshop can have as many sessions running as specified by the maximum for the portal. However, to achieve this maximum for a given workshop, only instances of that workshop can be created. In other words, the maximum capacity can be spread across a number of workshops or it can be used in its entirety by a single workshop.

If you do not set `portal.sessions.maximum`, you must set the capacity for each individual workshop as detailed in the following section. In only setting the capacities of each workshop and not an overall maximum for sessions, you cannot share the overall capacity of the training portal across multiple workshops.

Capacity of individual workshops

When you have more than one workshop, you can want to limit how many instances of each workshop you can have so that they cannot grow to the maximum number of sessions for the whole training portal. This means you can stop a specific workshop from using all of the capacity of the training portal. To do this, set the `capacity` field under the entry for the workshop:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: sample-workshops
spec:
  portal:
    sessions:
      maximum: 8
  workshops:
  - name: lab-asciidoc-sample
    capacity: 4
  - name: lab-markdown-sample
    capacity: 6
```

The value of `capacity` limits the number of workshop sessions for a specific workshop to that value. It must be less than or equal to the maximum number of workshops sessions for the portal, because the latter always sets the absolute limit.

Set reserved workshop instances

By default one instance of each of the listed workshops is created so when the initial user requests that workshop, it’s available for use immediately.

When such a reserved instance is allocated to a user, provided that the workshop capacity hasn’t been reached, a new instance of the workshop is created as a reserve ready for the next user. When a user ends a workshop and the workshop is at capacity, when the instance is deleted, a new reserve is created. The total of allocated and reserved sessions for a workshop cannot exceed the capacity for that workshop.

To override for a specific workshop how many reserved instances are kept on standby ready for users, you can set the `reserved` setting against the workshop:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: sample-workshops
spec:
  portal:
    sessions:
      maximum: 8
  workshops:
  - name: lab-asciidoc-sample
    capacity: 4
    reserved: 2
  - name: lab-markdown-sample
    capacity: 6
    reserved: 4
```

You can set the value of `reserved` to 0 if you never want any reserved instances for a workshop and only want instances of that workshop created on demand when required for a user. Creating instances of a workshop on demand can result in a user waiting longer to access a workshop session.

When workshop instances are always created on demand, the oldest reserved instance is terminated to allow a new session of a desired workshop to be created. This also happens when reserved instances tie up capacity that could be used for a new session of another workshop. This occurs if any caps for specific workshops are met.

Override initial number of sessions

The initial number of workshop instances created for each workshop is specified by `reserved` or 1 if the setting hasn't been provided.

In the case where `reserved` is set in order to keep workshop instances on standby, you can indicate that initially you want more than the reserved number of instances created. This is useful when running a workshop for a set period of time. You might create up-front instances of the workshop corresponding to 75% of the expected number of attendees but with a smaller reserve number. With this configuration, new reserve instances only start to be created when the total number approaches 75% and all extra instances created up front have been allocated to users. This ensures you have enough instances ready for when most people come, but you can also create other instances later if necessary:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: kubernetes-fundamentals
spec:
  portal:
    sessions:
      maximum: 100
  workshops:
  - name: lab-kubernetes-fundamentals
    initial: 75
    reserved: 5
```

Setting defaults for all workshops

If you have a list of workshops, and they all must be set with the same values for `capacity`, `reserved`, and `initial`, rather than add settings to each, you can set defaults to apply to all workshops under the `portal` section:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: sample-workshops
spec:
  portal:
    sessions:
      maximum: 10
    capacity: 6
    reserved: 2
    initial: 4
  workshops:
  - name: lab-asciidoc-sample
  - name: lab-markdown-sample
```

Set caps on individual users

By default a single user can run more than one workshop at a time. You can cap this to ensure that workshops run only one at a time. This prevents a user from wasting resources by starting more than one workshop and only working on one without shutting the other down.

To apply a limit on how many concurrent workshop sessions a user can start, use the `portal.sessions.registered` setting:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: sample-workshops
spec:
  portal:
    sessions:
      maximum: 8
      registered: 1
  workshops:
  - name: lab-asciidoc-sample
    capacity: 4
    reserved: 2
  - name: lab-markdown-sample
    capacity: 6
    reserved: 4
```

This limit also applies to anonymous users when anonymous access is enabled through the training portal web interface or if sessions are being created through the REST API. To set a limit on anonymous users, you can set `portal.sessions.anonymous` instead:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: sample-workshops
spec:
  portal:
    sessions:
      maximum: 8
      anonymous: 1
  workshops:
  - name: lab-asciidoc-sample
    capacity: 4
    reserved: 2
  - name: lab-markdown-sample
    capacity: 6
    reserved: 4
```

Expiration of workshop sessions

After you reach the maximum capacity, no more workshops sessions can be created. After a workshop session is allocated to a user, it cannot be reassigned to another user.

If you are running a supervised workshop, set the capacity higher than the anticipated number of users in case you have more users than you expect. Use the setting for the reserved number of instances. This way, even if you set a higher capacity than needed, workshop sessions are only created as required and not all up front.

In supervised workshops, when the training is over, delete the whole training environment. All workshop sessions are then deleted.

To host a training portal over an extended period but don’t know when users want to do a workshop, you can set up workshop sessions to expire after a set time. When expired, the workshop session is deleted and a new workshop session can be created in its place.

The maximum capacity is therefore the maximum at any one point in time, while the number can grow and shrink over time. So over an extended time, you can handle many more sessions than the set maximum capacity. The maximum capacity ensures you don’t try to allocate more workshop sessions than you have resources for at a given time.

To set a maximum time allowed for a workshop session, use the `expires` setting:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  workshops:
  - name: lab-markdown-sample
    capacity: 8
    reserved: 1
    expires: 60m
```

The value needs to be an integer, followed by a suffix of ‘s’, ‘m’ or ‘h’, corresponding to seconds, minutes, or hours.

The time period is calculated from when the workshop session is allocated to a user. When the time period is up, the workshop session is automatically deleted.

When an expiration period is specified, or when a user finishes a workshop or restarts the workshop, the workshop is also deleted.

To cope with users who claim a workshop session, but leave and don't use it, you can set a time period for when a workshop session with no activity is deemed orphaned and so is deleted. Do this using the `orphaned` setting:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  workshops:
  - name: lab-markdown-sample
    capacity: 8
    reserved: 1
    expires: 60m
    orphaned: 5m
```

Avoid this setting for supervised workshops where the whole event only lasts a certain length of time. This prevents a user's session from being deleted when the user takes breaks and the computer goes to sleep.

The `expires` and `orphaned` settings can also be set against `portal` to apply them to all workshops.

Updates to workshop environments

The list of workshops for an existing training portal can be changed by modifying the training portal definition applied to the Kubernetes cluster.

If you remove a workshop from the list of workshops, the workshop environment is marked as stopping and is deleted when all active workshop sessions have completed.

If you add a workshop to the list of workshops, a new workshop environment for it is created.

Changes to settings, such as the maximum number of sessions for the training portal or capacity settings for individual workshops, are applied to existing workshop environments.

By default a workshop environment is left unchanged if the corresponding workshop definition is changed. So in the default configuration, you must explicitly delete the workshop from the list of workshops managed by the training portal and then add it back again if the workshop definition changed.

If you prefer that workshop environments be replaced when the workshop definition changes, enable this by using the `portal.updates.workshop` setting:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    sessions:
      maximum: 8
    updates:
      workshop: true
  workshops:
  - name: lab-markdown-sample
    reserved: 1
    expires: 60m
    orphaned: 5m
```

When using this option, use the `portal.sessions.maximum` setting to limit the number of workshop sessions that can be run for the training portal as a whole. When replacing the workshop environment, the old workshop environment is retained if there is still an active workshop session being used. If the limit isn't set, the new workshop environment is still able to grow to its specific capacity and is not limited by how many workshop sessions are running against old instances of the workshop environment.

Overall, VMware recommends updating workshop environments when workshop definitions change only in development environments when working on workshop content. This is an especially good practice until you are familiar with how the training portal replaces existing workshop environments, and the resource implications of having old and new instances of a workshop environment running at the same time.

Override the ingress domain

To access a workshop instance using a public URL, specify an ingress domain. If an ingress domain isn't specified, the default ingress domain that the Learning Center Operator is configured with is used.

When setting a custom domain, DNS must have been configured with a wildcard domain to forward all requests for sub-domains of the custom domain to the ingress router of the Kubernetes cluster.

To provide the ingress domain, set the `portal.ingress.domain` field:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    ingress:
      domain: learningcenter.tanzu.vmware.com
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

If overriding the domain, by default the workshop session is exposed using a HTTP connection. For a secure HTTPS connection, you must have access to a wildcard SSL certificate for the domain. A secret of type `tls` should be created for the certificate in the `learningcenter` namespace or the namespace where the Learning Center Operator is deployed. The name of that secret must be set in the `portal.ingress.secret` field:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    ingress:
      domain: learningcenter.tanzu.vmware.com
      secret: learningcenter.tanzu.vmware.com-tls
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

You can terminate HTTPS connections by using an external load balancer instead of specifying a secret for ingresses managed by the Kubernetes ingress controller. In that case, when routing traffic into the Kubernetes cluster as HTTP connections, you can override the ingress protocol without specifying an ingress secret. Instead, set the `portal.ingress.protocol` field:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    ingress:
      domain: learningcenter.tanzu.vmware.com
      protocol: https
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

To override or set the ingress class, which dictates which ingress router is used when more than one option is available, you can add `portal.ingress.class`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    ingress:
      domain: learningcenter.tanzu.vmware.com
      secret: learningcenter.tanzu.vmware.com-tls
      class: nginx
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

Override the portal host name

The default host name given to the training portal is the name of the resource with `-ui` suffix, followed by the domain specified by the resource or the default inherited from the configuration of the Learning Center Operator.

To override the generated host name, you can set `portal.ingress.hostname`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    ingress:
      hostname: labs
      domain: learningcenter.tanzu.vmware.com
      secret: learningcenter.tanzu.vmware.com-tls
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

This causes the host name to be `labs.learningcenter.tanzu.vmware.com` rather than the default generated name for this example of `lab-markdown-sample-ui.learningcenter.tanzu.vmware.com`.

Set extra environment variables

To override any environment variables for workshop instances created for a specific work, provide the environment variables in the `env` field of that workshop:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
    env:
```

```
    - name: REPOSITORY-URL
      value: YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE
```

Where `YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE` is the Git repository URL for `lab-markdown-sample`. For example, `{YOUR-GIT-REPO-URL}/lab-markdown-sample`.

Values of fields in the list of resource objects can reference a number of predefined parameters. The available parameters are:

- `session_id` - A unique ID for the workshop instance within the workshop environment.
- `session_namespace` - The namespace created for and bound to the workshop instance. This is the namespace unique to the session and where a workshop can create its own resources.
- `environment_name` - The name of the workshop environment. For now this is the same as the name of the namespace for the workshop environment. Don't rely on them being the same, and use the most appropriate to cope with any future change.
- `workshop_namespace` - The namespace for the workshop environment. This is the namespace where all deployments of the workshop instances are created and where the service account that the workshop instance runs as exists.
- `service_account` - The name of the service account the workshop instance runs as and which has access to the namespace created for that workshop instance.
- `ingress_domain` - The host domain under which host names can be created when creating ingress routes.
- `ingress_protocol` - The protocol (http/https) used for ingress routes created for workshops.

The syntax for referencing one of the parameters is `$(parameter_name)`.

Override portal credentials

When a training portal is deployed, the user name for the admin and robot accounts uses the defaults of `learningcenter` and `robot@learningcenter`. The passwords for each account are randomly set.

For the robot account, the OAuth application client details used with the REST API are also randomly generated.

You can see what the credentials and client details are by running `kubectl describe` against the training portal resource. This will yield output that includes:

```
Status:
  learningcenter:
    Clients:
      Robot:
        Id:      ACZpcaLIT3qr725YWmXu8et9REl4HBg1
        Secret:  t5IfXbGZQThAKR43apoc9usOFVDv2BLE
    Credentials:
      Admin:
        Password:  0kGmMlYw46BZT2vCntyrRuFf1gQq5ohi
        Username:  learningcenter
      Robot:
        Password:  QrnY67ME9yGasNhq2OTbgWA4RzipUvo5
        Username:  robot@learningcenter
```

To override any of these values to set them to a predetermined value, you can add `credentials` and `clients` sections to the training portal specification.

To overload the credentials for the admin and robot accounts user:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    credentials:
      admin:
        username: admin-user
        password: top-secret
      robot:
        username: robot-user
        password: top-secret
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

To override the application client details for OAuth access by the robot account user:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    clients:
      robot:
        id: application-id
        secret: top-secret
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

Control registration type

By default the training portal web interface presents a registration page for users to create an account before selecting a workshop. If you only want to allow the administrator to log in, you can deactivate the registration page. Do this if you are using the REST API to create and allocate workshop sessions from a separate application:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    registration:
      type: one-step
      enabled: false
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

If rather than requiring users to register, you want to allow anonymous access, you can switch the registration type to anonymous:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    registration:
      type: anonymous
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

When a user visits the training portal home page in anonymous mode, an account for that user is automatically created and the user is logged in.

Specify an event access code

When deploying the training portal with anonymous access or open registration, anyone who knows the URL can access workshops. To at least restrict access to those who know a common event access code or password, you can set `portal.password`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    password: workshops-2020-07-01
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

When anonymous access is enabled and the training portal URL is accessed, users are asked to enter an event access code before they are redirected to the list of workshops or to the login page.

Make a list of workshops public

By default the index page providing the catalog of available workshop images is only available after a user has logged in, either through a registered account or as an anonymous user.

To make the catalog of available workshops public so they can be viewed before logging in, set the `portal.catalog.visibility` property:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    catalog:
      visibility: public
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

By default the catalog has visibility set to `private`. Use `public` to expose it.

This also makes it possible to access the list of available workshops from the catalog through the REST API, without authenticating against the REST API.

Use an external list of workshops

If you are using the training portal with registration deactivated, and you are using the REST API from a separate website to control creation of sessions, you can specify an alternate URL for providing the list of workshops.

This helps when the REST API creates a session and cookies are deleted or a session URL is shared with a different user. This means the value for the `index_url` supplied with the REST API request is lost.

To set the URL for the external site, use the `portal.index` property:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    index: https://www.example.com/
    registration:
      type: one-step
      enabled: false
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

If you supply this property, passing the `index_url` when creating a workshop session using the REST API is optional, and the value of this property is used. You can still supply `index_url` when using the REST API for a user to be redirected back to a sub-category of workshops on the site. The URL provided in the training portal definition then acts only as a fallback. That is, when the redirect URL becomes unavailable, it directs the user back to the top-level page for the external list of workshops.

If a user has logged into the training portal as the admin user, the user is not redirected to the external site and still sees the training portal's list of workshops.

Override portal title and logo

By default the web interface for the training portal displays a generic Learning Center logo and a page title of "Workshops." To override these, you can set `portal.title` and `portal.logo`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    title: Workshops
    logo: data:image/png;base64,....
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

The `logo` field should be a graphical image provided in embedded data URI format. The image is displayed with a fixed height of "40px". The field can also be a URL for an image stored on a remote web server.

Allow the portal in an iframe

By default it is prohibited to display the web interface for the training portal in an iframe of another web site, because of content security policies applying to the training portal website.

To enable the ability to iframe the full training portal web interface or even a specific workshop session created using the REST API, provide the host name of the site that embeds it. Do this by using the `portal.theme.frame.ancestors` property:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    theme:
      frame:
        ancestors:
        - https://www.example.com
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

The property is a list of hosts, not a single value. To use a URL for the training portal in an iframe of a page, which, in turn, is embedded in another iframe of a page on a different site, list the host names of all sites.

Because the sites that embed iframes must be secure and use HTTPS, they cannot use plain HTTP. Browser policies prohibit promoting cookies to an insecure site when embedding using an iframe. If cookies cannot be stored, a user cannot authenticate against the workshop session.

Collect analytics on workshops

To collect analytics data on usage of workshops, supply a webhook URL. When you supply a webhook URL, events are posted to the webhook URL, including:

- Workshops started
- Pages of a workshop viewed
- Expiration of a workshop
- Completion of a workshop
- Termination of a workshop

For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  analytics:
    webhook:
      url: https://metrics.learningcenter.tanzu.vmware.com/?client=name&token=password
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

At present there is no metrics collection service compatible with the portal webhook reporting mechanism, so create a custom service or integrate it with any existing web front end for the portal REST API service.

If the collection service needs to be provided with a client ID or access token, it must accept using query string parameters set in the webhook URL.

Include the details of the event as HTTP POST data by using the `application/json` content type:

```
{
  "portal": {
    "name": "lab-markdown-sample",
    "uid": "91dfa283-fb60-403b-8e50-fb30943ae87d",
    "generation": 3,
    "url": "https://lab-markdown-sample-ui.learningcenter.tanzu.vmware.com"
  },
  "event": {
    "name": "Session/Started",
    "timestamp": "2021-03-18T02:50:40.861392+00:00",
    "user": "c66db34e-3158-442b-91b7-25391042f037",
    "session": "lab-markdown-sample-w01-s001",
    "environment": "lab-markdown-sample-w01",
    "workshop": "lab-markdown-sample",
    "data": {}
  }
}
```

When an event has associated data, it is included in the `data` dictionary:

```
{
  "portal": {
    "name": "lab-markdown-sample",
    "uid": "91dfa283-fb60-403b-8e50-fb30943ae87d",
    "generation": 3,
    "url": "https://lab-markdown-sample-ui.learningcenter.tanzu.vmware.com"
  },
  "event": {
    "name": "Workshop/View",
    "timestamp": "2021-03-18T02:50:44.590918+00:00",
    "user": "c66db34e-3158-442b-91b7-25391042f037",
    "session": "lab-markdown-sample-w01-s001",
    "environment": "lab-markdown-sample-w01",
    "workshop": "lab-markdown-sample",
    "data": {
      "current": "workshop-overview",
      "next": "setup-environment",
      "step": 1,
      "total": 4
    }
  }
}
```

The `user` field is the same portal user identity returned by the REST API when creating workshop sessions.

The event stream only produces events for things as they happen. For a snapshot of all current workshop sessions, use the REST API to request the catalog of available workshop environments, enabling the inclusion of current workshop sessions.

Track using Google Analytics

To record analytics data on usage of workshops by using Google Analytics, enable tracking by supplying a tracking ID for Google Analytics:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  analytics:
    google:
      trackingId: UA-XXXXXXX-1
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

Custom dimensions are used in Google Analytics to record details about the workshop a user is taking, including through which training portal and cluster it was accessed. So you can use the same Google Analytics tracking ID for multiple training portal instances running on different Kubernetes clusters.

To support use of custom dimensions in Google Analytics, configure the Google Analytics property with the following custom dimensions. They must be added in the order shown, because Google Analytics doesn't allow you to specify the index position for a custom dimension. It allocates them for you. You can't already have custom dimensions defined for the property, as the new custom dimensions must start at index of 1.

Custom Dimension Name	Index
workshop_name	1
session_namespace	2
workshop_namespace	3
training_portal	4
ingress_domain	5
ingress_protocol	6

In addition to custom dimensions against page accesses, events are also generated. These include:

- Workshop/Start
- Workshop/Finish

- Workshop/Expired

If you provide a Google Analytics tracking ID with the `TrainingPortal` resource definition, it takes precedence over the `SystemProfile` resource definition.

Note: Google Analytics is not a reliable way to collect data. Individuals or corporate firewalls can block the reporting of Google Analytics data. For more precise statistics, use the webhook URL for collecting analytics with a custom data collection platform.

Configure the SystemProfile resource

This topic describes how you use the `SystemProfile` custom resource to configure the Learning Center operator.

You can use the default system profile to set defaults for ingress and image pull secrets. You can also select an alternate profile for specific deployments if required.

Note: Changes made to the `SystemProfile` custom resource, or changes made by means of environment variables, don't take effect on already deployed `TrainingPortals`. You must recreate those for the changes to be applied. You only need to recreate the `TrainingPortal` resources, because this resource takes care of recreating the `WorkshopEnvironments` with the new values.

Operator default system profile

The Learning Center Operator, by default, uses an instance of the `SystemProfile` custom resource if it exists, named `default-system-profile`. You can override the name of the resource used by the Learning Center Operator as the default by setting the `SYSTEM_PROFILE` environment variable on the deployment for the Learning Center Operator. For example:

```
kubectl set env deployment/learningcenter-operator -e SYSTEM_PROFILE=default-system-pr
ofile -n learningcenter
```

The Learning Center Operator automatically detects and uses any changes to an instance of the `SystemProfile` custom resource. You do not need to redeploy the operator when changes are made.

Defining configuration for ingress

The `SystemProfile` custom resource replaces the use of environment variables to configure details such as the ingress domain, secret, and class.

Instead of setting `INGRESS_DOMAIN`, `INGRESS_SECRET`, and `INGRESS_CLASS` environment variables, create an instance of the `SystemProfile` custom resource named `default-system-profile`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  ingress:
    domain: learningcenter.tanzu.vmware.com
    secret: learningcenter.tanzu.vmware.com-tls
    class: nginx
```

If you terminate HTTPS connections by using an external load balancer and not by specifying a secret for ingresses managed by the Kubernetes ingress controller, then routing traffic into the Kubernetes cluster as HTTP connections, you can override the ingress protocol without specifying an ingress secret:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  ingress:
    domain: learningcenter.tanzu.vmware.com
    protocol: https
    class: nginx
```

Defining container image registry pull secrets

To work with custom workshop images stored in a private image registry, the system profile can define a list of image pull secrets. Add this to the service accounts used to deploy and run the workshop images. Set the `environment.secrets.pull` property to the list of secret names:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  environment:
    secrets:
      pull:
      - private-image-registry-pull
```

The secrets containing the image registry credentials must exist within the `learningcenter` namespace or the namespace where the Learning Center Operator is deployed. The secret resources must be of type `kubernetes.io/dockerconfigjson`.

The secrets are added to the workshop namespace and are not visible to a user. No secrets are added to the namespace created for each workshop session.

Some container images are used as part of Learning Center itself, such as the container image for the training portal web interface and the builtin base workshop images. If you have copied these from the public image registries and stored them in a local private registry, use the `registry` section instead of the above setting. For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  registry:
    secret: learningcenter-image-registry-pull
```

The `registry.secret` is the name of the secret containing the image registry credentials. This must be present in the `learningcenter` namespace or the namespace where the Learning Center Operator is deployed.

Defining storage class for volumes

Deployments of the training portal web interface and the workshop sessions make use of persistent volumes. By default the persistent volume claims do not specify a storage class for the volume. Instead, they rely on the Kubernetes cluster to specify a default storage class that works. If the Kubernetes cluster doesn’t define a suitable default storage class or you need to override it, you can set the `storage.class` property. For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  storage:
    class: default
```

This only applies to persistent volume claims setup by the Learning Center Operator. If a user executes steps in a workshop that include making persistent volume claims, these are not automatically adjusted.

Defining storage group for volumes

The cluster must apply pod security policies where persistent volumes are used by Learning Center for the training portal web interface and workshop environments. These security policies ensure that permissions of persistent volumes are set correctly so they can be accessed by containers mounting the persistent volume. When the pod security policy admission controller is not enabled, the cluster institutes a fallback to enable access to volumes by enabling group access using the group ID of `0`.

In situations where the only class of persistent storage available is NFS or similar, you might have to override the group ID applied and set it to an alternate ID dictated by the file system storage provider. If this is required, you can set the `storage.group` property. For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  storage:
    group: 1
```

Overriding the group ID to match the persistent storage relies on the group having write permission to the volume. If only the owner of the volume has permission, this does not work.

In this case, change the owner/group and permissions of the persistent volume such that the owner matches the user ID a container runs at. Alternatively, set the group to a known ID that is added as a supplemental group for the container and update the persistent volume to be writable to this group. This must be done by an `init` container running in the pod mounting the persistent volume.

To trigger this change of ownership and permissions, set the `storage.user` property. For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  storage:
    user: 1
    group: 1
```

This results in:

- The `init` container running as the root user.
- The owner of the mount directory of the persistent volume being set to `storage.user`.
- The group being set to `storage.group`.
- The directory made group-writable.

The group is then added as the supplemental group to containers using the persistent volume. So they can write to the persistent volume, regardless of what user ID the container runs as. To that end, the specific value of `storage.user` doesn't matter, but you might need to set it to a specific user ID based on requirements of the storage provider.

Both these variations on the settings only apply to the persistent volumes used by Learning Center itself. If a workshop asks users to create persistent volumes, those instructions, or the resource definitions used, might need to be modified to work where the available storage class requires access as a specific user or group ID.

Further, the second method using the `init` container to fix permissions does not work if pod security policies are enforced. The ability to run a container as the root user is blocked in that case due to the restricted PSP, which is applied to workshop instances.

Restricting network access

Any processes running from the workshop container, and any applications deployed to the session namespaces associated with a workshop instance, can contact any network IP addresses accessible from the cluster. To restrict access to IP addresses or IP subnets, set `network.blockCIDRs`. This must be a CIDR block range corresponding to the subnet or a portion of a subnet you want to block. A Kubernetes `NetworkPolicy` is used to enforce the restriction. So the Kubernetes cluster must use a network layer supporting network policies, and the necessary Kubernetes controllers supporting network policies must be enabled when the cluster is installed.

If deploying to AWS, it is important to block access to the AWS endpoint for querying EC2 metadata, because it can expose sensitive information that workshop users should not haves access to. By default Learning Center will block the AWS endpoint on the TAP SystemProfile. If you need to replicate this block to other SystemProfiles, the configuration is as follows:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  network:
    blockCIDRs:
    - 169.254.169.254/32
    - fd00:ec2::254/128
```

Running Docker daemon rootless

If `docker` is enabled for workshops, Docker-in-Docker is run using a sidecar container. Because of the current state of running Docker-in-Docker and portability across Kubernetes environments, the `docker` daemon by default runs as `root`. Because a privileged container is also being used, this represents a security risk. Only run workshops requiring `docker` in disposable Kubernetes clusters or for users whom you trust.

You can partly mediate the risks of running `docker` in the Kubernetes cluster by running the `docker` daemon in rootless mode. However, not all Kubernetes clusters can support this due to the Linux kernel configuration or other incompatibilities.

To enable rootless mode, you can set the `dockerd.rootless` property to `true`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  dockerd:
    rootless: true
```

Use of `docker` can be made even more secure by avoiding the use of a privileged container for the `docker` daemon. This requires that you set up a specific configuration for nodes in the Kubernetes cluster. With this configuration, you can disallow the use of a privileged container by setting `dockerd.privileged` to `false`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  dockerd:
    rootless: true
    privileged: false
```

For further details about the requirements for running rootless Docker-in-Docker and using a non-privileged container, see the Docker documentation.

Overriding network packet size

When you enable support for building container images using `docker` for workshops, because of network layering that occurs when doing `docker build` or `docker run`, you must adjust the network packet size (MTU) used for containers run from `dockerd` hosted inside the workshop container.

The default MTU size for networks is 1500, but, when containers are run in Kubernetes, the size available to containers is often reduced. To deal with this possibility, the MTU size used when `dockerd` is run for a workshop is set as 1400 instead of 1500.

You might need to override this value to an even lower value if you experience problems building or running images with `docker` support. These problems could include errors or timeouts in pulling images or when pulling software packages such as PyPi, npm, and so on.

To lower the value, set the `dockerd.mtu` property:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  dockerd:
    mtu: 1400
```

To discover the maximum viable size, access the `docker` container run with a workshop and run `ifconfig eth0`. This yields something similar to:

```
eth0      Link encap:Ethernet  HWaddr 02:42:AC:11:00:07
          inet addr:172.17.0.7  Bcast:172.17.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1350  Metric:1
          RX packets:270018 errors:0 dropped:0 overruns:0 frame:0
          TX packets:283882 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:86363656 (82.3 MiB)  TX bytes:65183730 (62.1 MiB)
```

If the `MTU` size is less than 1400, use the value given, or a smaller value, for the `dockerd.mtu` setting.

Image registry pull through cache

When running or building container images with `docker`, if the container image is hosted on Docker Hub, it is pulled down directly from the Docker Hub for each separate workshop session of that workshop.

Because the image is pulled from Docker Hub, this can be slow for all users, especially for large images. With Docker Hub introducing limits on how many images can be pulled anonymously from an IP address within a set period, this also can result in the cap on image pulls being reached. This prevents the workshop from being used until the period expires.

Docker Hub has a higher limit when pulling images as an authenticated user, but with the limit applied to the user rather than by IP address. For authenticated users with a paid plan on Docker Hub, there is no limit.

To attempt to avoid the impact of the limit, the first thing you can do is enable an image registry mirror with image pull-through. This is enabled globally and results in an instance of an image registry mirror being created in the workshop environment of workshops that enable `docker` support. This mirror is used for all workshops sessions created against that workshop environment. When the first user attempts to pull an image, it is pulled down from Docker Hub and cached in the mirror. Subsequent users are served up from the image registry mirror, avoiding the need to pull the image from Docker Hub again. Subsequent users also see a speed up in pulling the image, because the mirror is deployed to the same cluster.

To enable the use of an image registry mirror against Docker Hub, use:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  dockerd:
    mirror:
      remote: https://registry-1.docker.io
```

For authenticated access to Docker Hub, create an access token under your Docker Hub account. Then set the `username` and `password` using the access token as the `password`. Do not use the password for the account itself. Using an access token makes it easier to revoke the token if necessary.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  dockerd:
    mirror:
      remote: https://registry-1.docker.io
      username: username
      password: access-token
```

An access token provides write access to Docker Hub. It is therefore also recommended you use a separate robot account in Docker Hub that is not used to host images and doesn't have write access to any other organizations. In other words, use it purely for reading images from Docker Hub.

If this is a free account, the higher limit on image pulls then applies. If the account is paid, there might be higher limits or no limit all all.

The image registry mirror is only used when running or building images using support for running `docker`. The mirror does not come into play when creating deployments in Kubernetes, which make use of images hosted on Docker Hub. Use of images from Docker Hub in deployments is still subject to the limit for anonymous access, unless you supply image registry credentials for the deployment so an authenticated user is used.

Setting default access credentials

When deploying a training portal using the `TrainingPortal` custom resource, the credentials for accessing the portal are unique for each instance. Find the details of the credentials by viewing status information added to the custom resources by using `kubectl describe`.

To override the credentials for the portals so the same set of credentials are used for each, add the desired values to the system profile.

To override the user name and password for the admin and robot accounts, use `portal.credentials`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  portal:
    credentials:
      admin:
        username: learningcenter
        password: admin-password
      robot:
        username: robot@learningcenter
        password: robot-password
```

To override the client ID and secret used for OAuth access by the robot account, use `portal.clients`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  portal:
    clients:
      robot:
        id: robot-id
        secret: robot-secret
```

If the `TrainingPortal` has specified credentials or client information, they still take precedence over the values specified in the system profile.

Overriding the workshop images

When a workshop does not define a workshop image to use and instead downloads workshop content from GitHub or a web server, it uses the `base-environment` workshop image. The workshop content is then added to the container, overlaid on this image.

The version of the `base-environment` workshop image used is the most up-to-date, compatible version of the image available for that version of the Learning Center Operator when it was released.

If necessary you can override the version of the `base-environment` workshop image used by defining a mapping under `workshop.images`. For workshop images supplied as part of the Learning Center project, you can override the short names used to refer to them.

The short versions of the recognized names are:

- `base-environment:*` is a tagged version of the `base-environment` workshop image matched with the current version of the Learning Center Operator.

To override the version of the `base-environment` workshop image mapped to by the `*` tag, use:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  workshop:
    images:
      "base-environment:*": "registry.tanzu.vmware.com/learning-center/base-environmen
t:latest"
```

It is also possible to override where images are pulled from for any arbitrary image. This could be used where you want to cache the images for a workshop in a local image registry and avoid going outside of your network, or the cluster, to get them. This means you wouldn’t need to override the workshop definitions for a specific workshop to change it. For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  workshop:
    images:
      "{YOUR-REGISTRY-URL}/lab-k8s-fundamentals:main": "registry.test/lab-k8s-fundamen
tals:main"
```

Tracking using Google Analytics

If you want to record analytics data on usage of workshops using Google Analytics, you can enable tracking by supplying a tracking ID for Google Analytics. For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  analytics:
    google:
      trackingId: UA-XXXXXXX-1
```

Custom dimensions are used in Google Analytics to record details about the workshop a user is taking and through which training portal and cluster it was accessed. You can therefore use the same Google Analytics tracking ID with Learning Center running on multiple clusters.

To support use of custom dimensions in Google Analytics, you must configure the Google Analytics property with the following custom dimensions. They must be added in the order shown, because Google Analytics doesn’t allow you to specify the index position for a custom dimension and allocates them for you. You can’t already have defined custom dimensions for the property, because the new custom dimensions must start at index of 1.

Custom Dimension Name	Index
workshop_name	1
session_namespace	2
workshop_namespace	3
training_portal	4
ingress_domain	5
ingress_protocol	6

In addition to custom dimensions against page accesses, events are also generated. These include:

- Workshop/Start
- Workshop/Finish
- Workshop/Expired

However, Google Analytics is not a reliable way to collect data. This is because individuals or corporate firewalls can block the reporting of Google Analytics data. For more precise statistics, use the webhook URL for collecting analytics with a custom data collection platform. Configuration of a webhook URL for analytics can only be specified on the `TrainingPortal` definition and cannot be specified globally on the `SystemProfile` configuration.

Overriding styling of the workshop

If using the REST API to create/manage workshop sessions, and the workshop dashboard is then embedded into an iframe of a separate site, you can perform minor styling changes of the dashboard, workshop content, and portal to match the separate site. To do this, provide CSS styles under `theme.dashboard.style`, `theme.workshop.style` and `theme.portal.style`. For dynamic styling or for adding hooks to report on progress through a workshop to a separate service, supply JavaScript as part of the theme under `theme.dashboard.script`, `theme.workshop.script`, and `theme.portal.script`. For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  theme:
    dashboard:
      script: |
        console.log("Dashboard theme overrides.");
      style: |
        body {
          font-family: "Comic Sans MS", cursive, sans-serif;
        }
    workshop:
      script: |
        console.log("Workshop theme overrides.");
      style: |
        body {
          font-family: "Comic Sans MS", cursive, sans-serif;
        }
    portal:
      script: |
        console.log("Portal theme overrides.");
      style: |
        body {
          font-family: "Comic Sans MS", cursive, sans-serif;
        }
```

Additional custom system profiles

If the default system profile is specified, it is used by all deployments managed by the Learning Center Operator, unless it was overridden by the system profile to use for a specific deployment. You can set the name of the system profile for deployments by setting the `system.profile` property of `TrainingPortal`, `WorkshopEnvironment`, and `WorkshopSession` custom resources. For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  system:
    profile: learningcenter-tanzu-vmware-com-profile
  workshops:
  - name: lab-markdown-sample
    capacity: 1
```

Configure the WorkshopSession resource

This topic describes how you configure the `WorkshopSession` custom resource, which defines a Learning Center workshop session.

Specifying the session identity

When running training for multiple people, typically you'll use the `TrainingPortal` custom resource to set up a training environment. Alternatively, you can set up a workshop environment by using the `WorkshopEnvironment` custom resource, and then create requests for workshop instances by using the `WorkshopRequest` custom resource. If you're creating requests for workshop instances, and you need more control over how the workshop instances are set up, you can use `WorkshopSession` custom resource instead of `WorkshopRequest`.

To specify the workshop environment the workshop instance is created against, set the `environment.name` field of the specification for the workshop session. You must also specify the session ID for the workshop instance. For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopSession
metadata:
  name: lab-markdown-sample-user1
spec:
  environment:
    name: lab-markdown-sample
  session:
    id: user1
```

The `name` of the workshop specified in the `metadata` of the training environment must be globally unique for the workshop instance you're creating. You must create a separate `WorkshopSession` custom resource for each workshop instance.

The session ID must be unique within the workshop environment that you're creating the workshop instance against.

Specifying the login credentials

You can control access to each workshop instance using login credentials. This ensures one workshop attendee cannot interfere with another.

To set login credentials for a workshop instance, set the `session.username` and `session.password` fields. For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopSession
metadata:
  name: lab-markdown-sample
spec:
  environment:
    name: lab-markdown-sample-user1
  session:
    username: learningcenter
    password: lab-markdown-sample
```

If you do not specify login credentials, the workshop instance has no access controls and anyone can access it.

Specifying the ingress domain

To access the workshop instance by using a public URL, you must specify an ingress domain. If an ingress domain isn't specified, use the default ingress domain that the Learning Center operator was configured with.

When setting a custom domain, configure DNS with a wildcard domain to forward all requests for sub-domains of the custom domain to the ingress router of the Kubernetes cluster.

To provide the ingress domain, you can set the `session.ingress.domain` field. For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopSession
metadata:
  name: lab-markdown-sample
spec:
  environment:
    name: lab-markdown-sample-user1
  session:
    ingress:
      domain: training.learningcenter.tanzu.vmware.com
```

You can create a full host name for the session by prefixing the ingress domain with a host name constructed from the name of the workshop environment and the session ID.

If overriding the domain, by default, the workshop session is exposed by using a HTTP connection. If you require a secure HTTPS connection, you must have access to a wildcard SSL certificate for the domain.

You must create a secret of type `tls` for the certificate in the `learningcenter` namespace or in the namespace where the Learning Center operator is deployed. You must then set the name of that secret in the `session.ingress.secret` field. For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopSession
metadata:
  name: lab-markdown-sample
spec:
  environment:
    name: lab-markdown-sample-user1
  session:
    ingress:
      domain: training.learningcenter.tanzu.vmware.com
      secret: training.learningcenter.tanzu.vmware.com-tls
```

You can terminate HTTPS connections by using an external load balancer rather than by specifying a secret for ingresses managed by the Kubernetes ingress controller. When routing traffic into the Kubernetes cluster as HTTP connections, you can override the ingress protocol without specifying an ingress secret by setting the `session.ingress.protocol` field.

For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopSession
metadata:
  name: lab-markdown-sample
spec:
  environment:
    name: lab-markdown-sample-user1
  session:
    ingress:
      domain: training.learningcenter.tanzu.vmware.com
      protocol: https
```

To override or set the ingress class, add `session.ingress.class`. This dictates which ingress router is used when more than one option is available.

For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopSession
metadata:
  name: lab-markdown-sample
spec:
  environment:
    name: lab-markdown-sample-user1
  session:
    ingress:
      domain: training.learningcenter.tanzu.vmware.com
      secret: training.learningcenter.tanzu.vmware.com-tls
      class: nginx
```

Setting the environment variables

To set the environment variables for the workshop instance, provide the environment variables in the `session.env` field.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopSession
metadata:
  name: lab-markdown-sample
spec:
  environment:
    name: lab-markdown-sample
  session:
    id: user1
    env:
    - name: REPOSITORY-URL
      value: YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE
```

Where `YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE` is the Git repository URL for `lab-markdown-sample`. For example, `{YOUR-GIT-REPO-URL}/lab-markdown-sample`.

Values of fields in the list of resource objects can reference a number of predefined parameters. Available parameters are:

- `session_id` is a unique ID for the workshop instance within the workshop environment.
- `session_namespace` is the namespace created for and bound to the workshop instance. This is the namespace unique to the session and where a workshop can create their own resources.
- `environment_name` is the name of the workshop environment. For now this is the same as the name of the namespace for the workshop environment. Don't rely on them being the

same, and use the most appropriate to cope with any future change.

- `workshop_namespace` is the namespace for the workshop environment. This is the namespace where all deployments of the workshop instances are created, and where the service account that the workshop instance runs as exists.
- `service_account` is the name of the service account the workshop instance runs as, and which has access to the namespace created for that workshop instance.
- `ingress_domain` is the host domain under which host names can be created when creating ingress routes.
- `ingress_protocol` is the protocol (http/https) used for ingress routes created for workshops.

The syntax for referencing one of the parameters is `$(parameter_name)`.

If the workshop environment had specified a set of extra environment variables to be set for workshop instances, it is up to you to incorporate those in the set of environment variables you list under `session.env`. That is, anything listed in `session.env` of the `WorkshopEnvironment` custom resource of the workshop environment is ignored.

Enable anonymous access to a Learning Center training portal

This topic describes how you enable anonymous access to a Learning Center training portal. The REST API with client authentication provides a means to have the portal create and manage workshop sessions on your behalf but allow a separate system handle user authentication.

If you do not need to authenticate users but still want to provide your own front end from which users select a workshop, such as when integrating workshops into an existing web property, you can enable anonymous mode and redirect users to a URL for workshop session creation.

Note: Anonymous mode is only recommended for temporary deployments and not for a permanent web site providing access to workshops.

Enabling anonymous access

Set the registration type to `anonymous` to enable full anonymous access to the training portal:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    registration:
      type: anonymous
  workshops:
  ...
```

Note: Users can still visit the training portal directly and view the catalog of available workshops, so instead of linking to the main page of the training portal, link from your custom index page to the individual links for creating each workshop.

Triggering workshop creation

Direct users' browsers to a URL that is specific to a workshop to trigger creation and allocation of the workshop.

The URL format looks like this:

```
TRAINING-PORTAL-URL/workshops/environment/NAME/create/?index_url=INDEX
```

Where:

- `NAME` is the name of the workshop environment corresponding to the workshop that you creates.
- `INDEX` is the URL of your custom index page that contains the workshops.

The user is redirected back to this index page when:

- a user completes the workshop
- an error occurs

When a user is redirected back to the index page, a query string parameter is supplied to display a banner or other indication about why the user was returned.

The name of the query string parameter is `notification` and the possible values are:

- `session-deleted` - Used when the workshop session is completed or restarted.
- `workshop-invalid` - Used when the name of the workshop environment created is invalid.
- `session-unavailable` - Used when capacity is reached and a workshop session cannot be created.
- `session-invalid` - Used when an attempt is made to access a session that doesn't exist. This can occur when the workshop dashboard is refreshed after the workshop session is expired and deleted.

Enable anonymous access to a Learning Center training portal

This topic describes how you enable anonymous access to a Learning Center training portal. The REST API with client authentication provides a means to have the portal create and manage workshop sessions on your behalf but allow a separate system handle user authentication.

If you do not need to authenticate users but still want to provide your own front end from which users select a workshop, such as when integrating workshops into an existing web property, you can enable anonymous mode and redirect users to a URL for workshop session creation.

Note: Anonymous mode is only recommended for temporary deployments and not for a permanent web site providing access to workshops.

Enabling anonymous access

Set the registration type to `anonymous` to enable full anonymous access to the training portal:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    registration:
      type: anonymous
  workshops:
  ...
```

Note: Users can still visit the training portal directly and view the catalog of available workshops, so instead of linking to the main page of the training portal, link from your custom index page to the individual links for creating each workshop.

Triggering workshop creation

Direct users' browsers to a URL that is specific to a workshop to trigger creation and allocation of the workshop.

The URL format looks like this:

```
TRAINING-PORTAL-URL/workshops/environment/NAME/create/?index_url=INDEX
```

Where:

- `NAME` is the name of the workshop environment corresponding to the workshop that you creates.
- `INDEX` is the URL of your custom index page that contains the workshops.

The user is redirected back to this index page when:

- a user completes the workshop
- an error occurs

When a user is redirected back to the index page, a query string parameter is supplied to display a banner or other indication about why the user was returned.

The name of the query string parameter is `notification` and the possible values are:

- `session-deleted` - Used when the workshop session is completed or restarted.
- `workshop-invalid` - Used when the name of the workshop environment created is invalid.
- `session-unavailable` - Used when capacity is reached and a workshop session cannot be created.
- `session-invalid` - Used when an attempt is made to access a session that doesn't exist. This can occur when the workshop dashboard is refreshed after the workshop session is expired and deleted.

Use the Learning Center workshop catalog

A single training portal can host one or more workshops. This topic describes how you use the workshop catalog to list the available workshops and get information about them using the REST API.

Listing available workshops

The URL sub path for accessing the list of available workshop environments is `/workshops/catalog/environments/`. When making the request, you must supply the access token in the HTTP `Authorization` header with type set as `Bearer`:

```
curl -v -H "Authorization: Bearer <access-token>" \
<training-portal-url>/workshops/catalog/environments/
```

The JSON response looks like this:

```
{
  "portal": {
    "name": "learningcenter-tutorials",
    "uid": "9b82a7b1-97db-4333-962c-97be6b5d7ee0",
    "generation": 451,
    "url": "<training_portal_url>",
    "sessions": {
      "maximum": 10,
      "registered": 0,
      "anonymous": 0,
      "allocated": 0
    }
  },
  "environments": [
    {
      "name": "learningcenter-tutorials-w01",
      "state": "RUNNING",
      "workshop": {
        "name": "lab-et-self-guided-tour",
        "id": "15e5f1a569496f335049bb00c370ee20",
        "title": "Workshop Building Tutorial",
        "description": "A guided tour of how to build a workshop for your team's learn
ing center.",
        "vendor": "",
        "authors": [],
        "difficulty": "",
        "duration": "",
        "tags": [],
        "logo": "",
        "url": "<workshop_repository_url>"
      },
      "duration": 1800,
      "capacity": 10,
      "reserved": 0,
      "allocated": 0,
      "available": 0
    }
  ]
}
```

For each workshop listed under `environments`, where a field listed under `workshop` has the same name as appears in the `Workshop` custom resource, it has the same meaning. The `id` field is an additional field that can uniquely identify a workshop based on the name of the workshop image, the Git repository for the workshop, or the website hosting the workshop instructions. The value of the `id` field does not rely on the name of the `Workshop` resource and must be the same if the same workshop details are used but the name of the `Workshop` resource is different.

The `duration` field provides the time in seconds after which the workshop environment expires. The value is `null` if there is no expiration time for the workshop.

The `capacity` field is the maximum number of workshop sessions that you can create for the workshop.

The `reserved` field indicates how many instances of the workshop are reserved as hot spares. These are used to service requests for a workshop session. If no reserved instances are available and capacity has not been reached, a new workshop session is created on demand.

The `allocated` field indicates how many workshop sessions are active and currently allocated to a user.

The `available` field indicates how many workshop sessions are available for immediate allocation. This is never more than the number of reserved instances.

Under `portal.sessions`, the `allocated` field indicates the total number of allocated sessions across all workshops hosted by the portal.

Where `maximum`, `registered`, and `anonymous` are nonzero, they are the limit on the number of workshops run.

- The `maximum` is the total number of workshop sessions that can be run by the portal across all workshops. If `allocated` for the whole portal has reached `maximum`, no more workshop sessions are created.
- The value of `registered` when nonzero indicates a cap on the number of workshop sessions a single registered portal user can have running at the one time.
- The value of `anonymous` when nonzero indicates a cap on the number of workshop sessions an anonymous user can have running at the one time. Anonymous users are users created as a result of the REST API being used or if anonymous access is enabled when the user accesses the portal through the web interface.

By default, only workshop environments currently marked with a `state` of `RUNNING` are returned, that is, those workshop environments which are taking new workshop session requests. If you also want to see the workshop environments which are currently in the process of being shut down, you must provide the `state` query string parameter to the REST API call and indicate which states workshop environments to return for.

```
curl -v -H "Authorization: Bearer <access-token>" \
https://lab-markdown-sample-ui.test/workshops/catalog/environments/?state=RUNNING&stat
e=STOPPING
```

You can include the `state` query string parameter more than once to see workshop environments in both `RUNNING` and `STOPPING` states.

If anonymous access to the list of workshop environments is enabled and you are not authenticated when using the REST API endpoint, only workshop environments in a running state are returned.

Use session management for your Learning Center workshops

This topic describes how you use the REST API endpoints for session management, which allows you to request a workshop session to be allocated.

Deactivating portal user registration

When you use the REST API to trigger creation of workshop sessions, VMware recommends that you deactivate user registration through the training portal web interface. This means that only the admin user is able to directly access the web interface for the training portal.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: learningcenter-tutorials
spec:
  portal:
    registration:
      type: one-step
      enabled: false
  workshops:
  ...
```

Requesting a workshop session

The form of the URL sub path for requesting the allocation of a workshop environment by using the REST API is `/workshops/environment/<name>/request/`. The name segment must be replaced with the name of the workshop environment. When making the request, the access token must be supplied in the HTTP `Authorization` header with type set as `Bearer`:

```
curl -v -H "Authorization: Bearer <access-token>" \
<training-portal-url>/workshops/environment/<name>/request/?index_url=https://hub.tes
t/
```

You can supply a query string parameter, `index_url`. When you restart the workshop session from the workshop environment web interface, the session is deleted and the user is redirected to the supplied URL. This URL is that of your front end web application that has requested the workshop session, allowing users to select a different workshop.

The value of the `index_url` is not available if session cookies are cleared or a session URL is shared with another user. In this case, a user is redirected back to the training portal URL instead. You can override the global default for this case by specifying the index URL as part of the `TrainingPortal` configuration.

When successful, the JSON response from the request is of the form:

```
{
    "name": "educaes-tutorials-w01-s001",
    "user": "8d2d0c8b-6ff5-4244-b136-110fd8d8431a",
    "url": "/workshops/session/learningcenter-tutorials-w01-s001/activate/?token=6UIW4
D8Bhf0egVmsEKYlaOcTywrpQJGi&index_url=https%3A%2F%2Fhub.test%2F",
    "workshop": "learningcenter-tutorials",
    "environment": "learningcenter-tutorials-w01",
    "namespace": "learningcenter-tutorials-w01-s001"
}
```

This includes the name of the workshop session, an ID for identifying the user, and both a URL path with an activation token and an index URL included as query string parameters.

Redirect the user's browser to this URL path on the training portal host. Accessing the URL activates the workshop session and then redirects the user to the workshop dashboard.

If the workshop session is not activated, which confirms allocation of the session, the session is deleted after 60 seconds.

When a user is redirected back to the URL for the index page, a query string parameter is supplied to give the reason the user is being returned. You can use this to display a banner or other indication as to why the user was returned.

The name of the query string parameter is `notification` and the possible values are:

- `session-deleted` - Used when the workshop session is completed or restarted.
- `workshop-invalid` - Used when the name of the workshop environment supplied while attempting to create the workshop is invalid.
- `session-unavailable` - Used when capacity is reached, and a workshop session cannot be created.
- `session-invalid` - Used when an attempt is made to access a session that doesn't exist. This can occur when the workshop dashboard is refreshed sometime after the workshop session expired and was deleted.

In prior versions, the name of the session was returned through the "session" property, whereas the "name" property is now used. To support older code using the REST API, the "session" property is still returned, but it is deprecated.

Associating sessions with a user

When the workshop session is requested, a unique user account is created in the training portal each time. You can identify this account by using the `user` identifier, which is returned in the response.

The front end using the REST API to create workshop sessions can track user activity so that the training portal associates all workshop sessions created by the same user. Supply the `user` identifier with subsequent requests by the same user in the request parameter:

```
curl -v -H "Authorization: Bearer <access-token>" \
https://lab-markdown-sample-ui.test/workshops/environment/<name>/request/?index_url=ht
tps://hub.test/&user=<user>
```

If the supplied ID matches a user in the training portal, the training portal uses it internally and returns the same value for `user` in the response.

When the user does match, and if there is already a workshop session allocated to the user for the workshop being requested, the training portal returns a link to the existing workshop session, rather than requesting that the user create a new workshop session.

If the user is not a match, possibly because the training portal was completely redeployed since the last time it was accessed, the training portal returns a new user identifier.

The first time you make a request to create a workshop session for a user where `user` is not supplied, you can optionally supply request parameters for the following to set these as the user details in the training portal.

- `email` - The email address of the user.
- `first_name` - The first name of the user.
- `last_name` - The last name of the user.

These details will be accessible through the admin pages of the training portal.

When sessions are associated with a user, you can query all active sessions for that user across the different workshops hosted by the instance of the training portal:

```
curl -v -H "Authorization: Bearer <access-token>" \
<training-portal-url>/workshops/user/<user>/sessions/
```

The response is of the form:

```
{
  "user": "8d2d0c8b-6ff5-4244-b136-110fd8d8431a",
  "sessions": [
    {
      "name": "learningcenter-tutorials-w01-s001",
      "workshop": "learningcenter-tutorials",
      "environment": "learningcenter-tutorials-w01",
      "namespace": "learningcenter-tutorials-w01-s001",
      "started": "2020-07-31T03:57:33.942Z",
      "expires": "2020-07-31T04:57:33.942Z",
      "countdown": 3353,
      "extendable": false
    }
  ]
}
```

After a workshop has expired or has otherwise been shut down, the training portal no longer returns an entry for the workshop.

Listing all workshop sessions

To get a list of all running workshops sessions allocated to users, provide the `sessions=true` flag to the query string parameters of the REST API call. This lists the workshop environments available through the training portal.

```
curl -v -H "Authorization: Bearer <access-token>" |
<training-portal-url>/workshops/catalog/environments/?sessions=true
```

The JSON response is of the form:

```
{
  "portal": {
    "name": "learningcenter-tutorials",
    "uid": "9b82a7b1-97db-4333-962c-97be6b5d7ee0",
    "generation": 476,
    "url": "<training-portal-url>",
    "sessions": {
      "maximum": 10,
      "registered": 0,
      "anonymous": 0,
      "allocated": 1
    }
  },
  "environments": [
    {
      "name": "learningcenter-tutorials-w01",
      "state": "RUNNING",
      "workshop": {
        "name": "lab-et-self-guided-tour",
        "id": "15e5f1a569496f335049bb00c370ee20",
        "title": "Workshop Building Tutorial",
        "description": "A guided tour of how to build a workshop for your team's learn
ing center.",
        "vendor": "",
        "authors": [],
        "difficulty": "",
        "duration": "",
        "tags": [],
        "logo": "",
        "url": "<workshop-repository-url>"
      },
      "duration": 1800,
      "capacity": 10,
      "reserved": 0,
      "allocated": 1,
      "available": 0,
      "sessions": [
        {
          "name": "learningcenter-tutorials-w01-s002",
          "state": "RUNNING",
          "namespace": "learningcenter-tutorials-w01-s002",
          "user": "672338f3-4085-4782-8d9b-ae1637e1c28c",
          "started": "2021-11-05T15:50:04.824Z",
          "expires": "2021-11-05T16:20:04.824Z",
          "countdown": 1737,
          "extendable": false
        }
      ]
    }
  ]
}
```

No workshop sessions are returned if anonymous access to this REST API endpoint is enabled and you are not authenticated against the training portal.

Only workshop environments with a `state` of `RUNNING` are returned by default. To see workshop environments that are shut down and any workshop sessions that still haven't been completed, supply the `state` query string parameter with value `STOPPING`.

```
curl -v -H "Authorization: Bearer <access-token>" \
<training-portal-url>/workshops/catalog/environments/?sessions=true&state=RUNNING&stat
e=STOPPING
```

Include the `state` query string parameter more than once to see workshop environments in both `RUNNING` and `STOPPING` states.

Use client authentication for Learning Center

This topic describes how you can use the portal REST API to integrate access to workshops into an existing website or to create a custom web interface for accessing workshops hosted across one or more training portals.

The training portal web interface is a quick way of providing access to a set of workshops when running a supervised training workshop. The REST API gives you access to the list of workshops hosted by a training portal instance and allow you to request and access workshop sessions. This bypasses the training portal's own user registration and log in so you can implement whatever access controls you need. This can include anonymous access or access integrated into an organization's single sign-on system.

Querying the credentials

To provide access to the REST API, a robot account is automatically provisioned. Obtain the login credentials and details of the OAuth client endpoint used for authentication by querying the resource definition for the training portal after it is created and the deployment completed. If using `kubectl describe`, use:

```
kubectl describe trainingportal.learningcenter.tanzu.vmware.com/<training-portal-name>
```

The status section of the output reads:

```
Status:
  learningcenter:
    Clients:
      Robot:
        Id:      ACZpcaLIT3qr725YWmXu8et9REl4HBg1
        Secret:  t5IfXbGZQThAKR43apoc9usOFVDv2BLE
    Credentials:
      Admin:
        Password:  0kGmMlYw46BZT2vCntyrRuFf1gQq5ohi
        Username:  learningcenter
      Robot:
        Password:  QrnY67ME9yGasNhq2OTbgWA4RzipUvo5
        Username:  robot@learningcenter
```

Use the admin login credentials when you log in to the training portal web interface to access admin pages.

Use the robot login credentials if you want to access the REST API.

Requesting an access token

Before you can make requests against the REST API to query details about workshops or request a workshop session, you must log in through the REST API to get an access token.

This is done from any front-end web application or provisioning system, but the step is equivalent to making a REST API call by using `curl` of:

```
curl -v -X POST -H \
"Content-Type: application/x-www-form-urlencoded" \
-d "grant_type=password&username=robot@learningcenter&password=<robot-password>" \
-u "<robot-client-id>:<robot-client-secret>" \
<training-portal-url>/oauth2/token/
```

The URL sub path is `/oauth2/token/`.

Upon success, the output is a JSON response consisting of:

```
{
    "access_token": "tg31ied56fOo4axuhuZLHj5JpUYCEL",
    "expires_in": 36000,
    "token_type": "Bearer",
    "scope": "user:info",
    "refresh_token": "1ryXhXbNA9RsTRuCE8fDAyZToJmp30"
}
```

Refreshing the access token

The access token that is provided expires: it needs to be refreshed before it expires if in use by a long-running application.

To refresh the access token, use the equivalent of:

```
curl -v -X POST -H \
"Content-Type: application/x-www-form-urlencoded" \
-d "grant_type=refresh_token&refresh_token=<refresh-token>& \client_id=<robot-client-i
d>&client_secret=<robot-client-secret>" \
https://lab-markdown-sample-ui.test/oauth2/token/
```

As with requesting the initial access token, the URL sub path is `/oauth2/token/`.

The JSON response is of the same format as if a new token was requested.

Troubleshoot Learning Center

This topic gives you troubleshooting and recovery steps for Learning Center known issues.

Training portal stays in pending state

The training portal stays in a “pending” state.

The Training Portal custom resource (CR) has a status property. To see the status, run:

```
kubectl get trainingportals.learningcenter.tanzu.vmware.com
```

Explanation

If the status stays in a pending state, the TLS secret `tls` might not be available. Other errors can also cause the status to stay in a pending state, so it is important to check the operator and portal logs to execute the right steps.

Solution

1. Access the operator logs by running:

   ```
   kubectl logs deployment/learningcenter-operator -n learningcenter
   ```

   Access the portal logs by running:

   ```
   kubectl logs deployment/learningcenter-portal -n {PORTAL_NAMESPACE}
   ```

2. Check whether the TLS secret `tls` is available. The TLS secret must be on the Learning Center operator namespace (by default `learningcenter`). If the TLS secret is not on the Learning Center operator namespace, the operator logs contain the following error:

   ```
   ERROR:kopf.objects:Handler 'learningcenter' failed temporarily: TLS secret tls
   is not available
   ```

3. Follow the steps in Enforcing Secure Connections in *Learning Center Operator* to create the TLS secret.
4. Redeploy the `trainingPortal` resource.

image-policy-webhook-service not found

You are installing a Tanzu Application Platform profile and you get this error:

```
Internal error occurred: failed calling webhook "image-policy-webhook.signing.run.tanz
u.vmware.com": failed to call webhook: Post "https://image-policy-webhook-service.imag
e-policy-system.svc:443/signing-policy-check?timeout=10s": service "image-policy-webho
ok-service" not found
```

Explanation

This is a race condition error among some Tanzu Application Platform packages.

Solution

To recover from this error you only need to redeploy the trainingPortal resource.

Updates to Tanzu Application Platform values file not reflected in Learning Center Training Portal

If you installed Learning Center through Tanzu profiles, then your installation made use of a tap-values.yaml file where configurations were specified for Learning Center. If you make updates to these configurations using this command:

```
tanzu package installed update tap --package-name tap.tanzu.vmware.com --version {VERS
ION} -f tap-values.yml -n tap-install
```

then the changes are not reflected in the deployed Learning Center Training Portal resource. Tap package updates currently `DO NOT` update running Learning Center Training Portal resources.

Run one of these commands to validate changes made to parameters provided to the Learning Center Operator. These parameters include ingressDomain, TLS secret, ingressClass, and others.

Command:

```
kubectl describe systemprofile
```

Command:

```
kubectl describe pod  -n learningcenter
```

Explanation

By design, the training portal resources do not react to any changes on the parameters provided when the training portals were created. This prevents any change on the `trainingportal` resource from affecting any online user running a workshop.

Solution

You must restart the operator resource by first deleting the operator pod:

```
kubectl delete pod -n learningcenter learningcenter-operator-$OPERATOR_POD_NAME
```

Then delete the training portal resource. Redeploy `trainingportal` in a maintenance window where Learning Center is unavailable while the`systemprofile` is updated.

Increase your cluster's resources

If you don't have enough nodes or enough resources on nodes for deploying the workloads, node pressure might occur. In this case, follow your cloud provider's instructions on how to scale out or scale up your cluster.

Kubernetes Api Timeout error

The following operator error log means there is a connection error with the Kubernetes API server:

```
operator-log: unexpected error occurred. Read timed out.
```

This error has been found when running Learning Center with the Azure AkS cloud provider.

Solution

To fix this error:

1. Delete the operator pod on the learningcenter namespace.
2. Delete the training portal once the operator is running again by using:

```
kubectl delete trainingportals $PORTAL_NAME
```

1. Redeploy the `trainingPortal` resource.

No URL returned to your trainingportal

After deploying the Learning Center Operator and Trainingportal resources, the following command can yield the resource with no URL, even though your resources deployed correctly and are running:

```
kubectl get trainingportals
```

You also already specified learningcenter.mydomain.com in your tap values YAML file if installed through Tanzu Application Platform. See specifying ingress domain

Solution

Learning center requires that you use a wildcard domain (Wildcard DNS entry) to access your training portal in the browser. This configuration must be done in your DNS provider with a rule that points your wildcard domain to your IP/Load balancer.

For example, if using the default workshop on an Elastic Kubernetes Service (EKS) cluster, your URL could look something like:

`learning-center-guided.learningcenter.yourdomain.com`

Where learningcenter.yourdomain.com needs a DNS configuration made to point to your default ingress controller.

In this case, the wildcard domain configuration needed is `*.learningcenter.yourdomain.com`.

After this configuration is made, you might need to restart your operator resource by deleting and redeploying to see the URL update.

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 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.

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..
>
> **Note:** The Supply Chain Choreographer is now bundled with the Cartographer Conventions. For information on configuring and using Cartographer Conventions, see Creating conventions.

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/0.4.0 --values-schema
--namespace tap-install

KEY                   DEFAULT  TYPE    DESCRIPTION
aws_iam_role_arn               string  Optional: Arn role that has access to pul
l images from ECR container registry
ca_cert_data                   string  Optional: PEM Encoded certificate data fo
r image registries with private CA.
excluded_components   []       array   Optional: List of components to exclude f
rom installation (e.g. [conventions])
```

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-name cartographer.tanzu.vmware.com \
  --version 0.4.0
```

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 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 is responsible for delivering to an environment the Kubernetes configuration that has been produced by the Out of the Box Supply Chains, including Basic, Testing, and Testing With Scanning.

Prerequisites

To make use of this package you must have installed:

- Supply Chain Cartographer
- Out of the Box Templates

Usage

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
                                                ...
```

As a prerequisite to the Basic, Testing, and Testing With Scanning Out of the Box Supply Chains, you must install this package to have Workloads delivered properly.

Consumers do not interact directly with this package. Instead, this package is used once a carto.run/Deliverable object is created by the supply chains to express the intention of having the Workloads that go through them delivered to an environment. At this time, the environment is the same Kubernetes cluster as the Supply Chains.

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 is responsible for delivering to an environment the Kubernetes configuration that has been produced by the Out of the Box Supply Chains, including Basic, Testing, and Testing With Scanning.

Prerequisites

To make use of this package you must have installed:

- Supply Chain Cartographer
- Out of the Box Templates

Usage

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
                                                ...
```

As a prerequisite to the Basic, Testing, and Testing With Scanning Out of the Box Supply Chains, you must install this package to have Workloads delivered properly.

Consumers do not interact directly with this package. Instead, this package is used once a carto.run/Deliverable object is created by the supply chains to express the intention of having the Workloads that go through them delivered to an environment. At this time, the environment is the same Kubernetes cluster as the Supply Chains.

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.


   git_implementation     go-git   string  Which git client library to use.
                                           Valid options are go-git or libgit2.
   ```

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-name 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'
```

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 included 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
- 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

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 gets built by the supply chains 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 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 ships with 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 be able 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 we created for this workload, bind the `workload` ClusterRole to the ServiceAccount we 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 just 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 v1.2, see https://github.com/pivotal/docs-tap/blob/main/authn-authz/overview.md.

Developer workload

With the developer namespace set up with the preceding objects (secret, serviceaccount, and rolebinding), you can create the workload object.

To do so, make use of the `apps` plug-in from the Tanzu CLI:

```
tanzu apps workload create [flags] [workload-name]
```

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 a prebuilt image, for more information about how to leverage 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).

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 included 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
- 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

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 gets built by the supply chains 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 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 ships with 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 be able 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 we created for this workload, bind the `workload` ClusterRole to the ServiceAccount we 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 just 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 v1.2, see https://github.com/pivotal/docs-tap/blob/main/authn-authz/overview.md.

Developer workload

With the developer namespace set up with the preceding objects (secret, serviceaccount, and rolebinding), you can create the workload object.

To do so, make use of the `apps` plug-in from the Tanzu CLI:

```
tanzu apps workload create [flags] [workload-name]
```

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 a prebuilt image, for more information about how to leverage 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).

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/0.7.0 \
     --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).

git_implementation                     Determines which git client library to u
se. Valid options are go-git or
                                       libgit2.

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.ssh_secret                      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 (TBS) Cl
usterBuilder 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 (TBS) Image objects
as well 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

gitops:
  server_address: https://github.com/
  repository_owner: vmware-tanzu
  branch: main
  username: supplychain
  email: supplychain
  commit_message: supplychain@cluster.local
  ssh_secret: git-ssh
  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-name ootb-supply-chain-basic.tanzu.vmware.com \
  --version 0.7.0 \
  --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
- 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

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 make sure you have configured the namespace correctly, it is important that the namespace has 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.

Below you will find details about the new objects compared to Out of the Box Supply Chain Basic.

Updates to the developer Namespace

In order 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.

So, aside from the objects previously defined in the Out of the Box Supply Chain Basic section, you need to include one more:

- `tekton/Pipeline`: the definition of a series of tasks to run against the source code that has been found by earlier resources in the Supply Chain.

Tekton/Pipeline

Despite the full liberty around tasks to run, the Tekton or pipeline object **must** be labelled with `apps.tanzu.vmware.com/pipeline: test`, and define that it expects to take two parameters:

- `source-url`, an HTTP address where a `.tar.gz` file containing all the source code to be tested can be 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
        steps:
          - name: test
            image: gradle
            script: |-
              cd `mktemp -d`
              wget -qO- $(params.source-url) | tar xvz -m
              ./mvnw test
```

At this point, 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, the latest PipelineRun created can be deleted, which triggers a re-run.

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 template to include labels that differentiate the pipelines. Then configure the labels to differentiate between pipelines, as shown in the following example:

```
  selector:
     resource:
       apiVersion: tekton.dev/v1beta1
       kind: Pipeline
     matchingLabels:
       apps.tanzu.vmware.com/pipeline: test
+          apps.tanzu.vmware.com/language: #@ data.values.workload.metadata.labe
ls["apps.tanzu.vmware.com/language"]
```

The following example shows one namespace per-language pipeline:

```
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 ./...
```

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 by means of the `has-tests` label.

For example:

```
tanzu apps workload create tanzu-java-web-app \
  --git-branch main \
  --git-repo https://github.com/sample-accelerators/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/sample-accelerators/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
- 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

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 make sure you have configured the namespace correctly, it is important that the namespace has 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.

Below you will find details about the new objects compared to Out of the Box Supply Chain Basic.

Updates to the developer Namespace

In order 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.

So, aside from the objects previously defined in the Out of the Box Supply Chain Basic section, you need to include one more:

- `tekton/Pipeline`: the definition of a series of tasks to run against the source code that has been found by earlier resources in the Supply Chain.

Tekton/Pipeline

Despite the full liberty around tasks to run, the Tekton or pipeline object **must** be labelled with `apps.tanzu.vmware.com/pipeline: test`, and define that it expects to take two parameters:

- `source-url`, an HTTP address where a `.tar.gz` file containing all the source code to be tested can be 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
        steps:
          - name: test
            image: gradle
            script: |-
              cd `mktemp -d`
              wget -qO- $(params.source-url) | tar xvz -m
              ./mvnw test
```

At this point, 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, the latest PipelineRun created can be deleted, which triggers a re-run.

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 template to include labels that differentiate the pipelines. Then configure the labels to differentiate between pipelines, as shown in the following example:

  ```
    selector:
       resource:
         apiVersion: tekton.dev/v1beta1
         kind: Pipeline
       matchingLabels:
         apps.tanzu.vmware.com/pipeline: test
  +         apps.tanzu.vmware.com/language: #@ data.values.workload.metadata.labe
  ls["apps.tanzu.vmware.com/language"]
  ```

  The following example shows one namespace per-language pipeline:

  ```
  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 ./...
```

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 by means of the `has-tests` label.

For example:

```
tanzu apps workload create tanzu-java-web-app \
  --git-branch main \
  --git-repo https://github.com/sample-accelerators/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/sample-accelerators/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).

git_implementation                     Determines which git client library to u
se. Valid options are go-git or
                                       libgit2.

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.ssh_secret                      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 (TBS) 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 (TBS) Image objects as well as de
ploying 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

gitops:
  server_address: https://github.com/
  repository_owner: vmware-tanzu
  branch: main
  username: supplychain
  email: supplychain
  commit_message: supplychain@cluster.local
  ssh_secret: git-ssh
  commit_strategy: direct

cluster_builder: default
service_account: default
```

Important

it's **required** that the `gitops.repository_prefix` field ends with a `/`.

4. With the configuration ready, install the package by running:

```
tanzu package install ootb-supply-chain-testing \
  --package-name ootb-supply-chain-testing.tanzu.vmware.com \
  --version 0.7.0 \
  --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.

For workloads that use 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. 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
- 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

Prerequisites

To use this supply chain, verify that:

- Tanzu Application Platform GUI is configured to enable CVE scan results. This configuration allows the Supply Chain Choreographer TAP GUI plug-in to retrieve metadata about project packages and their vulnerabilities.
- 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**.
- Developer namespace is configured with the objects per Out of the Box Supply Chain With Testing guidance. This supply chain is in addition to the Supply Chain with testing.
- (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.

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 have 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 on 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 jobs are created for scanning the source code. For more information, see ScanTemplate section.
- **image scan template**: A template of how jobs are created for scanning the image produced by the supply chain. For more information, see ScanTemplate section.

Below you will find 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 overriden 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 following commands. See Tanzu apps workload commands.

```
tanzu apps workload update WORKLOAD --param "scanning_source_policy=SCAN-POLIC
Y" -n DEV-NAMESPACE
tanzu apps workload update WORKLOAD --param "scanning_source_template=SCAN-TEMP
LATE" -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.

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': 'component-a'
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 to be used by a Job to run a particular scan (image or source). When an ImageScan or SourceScan is instantiated by the supply chain, they reference these templates which must live in the same namespace as the workload with the names matching the ones below:

- source scanning (`blob-source-scan-template`)
- image scanning (`private-image-scan-template`)

If you are targeting a namespace that does not match the one configured in the Tanzu Application Platform profiles, for example if `grype.namespace` is not the same as the one you are writing the workload to, you can install these in such namespace by making use of the `tanzu package install` command as described in Install Supply Chain Security Tools - Scan:

1. Create a file named `ootb-supply-chain-basic-values.yaml` that specifies the corresponding values to the properties you want to change. For example:

   ```
   grype:
     namespace: YOUR-DEV-NAMESPACE
     targetImagePullSecret: registry-credentials
   ```

2. With the configuration ready, install the templates by running:

   ```
   tanzu package install grype-scanner \
     --package-name grype.scanning.apps.tanzu.vmware.com \
     --version 1.0.0 \
     --namespace YOUR-DEV-NAMESPACE
   ```

Note: Although you can customize the templates, if you are just following the Getting Started guide then it is recommended you stick with what is provided in the installation of `grype.scanning.apps.tanzu.vmware.com`. This is created in the same namespace as configured by using `grype.namespace` in either Tanzu Application Platform profiles or individual component installation as in the earlier example. 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. For 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 template to include labels that differentiate the pipelines. Then configure the labels to differentiate between pipelines. For example:

```
  selector:
     resource:
       apiVersion: tekton.dev/v1beta1
       kind: Pipeline
     matchingLabels:
       apps.tanzu.vmware.com/pipeline: test
+         apps.tanzu.vmware.com/language: #@ data.values.workload.metadata.labe
ls["apps.tanzu.vmware.com/language"]
```

The following example shows one namespace per-language pipeline:

```
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 ./...
```

Developer workload

With the ScanPolicy and ScanTemplate objects, with the required names set, submitted to the same namespace where the workload are 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/sample-accelerators/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/sample-accelerators/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 being built or images from being deployed that contain vulnerabilities that are in violation of the user-defined scan policy. Refer to the guidelines provided in Triaging and Remediating CVEs to learn how to handle these vulnerabilities and unblock your Supply Chain.

Scan Image using Snyk

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.

For workloads that use 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. 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
- 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

Prerequisites

To use this supply chain, verify that:

- Tanzu Application Platform GUI is configured to enable CVE scan results. This configuration allows the Supply Chain Choreographer TAP GUI plug-in to retrieve metadata about project packages and their vulnerabilities.
- 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**.
- Developer namespace is configured with the objects per Out of the Box Supply Chain With Testing guidance. This supply chain is in addition to the Supply Chain with testing.
- (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.

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/workload-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 have 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 on 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 jobs are created for scanning the source code. For more information, see ScanTemplate section.

- **image scan template**: A template of how jobs are created for scanning the image produced by the supply chain. For more information, see ScanTemplate section.

Below you will find 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 overriden 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 following commands. See Tanzu apps workload commands.

  ```
  tanzu apps workload update WORKLOAD --param "scanning_source_policy=SCAN-POLIC
  Y" -n DEV-NAMESPACE
  tanzu apps workload update WORKLOAD --param "scanning_source_template=SCAN-TEMP
  LATE" -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.

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': 'component-a'
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 := input.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 := vuln.ratings.rating[_].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 to be used by a Job to run a particular scan (image or source). When an ImageScan or SourceScan is instantiated by the supply chain, they reference these templates which must live in the same namespace as the workload with the names matching the ones below:

- source scanning (`blob-source-scan-template`)
- image scanning (`private-image-scan-template`)

If you are targeting a namespace that does not match the one configured in the Tanzu Application Platform profiles, for example if `grype.namespace` is not the same as the one you are writing the workload to, you can install these in such namespace by making use of the `tanzu package install` command as described in Install Supply Chain Security Tools - Scan:

1. Create a file named `ootb-supply-chain-basic-values.yaml` that specifies the corresponding values to the properties you want to change. For example:

   ```
   grype:
     namespace: YOUR-DEV-NAMESPACE
     targetImagePullSecret: registry-credentials
   ```

2. With the configuration ready, install the templates by running:

   ```
   tanzu package install grype-scanner \
     --package-name grype.scanning.apps.tanzu.vmware.com \
     --version 1.0.0 \
     --namespace YOUR-DEV-NAMESPACE
   ```

Note: Although you can customize the templates, if you are just following the Getting Started guide then it is recommended you stick with what is provided in the installation of `grype.scanning.apps.tanzu.vmware.com`. This is created in the same namespace as configured by using `grype.namespace` in either Tanzu Application Platform profiles or individual component installation as in the earlier example. 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. For 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 template to include labels that differentiate the pipelines. Then configure the labels to differentiate between pipelines. For example:

```
  selector:
     resource:
       apiVersion: tekton.dev/v1beta1
       kind: Pipeline
     matchingLabels:
       apps.tanzu.vmware.com/pipeline: test
+         apps.tanzu.vmware.com/language: #@ data.values.workload.metadata.labe
ls["apps.tanzu.vmware.com/language"]
```

The following example shows one namespace per-language pipeline:

```
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 ./...
```

Developer workload

With the ScanPolicy and ScanTemplate objects, with the required names set, submitted to the same namespace where the workload are 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/sample-accelerators/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/sample-accelerators/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 being built or images from being deployed that contain vulnerabilities that are in violation of the user-defined scan policy. Refer to the guidelines provided in Triaging and Remediating CVEs to learn how to handle these vulnerabilities and unblock your Supply Chain.

Scan Image using Snyk

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.

Prerequisites

- 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

To install Out of the Box Supply Chain with Testing and Scanning:

1. 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. Verify `ootb-supply-chain-testing` 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` 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'
```

2. 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/0.7.0 \
  --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).

git_implementation                     Determines which git client library to u
se. Valid options are go-git or
                                       libgit2.

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.ssh_secret                      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 (TBS) 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 (TBS) Image objects as well as de
ploying the
                          application.
```

3. 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

gitops:
  server_address: https://github.com/
  repository_owner: vmware-tanzu
  branch: main
  username: supplychain
  email: supplychain
  commit_message: supplychain@cluster.local
  ssh_secret: git-ssh
  commit_strategy: direct

cluster_builder: default
service_account: default
```

Important

The `gitops.repository_prefix` field must end with `/`.

4. With the configuration ready, install the package by running:

```
tanzu package install ootb-supply-chain-testing-scanning \
  --package-name ootb-supply-chain-testing-scanning.tanzu.vmware.com \
  --version 0.7.0 \
  --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.

In Cartographer, a supply chain is defined as a directed acyclic graph of resources choreographed by the Cartographer controllers, with the definition of the shape of such resources defined by templates.

This package contains a series of reusable templates used by:

- 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 a prerequisite of the Out of the Box Supply Chains, you must install this package to have Workloads delivered properly.

Out of the Box Templates for Supply Chain Choreographer

This topic describes the templates you can use with Supply Chain Choreographer.

In Cartographer, a supply chain is defined as a directed acyclic graph of resources choreographed by the Cartographer controllers, with the definition of the shape of such resources defined by templates.

This package contains a series of reusable templates used by:

- 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 a prerequisite of the Out of the Box Supply Chains, you must install this package to have Workloads delivered properly.

Install Out of the Box Templates

This document describes how to install Out of the Box Templates 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-name 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'
```

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 (TBS) for building container images out of source code.

You can configure a **platform operator** by using `tap-values`:

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.

Note: 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.

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 a prebuilt 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/sample-accelerators/tanzu-java-web-app` Git repository, run:

```
tanzu apps workload create tanzu-java-web-app \
  --app tanzu-java-web-app \
  --type web \
  --git-repo https://github.com/sample-accelerators/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/sample-accelerators/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
                                           gitops_ssh_secret Workload parameter
```

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:
  gitops:
    ssh_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:

```
gitops:
  ssh_secret: GIT-SECRET-NAME
```

You can also override the default secret name directly in the workload by using the `gitops_ssh_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/sample-accelerators/tanzu-java-web-app \
  --git-branch main \
  --param gitops_ssh_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: gitops_ssh_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/sample-accelerators/tanzu-java-web-app
```

Note: A secret reference is only provided to `GitRepository` if `gitops_ssh_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

Despite both the package value and workload parameter being called `gitops.ssh_secret`, you can use HTTP(S) transports as well:

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
   ```

3. With the secret created with the name matching the one configured for `gitops.ssh_secret`, 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 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
```

3. With the secret created with the name matching the one configured for `gitops.ssh_secret`, 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 information about how to generate the keys and set up SSH with the Git server, see Git Authentication's SSH section.

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/v1beta1
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/sample-accelerators/tanzu-java-web-app
```

```
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 (either `libgit2` or `go-git`) to fetch the source code.
- `gitops_ssh_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):

- `git_implementation`: the same as `gitImplementation` workload parameter
- `gitops.ssh_secret`: the same as `gitops_ssh_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
```

The `tanzu` CLI is used for creating workloads that define Maven artifacts as source.

To create a workload that defines a specific version of a maven artifact as source, 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 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/sample-accelerators/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 Dockerfile-based builds with Supply Chain Choreographer

This topic explains how you can use Dockerfile-based builds with Supply Chain Choreographer.

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.

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, and so on to a build)	`- --build-arg=FOO=BAR`

For example, assuming that you want to build a container image our of a repository named `github.com/foo/bar` whose 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 foo \
  --git-repo https://github.com/foo/bar \
  --git-branch dev \
  --param dockerfile=./Dockerfile

  Create workload:
        1 + |---
        2 + |apiVersion: carto.run/v1alpha1
        3 + |kind: Workload
        4 + |metadata:
        5 + |  name: foo
        6 + |  namespace: default
        7 + |spec:
        8 + |  params:
        9 + |  - name: dockerfile
       10 + |    value: ./Dockerfile
       11 + |  source:
       12 + |    git:
       13 + |      ref:
       14 + |        branch: dev
       15 + |      url: https://github.com/foo/bar
```

Similarly, if the context to be used for the build must be set to a different directory within the repository, you can make use of the `docker_build_context` to change that:

```
$ tanzu apps workload create foo \
  --git-repo https://github.com/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 `tap-values.yaml`, 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.

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 both HTTP(s) and SSH, do not use the same server for multiple secrets to avoid a Tekton error.

HTTP

For any action upon an HTTP(s)-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.

After you create the secret, attach it to the `ServiceAccount` configured for the workload by including it in its set of secrets. 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
```

SSH

Aside from using HTTP(S) as a transport, the supply chains also allow you to use SSH.

> **Important**
>
> If you want to use the pull request feature, you must use HTTP(S) 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
   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
```

6. After you create the secret, attach it to the ServiceAccount configured for the workload by including it in its set of secrets. 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
```

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 an 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: pivotal
     - name: gitops_repository_name
       value: kpack
```

Resulting gitops repository: `https://github.com/pivotal/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: pivotal
     - 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
  params:
      - 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/sample-accelerators/tanzu-java-web-app \
  --git-branch main \
  --param gitops_ssh_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/pivotal/kpack
```

Resulting gitops repository: `https://github.com/vmware-tanzu/cartographer` (The deprecated param `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: pivotal
    - 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

Authentication

The pull request approach requires HTTP(S) authentication with a token.

Pull request functionality is not a part of the Git specification, but is included by most Git server providers. You must authenticate with those providers using a token.

In the Kubernetes secret that holds the Git credentials, the password text box must be filled with a token. When generating a token, ensure that it is given proper scope:

For HTTP(S) authentication, use one of the following tokens:

- 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.
- `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
```

And 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-auth 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  # `git-ssh` is the default name.
                         #   - operators can change such default by using the
                         #     `gitops.ssh_secret` property in `tap-values.yaml`
                         #   - developers can override by using the workload parameter
                         #     named `gitops_ssh_secret`.
  annotations:
    tekton.dev/git-0: GIT-SERVER        # ! required
type: kubernetes.io/basic-auth          # ! required
stringData:
  username: GIT-USERNAME
  password: GIT-PASSWORD
```

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.

Note: If you want to use the pull request approach, the password field must be filled with a token. See the Pull Requests section for more information.

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  # `git-ssh` is the default name.
                        #   - operators can change such default through the
                        #     `gitops.ssh_secret` property in `tap-values.yaml`
                        #   - developers can override by using the workload parameter
                        #     named `gitops_ssh_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
```

For more 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 after `$(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/sample-accelerators/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/sample-accelerators/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_ssh_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: `ssh-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`

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.

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 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` field set
  - ClusterSupplyChain/**source-to-url**
    - label `apps.tanzu.vmware.com/workload-type: web`
- *ootb-supply-chain-testing*
  - ClusterSupplyChain/**testing-image-to-url**
    - label `apps.tanzu.vmware.com/workload-type: web`
    - `workload.spec.image` field 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` field 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.

Currently, 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**
- ClusterRole/**ootb-templates-app-viewer**
- ClusterRole/**ootb-templates-deliverable**
- ClusterRole/**ootb-templates-workload**
- ClusterRunTemplate/**tekton-source-pipelinerun**
- ClusterRunTemplate/**tekton-taskrun**
- ClusterSourceTemplate/**delivery-source-template**
- ClusterSourceTemplate/**source-scanner-template**
- ClusterSourceTemplate/**source-template**
- ClusterSourceTemplate/**testing-pipeline**
- ClusterTask/**git-writer**
- ClusterTask/**image-writer**
- ClusterTemplate/**config-writer-template**
- ClusterTemplate/**deliverable-template**

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 desired supply chain 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've 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 result in 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 currently 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.

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 - Scan, you can build and deploy secure, trusted software that complies with your corporate security requirements. Supply Chain Security Tools - 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.

Use cases

The following use cases apply to Supply Chain Security Tools - Scan:

- Use your scanner as a plug-in, 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 Supply Chain Security Tools - Store from where they are queried.

Supply Chain Security Tools - Scan features

The following Supply Chain Security Tools - Scan features enable the Use cases:

- Kubernetes controllers to run scan jobs.
- Custom Resource Definitions (`CRD`s) 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, it is important to remember that there are limitations present that impact its efficacy. The following examples illustrate the limitations that are prevalent in most scanners today:

Missed CVEs

One limitation 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 just 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 subcomponent 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 limitations:

- Scan more continuously and comprehensively to identify and remediate zero-day vulnerabilities quicker. Comprehensive scanning can be achieved by:
    - scanning earlier in the development cycle to ensure issues can be addressed 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 includes the ability to 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 Java and Node.js based projects. Since 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.

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 - Scan, you can build and deploy secure, trusted software that complies with your corporate security requirements. Supply Chain Security Tools - 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.

Use cases

The following use cases apply to Supply Chain Security Tools - Scan:

- Use your scanner as a plug-in, 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 Supply Chain Security Tools - Store from where they are queried.

Supply Chain Security Tools - Scan features

The following Supply Chain Security Tools - Scan features enable the Use cases:

- Kubernetes controllers to run scan jobs.
- Custom Resource Definitions (`CRD`s) 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, it is important to remember that there are limitations present that impact its efficacy. The following examples illustrate the limitations that are prevalent in most scanners today:

Missed CVEs

One limitation 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 just 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 subcomponent 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 limitations:

- Scan more continuously and comprehensively to identify and remediate zero-day vulnerabilities quicker. Comprehensive scanning can be achieved by:
    - scanning earlier in the development cycle to ensure issues can be addressed 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 includes the ability to 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 Java and Node.js based projects. Since 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 Supply Chain Security Tools - Scan. For more information about profiles, see Components and installation profiles.

Prerequisites

Before installing Supply Chain Security Tools - 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 Supply Chain Security Tools - Store can be handled in:
    - **Single Cluster:** The Supply Chain Security Tools - Store is present in the same cluster where Supply Chain Security Tools - Scan and the `ScanTemplates` will be present.

- **Multi-Cluster:** The Supply Chain Security Tools - Store is present in a different cluster (e.g.: view cluster) where the Supply Chain Security Tools - Scan and `ScanTemplates` will be present.
- **Integration Deactivated:** The Supply Chain Security Tools - Scan deployment doesn't need to communicate with Supply Chain Security Tools - Store.

For information about SCST - Store, see Using the Supply Chain Security Tools - Store.

Install

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 both the Grype scanner and the Scan controller are installed.

To install Supply Chain Security Tools - Scan (Scan controller):

1. List version information for the package by running:

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

   For example:

   ```
   $ tanzu package available list scanning.apps.tanzu.vmware.com --namespace tap-install
   / 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 `v1.5.0 and later` or other supported scanners included with Tanzu Application Platform `v1.5 and later` and do not want to use the default SCST - Store integration, explicitly 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: ""
   ```

   If you're using the Grype Scanner `<1.2.0`, the scanning configuration needs to configure the store parameters. See the v1.1 docs for reference.

   You can retrieve any other configurable setting using the following command, and appending the key-value pair to the previous `scan-values.yaml` file:

   ```
   tanzu package available get scanning.apps.tanzu.vmware.com/VERSION --values-schema -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-name 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 Supply Chain Security Tools - Scan (Grype scanner):

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:

   You need to define the configuration for the Supply Chain Security Tools - Store integration in the `grype-values.yaml` file for the Grype Scanner:

   ```
   ---
   namespace: "DEV-NAMESPACE" # The developer namespace where the ScanTemplates ar
   e gonna 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 can be 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.

You can retrieve any other configurable setting using the following command, and appending 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.
```

3. Install the package by running:

```
tanzu package install grype-scanner \
  --package-name 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-name 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'
```

Upgrading 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 Supply Chain Security Tools - Scan by following the instructions in Install Supply Chain Security Tools - Scan.

This topic includes instructions for:

- Upgrading Supply Chain Security Tools - Scan
  - Prerequisites
  - General Upgrades for Supply Chain Security Tools - Scan
  - Upgrading to Version v1.2.0

Prerequisites

Before you upgrade Supply Chain Security Tools - Scan:

- Upgrade the Tanzu Application Platform by following the instructions in Upgrading Tanzu Application Platform

General Upgrades for Supply Chain Security Tools - Scan

When you're upgrading to any version of Supply Chain Security Tools - Scan these are some factors to accomplish this task successfully:

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.

Upgrading 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 Supply Chain Security Tools - 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.

   Single Cluster Deployment

   Edit the `tap-values.yaml` file you used to deploy Supply Chain Security Tools - Store to export the ca secret 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.

   Update Tanzu Application Platform to apply the changes:

   ```
   tanzu package installed update tap -f tap-values.yaml -n tap-install
   ```

   Multi-Cluster Deployment

   You must reapply the SecretExport by changing the toNamespace: scan-link-system to Namespace: `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 gonna 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).

   Now update Tanzu Application Platform to apply the changes:

```
tanzu package installed update tap -f tap-values.yaml -n tap-install
```

3. 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`, See the Release Notes.

4. 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.

   Create the `verify-upgrade.yaml` file in your system with the following content:

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: scanpolicy-sample
  labels:
    'app.kubernetes.io/part-of': 'component-a'
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "UnknownSeverity"
    notAllowedSeverities := ["Critical", "High"]
    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 := input.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 := vuln.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: scanpolicy-sample
```

   Deploy the resources

```
kubectl apply -f verify-upgrade.yaml -n <DEV-NAMESPACE>
```

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`.

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.

Prerequisites

Before installing Supply Chain Security Tools - Scan (Snyk Scanner):

- Install Supply Chain Security Tools - Scan. It must be present on the same cluster. The prerequisites for Scan are also required.
- Obtain a Snyk API Token from the Snyk Docs.

Install

To install Supply Chain Security Tools - Scan (Snyk scanner):

1. List version information for the package by running:

   ```
   tanzu package available list snyk.scanning.apps.tanzu.vmware.com --namespace ta
   p-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
   ```

2. (Optional) Make changes to the default installation settings by running:

   ```
   tanzu package available get snyk.scanning.apps.tanzu.vmware.com/VERSION --value
   s-schema -n tap-install
   ```

   Where `VERSION` is your package version number. For example, `1.0.0`.

   For example:

   ```
   $ tanzu package available get snyk.scanning.apps.tanzu.vmware.com/1.0.0 --value
   s-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.
```

3. Create a Snyk secret YAML file and insert the Snyk API token (base64 encoded) into the `snyk_token` key as follows:

```
apiVersion: v1
kind: Secret
metadata:
  name: snyk-token-secret
  namespace: my-apps
data:
  snyk_token: BASE64-SNYK-API-TOKEN
```

4. 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.

5. 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 Snyk 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 document.

```
---
namespace: DEV-NAMESPACE
targetImagePullSecret: TARGET-REGISTRY-CREDENTIALS-SECRET
snyk:
  tokenSecret:
    name: SNYK-TOKEN-SECRET
```

   - `DEV-NAMESPACE` is your developer namespace.

     Note: To use a namespace other than the default namespace, ensure 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. If built images are pushed to the same registry as the Tanzu Application Platform images, you can reuse the `tap-registry` secret created earlier in Add the Tanzu Application Platform package repository for this field.
   - `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 Supply Chain Security Tools - Store integration. The `values.yaml` file is slightly different for each configuration.

Using Supply Chain Security Tools - Store Integration: To persist the results found by the Snyk Scanner, you can enable the Supply Chain Security Tools - 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 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` Snyk Scanner is installed:

  ```
  #! ...
  metadataStore:
   #! The url where the Store deployment is accesible.
   #! 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 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 the 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 accesible.
   #! 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>"
     #! 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 Store Deployment.
     name: "<AUTH-SECRET-NAME>"
     #! The namespace where the secrets for the Store Deployment live.
     importFromNamespace: "<STORE-SECRETS-NAMESPACE>"
  ```

Without Supply Chain Security Tools - Store Integration: If you don't want to enable the Supply Chain Security Tools - Store integration, explicitly deactivate the integration by appending the next fields to the `values.yaml` file, since it's enabled by default:

```
# ...
metadataStore:
  url: "" # Configuration is moved, so set this string to empty.
```

6. Install the package by running:

   ```
   tanzu package install snyk-scanner \
     --package-name snyk.scanning.apps.tanzu.vmware.com \
     --version VERSION \
     --namespace tap-install \
     --values-file values.yaml
   ```

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:

~~~
```console
$ tanzu package install snyk-scanner \
  --package-name 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 integration with Snyk

To verify the integration with Snyk, apply the following `ImageScan` and its `ScanPolicy` in the developer namespace and review the result.

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': 'component-a'
   spec:
     regoFile: |
       package main

       notAllowedSeverities := ["Low"]
       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("%s %s %s", [comp, vuln.id, ratings])
    }
```

2. Apply the YAML file by running:

   ```
   kubectl apply -n $DEV_NAMESPACE -f <SCAN-POLICY-YAML>
   ```

3. Create the following ImageScan YAML:

   ```
   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. Apply the earlier created YAML:

   ```
   kubectl apply -n $DEV_NAMESPACE -f <IMAGE-SCAN-YAML>
   ```

5. To verify the integration, run:

   ```
   kubectl get imagescan sample-snyk-public-image-scan -n $DEV_NAMESPACE
   ```

   For example:

   ```
   kubectl get imagescan sample-snyk-public-image-scan -n $DEV_NAMESPACE
   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
   ```

6. Cleanup:

   ```
   kubectl delete imagescan sample-snyk-public-image-scan -n $DEV_NAMESPACE
   ```

Configure Supply Chains

In order to scan your images with Snyk instead of the default Grype scanner 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 later to your `tap-values.yaml` and perform a Tanzu Application Platform update.

```
ootb_supply_chain_testing_scanning:
  scanning:
    image:
      template: snyk-private-image-scan-template
      policy: snyk-scan-policy
```

Note: The Snyk Scanner integration is only available for an image scan, not a source scan.

Opt-out of using Snyk

You can opt out of using Snyk for either a specific supply chain or for all of Tanzu Application Platform.

Opt-out of Snyk for a Supply Chain

To opt-out of Snyk for a specific Supply Chain, reconfigure the supply chain to use another scanner:

- Edit the `ootb_supply_chain_testing_scanning.scanning.image.template` value to use a scan template that does not use Snyk, such as Grype.

  ```
  ootb_supply_chain_testing_scanning:
    scanning:
      image:
        template: "ALTERNATIVE-SCAN-TEMPLATE"
        policy: scan-policy
  ```

Opt-out of Snyk Entirely

To opt-out of Snyk for all of Tanzu Application Platform:

1. To uninstall Snyk, run:

   ```
   tanzu package installed delete snyk-scanner \
   --namespace tap-install
   ```

2. Follow the Opt-out of Snyk for a specific Supply Chain for all Supply Chains in the environment to not use Snyk and use another scanner such as Grype.

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.

CR Name	Use Case
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': 'component-a'
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "UnknownSeverity"
    notAllowedSeverities := ["Critical"]
    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 := input.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 := vuln.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-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 DEV-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-NAMESPACE
```

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': 'component-a'
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "UnknownSeverity"
    notAllowedSeverities := ["Critical"]
    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 := input.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 := vuln.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: 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.

3. Update the `tap-values.yaml` file to include the name of secret created earlier.

```
grype:
namespace: "MY-DEV-NAMESPACE"
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: v1
kind: Secret
metadata:
  name: image-secret
type: kubernetes.io/dockerconfigjson
data:
  .dockerconfigjson: <~/.docker/config.json base64 data>

---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ImageScan
metadata:
  name: sample-image-source-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:
   namespace: "MY-DEV-NAMESPACE"
   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.

Using Grype in offline and air-gapped environments for Supply Chain Security Tools - Scan

This topic tells you how to use Grype in offline and air-gapped environments for Supply Chain Security Tools (SCST) - Scan.

The `grype` CLI attempts to perform two over the Internet calls: one to verify for later versions of the CLI and another to update the vulnerability database before scanning.

You must deactivate both of these external calls. For the `grype` CLI to function in an offline or air-gapped environment, the vulnerability database must be hosted within the environment. You must configure the `grype` CLI with the internal URL.

The `grype` URL accepts environment variables to satisfy these needs.

For information about setting up an offline vulnerability database, see the Anchore Grype README in GitHub.

Overview

To enable Grype in offline air-gapped environments:

1. Create ConfigMap
2. Create Patch Secret
3. Configure tap-values.yaml to use `package_overlays`
4. Update Tanzu Application Platform

Use Grype

To use Grype in offline and air-gapped environments:

1. Create a ConfigMap that contains the public ca.crt to the file server hosting the Grype database files. Apply this ConfigMap to your developer namespace.

2. Create a secret that contains the ytt overlay to add the Grype environment variables to the ScanTemplates.

```
apiVersion: v1
kind: Secret
metadata:
  name: grype-airgap-overlay
  namespace: tap-install #! namespace where tap is installed
stringData:
  patch.yaml: |
    #@ load("@ytt:overlay", "overlay")

    #@overlay/match by=overlay.subset({"kind":"ScanTemplate","metadata":{"names
pace":"<DEV-NAMESPACE>"}}),expects="1+"
    #! developer namespace you are using
    ---
    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"
              - name: GRYPE_DB_AUTO_UPDATE
                value: "true"
              - name: GRYPE_DB_UPDATE_URL
                value: <INTERNAL-VULN-DB-URL> #! url points to the internal fil
e server
              - name: GRYPE_DB_CA_CERT
                value: "/etc/ssl/certs/custom-ca.crt"
              - name: GRYPE_DB_MAX_ALLOWED_BUILT_AGE #! see note on best practi
ces
                value: "120h"
            volumeMounts:
              #@overlay/append
              - name: ca-cert
                mountPath: /etc/ssl/certs/custom-ca.crt
                subPath: <INSERT-KEY-IN-CONFIGMAP> #! key pointing to ca certif
icate
        volumes:
        #@overlay/append
        - name: ca-cert
          configMap:
            name: <CONFIGMAP-NAME> #! name of the configmap created
```

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. Stale databases weaken your security posture. VMware reccomends updating the database daily. You can use the `GRYPE_DB_MAX_ALLOWED_BUILT_AGE` parameter to override the default in accordance with your security posture.

You can also add more certificates to the ConfigMap created earlier, to handle connections to a private registry for example, and mount them in the `volumeMounts` section if needed.

For example:

```
#! ...
volumeMounts:
  #@overlay/append
  #! ...
  - name: ca-cert
    mountPath: /etc/ssl/certs/another-ca.crt
    subPath: another-ca.cert #! key pointing to ca certificate
```

If you have more than one developer namespace and you want to apply this change to all of them, change the `overlay match` on top of the patch.yaml to the following:

```
#@overlay/match by=overlay.subset({"kind":"ScanTemplate"}),expects="1+"
```

3. [Optional] If Grype was installed by using a Tanzu Application Platform profile, you can skip to the next step.

If Grype was installed manually, you must update your `PackageInstall` to include the annotation to reference the overlay `Secret`.

```
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
name: grype
namespace: tap-install
annotations:
  ext.packaging.carvel.dev/ytt-paths-from-secret-name.0: grype-airgap-overlay
...
```

For more information, see Customize package installation.

1. Configure tap-values.yaml to use `package_overlays`. Add the following to your tap-values.yaml:

   ```
   package_overlays:
    - name: "grype"
      secrets:
         - name: "grype-airgap-overlay"
   ```

2. Update Tanzu Application Platform

   ```
   tanzu package installed update tap -f tap-values.yaml -n tap-install
   ```

```
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’s 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 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. First confirm that the required database schema for the installed Grype version is being used. Next, 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/releases/vulnerability-db_v5_2023-02-08T
08_17_20Z_6ef73016d160043c630f.tar.gz",
            "checksum": "sha256:aab8d369933c845878ef1b53bb5c26ee49b91ddc5cd87c9eb57ffb
203a88a72f"
    }]
  }
}
```

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.

1. The `built` parameters in the `listing.json` file are incorrectly formatted. The proper format is `yyyy-MM-ddTHH:mm:ssZ`.
2. The `url` which you modified to point at an internal endpoint is not reachable from within the cluster. For information about verifying connectivity, see Debug Grype database in a cluster.

Debug Grype database in a cluster

1. Describe the failed source or image scan to determine verify the name of the ScanTemplate being used:

```
kubectl describe sourcescan/imagescan SCAN-NAME -n DEV-NAMESPACE
```

Where `SCAN-NAME` is the name of the source/image scan that failed.

1. 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
   ```

2. Re-run the scan.
3. Get the name of the `scan-plugin` pod.

   ```
   kubectl get pods -n DEV-NAMESPACE
   ```

4. Get a shell to the container. See the Kubernetes documentation.

   ```
   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.

5. Inside the container, run Grype CLI commands to report database status and verify connectivity from cluster to mirror. See the Anchore Grype documentation in GitHub.

   - Report current status of Grype's database, such as location, build date, and checksum:

     ```
     grype db status
     ```

   - Download the listing file configured at `db.update-url` and show databases that are available for download:

```
grype db list
```

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. Enabling CVE scan causes Supply Chain Choreographer by using Tanzu Application Platform GUI to visualize your supply chain, including the scans, scan policy, and CVEs. See Enable CVE scan results. You can also use the Tanzu Insight plug-in to query packages and CVEs using a CLI. See Tanzu 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 Kubernetes Job where the Job creates a pod. Both the Job and pod are cleaned up after completion.

Before applying a new scan, you can set a watch on the Jobs, Pods, SourceScans, 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 SCST - Scan.

Debugging commands

Run these commands to get more logs and details about the errors around scanning. The Jobs and pods persist for a predefined amount of seconds before getting deleted. (deleteScanJobsSecondsAfterFinished is the tap pkg variable that defines this)

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 <sourcescan> -n <DEV-NAMESPACE>
```

Where `DEV-NAMESPACE` is the name of the developer namespace you want to use.

```
kubectl describe imagescan <imagescan> -n <DEV-NAMESPACE>
```

Where `DEV-NAMESPACE` is the name of the developer namespace 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 Multicluster setup.

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 Multicluster setup.

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.

   You can confirm this by base64 decoding both secrets and seeing if 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

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 job fails with the error as follows:

```
Job.batch "scan-${app}-${id}" is invalid: [spec.template.spec.volumes[2].secret.secret
Name: Required value, spec.template.spec.containers[0].volumeMounts[2].name: Not foun
d: "registry-cred"]
```

Deactivate Supply Chain Security Tools - 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-name 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:

- (Preferred method) Install a version of Tanzu Build Service that provides an SBOM with a compatible Syft Schema Version.
- 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 Verify integration with Snyk 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"}
```

Include the following in your `tap-values.yaml`:

```
metadata_store:
  ns_for_export_app_cert: "<DEV-NAMESPACE>"
```

However, if the earlier tap-values.yaml doesn't work, include:

```
metadata_store:
  ns_for_export_app_cert: "*"
```

Blob Source Scan is reporting wrong source URL

A Source Scan for a blob artifact can result in 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 Supply Chain Security Tools - 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 a Supply Chain that is blocked by failing scans

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 Application Platform GUI

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 for more details.

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
```

This error is caused by a connection error while attempting to connect to the local cluster URL. 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 Install Build profile.

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 shows this:

```
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
```

One possible reason is due to using Grype Scanner ScanTemplates shipped with versions before Supply Chain Security Tools - Scan v1.2.0 which are now deprecated and are no longer supported in v1.4.0+.

The two options to resolve this issue are:

1. Upgrade Grype Scanner to v1.2+ (preferably latest). This will automatically replace the old ScanTemplates with the upgraded ScanTemplates.
2. Create a ScanTemplate using this steps.
3. Create a ScanTemplate. Follow the steps in Create a scan template.

Incorrectly configured self-signed cert

If the pod logs show the following error:

```
x509: certificate signed by unknown authority
```

This indicates that the self-signed certificate might be incorrectly configured.

The `shared.ca_cert_data` installation value can contain a PEM-encoded CA bundle. The scanning component trusts the CAs contained in the bundle. You configure the self-signed certificate by using the shared top-level key.

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 were deprecated in Tanzu Application Platform v1.4.0. Previously these text boxes might be used to populate the `registry-credentials` secret. If you encounter the following error during installation:

```
UNAUTHORIZED: unauthorized to access repository
```

The recommended migration path for users who are 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

Prior to running a scan, the Grype scanner downloads a copy of its database. If the database fails to download, the following log message 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 have set up a mirror of the vulnerability database, 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.

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 a `ScanTemplate` to be defined. Run `kubectl get scantemplates` for the ScanTemplates provided with the scanner installation. These can be referenced, or see How to create a ScanTemplate.

Deploy scan custom resources

The scan controller defines two custom resources to create scanning jobs:

- 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: .tar.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 information.
    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. 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 `SourceScan` custom resource to the desired namespace on cluster by running:

   ```
   kubectl apply -f <path_to_the_cr>/<custom_resource_filename>.yaml -n <desired_namespace>
   ```

   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>component>*` 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" section to learn more.
     conditions: []

     # The URL of the vulnerability scan results in the Metadata Store integration.
     # 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 registry.
       # 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_namespace>
   ```

   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>metadata>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" section to learn more.
     conditions: []

     # The URL of the vulnerability scan results in the Metadata Store integration.
     # 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 a `ScanTemplate` to be defined. Run `kubectl get scantemplates` for the ScanTemplates provided with the scanner installation. These can be referenced, or see How to create a ScanTemplate.

Deploy scan custom resources

The scan controller defines two custom resources to create scanning jobs:

- 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. 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 `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: scanpolicy-sample
  labels:
    'app.kubernetes.io/part-of': 'component-a'
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "Unkn
ownSeverity"
    notAllowedSeverities := ["Low"]
    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 modify the following fields 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"]
    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.

Enable Tanzu Application Platform GUI to view ScanPolicy Resource

In order for the Tanzu Application Platform GUI 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 Application Platform GUI:

```
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: scanpolicy-sample
  labels:
    'app.kubernetes.io/part-of': 'component-a'
spec:
  regoFile: |
    ...
```

> **Note**
>
> The value for the label can be anything. 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': 'component-a'
spec:
  regoFile: |
    package policies

    default isCompliant = false

    ignoreSeverities := ["Low","Medium","High","Critical"]

    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
```

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.

Understanding 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 the execution of the scanning job. The Status field indicates whether the scan is still running or has already finished (i.e., if `Status: True`, the scan job is still running; if `Status: False`, the scan is done).

The Reason field is `JobStarted` while the scanning is running and `JobFinished` when it is done.

The Message field can 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 job result. The Status field indicates whether the scan finished successfully or if it encountered an error (i.e., 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 job.

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 may also indicate that the metadata store integration has not been configured or that there was an error sending. An error would usually be a misconfigured metadata store url or that the metadata store is inaccessible. Check the installation steps to ensure 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.

Understanding 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`.

Understanding 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.

Understanding 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 job is running.
- `Completed`: scan completed and no CVEs were found that violated the scanpolicy.
- `Failed`: scan completed but CVEs were found that violated the scan policy.
- `Error`: indication of an error (e.g., an invalid scantemplate or scanpolicy).

Note: The PHASE print column also shows this with `kubectl get` on a `SourceScan` or `ImageScan`.

Understanding ScannedBy

The `status.scannedBy` field is populated with the name, vendor, and scanner version that generates the security assessment report.

Understanding ScannedAt

The `status.scannedAt` field is populated with the latest date when the scanning was successfully finished.

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.

To Install Supply Chain Security Tools - Policy Controller, see Install Supply Chain Security Tools - Policy Controller

Known issues

Tanzu Application Platform v1.2.2 and earlier

`kubectl run` **pods fail to validate in non-default namespaces:** When policy verification occurs on an image deployed through `kubectl run` on a non-default namespace, the verification fails to create the keychain required if the image requires credentials.

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.

To Install Supply Chain Security Tools - Policy Controller, see Install Supply Chain Security Tools - Policy Controller

Known issues

Tanzu Application Platform v1.2.2 and earlier

`kubectl run` **pods fail to validate in non-default namespaces:** When policy verification occurs on an image deployed through `kubectl run` on a non-default namespace, the verification fails to create the keychain required if the image requires credentials.

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.

- If Supply Chain Security Tools - Sign is installed with an existing running Image Policy Webhook `ClusterImagePolicy`, see Migration From Supply Chain Security Tools - Sign.
- During configuration for this component, you are asked to provide a cosign public key to use 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 to generate your own keys and sign an image.

Caution: This component WILL REJECT `pods` if it is 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  1.0.0          2022-06-02 20:00:00 -0400 EDT
  policy.apps.tanzu.vmware.com  1.0.1          2022-06-08 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, `1.0.1`.

   For example:

```
$ tanzu package available get policy.apps.tanzu.vmware.com/1.0.1 --values-schem
a --namespace tap-install
| Retrieving package details for policy.apps.tanzu.vmware.com/1.0.1...

KEY                   DEFAULT        TYPE     DESCRIPTION
custom_ca_secrets     <nil>          array    List of custom CA secrets that sh
ould be included in the application container
                                              for registry communication. An ar
ray of secret references each containing a
                                              secret_name field with the secret
name to be referenced and a namespace field
                                              with the name of the namespace wh
ere the referred secret resides.
custom_cas            <nil>          array    List of custom CA contents that s
hould be included in the application container
                                              for registry communication. An ar
ray of items containing a ca_content field with
                                              the PEM-encoded contents of a cer
tificate authority.
requests_cpu          20m            string   The CPU request defines the minim
um CPU time for the Policy
                                              Controller manager. During CPU co
ntention, CPU request is used as
                                              a weighting where higher CPU requ
ests are allocated more CPU time.
                                              https://kubernetes.io/docs/concep
ts/configuration/manage-resources-containers/#meaning-of-cpu

deployment_namespace  cosign-system  string   Deployment namespace specifies th
e namespace where this component should be
                                              deployed to. If not specified, "c
osign-system" is assumed.
limits_cpu            200m           string   The CPU limit defines a hard ceil
ing on how much CPU time
                                              that the Policy Controller manage
r container can use.
                                              https://kubernetes.io/docs/concep
```

```
ts/configuration/manage-resources-containers/#meaning-of-cpu

limits_memory         200Mi          string   The memory limit defines a hard c
eiling on how much memory
                                              that the Policy Controller manage
r container can use.
                                              https://kubernetes.io/docs/concep
ts/configuration/manage-resources-containers/#meaning-of-memory

quota.pod_number      6              string   The maximum number of Policy Cont
roller Pods allowed to be created with the
                                              priority class system-cluster-cri
tical. This value must be enclosed in quotes
                                              (""). If this value is not specif
ied then a default value of 6 is used.
replicas              1              integer  The number of replicas to be crea
ted for the Policy Controller. This value must
                                              not be enclosed in quotes. If thi
s value is not specified then a default value
                                              of 1 is used.
requests_memory       20Mi           string   The memory request defines the mi
nium memory amount for the Policy Controller manager.
                                              https://kubernetes.io/docs/concep
ts/configuration/manage-resources-containers/#meaning-of-memory
```

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 fields: the `secret_name` field defines the name of the secret, and the `namespace` field 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 field named `ca_content`. The value of this field 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 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 for more details.

- `limits_memory`: This setting controls the maximum memory resource allocated to the Policy admission controller. The default value is “200Mi”. See Kubernetes documentation for more details.

- `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 field 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 limitations.

- `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 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 Kubernetes documentation for more details.

- `requests_memory`: This setting controls the minimum memory resource allocated to the Policy admission controller. The default value is “20Mi”. See Kubernetes documentation for more details.

4. Install the package:

```
tanzu package install policy-controller \
  --package-name 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, `1.0.1`.

   For example:

```
$ tanzu package install policy-controller \
    --package-name policy.apps.tanzu.vmware.com \
    --version 1.0.1 \
    --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.

Migration From Supply Chain Security Tools - Sign

This topic explains how you can migrate from Supply Chain Security Tools - Sign to Supply Chain Security Tools - Policy. For more information about additional features introduced in Policy Controller, see Configuring Supply Chain Security Tools - Policy.

Note: There is currently no equivalent of “AllowUnmatchedImages” VMware recommends that users sign public images and have the signature stored in their own repository that is referenced by specifying a source in the `ClusterImagePolicy` authorities.

Add Policy Controller Namespace to Image Policy Webhook

If there is an active Image Policy Webhook `ClusterImagePolicy`, it prevents Policy Controller from deploying. To ensure that Policy Controller deploys, update the Image Policy Webhook `ClusterImagePolicy` by adding `cosign-system` to the excluded namespaces. If an alternative `deployment_namespace` is specified for installing Policy Controller, exclude that namespace. For more information about how to exclude namespaces, see Configuring Supply Chain Security Tools - Sign

Enable Policy Controller on Namespaces

Policy Controller works with an opt-in system. Operators must update namespaces with the label `policy.sigstore.dev/include: "true"` to the namespace resource to enable Policy Controller verification.

```
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 can be 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.

Policy Controller ClusterImagePolicy

The Policy Controller `ClusterImagePolicy` does not have a name requirement. Image Policy Controller required that the `ClusterImagePolicy` be named `image-policy` and that there be only one `ClusterImagePolicy`. Multiple Policy Controller `ClusterImagePolicies` are applied. During validation, all `ClusterImagePolicy` that have an image `glob` pattern that matches the deploying image is evaluated. All matched `ClusterImagePolicies` must be valid. For a `ClusterImagePolicy` to be valid, at least one authority in the policy must successfully validate the signature of the deploying image.

Excluding Namespaces

The namespaces listed in `spec.verification.exclude.resources.namespaces[]` must have `policy.sigstore.dev/include` set to `false` or not be set. Therefore, they are exempted from Policy Controller validation.

Image Policy Webhook:

```
---
apiVersion: signing.apps.tanzu.vmware.com/v1beta1
kind: ClusterImagePolicy
metadata:
  name: image-policy
spec:
  verification:
    ...

    exclude:
      resources:
        namespaces:
        - image-policy-system
        - kube-system
        - cert-manager

    ...
```

Specifying Public Keys

`spec.verification.keys[].publicKey` from Image Policy Webhook is mapped to `spec.authorities[].key.data` for Policy Controller.

The `name` associated with each `key` is no longer required. Image Policy Webhook has direct association between `key` name and `imagePattern`. For Policy Controller, multiple `ClusterImagePolicy` resources are defined to create direct association between image patterns and key authorities.

Image patterns and keys are scoped to each `ClusterImagePolicy` resource.

Therefore, to have direct association be isolated between `key` and `imagePattern`, multiple Policy Controller `ClusterImagePolicy` must be created. Each `ClusterImagePolicy` has the image glob pattern defined and the associated key authorities defined.

Image Policy Webhook:

```
---
apiVersion: signing.apps.tanzu.vmware.com/v1beta1
kind: ClusterImagePolicy
metadata:
  name: image-policy
spec:
  verification:
    ...

    keys:
    - name: official-cosign-key
      publicKey: |
        -----BEGIN PUBLIC KEY-----
        MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEhyQCx0E9wQWSFI9ULGwy3BuRklnt
        IqozONbbdbqz11hlRJy9c7SG+hdcFl9jE9uE/dwtuwU2MqU9T/cN0YkWww==
        -----END PUBLIC KEY-----

    ...
```

Policy Controller:

```
---
apiVersion: policy.sigstore.dev/v1beta1
kind: ClusterImagePolicy
metadata:
  name: POLICY-NAME
spec:
  authorities:
  ...

  - key:
      data: |
        -----BEGIN PUBLIC KEY-----
        MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEhyQCx0E9wQWSFI9ULGwy3BuRklnt
        IqozONbbdbqz11hlRJy9c7SG+hdcFl9jE9uE/dwtuwU2MqU9T/cN0YkWww==
        -----END PUBLIC KEY-----

  ...
```

Where POLICY-NAME is the name of the cluster image policy you want to use.

Specifying Image Matching

spec.verification.images[].namePattern from Image Policy Webhook maps to spec.images[].glob for Policy Controller.

Policy Controller follows more closely to glob matching. For the Image Policy Webhook, registry.com/* wildcards all projects and images under the registry. However, glob matching uses / separator delimiting. Therefore, the glob wildcard matching equivalent is registry.com/**/*. The ** allows for recursive project path matching while the trailing * images found in the terminating project path.

If only one level of pathing is required, the glob pattern is registry.com/*/*.

Policy Controller also have defaults defined. 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, the glob pattern ** matches against all images.

Image Policy Webhook:

```
---
apiVersion: signing.apps.tanzu.vmware.com/v1beta1
kind: ClusterImagePolicy
metadata:
  name: image-policy
spec:
  verification:
    ...

    images:
    - namePattern: gcr.io/projectsigstore/cosign*
      keys:
      - name: official-cosign-key
      secretRef:
        name: your-secret
        namespace: your-namespace

    ...
```

Policy Controller:

```
---
apiVersion: policy.sigstore.dev/v1beta1
kind: ClusterImagePolicy
metadata:
  name: POLICY-NAME
spec:
  images:
  - glob: gcr.io/projectsigstore/cosign*
```

Configuring 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 all policies with matching image patterns, and where at least one valid signature is obtained from the authorities provided in the matched ClusterImagePolicy later in the topic. Within a single policy, every signature must be valid. When more than one policy has a matching image pattern, the image must match at least one signature from each ClusterImagePolicy.

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 can be 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`: The images block defines the patterns of images that must be subject to the `ClusterImagePolicy`. If multiple policies match a particular image, *ALL* of those policies must be satisfied for the image to be admitted.

  Policy Controller has defaults defined if the following globs are specified:

  - 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, it is important to include `index.docker.io` as the host for the glob.
- `authorities`: The authorities block defines the rules for discovering and validating signatures. Discovery is done by using the `sources` field, and is specified on any entry. Signatures are cryptographically verified using one of the `key` or `keyless` fields.

When a policy is selected to be evaluated against the matched image, the authorities are used to validate signatures. If at least one authority is satisfied and a signature is validated, the policy is validated.

`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 has defaults defined if the following globs are specified: - 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, it is important to include `index.docker.io` as the host for the glob.

A sample of a ClusterImagePolicy which matches against all images using glob:

```
apiVersion: policy.sigstore.dev/v1beta1
kind: ClusterImagePolicy
metadata:
  name: image-policy
spec:
  images:
  - glob: "**"
```

authorities

Authorities listed in the `authorities` block of the ClusterImagePolicy are `key` or `keyless` specifications.

Each `key` authority can contain a PEM-encoded ECDSA public key, a `secretRef`, or a `kms` path.

Important

Only ECDSA public keys are supported.

```
spec:
  authorities:
    - key:
        data: |
          -----BEGIN PUBLIC KEY-----
          ...
          -----END PUBLIC KEY-----
    - key:
        secretRef:
          name: secretName
    - key:
        kms: KMSPATH
```

Each keyless authority can contain a Fulcio URL, a Rekor URL, a certificate, or an array of identities.

```
spec:
  authorities:
    - keyless:
        url: https://fulcio.example.com
        ca-cert:
          data: Certificate Data
      ctlog:
        url: https://rekor.example.com
    - keyless:
        url: https://fulcio.example.com
        ca-cert:
          secretRef:
            name: secretName
    - keyless:
        identities:
          - issuer: https://accounts.google.com
            subject: .*@example.com
          - issuer: https://token.actions.githubusercontent.com
            subject: https://github.com/mycompany/*/.github/workflows/*@*
```

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.

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` field.

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` must be found in the namespace of the deploying Pod resource.

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: mysecret
    - keyless:
        url: https://fulcio.example.com
      source:
        - oci: registry.example.com/project/signature-location
          signaturePullSecrets:
            - name: mysecret
```

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)
```

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-#`.

Supply Chain Security Tools for VMware Tanzu - Sign

Caution: This component is being deprecated in favor of Supply Chain Security Tools - Policy Controller. To migrate from Supply Chain Security Tools - Sign to Supply Chain Security Tools - Policy Controller, please follow these steps

Supply Chain Security Tools - Sign provides an admission WebHook that:

- Verifies signatures on container images used by Kubernetes resources.
- Enforces policy by allowing or denying container images from running based on configuration.
- Adds metadata to verified resources according to their verification status.

It intercepts all resources that create Pods as part of their lifecycle:

- `Pod`s,
- `ReplicaSet`s
- `Deployment`s
- `Job`s
- `StatefulSet`s
- `DaemonSet`s
- `CronJob`s.

This component uses cosign as its backend for signature verification and is compatible only with cosign signatures. When cosign signs an image, it generates a signature in an OCI-compliant format and pushes it to the same registry where the image is stored. The signature is identified by a tag in the format `sha256-<image-digest>.sig`, where `<image-digest>` is the digest of the image that this signature belongs to. The WebHook needs credentials to access this artifact when hosted in a registry protected by authentication.

By default, once installed, this component does not include any policy resources and does not enforce any policy. The operator must create a `ClusterImagePolicy` resource in the cluster before the WebHook can perform any verifications. This `ClusterImagePolicy` resource contains all image patterns the operator wants to verify, and their corresponding cosign public keys.

Typically, the WebHook gets credentials from running resources and their service accounts to authenticate against private registries at admission time. There are other mechanisms that the WebHook uses for finding credentials. For more information about providing credentials, see Providing Credentials for the WebHook.

Supply Chain Security Tools for VMware Tanzu - Sign

Caution: This component is being deprecated in favor of Supply Chain Security Tools - Policy Controller. To migrate from Supply Chain Security Tools - Sign to Supply Chain Security Tools - Policy Controller, please follow these steps

Supply Chain Security Tools - Sign provides an admission WebHook that:

- Verifies signatures on container images used by Kubernetes resources.
- Enforces policy by allowing or denying container images from running based on configuration.
- Adds metadata to verified resources according to their verification status.

It intercepts all resources that create Pods as part of their lifecycle:

- `Pod`s,
- `ReplicaSet`s
- `Deployment`s
- `Job`s
- `StatefulSet`s
- `DaemonSet`s
- `CronJob`s.

This component uses cosign as its backend for signature verification and is compatible only with cosign signatures. When cosign signs an image, it generates a signature in an OCI-compliant format and pushes it to the same registry where the image is stored. The signature is identified by a tag in the format `sha256-<image-digest>.sig`, where `<image-digest>` is the digest of the image that this signature belongs to. The WebHook needs credentials to access this artifact when hosted in a registry protected by authentication.

By default, once installed, this component does not include any policy resources and does not enforce any policy. The operator must create a `ClusterImagePolicy` resource in the cluster before the WebHook can perform any verifications. This `ClusterImagePolicy` resource contains all image patterns the operator wants to verify, and their corresponding cosign public keys.

Typically, the WebHook gets credentials from running resources and their service accounts to authenticate against private registries at admission time. There are other mechanisms that the WebHook uses for finding credentials. For more information about providing credentials, see Providing Credentials for the WebHook.

Install Supply Chain Security Tools - Sign

Caution: This component is being deprecated in favor of Supply Chain Security Tools - Policy Controller. To migrate from Supply Chain Security Tools - Sign to Supply Chain Security Tools - Policy Controller, please follow these steps

Supply Chain Security Tools - Sign is released as part of Tanzu Application Platform's full, iterate and run profiles. Follow the instructions below to manually install this component.

Prerequisites

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- A container image registry that supports TLS connections. This component does not work with insecure registries.
- During configuration for this component, you are asked to provide a cosign public key to use to validate signed images. An example cosign public key is provided that can validate an image from the public cosign registry. If you want to provide your own key and images, follow the cosign quick start guide in GitHub to generate your own keys and sign an image.

Caution: This component rejects pods if the webhook fails or is incorrectly configured. If the webhook is preventing the cluster from functioning, see Supply Chain Security Tools - Sign Known Issues in the Tanzu Application Platform release notes for recovery steps.

Install

Note: v1alpha1 api version of the ClusterImagePolicy is no longer supported as the group name has been renamed from `signing.run.tanzu.vmware.com` to `signing.apps.tanzu.vmware.com`.

To install Supply Chain Security Tools - Sign:

1. List version information for the package by running:

   ```
   tanzu package available list image-policy-webhook.signing.apps.tanzu.vmware.com
   --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package available list image-policy-webhook.signing.apps.tanzu.vmware.c
   om --namespace tap-install
   - Retrieving package versions for image-policy-webhook.signing.apps.tanzu.vmwar
   e.com...
     NAME                                               VERSION        RELEASED-A
   T
     image-policy-webhook.signing.apps.tanzu.vmware.com  1.1.1          2022-03-30
   18:00:00 -0500 EST
   ```

2. (Optional) Make changes to the default installation settings by running:

   ```
   tanzu package available get image-policy-webhook.signing.apps.tanzu.vmware.com/
   VERSION --values-schema --namespace tap-install
   ```

   Where `VERSION` is the version number you discovered. For example, `1.1.1`.

For example:

```
$ tanzu package available get image-policy-webhook.signing.apps.tanzu.vmware.co
m/1.1.1 --values-schema --namespace tap-install
| Retrieving package details for image-policy-webhook.signing.apps.tanzu.vmwar
e.com/1.1.1...
  KEY                     DEFAULT               TYPE     DESCRIPTION
  allow_unmatched_images  false                 boolean  Feature flag for enabli
ng admission of images that do not match any patterns in the image policy confi
guration.
                                                        Set to true to allow im
ages that do not match any patterns into the cluster with a warning.

  custom_ca_secrets       <nil>                 array    List of custom CA secre
ts that should be included in the application container for registry communicat
ion.
                                                        An array of secret refe
rences each containing a secret_name field with the secret name to be reference
d
                                                        and a namespace field w
ith the name of the namespace where the referred secret resides.

  custom_cas              <nil>                 array    List of custom CA conte
nts that should be included in the application container for registry communica
tion.
                                                        An array of items conta
ining a ca_content field with the PEM-encoded contents of a certificate authori
ty.

  deployment_namespace    image-policy-system  string   Deployment namespace sp
ecifies the namespace where this component should be deployed to.
                                                        If not specified, "imag
e-policy-system" is assumed.

  limits_cpu              200m                  string   The CPU limit defines a
hard ceiling on how much CPU time that
                                                        the Image Policy Webhoo
k controller manager container can use.
                                                        https://kubernetes.io/d
ocs/concepts/configuration/manage-resources-containers/#meaning-of-cpu

  limits_memory           256Mi                 string   The memory limit define
s a hard ceiling on how much memory that
                                                        the Image Policy Webhoo
k controller manager container can use.
                                                        https://kubernetes.io/d
ocs/concepts/configuration/manage-resources-containers/#meaning-of-memory

  quota.pod_number        5                     string   The maximum number of I
mage Policy Webhook Pods allowed to be created with the priority class
                                                        system-cluster-critica
l. This value must be enclosed in quotes (""). If this value is not
                                                        specified then a defaul
t value of 5 is used.
  replicas                1                     integer  The number of replicas
to be created for the Image Policy Webhook. This value must not be enclosed
                                                        in quotes. If this valu
e is not specified then a default value of 1 is used.
  requests_cpu            100m                  string   The CPU request defines
the minimum CPU time for the Image Policy
                                                        Webhook controller mana
ger. During CPU contention, CPU request is used
                                                        as a weighting where hi
gher CPU requests are allocated more CPU time.
                                                        https://kubernetes.io/d
ocs/concepts/configuration/manage-resources-containers/#meaning-of-cpu

  requests_memory         50Mi                  string   The memory request defi
nes the minium memory amount for the Image Policy Webhook controller manager.
                                                        https://kubernetes.io/d
ocs/concepts/configuration/manage-resources-containers/#meaning-of-memory
```

3. Create a file named `scst-sign-values.yaml` and add the settings you want to customize:

   - `allow_unmatched_images`:

     - **For non-production environments**: To warn the user when images do not match any pattern in the policy, but still allow them into the cluster, set `allow_unmatched_images` to `true`.

       ```
       ---
       allow_unmatched_images: true
       ```

     - **For production environments**: To deny images that match no patterns in the policy set `allow_unmatched_images` to `false`.

       ```
       ---
       allow_unmatched_images: false
       ```

       Note

       : For a quicker installation process VMware recommends that you set `allow_unmatched_images` to `true` initially. This setting means that the webhook allows unsigned images to run if the image does not match any pattern in the policy. To promote to a production environment VMware recommends that you re-install the webhook with `allow_unmatched_images` set to `false`.

   - `custom_ca_secrets`: This setting controls which secrets to be added to the application container as custom certificate authorities (CAs). It enables communication with registries deployed with self-signed certificates. `custom_ca_secrets` consists of an array of items. Each item contains two fields: the `secret_name` field defines the name of the secret, and the `namespace` field 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` was informed.

   - `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 field named `ca_content`. The value of this field 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 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` was informed.

  - `deployment_namespace`: This setting controls the namespace to which this component is deployed. When not specified, the namespace `image-policy-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 Image Policy Webhook controller. The default value is “200m”. See Kubernetes documentation for more details.
  - `limits_memory`: This setting controls the maximum memory resource allocated to the Image Policy Webhook controller. The default value is “256Mi”. See Kubernetes documentation for more details.
  - `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 field is `5`. If your use case requires more than 5 pods, change this value to allow the number of replicas you intend to deploy.
  - `replicas`: This setting controls the default amount of replicas to be deployed by this component. The default value is `1`.

    For production environments: VMware recommends you increase the number of replicas to `3` to ensure availability of the component and better admission performance.
  - `requests_cpu`: This setting controls the minimum CPU resource allocated to the Image Policy Webhook controller. During CPU contention, this value is used as a weighting where higher values indicate more CPU time is allocated. The default value is “100m”. See Kubernetes documentation for more details.
  - `requests_memory`: This setting controls the minimum memory resource allocated to the Image Policy Webhook controller. The default value is “50Mi”. See Kubernetes documentation for more details.

4. Install the package:

```
tanzu package install image-policy-webhook \
  --package-name image-policy-webhook.signing.apps.tanzu.vmware.com \
  --version VERSION \
  --namespace tap-install \
  --values-file scst-sign-values.yaml
```

Where `VERSION` is the version number you discovered earlier. For example, `1.1.1`.

For example:

```
$ tanzu package install image-policy-webhook \
    --package-name image-policy-webhook.signing.apps.tanzu.vmware.com \
    --version 1.1.1 \
    --namespace tap-install \
    --values-file scst-sign-values.yaml

| Installing package 'image-policy-webhook.signing.apps.tanzu.vmware.com'
| Getting namespace 'default'
| Getting package metadata for 'image-policy-webhook.signing.apps.tanzu.vmware.
com'
| Creating service account 'image-policy-webhook-default-sa'
| Creating cluster admin role 'image-policy-webhook-default-cluster-role'
| Creating cluster role binding 'image-policy-webhook-default-cluster-rolebindi
ng'
| Creating secret 'image-policy-webhook-default-values'
/ Creating package resource
```

```
- Package install status: Reconciling

Added installed package 'image-policy-webhook' in namespace 'tap-install'
```

After you run the commands above your signing package will be running.

Note: This component requires extra configuration steps to work properly. See Configuring Supply Chain Security Tools - Sign for instructions on how to apply the required configuration.

Configure

The WebHook deployed by Supply Chain Security Tools - Sign requires extra input from the operator before it starts enforcing policies.

To configure your installed component properly, see Configuring Supply Chains Security Tools - Sign.

Known issues

See Supply Chain Security Tools - Sign Known Issues.

Configuring Supply Chain Security Tools - Sign

Caution: This component is being deprecated in favor of Supply Chain Security Tools - Policy Controller. To migrate from Supply Chain Security Tools - Sign to Supply Chain Security Tools - Policy Controller, follow these steps

This component requires extra configuration steps to start verifying your container images properly.

The instructions in this section only apply to the deployment namespace of Supply Chain Security Tools - Sign. In most cases, this namespace is rendered as the default namespace `image-policy-system`.

If you deployed Supply Chain Security Tools - Sign by using a customized namespace specified in the installation values file, replace `image-policy-system` with the namespace name that you specified in `deployment_namespace` before performing the configuration steps.

Create a `ClusterImagePolicy` resource

The cluster image policy is a custom resource containing the following properties:

- `spec.verification.keys`: A list of public keys complementary to the private keys that were used to sign the images.
- `spec.verification.images[].namePattern`: Image name patterns that the policy enforces. Each image name pattern maps to the required public keys. (Optional) Use a secret to authenticate the private registry where images and signatures matching a name pattern are stored.
- `spec.verification.exclude.resources.namespaces`: A list of namespaces where this policy is not enforced.

System namespaces specific to your cloud provider may need to be excluded from the policy. VMware also recommends configuring exclusions for Tanzu Application Platform system namespaces. This prevents the Image Policy Webhook from blocking components of Tanzu Application Platform.

To get a list of created namespaces, run:

```
kubectl get namespaces
```

Tanzu Application Platform system namespaces can include:

```
- accelerator-system
- api-portal
- app-live-view
- app-live-view-connector
- app-live-view-conventions
- build-service
- cartographer-system
- cert-injection-webhook
- cert-manager
- conventions-system
- developer-conventions
- flux-system
- image-policy-system
- kapp-controller
- knative-eventing
- knative-serving
- knative-sources
- kpack
- learning-center-guided-ui
- learning-center-guided-w01
- learningcenter
- metadata-store
- scan-link-system
- secretgen-controller
- service-bindings
- services-toolkit
- source-system
- spring-boot-convention
- stacks-operator-system
- tanzu-cluster-essentials
- tanzu-package-repo-global
- tanzu-system-ingress
- tap-gui
- tap-install
- tap-telemetry
- tekton-pipelines
- triggermesh
```

The following is an example `ClusterImagePolicy`:

```
---
apiVersion: signing.apps.tanzu.vmware.com/v1beta1
kind: ClusterImagePolicy
metadata:
    name: image-policy
spec:
  verification:
    exclude:
      resources:
        namespaces:
        - kube-system
        - <TAP system namespaces>
    keys:
    - name: first-key
      publicKey: |
        -----BEGIN PUBLIC KEY-----
        ...
        -----END PUBLIC KEY-----
    images:
    - namePattern: registry.example.org/myproject/*
      keys:
      - name: first-key
    - namePattern: registry.example.org/authproject/*
      secretRef:
        name: secret-name
        namespace: namespace-name
      keys:
      - name: first-key
```

The `name` for the `ClusterImagePolicy` resource must be `image-policy`.

Add any namespaces that run container images that are not signed in the `spec.verification.exclude.resources.namespaces` section, such as the `kube-system` namespace.

If no `ClusterImagePolicy` resource is created, all images are admitted into the cluster with the following warning:

```
Warning: clusterimagepolicies.signing.apps.tanzu.vmware.com "image-policy" not found.
Image policy enforcement was not applied.
```

The patterns are evaluated using the any of operator to admit container images. For each pod, the Image Policy Webhook iterates over the list of containers and init containers. The pod is verified when there is at least one key specified in `spec.verification.images[].keys[]` for each container image that matches `spec.verification.images[].namePattern`.

For a simpler installation process in a non-production environment, use the manifest below to create the `ClusterImagePolicy` resource. This manifest includes a cosign public key which signed the public cosign v1.2.1 image. The cosign public key validates the specified cosign images. Container images running in system namespaces are currently not signed. You must configure the Image Policy Webhook to allow these unsigned images by adding system namespaces to the `spec.verification.exclude.resources.namespaces` section.

```
cat <<EOF | kubectl apply -f -
apiVersion: signing.apps.tanzu.vmware.com/v1beta1
kind: ClusterImagePolicy
metadata:
  name: image-policy
spec:
  verification:
    exclude:
      resources:
        namespaces:
        - kube-system
        - accelerator-system
        - api-portal
        - app-live-view
        - app-live-view-connector
        - app-live-view-conventions
        - build-service
        - cartographer-system
        - cert-injection-webhook
        - cert-manager
        - conventions-system
        - developer-conventions
        - flux-system
        - image-policy-system
        - kapp-controller
        - knative-eventing
        - knative-serving
        - knative-sources
        - kpack
        - learning-center-guided-ui
        - learning-center-guided-w01
        - learningcenter
        - metadata-store
        - scan-link-system
        - secretgen-controller
        - service-bindings
        - services-toolkit
        - source-system
        - spring-boot-convention
        - stacks-operator-system
        - tanzu-cluster-essentials
        - tanzu-package-repo-global
        - tanzu-system-ingress
        - tap-gui
        - tap-install
        - tap-telemetry
        - tekton-pipelines
        - triggermesh
    keys:
```

```
    - name: cosign-key
      publicKey: |
        -----BEGIN PUBLIC KEY-----
        MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEhyQCx0E9wQWSFI9ULGwy3BuRklnt
        IqozONbbdbqz11hlRJy9c7SG+hdcFl9jE9uE/dwtuwU2MqU9T/cN0YkWww==
        -----END PUBLIC KEY-----
    images:
    - namePattern: gcr.io/projectsigstore/cosign*
      keys:
      - name: cosign-key
EOF
```

Provide credentials for the package

There are four ways the package reads credentials to authenticate to registries protected by authentication, in order:

1. Reading `imagePullSecrets` directly from the resource being admitted.
2. Reading `imagePullSecrets` from the service account the resource is running as.
3. Reading a `secretRef` from the `ClusterImagePolicy` resource applied to the cluster for the container image name pattern that matches the container being admitted.
4. Reading `imagePullSecrets` from the `image-policy-registry-credentials` service account in the deployment namespace.

Authentication fails in the following scenario:

- A valid credential is specified in the `ClusterImagePolicy` `secretRef` field, or in the `image-policy-registry-credentials` service account.
- An invalid credential is specified in the `imagePullSecrets` of the resource or in the service account the resource runs as.

To prevent this issue, choose a single authentication method to validate signatures for your resources.

If you use containerd-configured registry credentials or another mechanism that causes your resources and service accounts to not include an `imagePullSecrets` field, you must provide credentials to the WebHook using one of the following mechanisms:

1. Create secret resources in any namespace of your preference that grants read access to the location of your container images and signatures and include it as part of your policy configuration.
2. Create secret resources and include them in the `image-policy-registry-credentials` service account. The service account and the secrets must be created in the deployment namespace.

Provide secrets for authentication in your policy

You can provide secrets for authentication as part of the name pattern policy configuration provided your use case meets the following conditions:

- Your images and signatures reside in a registry protected by authentication.
- You do not have `imagePullSecrets` configured in your runnable resources or in the `ServiceAccount`s that your runnable resources use.
- You want this WebHook to check these container images.

See the following example:

```
---
apiVersion: signing.apps.tanzu.vmware.com/v1beta1
kind: ClusterImagePolicy
metadata:
  name: image-policy
spec:
  verification:
    exclude:
      resources:
        namespaces:
        - kube-system
    keys:
    - name: first-key
      publicKey: |
        -----BEGIN PUBLIC KEY-----
        ...
        -----END PUBLIC KEY-----
    images:
    - namePattern: registry.example.org/myproject/*
      # Your secret reference must be included here
      secretRef:
        name: your-secret
        namespace: your-namespace
      keys:
      - name: first-key
```

> **Note**
>
> : You may need to grant the service account `image-policy-controller-manager` in the deployment namespace RBAC permissions for the verbs `get` and `list` in the namespace that hosts your secrets.

VMware suggests the use of a set of credentials with the least amount of privilege that allows reading the signature stored in your registry.

Provide secrets for authentication in the `image-policy-registry-credentials` service account

If you prefer to provide your secrets in the `image-policy-registry-credentials` service account, follow these steps:

1. Create the required secrets in the deployment namespace (once per secret):

   ```
   kubectl create secret docker-registry SECRET-1 \
     --namespace image-policy-system \
     --docker-server=<server> \
     --docker-username=<username> \
     --docker-password=<password>
   ```

2. Create the `image-policy-registry-credentials` service account in the deployment namespace and add the secret name (one or more) in the previous step to the `imagePullSecrets` section:

   ```
   cat <<EOF | kubectl apply -f -
   apiVersion: v1
   kind: ServiceAccount
   metadata:
     name: image-policy-registry-credentials
     namespace: image-policy-system
   imagePullSecrets:
   - name: SECRET-1
   EOF
   ```

   Where `SECRET-1` is a secret that allows the WebHook to pull signatures from the private registry.

   Add additional secrets to `imagePullSecrets` as required.

Image name patterns

The container image names can be matched exactly or use a wildcard (*) that matches any number of characters.

Example name patterns:

Description	Pattern	Matches Image Name
Exact Match	registry.example.org/myproject/my-image:mytag	registry.example.org/myproject/my-image:mytag
Any Tag	registry.example.org/myproject/my-image	registry.example.org/myproject/my-image:mytag registry.example.org/myproject/my-image:other-tag
Any Tag	registry.example.org/myproject/my-image:*	registry.example.org/myproject/my-image:mytag registry.example.org/myproject/my-image:other-tag
Any Image and Tag	registry.example.org/myproject/*	registry.example.org/myproject/my-image:mytag registry.example.org/myproject/anotherimage:anothertag
Any Project	registry.example.org/*/my-image:mytag	registry.example.org/myproject/my-image:mytag registry.example.org/anotherproject/my-image:mytag
Any Project and Tag	registry.example.org/*/my-image	registry.example.org/myproject/my-image:mytag registry.example.org/myproject/my-image:anothertag
Registry	registry.example.org/*	registry.example.org/myproject/my-image:mytag registry.example.org/anotherproject/anotherimage:anothertag
Any Subdomain	*.example.org/*	my-registry.example.org/myproject/my-image:mytag registry.example.org/anotherproject/anotherimage:anothertag
Anything	*	my-registry.example.org/myproject/my-image:mytag registry.example.org/anotherproject/anotherimage:anothertag registry.io/project/image:tag

Note

: Providing a name pattern without specifying a tag acts as a wildcard for the tag even if other wildcards are specified. The pattern `registry.example.org/myproject/my-image` is the same as `registry.example.org/myproject/my-image:*`. In the same way, `*.example.org/project/image` is equivalent to `*.example.org/project/image:*`

Verify your configuration

If you are using the suggested key `cosign-key` shown in the previous section then you can run the following commands to check your configuration:

1. Verify that a signed image, validated with a configured public key, launches. Run:

   ```
   kubectl run cosign \
     --image=gcr.io/projectsigstore/cosign:v1.2.1 \
     --restart=Never \
     --command -- sleep 900
   ```

   For example:

   ```
   $ kubectl run cosign \
     --image=gcr.io/projectsigstore/cosign:v1.2.1 \
     --restart=Never \
   ```

```
  --command -- sleep 900
pod/cosign created
```

2. Verify that an unsigned image does not launch. Run:

   ```
   kubectl run bb --image=busybox --restart=Never
   ```

   For example:

   ```
   $ kubectl run bb --image=busybox --restart=Never
   Warning: busybox did not match any image policies. Container will be created as
   AllowUnmatchedImages flag is true.
   pod/bb created
   ```

3. Verify that an image signed with a key that does not match the configured public key will not launch. Run:

   ```
   kubectl run cosign-fail \
     --image=gcr.io/projectsigstore/cosign:v0.3.0 \
     --command -- sleep 900
   ```

   For example:

   ```
   $ kubectl run cosign-fail \
     --image=gcr.io/projectsigstore/cosign:v0.3.0 \
     --command -- sleep 900
   Error from server (The image: gcr.io/projectsigstore/cosign:v0.3.0 is not signe
   d.): admission webhook "image-policy-webhook.signing.apps.tanzu.com" denied the
   request: The image: gcr.io/projectsigstore/cosign:v0.3.0 is not signed.
   ```

Logs messages and reasons

Log messages follow a JSON format. Each log can contain the following keys:

Key	Description
level	Log level
ts	Timestamp
logger	Name of the logger component which provided the log message
msg	Log message
object	Relevant object that triggered the log message
error	A message for the error. Only present with “error” log level
stacktrace	A stacktrace for where the error occured. Only present with error level

The possible log messages the webhook emits and their explanations are summarized in the following table:

Log Message	Explanation
`clusterimagepolicies.signing.apps.tanzu.vmware.com "image-policy" not found. Image policy enforcement was not applied.`	The Image Policy was not created in the cluster and the webhook did not check any container images for signatures.
`<Namespace> is excluded. The ImagePolicy will not be applied.`	- An image policy is present in the cluster. - The namespace is present in the `verification.exclude.resources.namespaces` property of the policy. - Any container images trying to get created in this namespace will not be checked for signatures.

Log Message	Explanation
`Could not verify against any image policies for container image: <ContainerImage>.`	- An image policy is present in the cluster. - The `AllowUnMatchedImages` flag is set to `false` or is absent. - The namespace is not excluded. - Image of the container being installed does not match any pattern present in the policy and was rejected by the webhook.
`<ContainerImage> did not match any image policies. Container will be created as AllowUnmatchedImages flag is true.`	- An image policy is present in the cluster. - The `AllowUnMatchedImages` flag is set to `true`. - The namespace you are installing your resource in is not excluded. - Image of the container being installed does not match any pattern present in the policy and was allowed to be created.
`failed to find signature for image.`	- An image policy is present in the cluster. - The namespace you are installing your resource in is not excluded. - Image of the container being installed matches a pattern in the policy. - The webhook was not able to verify the signature.
`The image: <ContainerImage> is not signed.`	- An image policy is present in the cluster. - The namespace you are installing your resource in is not excluded. - Image of the container being installed matches a pattern in the policy. - The image is not signed.

`failed to decode resource`	- The resource type is not supported.
	- Currently supported v1 versions of:
	  ◦ Pod
	  ◦ Deployment
	  ◦ StatefulSet
	  ◦ DaemonSet
	  ◦ ReplicaSet
	  ◦ Job
	  ◦ CronJob (and v1beta1)

`failed to verify`	- An image policy is present in the cluster. - The namespace you are installing your resource in is not excluded. - Image of the container being installed matches a pattern. - The webhook can not verify the signature.

`matching pattern: <Pattern> against image <ContainerImage>`
`matching registry patterns: [{}]`
	- Provide the pattern that matches the container image.
	- Provide the corresponding `Image` configuration from the `ClusterImagePolicy` that matches the container image.

Log Message	Explanation
`service account not found`	- The fallback service account, "image-policy-registry-credentials", was not found in the namespace of which the webhook is installed. - The fallback service account is deprecated and was originally purposed to storing `imagePullSecrets` for container images and their co-located `cosign` signatures.
`unmatched image policy: <ContainerImage>`	Container image does not match any policy image patterns.

Overview of Supply Chain Security Tools for Tanzu – Store

This topic gives you an overview of Supply Chain Security Tools (SCST) – Store.

Overview

Supply Chain Security Tools - Store saves software bills of materials (SBoMs) to a database and allows you to query for image, source code, package, and vulnerability relationships. It integrates with Supply Chain Security Tools - Scan to automatically store the resulting source code and image vulnerability reports. It accepts CycloneDX input and outputs in both human-readable and machine-readable formats, including JSON, text, and CycloneDX.

The following is a quick demo of configuring the tanzu insight plug-in and querying the metadata store for CVEs and scan results.

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 - Scan of source code and image files. Use it to query by source code commit, image digest, and CVE identifier to understand security risks.

See Tanzu Insight plug-in overview to install, configure, and use `tanzu insight`.

Multicluster configuration

See Multicluster setup for information about how to set up SCST - Store in a multicluster setup.

Integrating with Tanzu Application Platform GUI

Using the Supply Chain Choreographer in Tanzu Application Platform GUI, you can visualize your supply chain. It uses to SCST - Store to show the packages and vulnerabilities in your source code and images.

To enable this feature, see Supply Chain Choreographer in Tanzu Application Platform GUI - Enable CVE scan results.

Additional documentation

Additional documentation includes information about the API, deployment details and configuration, AWS RDS configuration, other database backup recommendations, known issues, and other topics.

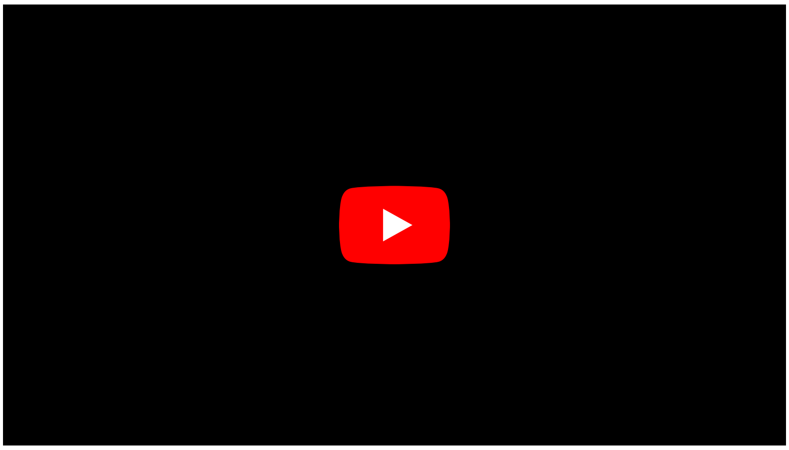
Overview of Supply Chain Security Tools for Tanzu – Store

This topic gives you an overview of Supply Chain Security Tools (SCST) – Store.

Overview

Supply Chain Security Tools - Store saves software bills of materials (SBoMs) to a database and allows you to query for image, source code, package, and vulnerability relationships. It integrates with Supply Chain Security Tools - Scan to automatically store the resulting source code and image vulnerability reports. It accepts CycloneDX input and outputs in both human-readable and machine-readable formats, including JSON, text, and CycloneDX.

The following is a quick demo of configuring the tanzu insight plug-in and querying the metadata store for CVEs and scan results.



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 - Scan of source code and image files. Use it to query by source code commit, image digest, and CVE identifier to understand security risks.

See Tanzu Insight plug-in overview to install, configure, and use `tanzu insight`.

Multicluster configuration

See Multicluster setup for information about how to set up SCST - Store in a multicluster setup.

Integrating with Tanzu Application Platform GUI

Using the Supply Chain Choreographer in Tanzu Application Platform GUI, you can visualize your supply chain. It uses to SCST - Store to show the packages and vulnerabilities in your source code and images.

To enable this feature, see Supply Chain Choreographer in Tanzu Application Platform GUI - Enable CVE scan results.

Additional documentation

Additional documentation includes information about the API, deployment details and configuration, AWS RDS configuration, other database backup recommendations, known issues, and other topics.

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.

The endpoint host should be set to `metadata-store.<ingress-domain>` (such as `metadata-store.example.domain.com`), where `<ingress-domain>` should match the 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.loadBalancer.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 ingress-cert -n metadata-store -o json | jq -r '.data."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.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.

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
- API details
- API walkthrough
- Failover, redundancy, and backups

Troubleshooting and logging

- Troubleshooting upgrading
- Log configuration and usage

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
- Multicluster setup
- Developer namespace setup
- Retrieve access tokens
- Create service accounts
- Create a service account with a custom cluster role

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
- API details
- API walkthrough
- Failover, redundancy, and backups

Troubleshooting and logging

- Troubleshooting upgrading
- Log configuration and usage

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
- Multicluster setup
- Developer namespace setup
- Retrieve access tokens
- Create service accounts
- Create a service account with a custom cluster role

API reference for Supply Chain Security Tools - Store

This topic contains API reference information for Supply Chain Security Tools - Store. See API walkthrough for an SCST - Store example.

Information

Version

0.0.1

Content negotiation

URI Schemes

- http
- https

Consumes

- application/json

Produces

- application/json

All endpoints

images

Method	URI	Name	Summary
POST	/api/imageReport	create image report	Create a new image report. Related packages and vulnerabilities are also created.
GET	/api/images	get images	Search image by id or digest.
GET	/api/packages/{IDorName}/images	get package images	List the images that contain the given package.
GET	/api/vulnerabilities/{CVEID}/images	get vulnerability images	List the images that contain the given vulnerability.

Operations

Method	URI	Name	Summary
GET	/api/health	health check	

Packages

Method	URI	Name	Summary
GET	/api/images/{IDorDigest}/packages	get image packages	List the packages in an image.
GET	/api/packages	get packages	Search packages by id, name and/or version.
GET	/api/sources/{IDorRepoorSha}/packages	get source packages	
GET	/api/sources/packages	get source packages query	List packages of the given source.
GET	/api/vulnerabilities/{CVEID}/packages	get vulnerability packages	List packages that contain the given CVE id.

Sources

Method	URI	Name	Summary
POST	/api/sourceReport	create source report	Create a new source report. Related packages and vulnerabilities are also created.
GET	/api/packages/{IDorName}/sources	get package sources	List the sources containing the given package.
GET	/api/sources	get sources	Search for sources by ID, repository, commit sha and/or organization.
GET	/api/vulnerabilities/{CVEID}/sources	get vulnerability sources	List sources that contain the given vulnerability.

Vulnerabilities

Method	URI	Name	Summary
GET	/api/images/{IDorDigest}/vulnerabilities	get image vulnerabilities	List vulnerabilities from the given image.
GET	/api/packages/{IDorName}/vulnerabilities	get package vulnerabilities	List vulnerabilities from the given package.
GET	/api/sources/{IDorRepoorSha}/vulnerabilities	get source vulnerabilities	
GET	/api/sources/vulnerabilities	get source vulnerabilities query	List vulnerabilities of the given source.
GET	/api/vulnerabilities	get vulnerabilities	Search for vulnerabilities by CVE id.

Paths

Create a new image report. Related packages and vulnerabilities are also created. (*CreateImageReport*)

```
POST /api/imageReport
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
Image	`body`	Image	`models.Image`		✓		

All responses

Code	Status	Description	Has headers	Schema
200	OK	Image		schema
default		ErrorMessage		schema

Responses

200 - Image

Status: OK

Schema

Image

Default Response

ErrorMessage

Schema

ErrorMessage

Create a new source report. Related packages and vulnerabilities are also created. (*CreateSourceReport*)

```
POST /api/sourceReport
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
Image	`body`	Source	`models.Source`		✓		

All responses

Code	Status	Description	Has headers	Schema
200	OK	Source		schema
default		ErrorMessage		schema

Responses

200 - Source

Status: OK

Schema

Source

Default Response

ErrorMessage

Schema

ErrorMessage

List the packages in an image. (*GetImagePackages*)

```
GET /api/images/{IDorDigest}/packages
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
IDorDigest	`path`	string	`string`		✓		

All responses

Code	Status	Description	Has headers	Schema
200	OK	Package		schema
default		ErrorMessage		schema

Responses

200 - Package

Status: OK

Schema

[][Package](#package)

Default Response

ErrorMessage

Schema

ErrorMessage

List vulnerabilities from the given image. (*GetImageVulnerabilities*)

```
GET /api/images/{IDorDigest}/vulnerabilities
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
IDorDigest	`path`	string	`string`		✓		

All responses

Code	Status	Description	Has headers	Schema
200	OK	Vulnerability		schema
default		ErrorMessage		schema

Responses

200 - Vulnerability

Status: OK

Schema

[][Vulnerability](#vulnerability)

Default Response

ErrorMessage

Schema

ErrorMessage

Search image by id or digest. (*GetImages*)

```
GET /api/images
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
digest	`query`	string	`string`				
id	`query`	int64 (formatted integer)	`int64`				

responses

Code	Status	Description	Has headers	Schema
200	OK	Image		schema
default		ErrorMessage		schema

Responses

200 - Image

Status: OK

Schema

Image

Default Response

ErrorMessage

Schema

ErrorMessage

List the images that contain the given package. (*GetPackageImages*)

```
GET /api/packages/{IDorName}/images
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
IDorName	`path`	string	`string`		✓		

All responses

Code	Status	Description	Has headers	Schema
200	OK	Image		schema
default		ErrorMessage		schema

Responses

200 - Image

Status: OK

Schema

[][Image](#image)

Default Response

ErrorMessage

Schema

ErrorMessage

List the sources containing the given package. (*GetPackageSources*)

```
GET /api/packages/{IDorName}/sources
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
IDorName	`path`	string	`string`		✓		

All responses

Code	Status	Description	Has headers	Schema
200	OK	Source		schema
default		ErrorMessage		schema

Responses

200 - Source

Status: OK

Schema

[][Source](#source)

Default Response

ErrorMessage

Schema

ErrorMessage

List vulnerabilities from the given package. (*GetPackageVulnerabilities*)

```
GET /api/packages/{IDorName}/vulnerabilities
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
IDorName	`path`	string	`string`		✓		

All responses

Code	Status	Description	Has headers	Schema
200	OK	Vulnerability		schema
default		ErrorMessage		schema

Responses

200 - Vulnerability

Status: OK

Schema

[][Vulnerability](#vulnerability)

Default Response

ErrorMessage

Schema

ErrorMessage

Search packages by id, name and/or version. (*GetPackages*)

```
GET /api/packages
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
id	`query`	int64 (formatted integer)	`int64`				Any of id or name must be provided
name	`query`	string	`string`				Any of id or name must be provided
version	`query`	string	`string`				

All responses

Code	Status	Description	Has headers	Schema
200	OK	Package		schema
default		ErrorMessage		schema

Responses

200 - Package

Status: OK

Schema

[][Package](#package)

Default Response

ErrorMessage

Schema

ErrorMessage

get source packages (*GetSourcePackages*)

```
GET /api/sources/{IDorRepoorSha}/packages
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
IDorRepoorSha	`path`	string	`string`		✓		

All responses

Code	Status	Description	Has headers	Schema
200	OK	Package		schema
default		ErrorMessage		schema

Responses

200 - Package

Status: OK

Schema

[][Package](#package)

Default Response

ErrorMessage

Schema

ErrorMessage

List packages of the given source. (*GetSourcePackagesQuery*)

```
GET /api/sources/packages
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
id	`query`	uint64 (formatted integer)	`uint64`				
repo	`query`	string	`string`				
sha	`query`	string	`string`				

All responses

Code	Status	Description	Has headers	Schema
200	OK	Package		schema
default		ErrorMessage		schema

Responses

200 - Package

Status: OK

Schema

[][Package](#package)

Default Response

ErrorMessage

Schema

ErrorMessage

get source vulnerabilities (*GetSourceVulnerabilities*)

```
GET /api/sources/{IDorRepoorSha}/vulnerabilities
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
IDorRepoorSha	`path`	string	`string`		✓		

All responses

Code	Status	Description	Has headers	Schema
200	OK	Vulnerability		schema
default		ErrorMessage		schema

Responses

200 - Vulnerability

Status: OK

Schema

[][Vulnerability](#vulnerability)

Default Response

ErrorMessage

Schema

ErrorMessage

List vulnerabilities of the given source. (*GetSourceVulnerabilitiesQuery*)

```
GET /api/sources/vulnerabilities
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
id	`query`	uint64 (formatted integer)	`uint64`				
repo	`query`	string	`string`				
sha	`query`	string	`string`				

All responses

Code	Status	Description	Has headers	Schema
200	OK	Vulnerability		schema
default		ErrorMessage		schema

Responses

200 - Vulnerability

Status: OK

Schema

[][Vulnerability](#vulnerability)

Default Response

ErrorMessage

Schema

ErrorMessage

Search for sources by ID, repository, commit sha and/or organization. (*GetSourcs*)

```
GET /api/sources
```

All responses

Code	Status	Description	Has headers	Schema
200	OK	Source		schema
default		ErrorMessage		schema

Responses

200 - Source

Status: OK

Schema

[][Source](#source)

Default Response

ErrorMessage

Schema

ErrorMessage

Search for vulnerabilities by CVE id. (*GetVulnerabilities*)

```
GET /api/vulnerabilities
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
CVEID	`query`	string	`string`		✓		

All responses

Code	Status	Description	Has headers	Schema
200	OK	Vulnerability		schema
default		ErrorMessage		schema

Responses

200 - Vulnerability

Status: OK

Schema

[][Vulnerability](#vulnerability)

Default Response

ErrorMessage

Schema

ErrorMessage

List the images that contain the given vulnerability. (*GetVulnerabilityImages*)

```
GET /api/vulnerabilities/{CVEID}/images
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
CVEID	`path`	string	`string`		✓		

All responses

Code	Status	Description	Has headers	Schema
200	OK	Image		schema
default		ErrorMessage		schema

Responses

200 - Image

Status: OK

Schema

[][Image](#image)

Default Response

ErrorMessage

Schema

ErrorMessage

List packages that contain the given CVE id. (*GetVulnerabilityPackages*)

```
GET /api/vulnerabilities/{CVEID}/packages
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
CVEID	`path`	string	`string`		✓		

All responses

Code	Status	Description	Has headers	Schema
200	OK	Package		schema
default		ErrorMessage		schema

Responses

200 - Package

Status: OK

Schema

[][Package](#package)

Default Response

ErrorMessage

Schema

ErrorMessage

List sources that contain the given vulnerability. (*GetVulnerabilitySources*)

```
GET /api/vulnerabilities/{CVEID}/sources
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
CVEID	`path`	string	`string`		✓		

All responses

Code	Status	Description	Has headers	Schema
200	OK	Source		schema
default		ErrorMessage		schema

Responses

200 - Source

Status: OK

Schema

[][Source](#source)

Default Response

ErrorMessage

Schema

ErrorMessage

health check (*HealthCheck*)

```
GET /api/health
```

All responses

Code	Status	Description	Has headers	Schema
200	OK			schema
default		ErrorMessage		schema

Responses

200

Status: OK

Schema

Default Response

ErrorMessage

Schema

ErrorMessage

Models

DeletedAt

- composed type NullTime

ErrorMessage

ErrorMessage wraps an error message in a struct so responses are properly marshalled as a JSON object.

Properties

Name	Type	Go type	Required	Default	Description	Example
Message	string	`string`			in: body	`something went wrong`

Image

Properties

Name	Type	Go type	Required	Default	Description	Example
Digest	string	`string`	✓			`9n38274ods897fmay487gsdyfga678wr82`
ID	uint64 (formatted integer)	`uint64`				
Name	string	`string`	✓			`myorg/application`
Packages	[][Package](#package)	`[]*Package`				
Registry	string	`string`	✓			`docker.io`
Sources	[][Source](#source)	`[]*Source`				

MethodType

Properties

Name	Type	Go type	Required	Default	Description	Example
CreatedAt	date-time (formatted string)	`strfmt.DateTime`				
DeletedAt	DeletedAt	`DeletedAt`				
ID	uint64 (formatted integer)	`uint64`				
Name	string	`string`				
Rating	[][Rating](#rating)	`[]*Rating`				
UpdatedAt	date-time (formatted string)	`strfmt.DateTime`				

Model

Model a basic GoLang struct which includes the following fields: ID, CreatedAt, UpdatedAt, DeletedAt It may be embedded into your model, or you may build your model without it type User struct { gorm.Model }

Properties

Name	Type	Go type	Required	Default	Description	Example
CreatedAt	date-time (formatted string)	`strfmt.DateTime`				
DeletedAt	DeletedAt	`DeletedAt`				
ID	uint64 (formatted integer)	`uint64`				
UpdatedAt	date-time (formatted string)	`strfmt.DateTime`				

NullTime

NullTime implements the Scanner interface to be used as a scan destination, similar to NullString.

Properties

Name	Type	Go type	Required	Default	Description	Example
Time	date-time (formatted string)	`strfmt.DateTime`				
Valid	boolean	`bool`				

Package

Properties

Name	Type	Go type	Required	Default	Description	Example
Homepage	string	`string`				
ID	uint64 (formatted integer)	`uint64`				
Images	[][Image](#image)	`[]*Image`				
Name	string	`string`				
PackageManager	string	`string`				
Sources	[][Source](#source)	`[]*Source`				
Version	string	`string`				
Vulnerabilities	[][Vulnerability](#vulnerability)	`[]*Vulnerability`				

Rating

Properties

Name	Type	Go type	Required	Default	Description	Example
ID	uint64 (formatted integer)	`uint64`				
MethodType	MethodType	`MethodType`				
MethodTypeID	uint64 (formatted integer)	`uint64`				
Score	double (formatted number)	`float64`				
Severity	string	`string`				
Vector	string	`string`				

Source

Properties

Name	Type	Go type	Required	Default	Description	Example
DeletedAt	DeletedAt	`DeletedAt`				
Host	string	`string`				`gitlab.com`
ID	uint64 (formatted integer)	`uint64`				
Images	[][Image](#image)	`[]*Image`				
Organization	string	`string`				`vmware`
Packages	[][Package](#package)	`[]*Package`				
Repository	string	`string`	✓			`myproject`
Sha	string	`string`	✓			`0eb5fcd1`

StringArray

[]string

Vulnerability

Properties

Name	Type	Go type	Required	Default	Description	Example
CNA	string	`string`				
CVEID	string	`string`	✓			`CVE-7467-2020`
Description	string	`string`				
ID	uint64 (formatted integer)	`uint64`				
Packages	[][Package](#package)	`[]*Package`				
Ratings	[][Rating](#rating)	`[]*Rating`				
References	StringArray	`StringArray`				
URL	string	`string`				

API walkthrough for Supply Chain Security Tools - Store

This topic includes an example API call that you can use with Supply Chain Security Tools - Store. For information about using the SCST - Store API, see full API documentation.

Using CURL to POST an image report

The following procedure explains how to use CURL to POST an image report.

1. Port Forward the metadata-store-app. Run:

```
kubectl port-forward service/metadata-store-app 8443:8443 -n metadata-store
```

2. Retrieve the `metadata-store-read-write-client` access token. See Retrieve access tokens. Run:

```
export METADATA_STORE_ACCESS_TOKEN=$(kubectl get secrets metadata-store-read-wr
ite-client -n metadata-store -o jsonpath="{.data.token}" | base64 -d)
```

3. Retrieve the CA Certificate and store it locally. Run:

```
kubectl get secret ingress-cert -n metadata-store -o json | jq -r '.data."ca.cr
t"' | base64 -d > /tmp/ca.crt
```

4. Run the Curl POST Command:

```
curl https://metadata-store.<ingress-domain>/api/imageReport \
    --cacert /tmp/ca.crt \
    -H "Authorization: Bearer ${METADATA_STORE_ACCESS_TOKEN}" \
    -H "Content-Type: application/json" \
    -X POST \
    --data "@<ABSOLUTE PATH TO THE POST BODY>"
```

5. Replace with the absolute path of the POST body.

6. The following is a sample POST body of a image report:

```
{
  "Name" : "burger-image-2",
  "Registry" : "test-registry",
  "Digest" : "test-digest@45asd61asasssdfsdfddssghjkdfsdfasdfasdsdasdassdfghjdd
asfddfsadfadfgfshdasdfsdfsdfsdasdsdfsdfadsdassdfdasdfaasdsdfsddfsdasgsasddffdgf
dasddfgdfssdfakasdasdasdsdasddasdsd23",
  "Sources" : [
    {
      "Repository" : "aaaaoslfdfggo",
      "Organization" : "pivotal",
      "Sha" : "1235assdfssadfacfddxdf41",
      "Host" : "http://oslo.io",
      "Packages" : [
        {
          "Name" : "Source package5",
          "Version" : "v2sfsfdd34",
          "PackageManager" : "test-manager",
          "Vulnerabilities" : [
            {
              "CVEID" : "0011",
              "PrimaryURL" : "http://www.mynamejeff.comm",
              "Description" : "Bye",
              "CNA" : "NVD",
              "Ratings": [{
                "Vector" : "AV:L/AC:L/Au:N/C:P/I:P/A:P",
                "Score" : 0,
                "MethodTypeID" : 1,
                "Severity":   "High"
              }],
              "References" : [""]
            }
          ]
        }
      ]
    }
  ],
  "Packages" : [
    {
      "Name" : "bob-dependency-35daasds56j",
      "Version" : "v2",
      "PackageManager" : "test-manager",
      "Vulnerabilities" : [
        {
          "CVEID" : "002",
          "PrimaryURL" : "http://www.mynamejeff.comm",
          "Description" : "Bye",
          "CNA" : "NVD",
          "Ratings": [{
            "Vector" : "AV:L/AC:L/Au:N/C:P/I:P/A:P",
            "Score" : 0,
            "MethodTypeID" : 1,
            "Severity":   "High"
          }],
          "References" : [""]
        }
      ]
    }
  ]
}
```

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:

- Two components — an API back end and a database. Each component includes:
  - service
  - deployment
  - replicaset
  - Pod
- Persistent volume claim
- External IP address (based on a deployment configuration set to use `LoadBalancer`).
- A Kubernetes secret to allow pulling SCST - Store images from a registry.
- A namespace called `metadata-store`.
- A service account with read-write privileges named `metadata-store-read-write-client`, and a corresponding secret for the service account. It's bound to a ClusterRole named `metadata-store-read-write`.
- A read-only ClusterRole named `metadata-store-read-only` that isn't bound to a service account. See Service Accounts.
- (Optional) An HTTPProxy object for ingress support.

Deployment configuration

All configurations are nested inside of `metadata_store` in your tap values deployment YAML.

Supported Network Configurations

The following connection methods are recommended based on Tanzu Application Platform setup:

- Single or multicluster with Contour = `Ingress`
- Single cluster without Contour and with `LoadBalancer` support = `LoadBalancer`
- Single cluster without Contour and without `LoadBalancer` = `NodePort`
- Multicluster without Contour = Not supported

For a production environment, VMware recommends that you install SCST - Store with ingress enabled.

By default, a database password is generated upon deployment. To configure a custom password, use the `db_password` property in the `metadata-store-values.yaml` during deployment.

Supported values include `LoadBalancer`, `ClusterIP`, `NodePort`. The `app_service_type` is set to `LoadBalancer` by default. If your environment does not support `LoadBalancer`, and you want to use `ClusterIP`, configure the `app_service_type` property in your deployment YAML:

```
app_service_type: "ClusterIP"
```

If you set the `ingress_enabled` to `"true"`, VMware recommends setting the `app_service_type` property to `"ClusterIP"`.

Service accounts

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. For example:

```
ingress_enabled: "true"
ingress_domain: "example.com"
app_service_type: "ClusterIP" # recommended setting
```

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 is meant to get users started using the metadata store. The default database deployment does not support many enterprise production requirements, including scaling, redundancy, or failover. 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 also 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 `db_password` property in the deployment YAML.

```
db_password: "PASSWORD-0123"
```

Where `PASSWORD-0123` is the same password used between deployments.

Important

There is a known issue related to changing database passwords Persistent Volume Retains Data.

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

AWS RDS

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
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 can configure and use your external PostgreSQL database for Supply Chain Security Tools (SCST) - Store.

Prerequisites

- Setup your external postgres database. Once the database instance starts, retrieve the following information:
    1. DB Instance Endpoint
    2. Master Username
    3. Master Password
    4. Database Name

Setup certificate and configuration

1. Create a security group to allow inbound connections from the cluster to the Postgres DB
2. Retrieve the corresponding CA Certificate that signed the Postgres TLS Certificate.
3. 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.

Validation

Verification was done using bitnami postgres. You can get more information from 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-location> --secret-file <secrets-file>
```

For example:

```
velero install --provider gcp --bucket <gcs-bucket-name> --plugins velero/velero-plugin-for-gcp:v1.3.0 --secret-file <gcp-json-credentials>
```

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 the Supply Chain Security Tools - Store supports.

Level	Description
Trace	Output extended debugging logs.

Level	Description
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 the Store is deployed at a specific verbosity level, all logs of that level and lower are outputted 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 the Metadata Store is configured to 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 no verbosity level is specified when the Store is installed, the level is set to `default`.

Error Logs

Error logs are always outputted 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 by running:

```
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 prefix `metadata-store-db-` and 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 see 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 the other 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 Github.

API endpoint log output

When an API endpoint handles a request, the Store generates two and five log lines. They are:

1. When the endpoint receives a request, it outputs a `Processing request` line. This logline is shown at the `default` verbosity level.
2. 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.
3. 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.
4. When the endpoint returns a response, it outputs a `Request response` line. This line is shown at the `default` verbosity level.
5. 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 some 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

Since 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 message 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`

Key	Type	Verbosity Level	Description
msg	string	all	A short description of the logged event.
hostname	string	all	The Kubernetes hostname 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` logline as key-value pairs. Afterward, they are appended to all other log lines 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 line contains the `digest` and `id` key, which represents the respective `digest` and `id` query parameters, as well as their values:

```
{"level":"info","ts":"2022-05-27T15:41:36.052063176Z","logger":"MetadataStore","ms
g":"Request parameters","hostname":"metadata-store-app-c7c8648f7-8dmdl","method":"GE
T","endpoint":"/api/images?digest=sha256%3A20521f76ff3d27f436e03dc666cc97a511bbe71e8e8
495f851d0f4bf57b0bab6","id":0,"digest":"sha256:20521f76ff3d27f436e03dc666cc97a511bbe71
e8e8495f851d0f4bf57b0bab6","name":""}
```

These key/value pairs show up in all subsequent log lines of the same call. For example:

```
{"level":"info","ts":"2022-05-27T15:41:36.057393519Z","logger":"MetadataStore","ms
g":"Request response","hostname":"metadata-store-app-c7c8648f7-8dmdl","method":"GE
T","endpoint":"/api/images?digest=sha256%3A20521f76ff3d27f436e03dc666cc97a511bbe71e8e8
495f851d0f4bf57b0bab6","id":0,"digest":"sha256:20521f76ff3d27f436e03dc666cc97a511bbe71
e8e8495f851d0f4bf57b0bab6","name":"","code":200,"response":"OK"}
```

This is done to ensure:

- The application interprets the values of the query or path parameters correctly.
- Help figure out which log lines are associated with a particular API request. Since there can 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`, is used to display this information instead of `default` because:

- Body payloads can be huge, containing 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 may be sensitive, and the user may not want them exposed in production environment logs.

SQL Query log output

Some Store logs display the executed SQL query commands when you set the verbosity level to `trace` or a failed SQL call occurs.

Note: Some information in these SQL Query trace logs might be sensitive, and the user 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":"2022-05-27T15:37:26.186960324Z","logger":"MetadataStore.gorm","msg":"sql call","hostname":"metadata-store-app-c7c8648f7-8dmdl","rows":1,"sql":"SELECT count(*) FROM information_schema.tables WHERE table_schema = CURRENT_SCHEMA() AND table_name = 'images' AND table_type = 'BASE TABLE'"}
```

It is similar to the API endpoint log output format, but also 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.

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

When attempting to look up CVE and affected packages, querying `insight source get` (or other `insight source` commands) might return zero results due to supply chain configuration and repository URL.

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 in GitHub.

Persistent volume retains data

Symptom

If **Supply Chain Security Tools - 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.

Solution

Caution: Changing the database password deletes your Supply Chain Security Tools - Store data.

To redeploy the app, either use the same database password or follow the following 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 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 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 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 Amazon Elastic Kubernetes Service (EKS) running Kubernetes v1.23

Symptom

When installing 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.

Store uses the default storage class which uses EBS volumes by default on EKS.

Solution

To install the Amazon EBS CSI Driver before installing Store or before upgrading to Kubernetes v1.23, see the Amazon EBS CSI Driver topic in the Amazon documentation.

Certificate Expiries

Symptom

The Insight CLI or the Scan Controller fails to connect to the 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
```

Troubleshoot upgrading Supply Chain Security Tools - Store

This topic describes how you can troubleshoot upgrading issues Supply Chain Security Tools (SCST) - Store.

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 using Tanzu to upgrade to a new version of the store, there is occasionally data corruption. Here is an example of how this shows up in the log:

```
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 steps to fix the issue so that the upgrade can proceed, see the SysOpsPro documentation.

Upgraded pod hanging

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
   ```

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 only has 1 replica. If the POD fails, the single instance restarts by normal Kubernetes behavior, but there is downtime. If the user is upgrading, some downtime is expected in most cases as well.

Users have the option to configure the number of replicas using the `app_replicas` field in the `scst-store-values.yaml` file.

Database

By default, the database has 1 replica, and restarts with some downtime if it were to fail.

Although the field `db_replicas` exists and is configurable by the user in the `scst-store-values.yaml` file, VMware discourages using it. The default internal database is not intended to be used in production. For production use AWS RDS. See instructions here.

For the default postgres database deployment (set by default or by setting `deploy_internal_db` to true), `Velero` may be used as the backup method. Read more about using `Velero` as back up here.

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

Optionally, users can configure TLS to use a custom certificate. In order to do that, follow these steps:

1. Place the certificates in secret
2. Modify the `tap-values.yaml` to use this secret

Place the certificates in secret

The certificate secret should be created before deploying Supply Chain Security Tools - Store. Create a Kubernetes object with kind `Secret` and type `kubernetes.io/tls`.

Update `tap-values.yaml`

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"
```

- `namespace`: The targeted namespace for secret consumption by the HTTPProxy.
- `secretName`: 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.

(Optional) Setting up custom ingress TLS certificate

Optionally, users can configure TLS to use a custom certificate. In order to do that, follow these steps:

1. Place the certificates in secret
2. Modify the `tap-values.yaml` to use this secret

Place the certificates in secret

The certificate secret should be created before deploying Supply Chain Security Tools - Store. Create a Kubernetes object with kind `Secret` and type `kubernetes.io/tls`.

Update `tap-values.yaml`

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"
```

- `namespace`: The targeted namespace for secret consumption by the HTTPProxy.
- `secretName`: The name of secret for consumption by the HTTPProxy.

Change server TLS Ciphers

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
```

`tls.server.rfcCiphers`: 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: "secretName"
    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
```

Additional resources

- Custom certificate configuration
- Ingress support

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 should 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`. See Custom certificate configuration for information about how to configure custom certificates.

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 ingress-cert -n metadata-store -o json | jq -r '.data."ca.crt"' |
base64 -d > insight-ca.crt
$ cat insight-ca.crt
```

Additional Resources

- Custom certificate configuration
- TLS configuration
- 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 metadata-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 > /dev/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 Supply Chain Security Tools - 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-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 "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 Supply Chain Security Tools - 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.

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 includes installing multiple profiles such as, View, Build, Run, and Iterate.

Overview

SCST - Store is deployed with the View profile. After installing the View profile, but before installing the Build profile, you must add configuration for SCST - Store to the Kubernetes cluster where you intend to install the Build profile. This topic explains how to add configuration which allows components in the Build cluster to communicate with SCST - Store in the View cluster.

Note

If you already deployed the Build profile, you can follow this procedure. However, in the Install Build profile step, instead of deploying the Build profile again, update your deployment using `tanzu package installed update`.

If you have already deployed the Build profile, you can still follow this guide. However, in the step Install Build profile, instead of deploying the Build profile again, you should update your deploying using `tanzu package installed update`.

Prerequisites

You must install the View profile. See Install View profile.

Procedure summary

1. Copy SCST - Store CA certificate from the View cluster.
2. Copy SCST - Store authentication token from the View cluster.
3. Apply the CA certificate and authentication token to the Kubernetes cluster where you intend to install the Build profile.
4. Install the Build profile.

Copy SCST - Store CA certificate from View cluster

With your kubectl targeted at the View cluster, you can view SCST - Store’s TLS CA certificate. Run these commands to copy the CA certificate into a file `store_ca.yaml`.

```
CA_CERT=$(kubectl get secret -n metadata-store CERT-NAME -o json | jq -r ".data.\"ca.c
rt\"")
cat <<EOF > store_ca.yaml
---
apiVersion: v1
kind: Secret
type: Opaque
metadata:
  name: store-ca-cert
  namespace: metadata-store-secrets
data:
  ca.crt: $CA_CERT
EOF
```

For example:

```
$ CA_CERT=$(kubectl get secret -n metadata-store ingress-cert -o json | jq -r ".dat
a.\"ca.crt\"")
$ cat <<EOF > store_ca.yaml
---
apiVersion: v1
kind: Secret
type: Opaque
metadata:
  name: store-ca-cert
  namespace: metadata-store-secrets
data:
  ca.crt: $CA_CERT
EOF
```

Copy SCST - Store authentication token from the View cluster

Copy the SCST - Store authentication token into an environment variable. You use this environment variable in the next step.

```
AUTH_TOKEN=$(kubectl get secrets metadata-store-read-write-client -n metadata-store -o
jsonpath="{.data.token}" | base64 -d)
```

Apply the CA certificate and authentication token to a new Kubernetes cluster

Before you deploy the Build profile, you must apply the CA certificate and authentication token from the earlier steps. Then the Build profile deployment has access to these values.

To apply the CA certificate and authentication token:

1. With your kubectl targeted at the Build cluster, create a namespace for the CA certificate and authentication token.

   ```
   kubectl create ns metadata-store-secrets
   ```

2. Apply the CA certificate `store_ca.yaml` secret YAML you generated earlier.

   ```
   kubectl apply -f store_ca.yaml
   ```

3. Create a secret to store the access token. This uses the `AUTH_TOKEN` environment variable.

   ```
   kubectl create secret generic store-auth-token \
     --from-literal=auth_token=$AUTH_TOKEN -n metadata-store-secrets
   ```

The cluster now has a CA certificate named `store-ca-cert` and authentication token named `store-auth-token` in the namespace `metadata-store-secrets`.

Install Build profile

If you came to this topic from the Install multicluster Tanzu Application Platform profiles topic after installing the View profile, return to that topic to install the Build profile.

The Build profile `values.yaml` contains configuration that references the secrets in the `metadata-store-secrets` namespace you created in this guide. The names of these secrets are hard coded in the example `values.yaml`.

More information about how Build profile uses the configuration

The secrets you created are used in the Build profile `values.yaml` to configure the Grype scanner which talks to SCST - Store. After performing a vulnerabilities scan, the Grype scanner sends the results to SCST - Store. Here's a snippet of what the configuration might look like.

```
...
grype:
  metadataStore:
    caSecret:
        name: store-ca-cert
        importFromNamespace: metadata-store-secrets
    authSecret:
        name: store-auth-token
        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.
- `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.

Configure developer namespaces

After you finish the entire Tanzu Application Platform installation process, you are ready to configure developer namespaces. To prepare developer namespaces, you must export the 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. Run the following command to create the `SecretExport` resources. You must have created and populated the `metadata-store-secrets` namespace.

```
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 secrets are exported.

Additional resources

- Ingress support
- Custom certificate configuration

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 included 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`, depending on the Kubernetes version:

```
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
```

Note

For Kubernetes v1.24 and later, services account secrets are no longer automatically created. This is why the example adds a `Secret` resource in the earlier YAML.

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 depending on the Kubernetes version:

```
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
```

Note

For Kubernetes v1.24 and later, services account secrets are no longer automatically created. This is why the example adds a `Secret` resource in the earlier YAML.

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`, depending on the Kubernetes version:

```
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
```

Note

For Kubernetes v1.24 and later, service account secrets are no longer automatically created. This is why the example adds a `Secret` resource in the earlier YAML.

Additional Resources

- Retrieve access tokens
- Create service accounts

Overview of Tanzu Developer Tools for VS Code

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 only on macOS for Java applications.

Extension 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 Workloads panel you can see any workload found within the cluster and namespace specified in the current kubectl context.

Overview of Tanzu Developer Tools for VS Code

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 only on macOS for Java applications.

Extension 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 Workloads panel you can see any workload found within the cluster and namespace specified in the current kubectl context.

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 v0.27.2 or later
- Tanzu CLI and plug-ins
- A cluster with the Tanzu Application Platform Full profile or Iterate profile

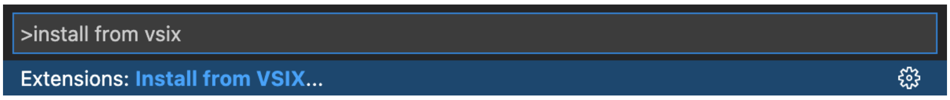
If you are an app developer, someone else in your organization might have already set up the Tanzu Application Platform environment.

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

Install

To install the extension:

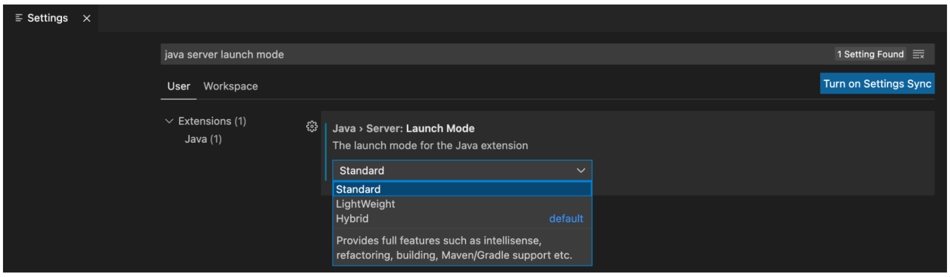
1. Sign in to VMware Tanzu Network and download Tanzu Developer Tools for Visual Studio Code.
2. Open VS Code.
3. Press cmd+shift+P to open the Command Palette and run `Extensions: Install from VSIX...`.
4. Select the extension file **tanzu-vscode-extension.vsix**.



>install from vsix

Extensions: Install from VSIX...

5. If you do not have the following extensions, and they do not automatically install, install them from VS Code Marketplace:
   - Debugger for Java
   - Language Support for Java(™) by Red Hat
   - YAML
6. Ensure Language Support for Java is running in Standard Mode. You can configure it in the **Settings** menu by going to **Code** > **Preferences** > **Settings** under **Java > Server: Launch Mode**.



Java › Server: Launch Mode

The launch mode for the Java extension

Standard

Standard
LightWeight
Hybrid default

Provides full features such as intellisense, refactoring, building, Maven/Gradle support etc.

When the JDK and Language Support for Java are configured correctly, you see that the integrated development environment creates a directory target where the code is compiled.

Configure

To configure VMware Tanzu Developer Tools for VS Code:

1. Ensure that you are targeting the correct cluster. For more informatiom, see the Kubernetes documentation.
2. Go to **Code** > **Preferences** > **Settings** > **Extensions** > **Tanzu Developer Tools** and set the following:
   - **Confirm Delete**: This controls whether the extension asks for confirmation when deleting a workload.
   - **Enable Live Hover**: For more information, see Integrating Live Hover by using Spring Boot Tools. Reload VS Code for this change to take effect.
   - **Source Image**: (Required) The registry location for publishing local source code. For example, `registry.io/yourapp-source`. This must include both a registry and a project name.
   - **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.

Uninstall

To uninstall VMware Tanzu Developer Tools for VS Code:

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

Next steps

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

Getting 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.

Set up Tanzu Developer Tools

To use the extension with a project, the project must have these required files:

- `workload.yaml`
- `catalog-info.yaml`
- `Tiltfile`

There are two ways to create these files:

- Using the VS Code snippets that Tanzu Developer Tools provide, which create templates in empty files that you then fill in with the required information. For more information about the snippets, see the VS Code documentation.
- Writing the files by setting up manually.

Create the `workload.yaml` file

The `workload.yaml` file provides instructions to the Supply Chain Choreographer to build and manage a workload.

The extension requires only one workload.yaml per project. The workload.yaml must be a single-document YAML file, not a multidocument YAML file.

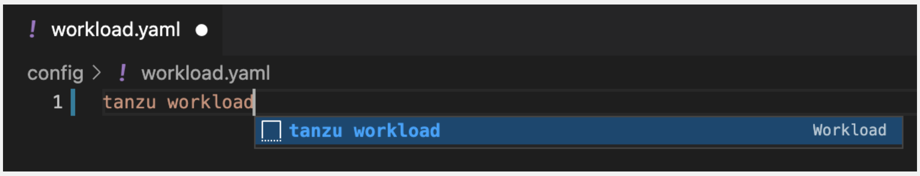
Before beginning to write your workload.yaml file, ensure that you know:

- 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.



```
tanzu workload
```

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 Tanzu apps workload apply in the Tanzu CLI documentation.

Create the catalog-info.yaml file

The catalog-info.yaml file enables the workloads of this project to appear in Tanzu Application Platform GUI.

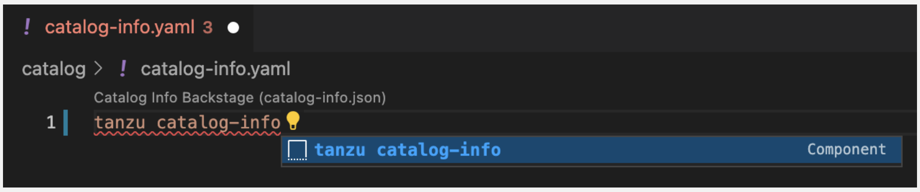
Before beginning to write your catalog-info.yaml file, ensure that you:

- 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.

   ```
   ! catalog-info.yaml 3 ●
   catalog > ! catalog-info.yaml
   Catalog Info Backstage (catalog-info.json)
   tanzu catalog-info
                     tanzu catalog-info                    Component
   ```

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.

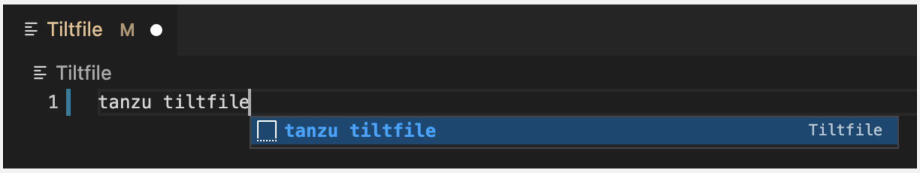
Before beginning to write your Tiltfile file, ensure that you know:

- The name of your application. For example, `my app`.
- The value of the source image. For example, `docker.io/mycompany/myapp`.
- 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.



```
tanzu tiltfile
```

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:

```
SOURCE_IMAGE = os.getenv("SOURCE_IMAGE", default='SOURCE-IMAGE')
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 +
       " --SOURCE-IMAGE " + SOURCE_IMAGE +
       " --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/c
omponent': 'run'}])
allow_k8s_contexts('CONTEXT-NAME')
```

Where:

- `SOURCE-IMAGE` is the value of source image.
- `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.

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 from there if it was not removed.

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. Open the Tiltfile and replace `your-registry.io/project` with your container registry.

Next steps

Proceed to Using Tanzu Developer Tools for VS Code.

Using Tanzu Developer Tools for VS Code

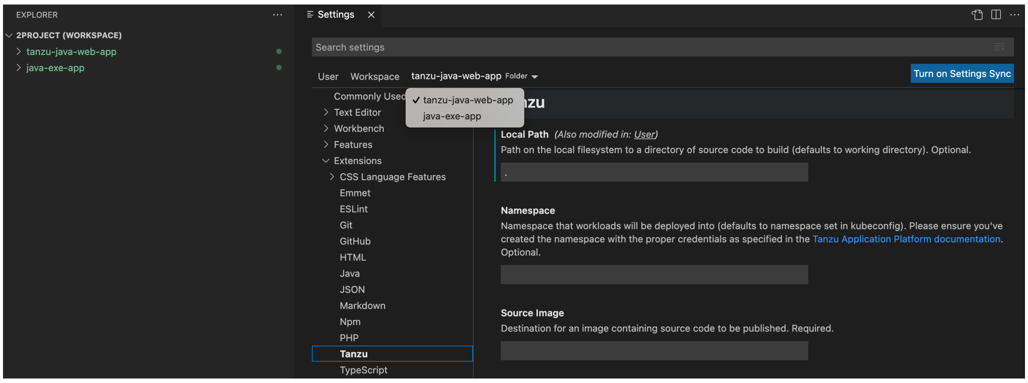
This topic tells you how to use VMware Tanzu Developer Tools for Visual Studio Code (VS Code).

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**.



Local Path *(Also modified in: User)*

Path on the local filesystem to a directory of source code to build (defaults to working directory). Optional.

.

Namespace

Namespace that workloads will be deployed into (defaults to namespace set in kubeconfig). Please ensure you've created the namespace with the proper credentials as specified in the Tanzu Application Platform documentation. Optional.

Source Image

Destination for an image containing source code to be published. Required.

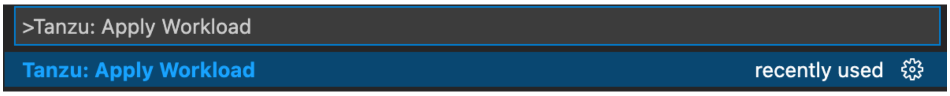
Apply a workload

The extension enables you to apply workloads on your Kubernetes cluster that has Tanzu Application Platform.

To apply a workload from the Command Palette:

1. Pressing ⇧⌘P on macOS to open the Command Palette.

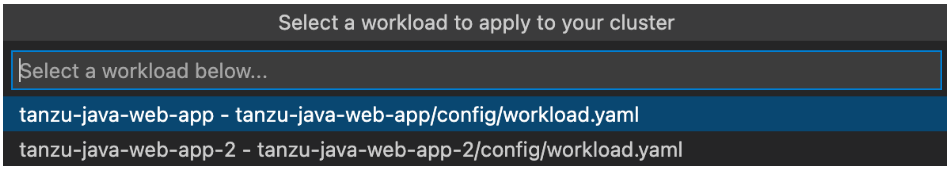
2. Run `Tanzu: Apply Workload`.



>Tanzu: Apply Workload

Tanzu: Apply Workload recently used

3. If there are multiple projects with workloads, select the workload to apply.



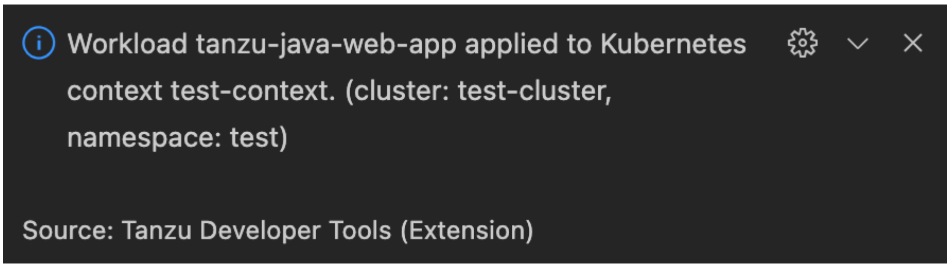
Select a workload to apply to your cluster

Select a workload below...

tanzu-java-web-app - tanzu-java-web-app/config/workload.yaml

tanzu-java-web-app-2 - tanzu-java-web-app-2/config/workload.yaml

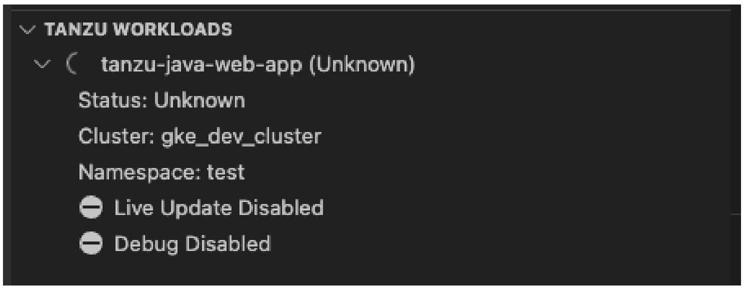
A notification appears showing that the workload was applied.



Workload tanzu-java-web-app applied to Kubernetes context test-context. (cluster: test-cluster, namespace: test)

Source: Tanzu Developer Tools (Extension)

A new workload appears on the Tanzu Workloads panel.



TANZU WORKLOADS

tanzu-java-web-app (Unknown)

Status: Unknown

Cluster: gke_dev_cluster

Namespace: test

Live Update Disabled

Debug Disabled

The Workloads panel shows the workloads running in the namespace that is defined in the current Kubernetes context.

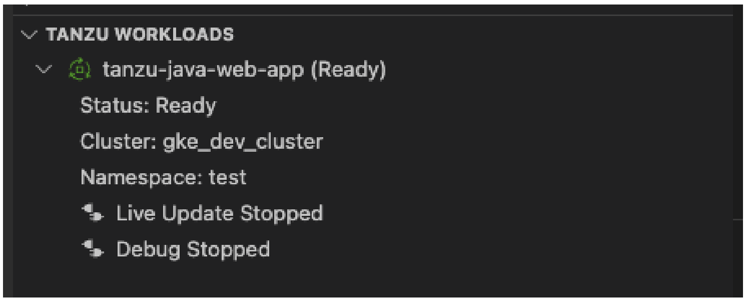
4. (Optional) See the context and namespace currently configured by running:

```
kubectl config get-contexts
```

5. (Optional) Set a namespace for the current context by running:

```
kubectl config set-context --current --namespace=YOUR-NAMESPACE
```

After the workload is deployed, the status on the Tanzu Workloads panel changes to `Ready`.



TANZU WORKLOADS

tanzu-java-web-app (Ready)

Status: Ready

Cluster: gke_dev_cluster

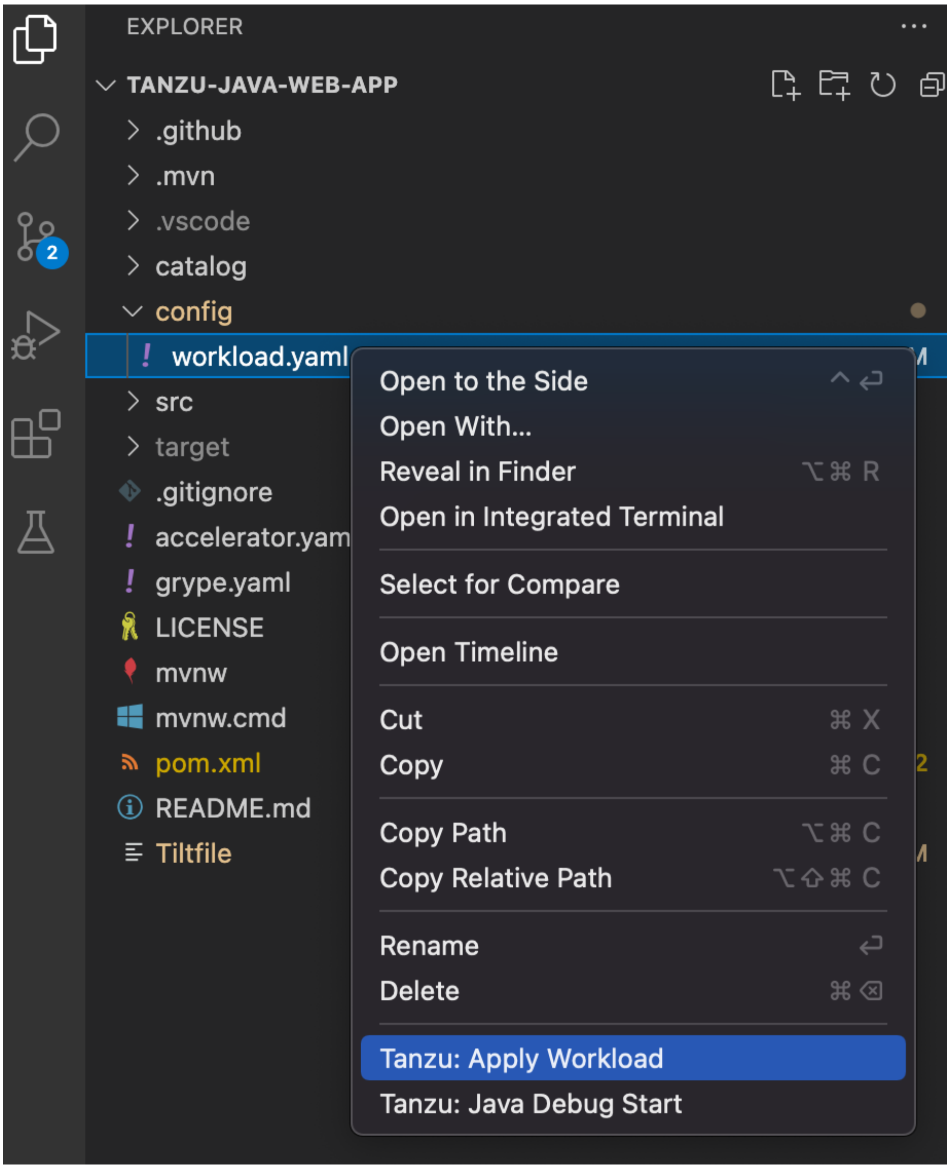
Namespace: test

Live Update Stopped

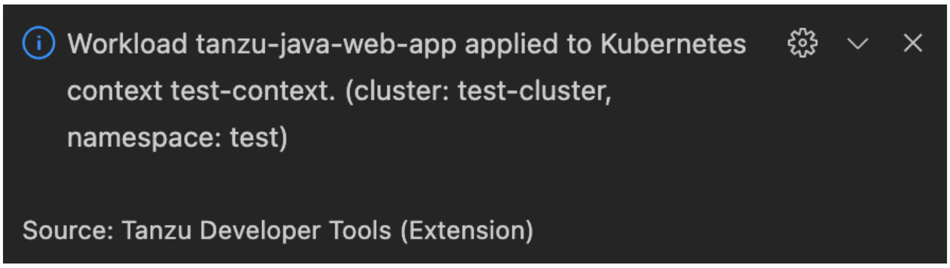
Debug Stopped

To apply a workload from the context menu:

1. Apply a workload from the context menu by right-clicking on your workload file and selecting **Tanzu: Apply Workload**.



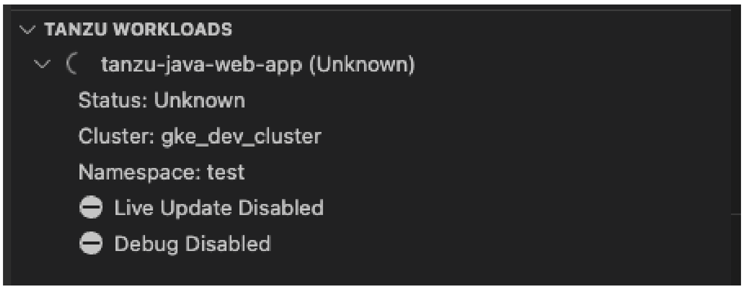
A notification appears showing that the workload was applied.



Workload tanzu-java-web-app applied to Kubernetes context test-context. (cluster: test-cluster, namespace: test)

Source: Tanzu Developer Tools (Extension)

A new workload appears on the Tanzu Workloads panel.



TANZU WORKLOADS
tanzu-java-web-app (Unknown)
Status: Unknown
Cluster: gke_dev_cluster
Namespace: test
Live Update Disabled
Debug Disabled

The workloads panel shows the workloads running in the namespace that is defined in the current Kubernetes context.

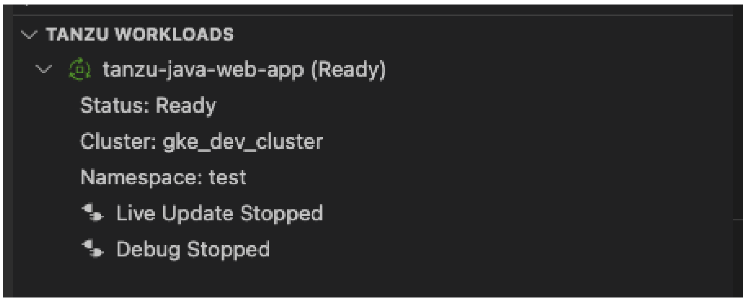
2. (Optional) See the context and namespace currently configured by running:

   ```
   kubectl config get-contexts
   ```

3. (Optional) Set a namespace for the current context by running:

   ```
   kubectl config set-context --current --namespace=YOUR-NAMESPACE
   ```

   After the workload is deployed, the status on the Tanzu Workloads panel changes to **Ready**.



TANZU WORKLOADS

tanzu-java-web-app (Ready)

Status: Ready

Cluster: gke_dev_cluster

Namespace: test

Live Update Stopped

Debug Stopped

Debugging on the cluster

The extension enables you to debug your application on your Kubernetes cluster that has Tanzu Application Platform.

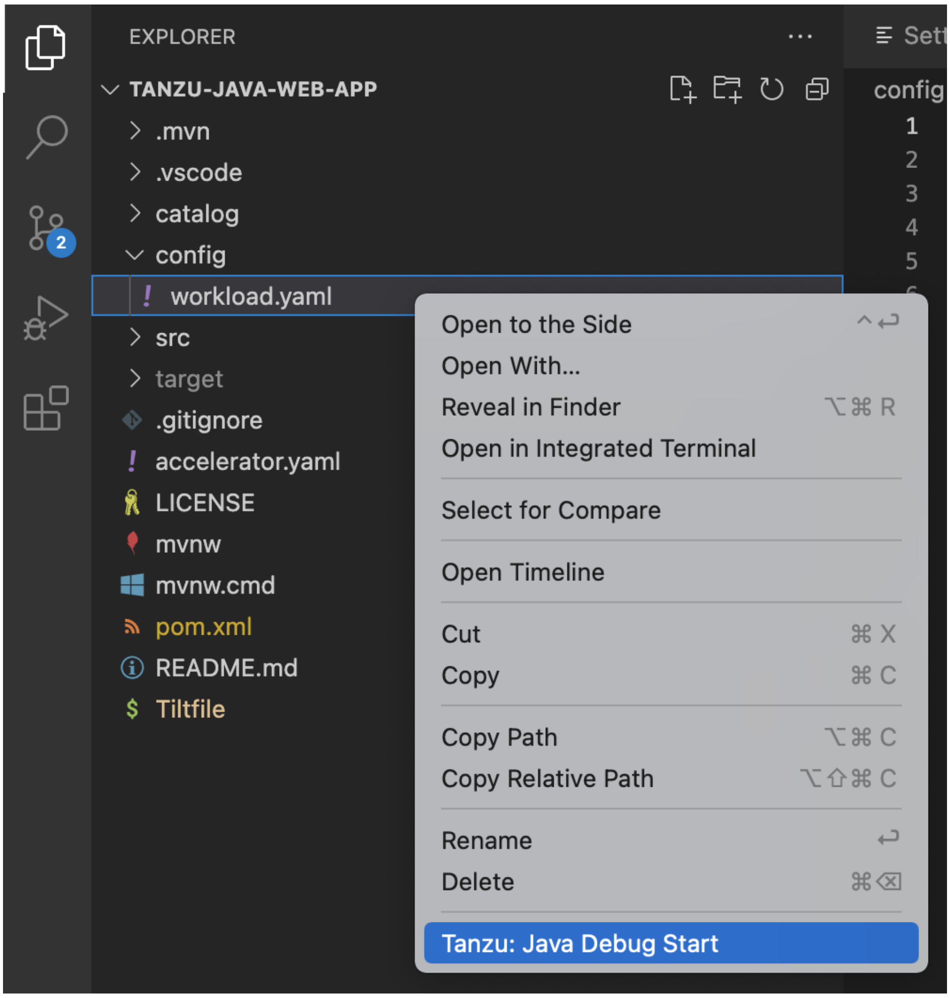
Debugging requires a `workload.yaml` file in your project. For information about creating a `workload.yaml` file, see Set up Tanzu Developer Tools.

Debugging on the cluster and Live Update cannot be used simultaneously. If you use Live Update for the current project, ensure that you stop the Tanzu Live Update Run Configuration before attempting to debug on the cluster. For more information, see Stop Live Update.

Start debugging on the cluster

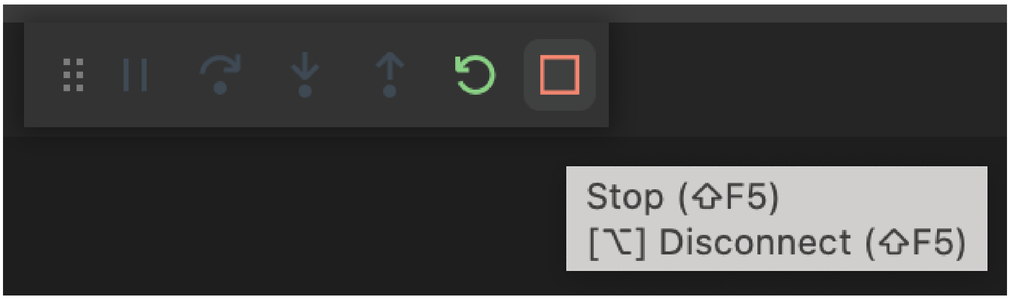
To start debugging on the cluster:

1. Add a breakpoint in your code.
2. Right-click the `workload.yaml` file in your project.
3. Select **Debug 'Tanzu Debug Workload...'** in the pop-up menu.

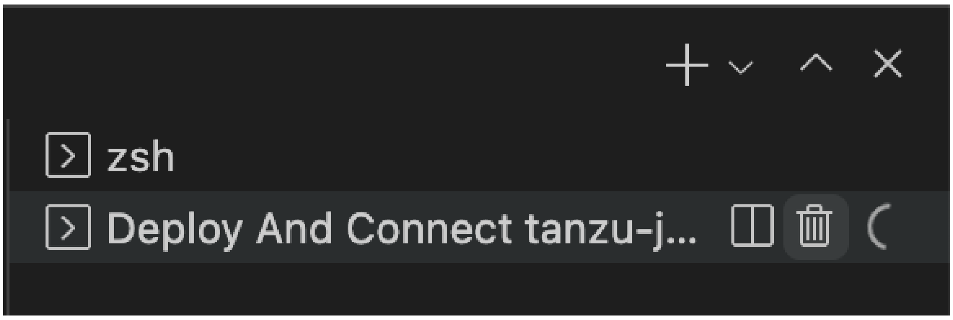


Stop Debugging on the cluster

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.



Live Update

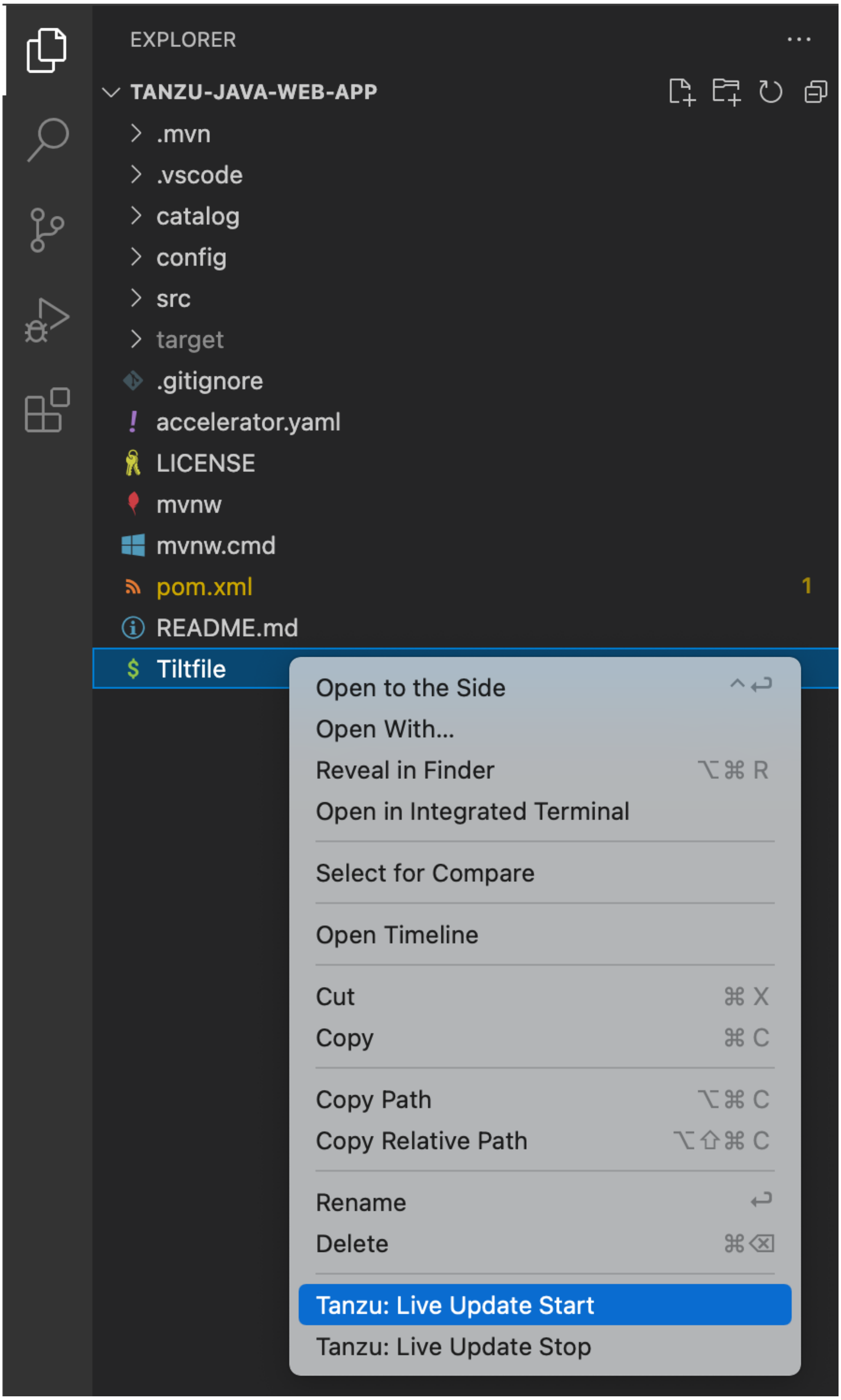
With the use of Live Update facilitated by Tilt, the extension enables you to deploy your workload once, save changes to the code, and see those changes reflected in the workload running on the cluster within seconds.

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 Set up Tanzu Developer Tools.

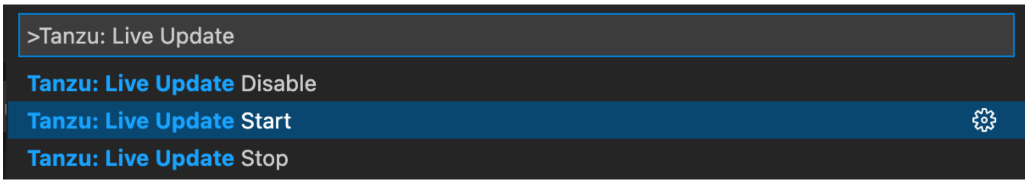
Live Update and Debugging on the cluster cannot be used simultaneously. If you are currently debugging on the cluster, stop debugging before attempting to use Live Update.

Start Live Update

You can start Live Update by right-clicking anywhere in the VS Code project explorer and then clicking **Tanzu: Live Update Start** in the pop-up menu.



Alternatively, you can press `⇧⌘P` to open the Command Palette and run the `Tanzu: Live Update Start` command.



>Tanzu: Live Update

Tanzu: Live Update Disable

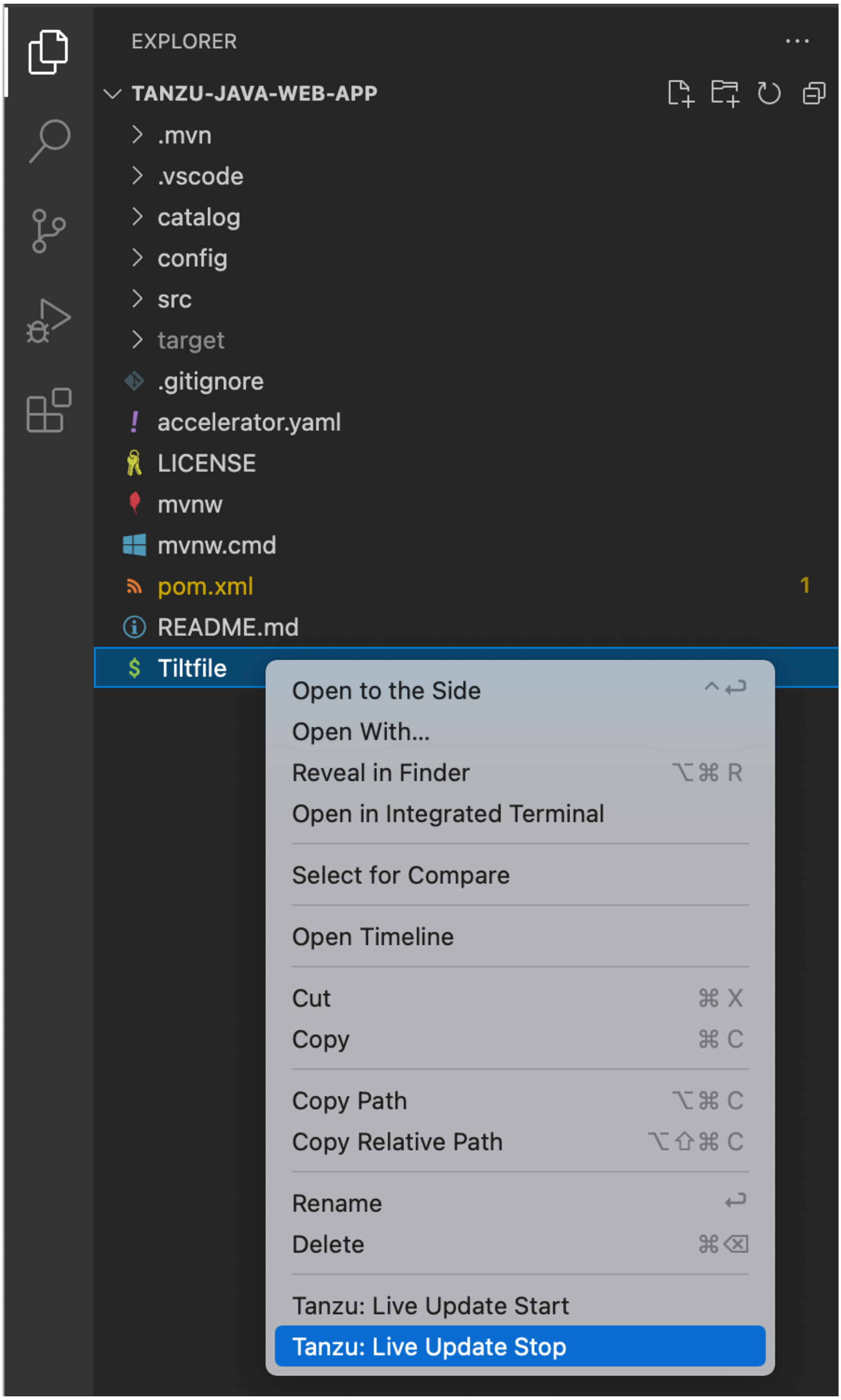
Tanzu: Live Update Start

Tanzu: Live Update Stop

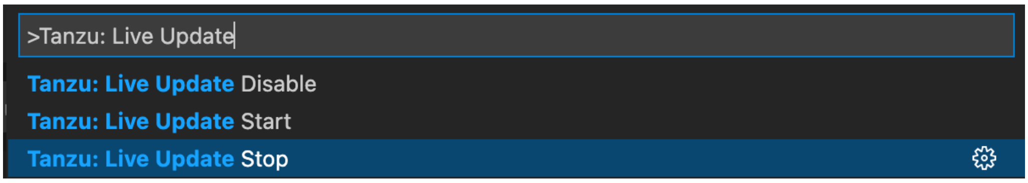
Stop Live Update

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.

You can stop Live Update by right-clicking your project's Tiltfile and selecting `Tanzu: Live Update Stop`.



Alternatively, you can press ⇧⌘P to open the Command Palette and then run `Tanzu: Live Update Stop`.



>Tanzu: Live Update

Tanzu: Live Update Disable

Tanzu: Live Update Start

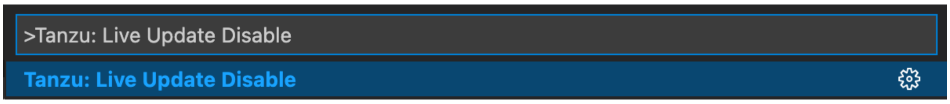
Tanzu: Live Update Stop

Deactivate Live Update

You can remove the Live Update capability from your application entirely. This option can be useful in a troubleshooting scenario. Deactivating Live Update redeploys your workload to the cluster and removes the Live Update capability.

To deactivate Live Update:

1. Press ⇧⌘P to open the Command Palette.
2. Run `Tanzu: Live Update Disable`.



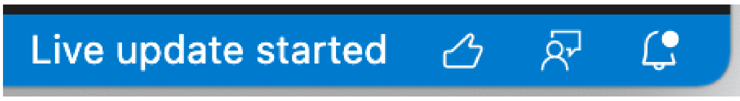
>Tanzu: Live Update Disable

Tanzu: Live Update Disable

3. Type the name of the workload for which you want to deactivate Live Update.

Live Update status

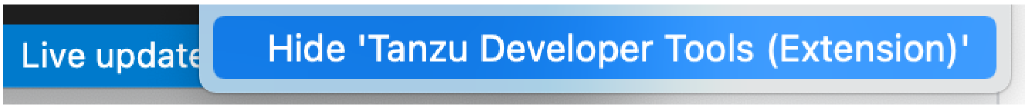
The current status of Live Update is visible on the right side of the status bar at the bottom of the VS Code window.



The Live Update status bar entry shows the following states:

- Live Update Stopped
- Live Update Starting…
- Live Update Running

To hide the Live Update status bar entry, right-click it and select **Hide ‘Tanzu Developer Tools (Extension)’**.

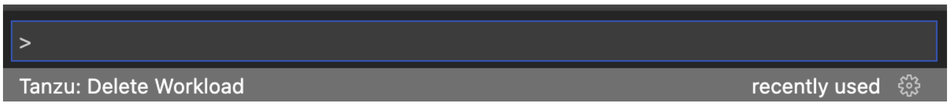


Delete a workload

The extension enables you to delete workloads on your Kubernetes cluster that has Tanzu Application Platform.

To delete a workload:

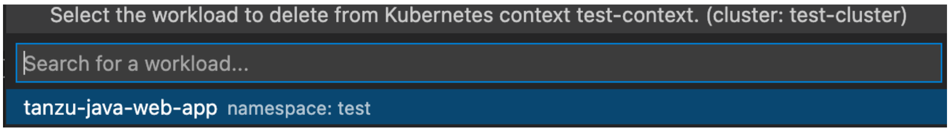
1. Open the Command Palette by pressing ⇧⌘P on macOS.
2. Run `Tanzu: Delete Workload`.



>

Tanzu: Delete Workload recently used

3. Select the workload to delete.

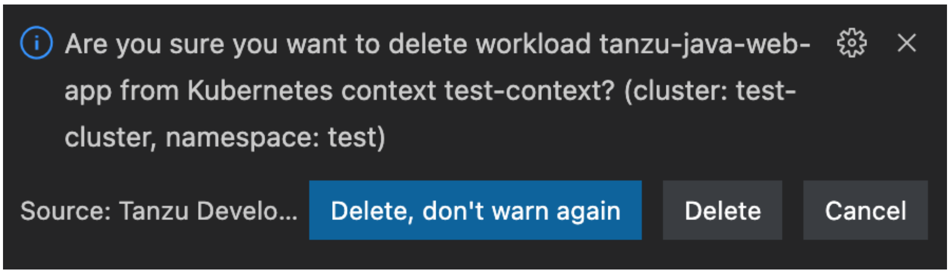


Select the workload to delete from Kubernetes context test-context. (cluster: test-cluster)

Search for a workload...

tanzu-java-web-app namespace: test

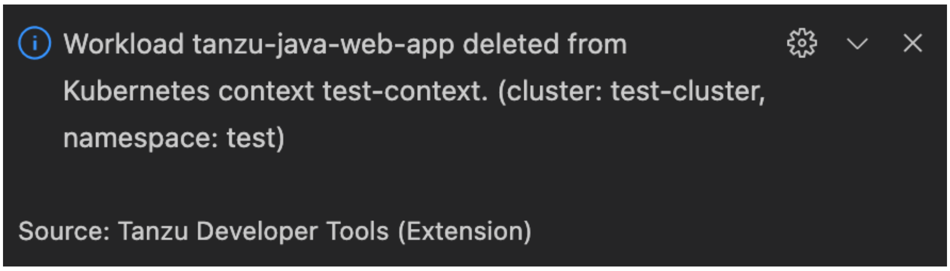
If the **Tanzu: Confirm Delete** setting is enabled, a message appears that prompts you to delete the workload and not warn again, delete the workload, or cancel.



Are you sure you want to delete workload tanzu-java-web-app from Kubernetes context test-context? (cluster: test-cluster, namespace: test)

Source: Tanzu Develo...

A notification appears showing that the workload was deleted.



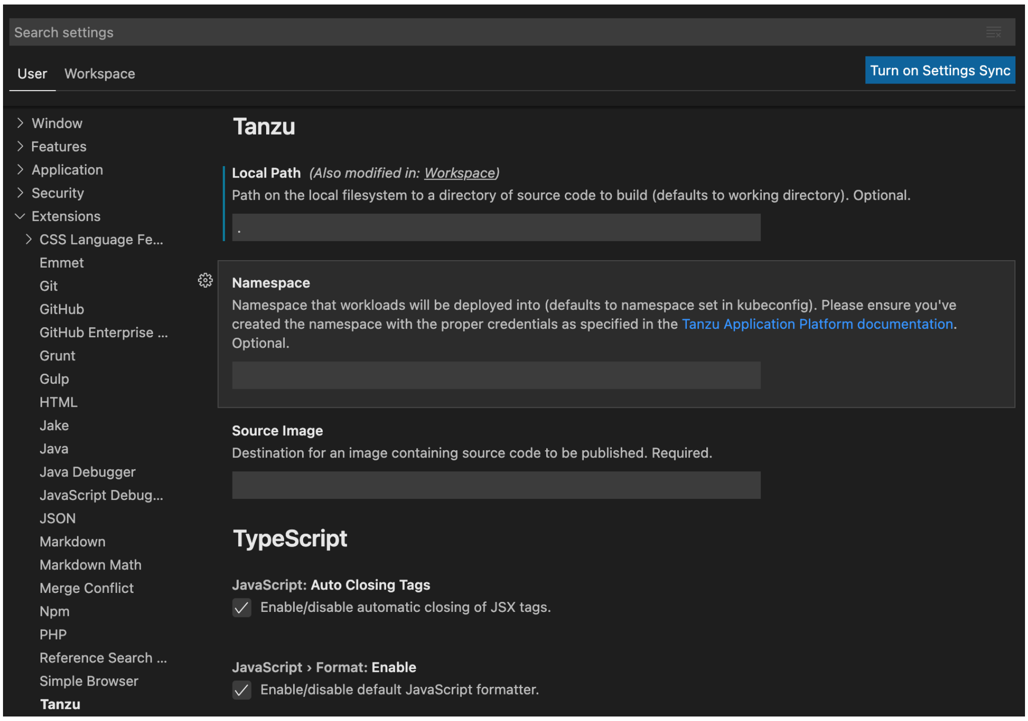
Workload tanzu-java-web-app deleted from Kubernetes context test-context. (cluster: test-cluster, namespace: test)

Source: Tanzu Developer Tools (Extension)

Switch namespaces

To switch the namespace where you created the workload:

1. Go to **Code** > **Preferences** > **Settings**.
2. Expand the **Extensions** section of the settings and select **Tanzu**.
3. In the **Namespace** option, add the namespace you want to deploy to. This is the `default` namespace by default.



Tanzu

Local Path *(Also modified in: Workspace)*
Path on the local filesystem to a directory of source code to build (defaults to working directory). Optional.
.

Namespace
Namespace that workloads will be deployed into (defaults to namespace set in kubeconfig). Please ensure you've created the namespace with the proper credentials as specified in the Tanzu Application Platform documentation. Optional.

Source Image
Destination for an image containing source code to be published. Required.

TypeScript

JavaScript: **Auto Closing Tags**
☑ Enable/disable automatic closing of JSX tags.

JavaScript › Format: **Enable**
☑ Enable/disable default JavaScript formatter.

Tanzu Workloads panel

The current state of the workloads is visible on the Tanzu Workloads panel in the bottom left corner of the VS Code window. The panel shows the current status of each workload, namespace, and cluster. It also shows whether Live Update and Debug is running, stopped, or deactivated.

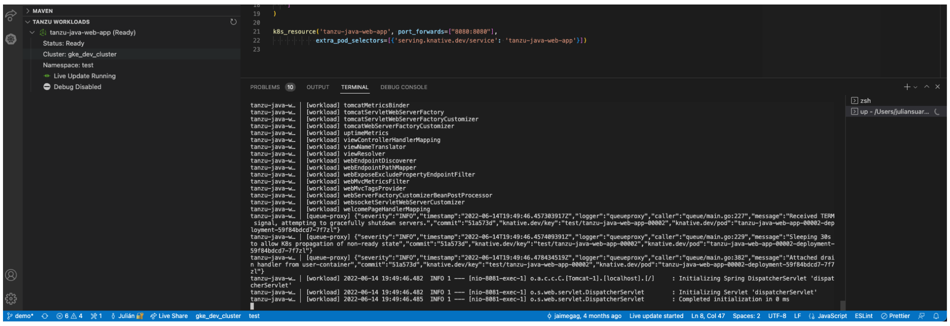
The Tanzu Workloads panel uses the cluster and namespace specified in the current kubectl context.

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
```



```
)

k8s_resource('tanzu-java-web-app', port_forwards=["8080:8080"],
            extra_pod_selectors=[{'serving.knative.dev/service': 'tanzu-java-web-app'}])
```

```
tanzu-java-w… | [workload] tomcatMetricsBinder
tanzu-java-w… | [workload] tomcatServletWebServerFactory
tanzu-java-w… | [workload] tomcatServletWebServerFactoryCustomizer
tanzu-java-w… | [workload] tomcatWebServerFactoryCustomizer
tanzu-java-w… | [workload] uptimeMetrics
tanzu-java-w… | [workload] viewControllerHandlerMapping
tanzu-java-w… | [workload] viewNameTranslator
tanzu-java-w… | [workload] viewResolver
tanzu-java-w… | [workload] webEndpointDiscoverer
tanzu-java-w… | [workload] webEndpointPathMapper
tanzu-java-w… | [workload] webExposeExcludePropertyEndpointFilter
tanzu-java-w… | [workload] webMvcMetricsFilter
tanzu-java-w… | [workload] webMvcTagsProvider
tanzu-java-w… | [workload] webServerFactoryCustomizerBeanPostProcessor
tanzu-java-w… | [workload] websocketServletWebServerCustomizer
tanzu-java-w… | [workload] welcomePageHandlerMapping
tanzu-java-w… | [queue-proxy] {"severity":"INFO","timestamp":"2022-06-14T19:49:46.457303917Z","logger":"queueproxy","caller":"queue/main.go:227","message":"Received TERM
 signal, attempting to gracefully shutdown servers.","commit":"51a573d","knative.dev/key":"test/tanzu-java-web-app-00002","knative.dev/pod":"tanzu-java-web-app-00002-dep
loyment-59f84bdcd7-7f7zl"}
tanzu-java-w… | [queue-proxy] {"severity":"INFO","timestamp":"2022-06-14T19:49:46.457409391Z","logger":"queueproxy","caller":"queue/main.go:229","message":"Sleeping 30s
to allow K8s propagation of non-ready state","commit":"51a573d","knative.dev/key":"test/tanzu-java-web-app-00002","knative.dev/pod":"tanzu-java-web-app-00002-deployment-
59f84bdcd7-7f7zl"}
tanzu-java-w… | [queue-proxy] {"severity":"INFO","timestamp":"2022-06-14T19:49:46.478434519Z","logger":"queueproxy","caller":"queue/main.go:382","message":"Attached drai
n handler from user-container","commit":"51a573d","knative.dev/key":"test/tanzu-java-web-app-00002","knative.dev/pod":"tanzu-java-web-app-00002-deployment-59f84bdcd7-7f7
zl"}
tanzu-java-w… | [workload] 2022-06-14 19:49:46.482  INFO 1 --- [nio-8081-exec-1] o.a.c.c.C.[Tomcat-1].[localhost].[/]     : Initializing Spring DispatcherServlet 'dispat
cherServlet'
tanzu-java-w… | [workload] 2022-06-14 19:49:46.482  INFO 1 --- [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
tanzu-java-w… | [workload] 2022-06-14 19:49:46.485  INFO 1 --- [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 0 ms
```

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.

Integrating Live Hover by using Spring Boot Tools (Experimental)

For more information about this feature, see the **Live application information hovers** section of VS Code documentation for Spring Boot Tools.

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 Tools extension v1.33 or later

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/sample-accelerators/tanzu-java-web-app`.

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, `./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:

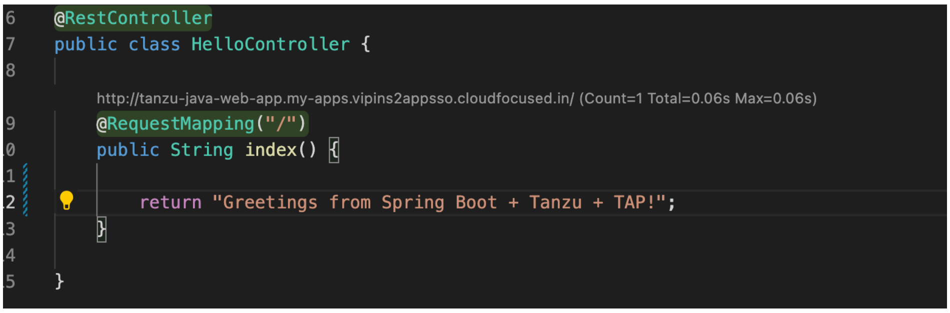
   ```
   kubectl get pods
   ```

   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.

Troubleshooting 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).

First debugging session ends prematurely

Symptom

When debugging an application with service bindings, the first debugging sessions ends prematurely.

Cause

This issue arises as a consequence of the services being late-bound.

Solution

Start another debugging session.

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.

Warning notification when canceling an action

Symptom

When running `Tanzu: Debug Start`, `Tanzu: Live Update Start`, or `Tanzu: Apply`, a quick-pick list appears when there are multiple options. If you cancel, by either pressing the ESC key or clicking outside the list, a warning notification appears that says no workloads or Tiltfiles were found.

Cause

GKE authentication was extracted into a separate plug-in and is no longer inside the Kubernetes client or libraries.

Solution

No action is needed. You can ignore this warning.

Live update might not work when using server or worker Workload types

Symptom

When using `server` or `worker` as a workload type, live update might not work.

Cause

The default pod selector used to check when a pod is ready to do live update is incorrectly using the label `'serving.knative.dev/service': '<workload_name>'`. This label is not present on `server` or `worker` workloads.

Solution

Go to the project's `Tiltfile`, look for the `k8s_resource` line, and modify the `extra_pod_selectors` parameter to use any pod selector that matches your workload. For example:

```
extra_pod_selectors=[{'carto.run/workload-name': '<workload_name>', 'app.kubernetes.i
o/component': 'run', 'app.kubernetes.io/part-of': '<workload_name>'}]
```

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.de
v/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.

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.

The extension currently only supports Java apps.

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.

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.

The extension currently only supports Java apps.

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.

Next steps

Follow the steps to install the extension.

Installing 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 macOS and Java applications. The extension currently supports IntelliJ IDEA v2021.1 to v2022.1.

Prerequisites

Before installing the extension, you must have:

- IntelliJ
- kubectl
- Tilt v0.27.2 or later
- Tanzu CLI and plug-ins
- A cluster with the Tanzu Application Platform Full profile or Iterate profile

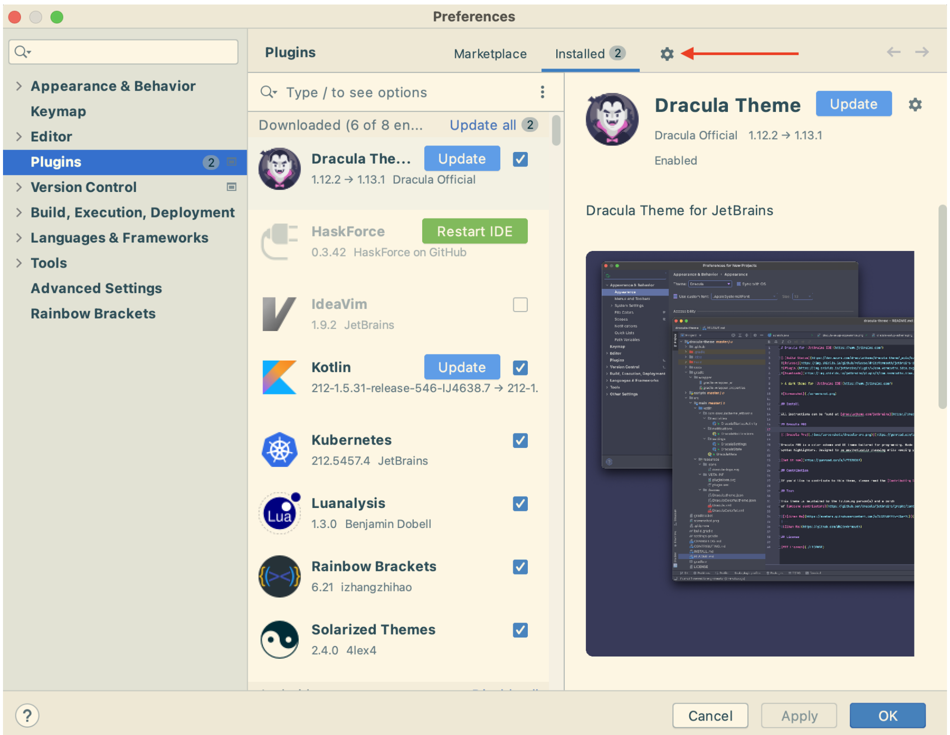
If you are an app developer, someone else in your organization might have already set up the Tanzu Application Platform environment.

Install

To install VMware Tanzu Developer Tools for IntelliJ:

1. Download VMware Tanzu Developer Tools for IntelliJ from the VMware Tanzu Network.
2. Open IntelliJ.
3. Open the **Preferences** pane and then go to **Plugins**.

4. Click the gear icon and then click **Install Plugin from disk…**.



5. Use the file picker to select the ZIP file downloaded from the VMware Tanzu Network.

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.

Getting Started with Tanzu Developer Tools for IntelliJ

This topic describes how to set up the Tanzu Developer Tools for IntelliJ extension for your project.

Overview

The Tanzu Developer Tools for IntelliJ extension makes use of the following files within your project.

- `workload.yaml`: Required
- `catalog-info.yaml`: Required for Tanzu Application Platform GUI integration
- `Tiltfile`: Required for live update
- `.tanzuignore`: Recommended

You can create these files manually using the instructions in this topic, or use the files in the View an example project section.

Prerequisites

Before you get started, ensure you have completed Installing Tanzu Developer Tools for IntelliJ.

Create the workload.yaml file

In your project, you must include a file named `workload.yaml`, for example, `my-project/config/workload.yaml`.

The `workload.yaml` file provides instructions to the Supply Chain Choreographer about how to build and manage a workload. For more information, see the Supply Chain Choreographer documentation.

The Tanzu Developer Tools for IntelliJ extension requires only one `workload.yaml` file per project. The `workload.yaml` must be a single-document YAML file, not a multi-document YAML file.

Example workload.yaml

The following is an example `workload.yaml`:

```
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 about the Tanzu CLI command, see Tanzu apps workload create in the Tanzu CLI documentation.

Create the catalog-info.yaml file

In your project, you must include a file named `catalog-info.yaml`, for example, `my-project/catalog/catalog-info.yaml`.

The `catalog-info.yaml` file enables the workloads created with the Tanzu Developer Tools for IntelliJ extension to be visible in the Tanzu Application Platform GUI. For more information, see the Tanzu Application Platform GUI documentation.

Example catalog-info.yaml

The following is an example `catalog-info.yaml`:

```
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), for example, `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.

Example Tiltfile

The following is an example `Tiltfile`:

```
SOURCE_IMAGE = os.getenv("SOURCE_IMAGE", default='SOURCE-IMAGE-VALUE')
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 +
       " --source-image " + SOURCE_IMAGE +
       " --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=[{'serving.knative.dev/service': 'APP-NAME'}])
allow_k8s_contexts('CONTEXT-NAME')
```

Where:

- `SOURCE-IMAGE-VALUE` is your source image.
- `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.

Example `.tanzuignore`

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 will 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 determine 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 tanzu-java-web-app repository from GitHub.
2. Open the `Tiltfile` and replace `your-registry.io/project` with your registry.

Next steps

- Using the Tanzu Developer Tools for IntelliJ extension.

Using Tanzu Developer Tools for IntelliJ

Ensure the project you want to use the extension with has the required files specified in Getting started.

The Tanzu Developer Tools 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.

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.

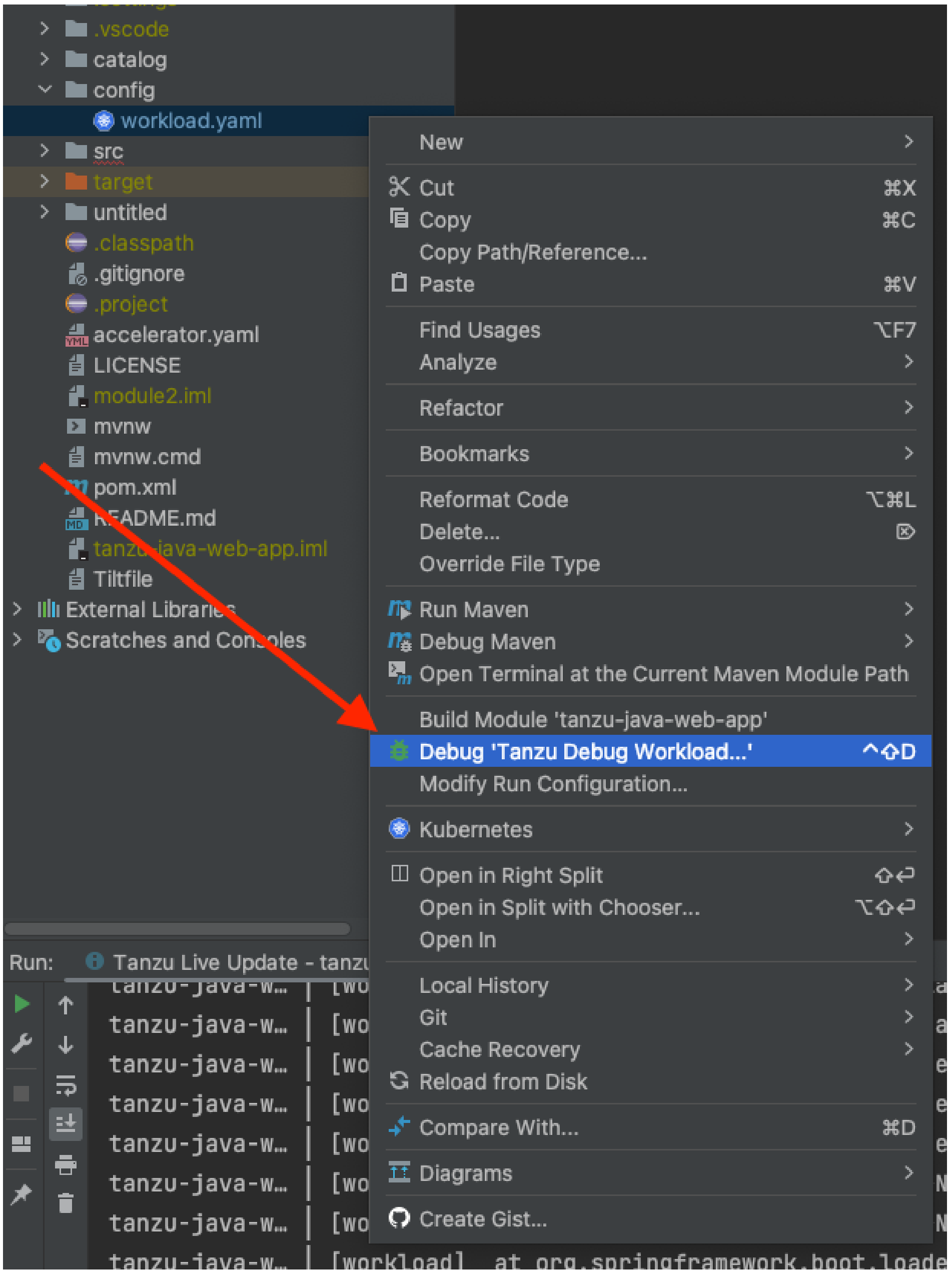
Debugging on the cluster and Live Update cannot be used simultaneously. If you use Live Update for the current project, ensure that you stop the Tanzu Live Update Run Configuration before attempting to debug on the cluster.

Start Debugging on the Cluster

To start debugging on the cluster:

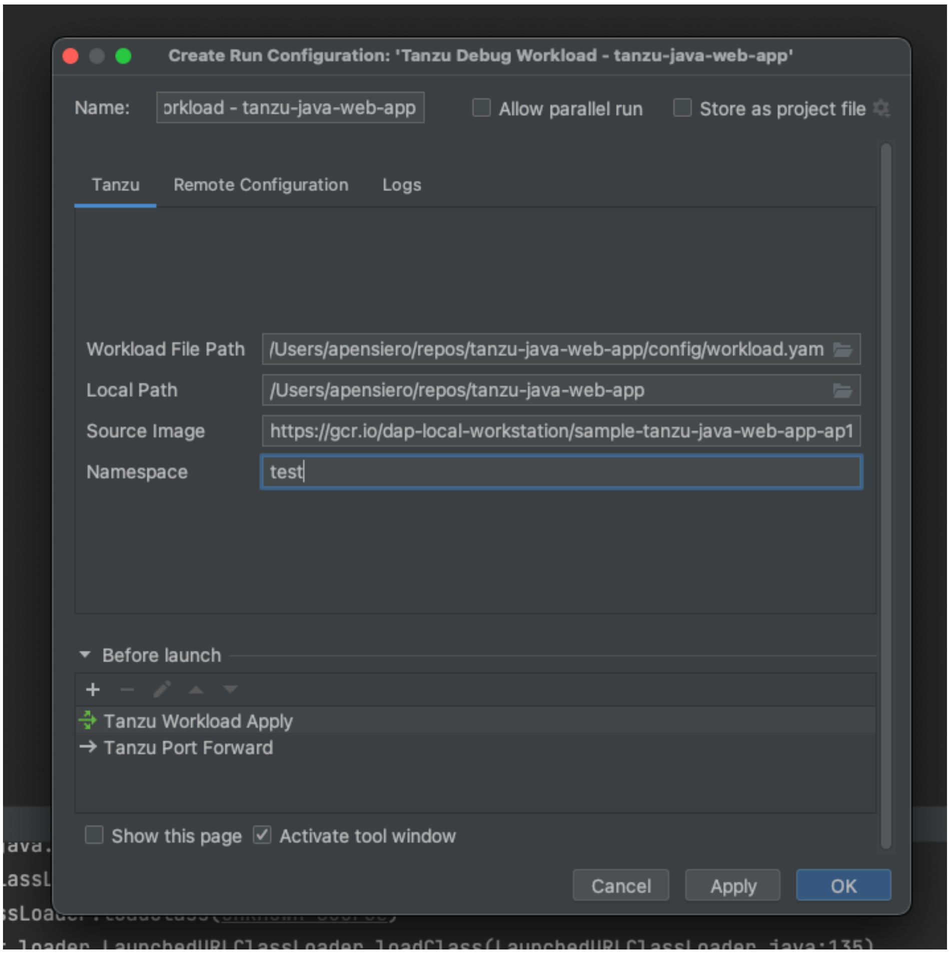
1. Add a breakpoint in your code.

2. Right-click the `workload.yaml` file in your project.
3. Select **Debug 'Tanzu Debug Workload...'** in the pop-up menu.



```
tanzu-java-w… | [wo
tanzu-java-w… | [wo
tanzu-java-w… | [wo
tanzu-java-w… | [wo
tanzu-java-w… | [wo
tanzu-java-w… | [wo
tanzu-java-w… | [wo
tanzu-java-w… | [workload]  at org.springframework.boot.loade
```

4. Ensure 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.
   - **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.



Create Run Configuration: 'Tanzu Debug Workload - tanzu-java-web-app'

Name: orkload - tanzu-java-web-app

□ Allow parallel run □ Store as project file

Workload File Path: /Users/apensiero/repos/tanzu-java-web-app/config/workload.yam

Local Path: /Users/apensiero/repos/tanzu-java-web-app

Source Image: https://gcr.io/dap-local-workstation/sample-tanzu-java-web-app-ap1

Namespace: test

Before launch

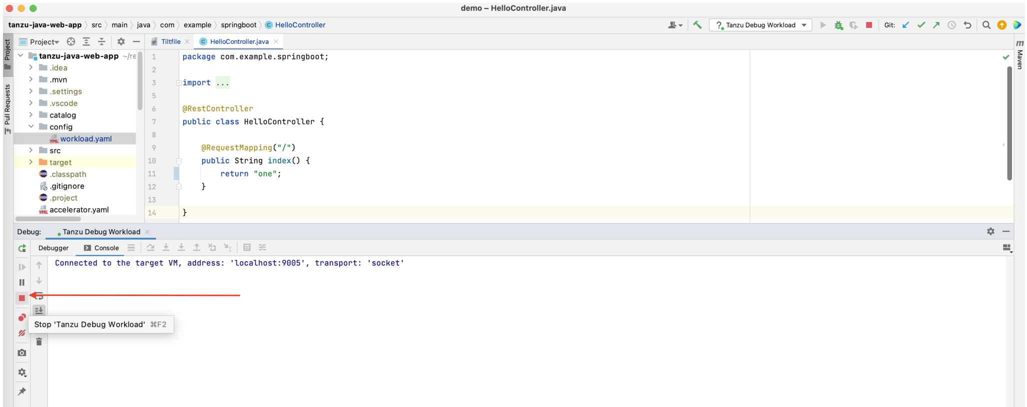
- Tanzu Workload Apply
- Tanzu Port Forward

□ Show this page ☑ Activate tool window

5. 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.



demo – HelloController.java

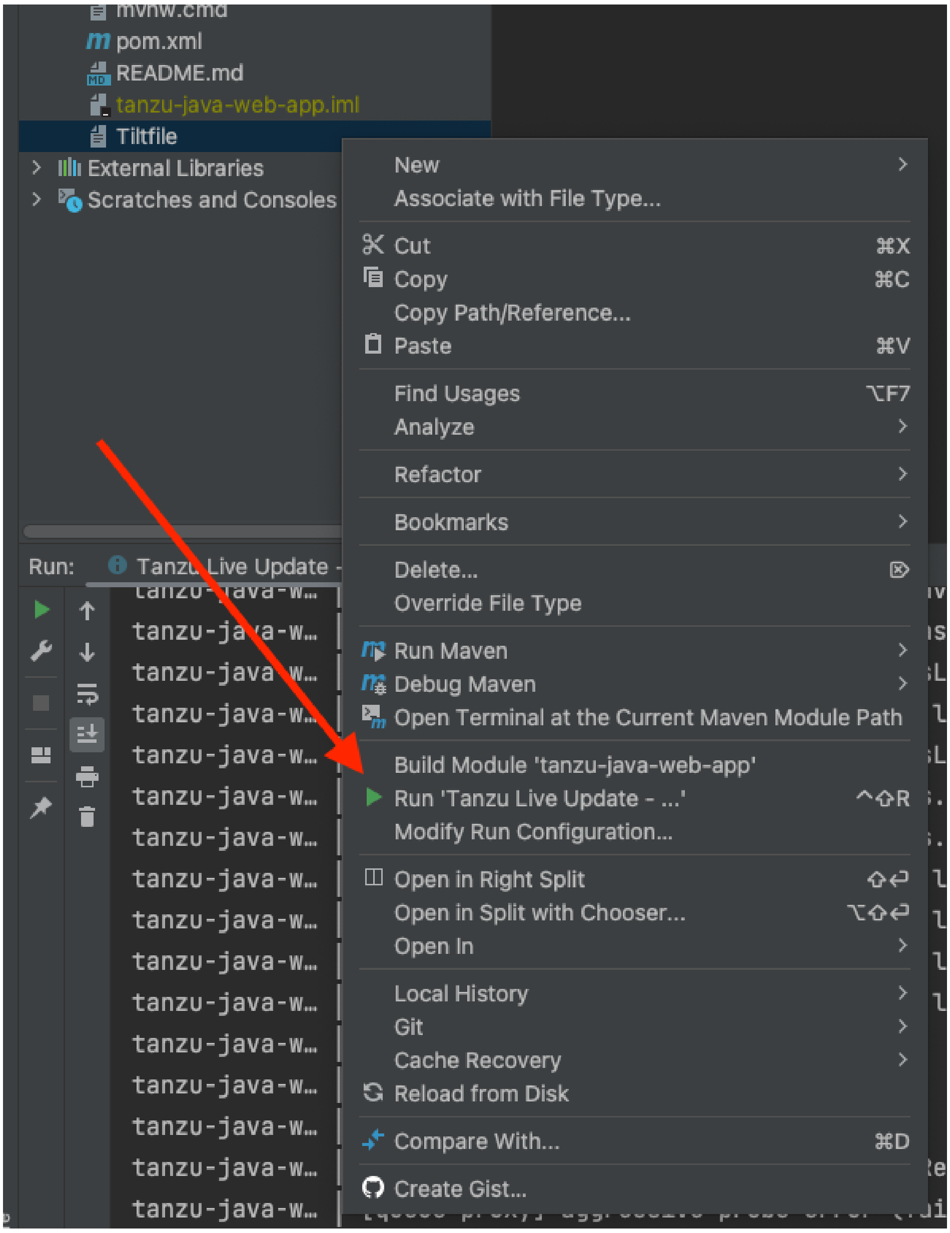
```
package com.example.springboot;

import ...

@RestController
public class HelloController {

    @RequestMapping("/")
    public String index() {
        return "one";
    }

}
```

```
Connected to the target VM, address: 'localhost:9005', transport: 'socket'
```

Start Live Update

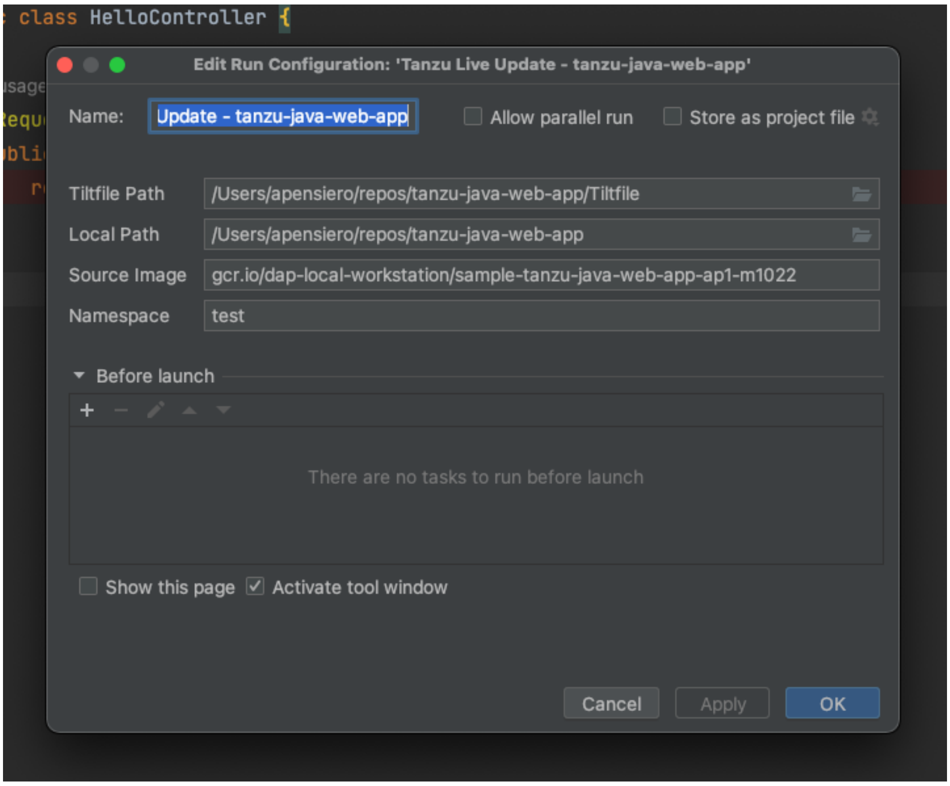
1. Stop any running sessions. Only one Live Update session can be active at a time.

2. Right-click your project's Tiltfile and select **Run 'Tanzu Live Update - …'**.



3. Ensure 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.
   - **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.



Edit Run Configuration: 'Tanzu Live Update - tanzu-java-web-app'

Name: Update - tanzu-java-web-app ☐ Allow parallel run ☐ Store as project file

Tiltfile Path: /Users/apensiero/repos/tanzu-java-web-app/Tiltfile

Local Path: /Users/apensiero/repos/tanzu-java-web-app

Source Image: gcr.io/dap-local-workstation/sample-tanzu-java-web-app-ap1-m1022

Namespace: test

Before launch

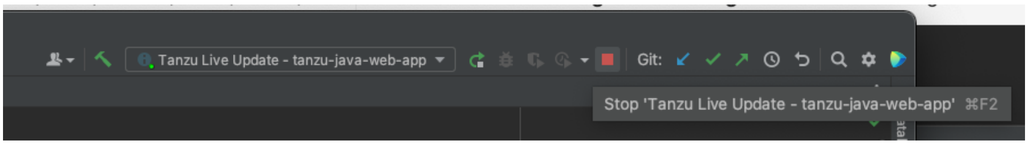
There are no tasks to run before launch

☐ Show this page ☑ Activate tool window

Important: 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 currently running Tanzu Live Update Run Configuration.



Stop 'Tanzu Live Update - tanzu-java-web-app' ⌘F2

Troubleshooting Tanzu Developer Tools for IntelliJ

This topic helps you troubleshoot issues with Tanzu Developer Tools for IntelliJ.

First debugging session ends prematurely

Symptom

When debugging an application with service bindings, the first debugging sessions ends prematurely.

Cause

This issue arises as a consequence of the services being late-bound.

Solution

Start another debugging session.

No support for multiple IntelliJ windows

Symptom: A notification appears saying that the Tanzu Language Server has failed to start.

Cause: You have more than one IntelliJ Project open in a single window.

Solution: Close the other windows, quit IntelliJ, and re-open IntelliJ.

Unqualified paths for Workload File Path and Local Path properties

Symptom: Debug and Live Update are unable to find their paths

Cause: Unqualified paths for the Workload File Path and Local Path properties on Tanzu Debug and Tanzu Live Update Run Configurations.

Solution: Make the Workload File Path and Local Path properties on your Debug and Live Update configurations the fully qualified paths. Use the file picker on each input field to help you enter the correct values.

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 might not work when using server or worker Workload types

Symptom

When using `server` or `worker` as a workload type, live update might not work.

Cause

The default pod selector used to check when a pod is ready to do live update is incorrectly using the label `'serving.knative.dev/service': '<workload_name>'`. This label is not present on `server` or `worker` workloads.

Solution

Go to the project's `Tiltfile`, look for the `k8s_resource` line, and modify the `extra_pod_selectors` parameter to use any pod selector that matches your workload. For example:

```
extra_pod_selectors=[{'carto.run/workload-name': '<workload_name>', 'app.kubernetes.i
o/component': 'run', 'app.kubernetes.io/part-of': '<workload_name>'}]
```

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.de
v/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.

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. In their own words:

“Tilt automates all the steps from a code change to a new process: watching files, building container images, and bringing your environment up-to-date.”

This means that while using Live Update, all you have to do is save your code changes to see them reflected in your application running on your cluster. No redeploy is necessary.

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 the Tanzu Application Platform GUI.

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."

Overview of API portal for VMware Tanzu

You can use API portal for VMware Tanzu to find APIs you can use in your own applications. 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.

For more information about Tanzu API portal, see API portal for VMware Tanzu.

Overview of API portal for VMware Tanzu

You can use API portal for VMware Tanzu to find APIs you can use in your own applications. 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.

For more information about Tanzu API portal, see API portal for VMware Tanzu.

Install Tanzu API portal

This topic tells you how to install 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 Tanzu API portal:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.

Install

To install Tanzu API portal:

1. Check 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) Make changes to the default installation settings by running:

   ```
   tanzu package available get api-portal.tanzu.vmware.com/VERSION-NUMBER --values
   -schema --namespace tap-install
   ```

   Where `VERSION-NUMBER` is the version of the package listed in step 1.

   For example:

   ```
   $ tanzu package available get api-portal.tanzu.vmware.com/1.0.3 --values-schema
   --namespace tap-install
   ```

   For more information about values schema options, see the individual product documentation.

3. Install API portal by running:

   ```
   tanzu package install api-portal -n tap-install -p api-portal.tanzu.vmware.com
   -v 1.0.3
   ```

   For example:

   ```
   $ tanzu package install api-portal -n tap-install -p api-portal.tanzu.vmware.co
   m -v 1.0.3

   / 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'
   ```

Overview of Tanzu Application Platform GUI

Tanzu Application Platform GUI (commonly called TAP GUI) 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 Application Platform GUI is built from the Cloud Native Computing Foundation's project Backstage.

Tanzu Application Platform GUI consists of the following components:

- **Your organization catalog:**

  The catalog serves as the primary visual representation of your running services (components) and applications (systems).

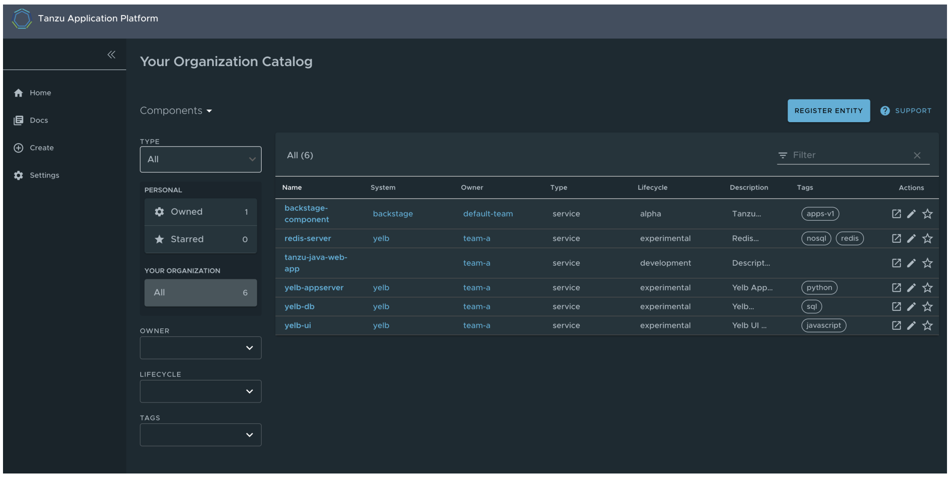
- **Tanzu Application Platform GUI plug-ins:**

  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.



Name	System	Owner	Type	Lifecycle	Description	Tags	Actions
backstage-component	backstage	default-team	service	alpha	Tanzu...	apps-v1	
redis-server	yelb	team-a	service	experimental	Redis...	nosql redis	
tanzu-java-web-app		team-a	service	development	Descript...		
yelb-appserver	yelb	team-a	service	experimental	Yelb App...	python	
yelb-db	yelb	team-a	service	experimental	Yelb...	sql	
yelb-ui	yelb	team-a	service	experimental	Yelb UI ...	javascript	

- **A Git repository:**

  Tanzu Application Platform GUI stores the following in a Git repository:

  - The structure for your application catalog.
  - Your technical documentation about the catalog items, if you enable Tanzu Application Platform GUI 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 Application Platform GUI

Tanzu Application Platform GUI (commonly called TAP GUI) 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 Application Platform GUI is built from the Cloud Native Computing Foundation's project Backstage.

Tanzu Application Platform GUI consists of the following components:

- **Your organization catalog:**

  The catalog serves as the primary visual representation of your running services (components) and applications (systems).

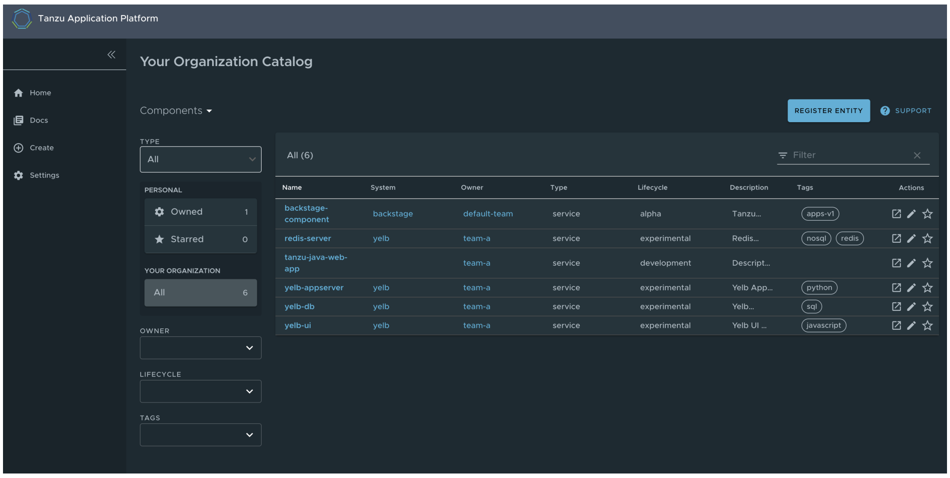
- **Tanzu Application Platform GUI plug-ins:**

  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.



Name	System	Owner	Type	Lifecycle	Description	Tags	Actions
backstage-component	backstage	default-team	service	alpha	Tanzu...	apps-v1	
redis-server	yelb	team-a	service	experimental	Redis...	nosql redis	
tanzu-java-web-app		team-a	service	development	Descript...		
yelb-appserver	yelb	team-a	service	experimental	Yelb App...	python	
yelb-db	yelb	team-a	service	experimental	Yelb...	sql	
yelb-ui	yelb	team-a	service	experimental	Yelb UI ...	javascript	

- **A Git repository:**

  Tanzu Application Platform GUI stores the following in a Git repository:

  - The structure for your application catalog.
  - Your technical documentation about the catalog items, if you enable Tanzu Application Platform GUI 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 Application Platform GUI

This topic tells you how to install Tanzu Application Platform GUI (commonly called TAP GUI) 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 Application Platform GUI. For more information about profiles, see Components and installation profiles.

Prerequisites

Before installing Tanzu Application Platform GUI:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see the Tanzu Application Platform Prerequisites.
- Create a Git repository for Tanzu Application Platform GUI software catalogs, with a token allowing read access. Supported Git infrastructure includes:
  - GitHub
  - GitLab
  - Azure DevOps
- Install Tanzu Application Platform GUI 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 **tap-gui-catalogs-latest** folder.
  3. Extract Tanzu Application Platform GUI Blank Catalog to your Git repository. This serves as the configuration location for your organization's Catalog inside Tanzu Application Platform GUI.

Procedure

To install Tanzu Application Platform GUI 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 code:

   ```
   service_type: ClusterIP
   ingressEnabled: true
   ingressDomain: "INGRESS-DOMAIN"
   app_config:
     app:
       baseUrl: http://tap-gui.INGRESS-DOMAIN
     catalog:
       locations:
         - type: url
           target: https://GIT-CATALOG-URL/catalog-info.yaml
     backend:
       baseUrl: http://tap-gui.INGRESS-DOMAIN
       cors:
         origin: http://tap-gui.INGRESS-DOMAIN
   ```

   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 Application Platform GUI Catalog**) or a Backstage-compliant catalog that you've already built and posted on the Git infrastructure specified in Adding Tanzu Application Platform GUI integrations.

4. Install the package by running:

   ```
   tanzu package install tap-gui \
    --package-name tap-gui.tanzu.vmware.com \
    --version VERSION -n tap-install \
    -f 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-name tap-gui.tanzu.vmware.com --versio
   n 1.0.1 -n \
   tap-install -f 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 Application Platform GUI, use the service you exposed in the `service_type` field in the values file.

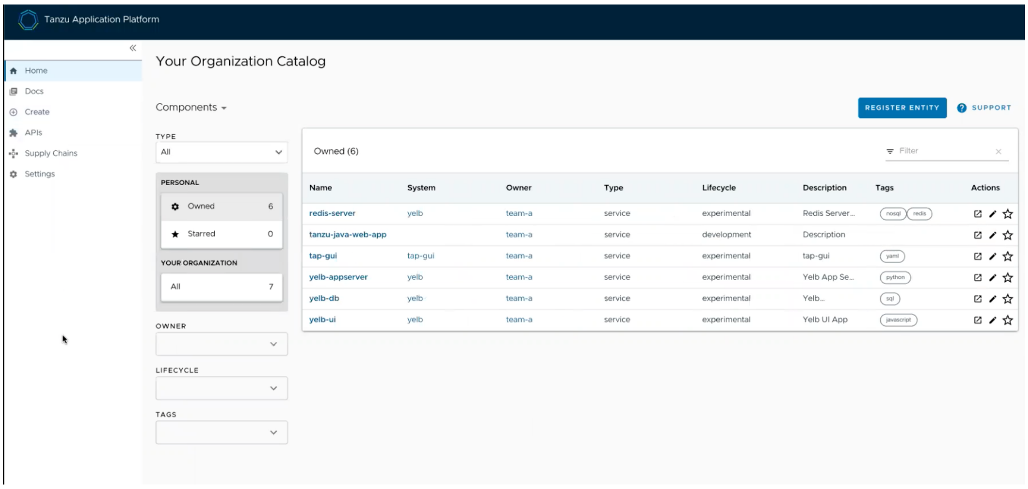
Customizing branding

This section describes how to customize the branding within the Tanzu Application Platform GUI portal:

- Customize logo and portal name on the top banner
- Customize Software Catalog page
- Customize Authentication page

Customize logo and portal name on the top banner

You can customize the logo and the name displayed in the top banner in the Tanzu Application Platform GUI portal. By default, the portal displays the VMware Tanzu logo and **Tanzu Application Platform** as the name.



Name	System	Owner	Type	Lifecycle	Description	Tags	Actions
redis-server	yelb	team-a	service	experimental	Redis Server...	nosql redis	
tanzu-java-web-app		team-a	service	development	Description		
tap-gui	tap-gui	team-a	service	experimental	tap-gui	yaml	
yelb-appserver	yelb	team-a	service	experimental	Yelb App Se...	python	
yelb-db	yelb	team-a	service	experimental	Yelb...	sql	
yelb-ui	yelb	team-a	service	experimental	Yelb UI App	javascript	

Customize branding

To customize the branding in your portal:

1. Provide additional configuration parameters to the `app_config` section of your `tap-values.yaml` file:

   ```
   tap_gui:
     app_config:
       customize:
         custom_logo: 'BASE-64-IMAGE'
         custom_name: 'PORTAL-NAME'
   ```

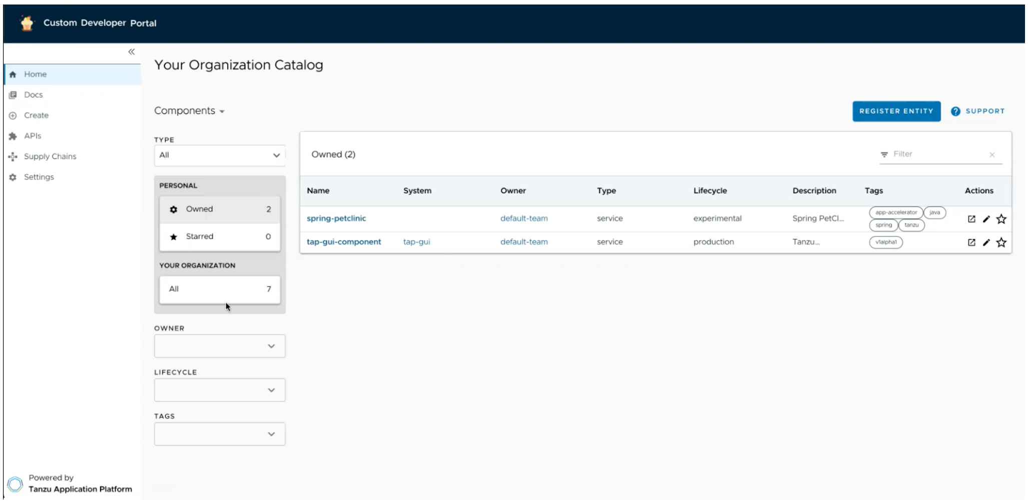
   Where:

   - `BASE-64-IMAGE` is the image encoded in base64. VMware recommends a 512-pixel by 512-pixel PNG image with a transparent background.
   - `PORTAL-NAME` is the name of your portal, such as `Our Custom Developer Experience Portal`.

2. Reinstall your Tanzu Application Platform GUI package by following steps in Upgrading Tanzu Application Platform.

After the updated values configuration file is applied in Tanzu Application Platform GUI, you see the customized version of your portal.

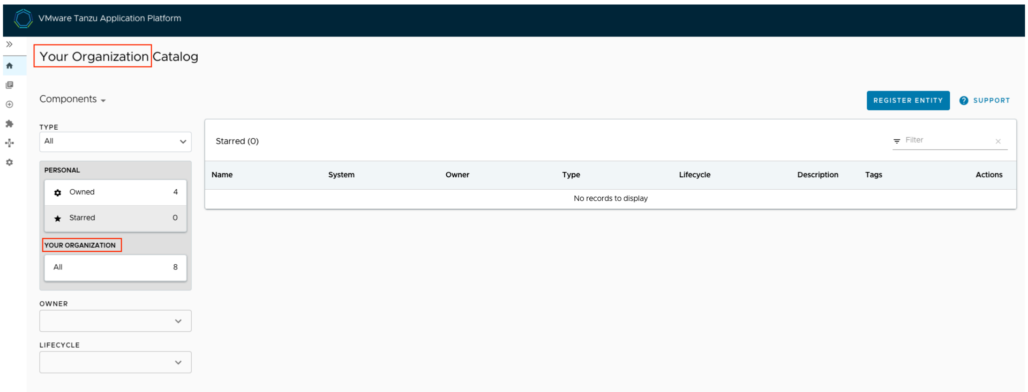
If there is an error in `BASE-64-IMAGE` or `PORTAL-NAME`, Tanzu Application Platform GUI reverts to the original branding template.



Name	System	Owner	Type	Lifecycle	Description	Tags	Actions
spring-petclinic		default-team	service	experimental	Spring PetCl...	app-accelerator java spring tanzu	
tap-gui-component	tap-gui	default-team	service	production	Tanzu...	vlalpha1	

Customize Software Catalog page

You can customize the name of your organization on the Software Catalog page of Tanzu Application Platform GUI portal. By default, the portal displays **Your Organization** in front of the **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:

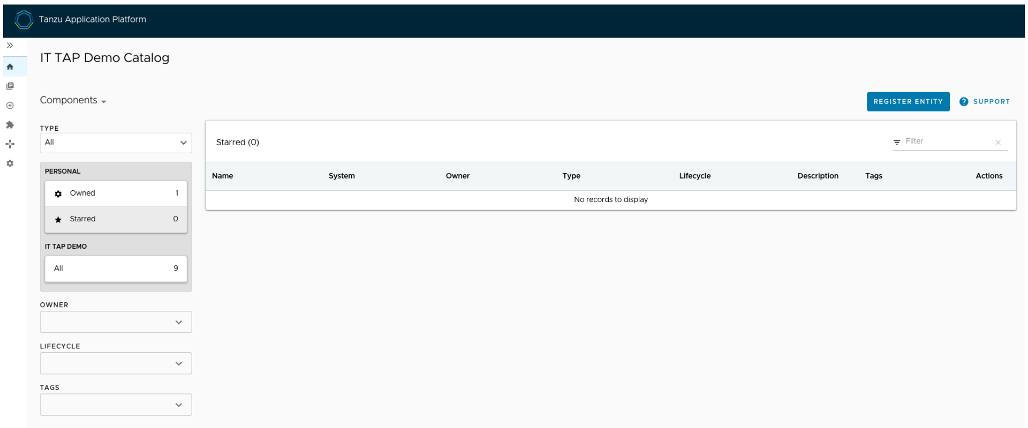
   ```
   tap_gui:
     app_config:
       organization:
         name: 'ORG-NAME'
   ```

   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 Application Platform GUI package by following the steps in Upgrading Tanzu Application Platform.

After the updated values configuration file is applied in Tanzu Application Platform GUI, you see the customized version of your portal.

If there is an error in the provided configuration parameters, Tanzu Application Platform GUI reverts to the original organization name.



Customize Authentication page

To customize the portal name on the **Authentication** page and the name of the browser tab for Tanzu Application Platform GUI:

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 Application Platform GUI package by following the steps in Upgrading Tanzu Application Platform.

After the updated values configuration file is applied in Tanzu Application Platform GUI, you see the customized version of your portal.

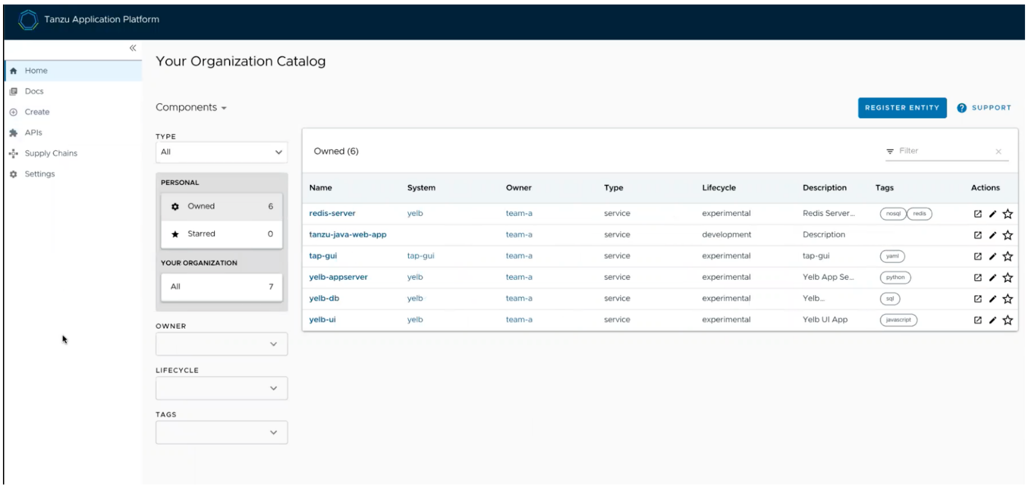
Customizing branding

This section describes how to customize the branding within the Tanzu Application Platform GUI portal:

- Customize logo and portal name on the top banner
- Customize Software Catalog page
- Customize Authentication page

Customize logo and portal name on the top banner

You can customize the logo and the name displayed in the top banner in the Tanzu Application Platform GUI portal. By default, the portal displays the VMware Tanzu logo and **Tanzu Application Platform** as the name.



Name	System	Owner	Type	Lifecycle	Description	Tags	Actions
redis-server	yelb	team-a	service	experimental	Redis Server...	nosql redis	
tanzu-java-web-app		team-a	service	development	Description		
tap-gui	tap-gui	team-a	service	experimental	tap-gui	yaml	
yelb-appserver	yelb	team-a	service	experimental	Yelb App Se...	python	
yelb-db	yelb	team-a	service	experimental	Yelb...	sql	
yelb-ui	yelb	team-a	service	experimental	Yelb UI App	javascript	

Customize branding

To customize the branding in your portal:

1. Provide additional configuration parameters to the `app_config` section of your `tap-values.yaml` file:

```
tap_gui:
  app_config:
    customize:
      custom_logo: 'BASE-64-IMAGE'
      custom_name: 'PORTAL-NAME'
```

Where:

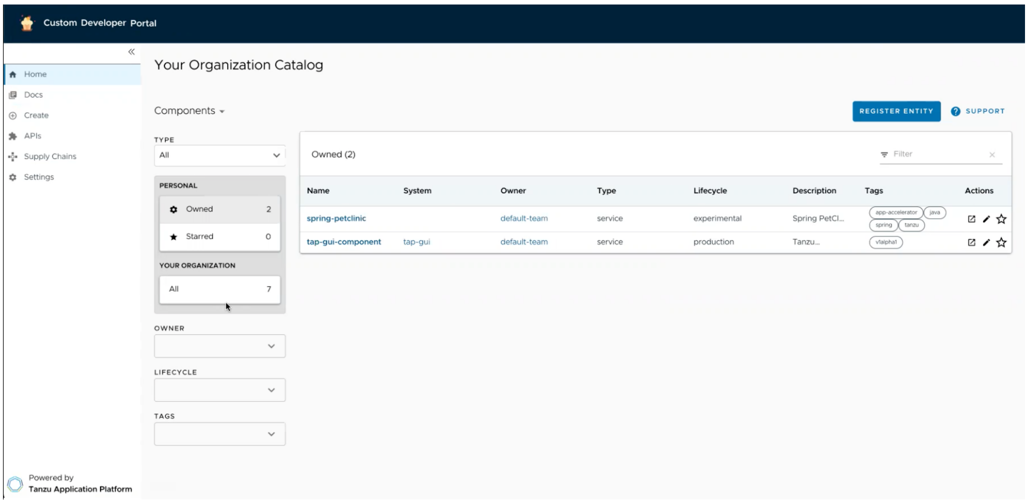
- `BASE-64-IMAGE` is the image encoded in base64. VMware recommends a 512-pixel by 512-pixel PNG image with a transparent background.

  - PORTAL-NAME is the name of your portal, such as Our Custom Developer Experience Portal.

2. Reinstall your Tanzu Application Platform GUI package by following steps in Upgrading Tanzu Application Platform.

After the updated values configuration file is applied in Tanzu Application Platform GUI, you see the customized version of your portal.

If there is an error in BASE-64-IMAGE or PORTAL-NAME, Tanzu Application Platform GUI reverts to the original branding template.

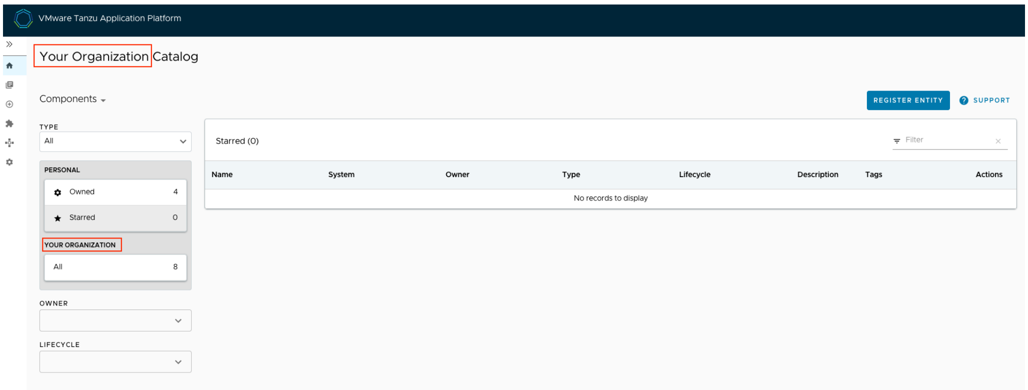


Owned (2)

Name	System	Owner	Type	Lifecycle	Description	Tags	Actions
spring-petclinic		default-team	service	experimental	Spring PetCl...	app-accelerator java spring tanzu	
tap-gui-component	tap-gui	default-team	service	production	Tanzu...	vlalpha1	

Customize Software Catalog page

You can customize the name of your organization on the Software Catalog page of Tanzu Application Platform GUI portal. By default, the portal displays **Your Organization** in front of the **Catalog** and in the selection box.



Starred (0)

Name	System	Owner	Type	Lifecycle	Description	Tags	Actions
No records to display							

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:

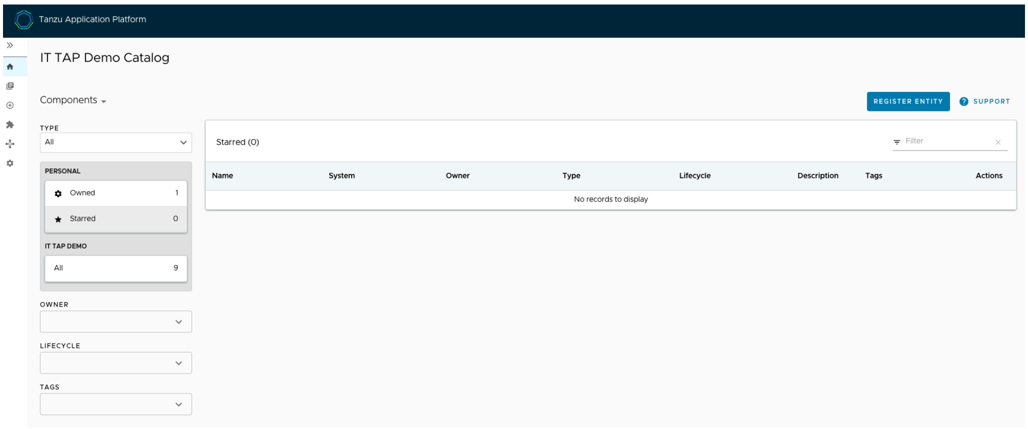
    ```
    tap_gui:
      app_config:
        organization:
          name: 'ORG-NAME'
    ```

    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 Application Platform GUI package by following the steps in Upgrading Tanzu Application Platform.

After the updated values configuration file is applied in Tanzu Application Platform GUI, you see the customized version of your portal.

If there is an error in the provided configuration parameters, Tanzu Application Platform GUI reverts to the original organization name.



Name	System	Owner	Type	Lifecycle	Description	Tags	Actions
No records to display							

Customize Authentication page

To customize the portal name on the **Authentication** page and the name of the browser tab for Tanzu Application Platform GUI:

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 Application Platform GUI package by following the steps in Upgrading Tanzu Application Platform.

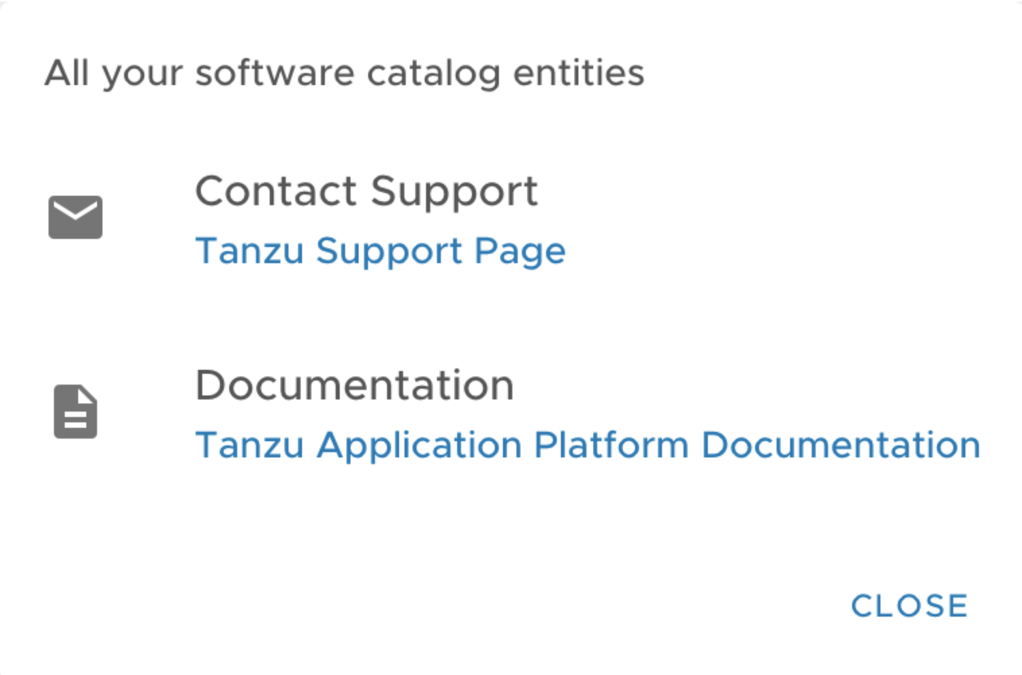
After the updated values configuration file is applied in Tanzu Application Platform GUI, you see the customized version of your portal.

Customizing the Support menu

This topic describes how to customize the support menu.

Overview

Many important pages of Tanzu Application Platform GUI 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. For example, the default menu on the Catalog page looks similar to the following image:



All your software catalog entities

✉ Contact Support
Tanzu Support Page

🗎 Documentation
Tanzu Application Platform Documentation

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
```

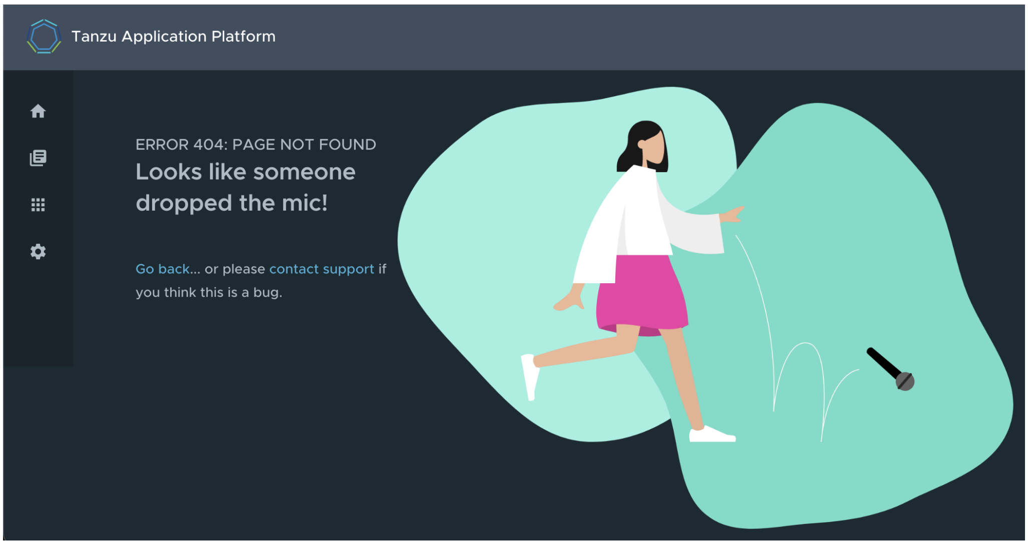
Structure of the support configuration

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 such as this one:



ERROR 404: PAGE NOT FOUND

Looks like someone dropped the mic!

Go back... or please contact support if you think this is a bug.

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.

Accessing Tanzu Application Platform GUI

This topic tells you how to access Tanzu Application Platform GUI (commonly called TAP GUI) 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 Application Platform GUI 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 Application Platform GUI 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 Application Platform GUI 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 Application Platform GUI 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"
   ```

```
tap_gui:
  service_type: ClusterIP
```

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:
     service_type: ClusterIP
   # 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
   ```

   This snippet is from a values file in the Configure Tanzu Application Platform GUI section of the Profiles installation topic. The new host names are populated based on the example host name `tap-gui.example.com`.

4. Update your package installation with your changed `tap-values.yaml` file by running:

   ```
   tanzu package installed update tap --package-name tap.tanzu.vmware.com --versio
   n VERSION-NUMBER \
   --values-file tap-values.yaml -n tap-install
   ```

   Where `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.2.2`.

5. Use a web browser to access Tanzu Application Platform GUI 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.

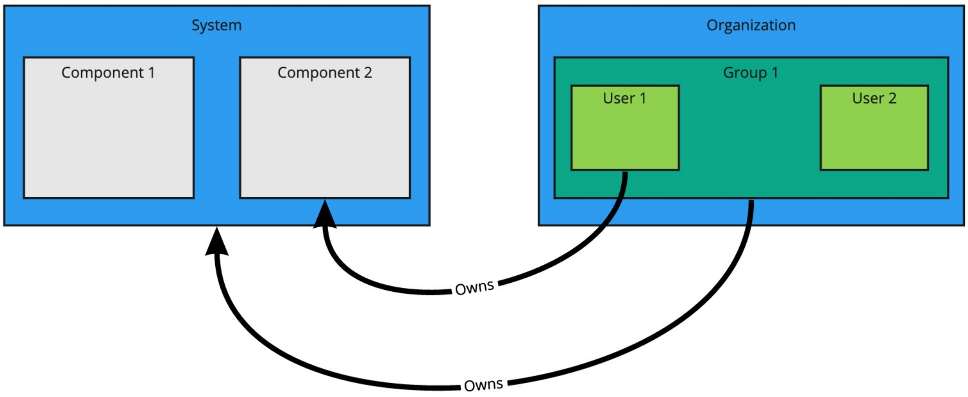
Adding catalog entities

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 Application Platform GUI prerequisites as a foundation for creating user, group, system, and main component YAML files.

Relationship Diagram:



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: []
```

More information about user entities and group entities is available in 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 Application Platform GUI System
spec:
  owner: default-team
```

More information about system entities is available in 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 Application Platform GUI 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
```

More information about component entities is available in 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:

1. Use static configuration to add or change catalog locations.

- 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
  ```

- 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 you see 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.

2. Update the package to include the catalog by running:

   ```
   tanzu package installed update backstage \
    --version PACKAGE-VERSION \
    -f VALUES-FILE
   ```

3. Verify the status of this update by running:

   ```
   tanzu package installed list
   ```

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

1. Download the appropriate 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

1. From the Tanzu Application Platform downloads page, click **tap-gui-catalogs-latest** > **Tanzu Application Platform GUI Yelb Catalog**.
2. Follow the earlier steps for Adding catalog entities to add `catalog-info.yaml`.

Viewing resources on multiple clusters in Tanzu Application Platform GUI

You can configure Tanzu Application Platform GUI (commonly called TAP GUI) to retrieve Kubernetes object details from multiple clusters and then surface those details in the various Tanzu Application Platform GUI 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 Application Platform GUI 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']
  verbs: ['get', 'watch', 'list']
- apiGroups: ['apps']
  resources: ['deployments', 'replicasets']
  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.carto.run']
  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: ['tekton.dev']
  resources:
  - taskruns
  - pipelineruns
  verbs: ['get', 'watch', 'list']
- apiGroups: ['kappctrl.k14s.io']
  resources:
  - apps
  verbs: ['get', 'watch', 'list']
- apiGroups: ['conventions.carto.run']
  resources:
  - podintents
  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.

   ```
   CLUSTER_URL=$(kubectl config view --minify -o jsonpath='{.clusters[0].cluster.s
   erver}')

   CLUSTER_TOKEN=$(kubectl -n tap-gui get secret $(kubectl -n tap-gui get sa tap-g
   ui-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
   ```

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[0].
   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 Application Platform GUI to view resources on multiple clusters later.

Update Tanzu Application Platform GUI to view resources on multiple clusters

The clusters must be identified to Tanzu Application Platform GUI 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 Application Platform GUI, 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
```

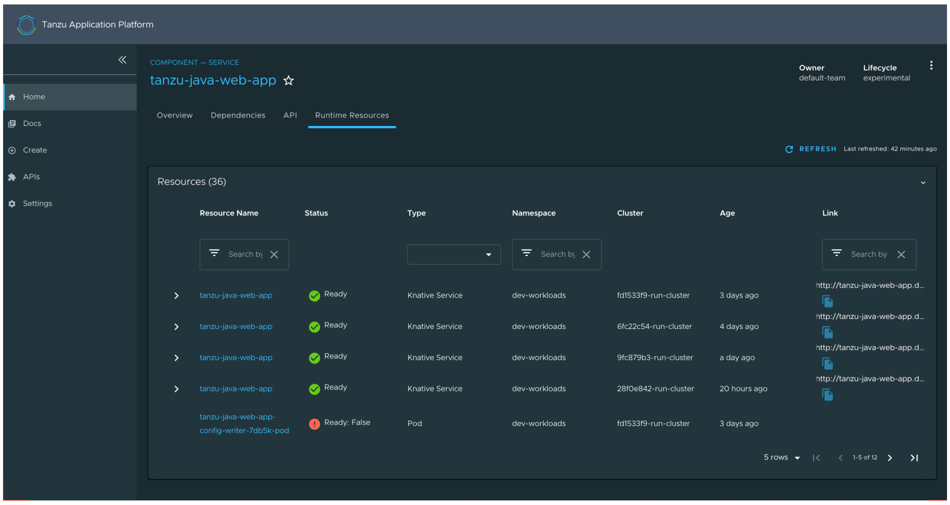
3. Wait a moment for the `tap` and `tap-gui` packages to update and then verify that `STATUS` is `Reconcile succeeded` by running:

```
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. Navigate 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.



Resource Name	Status	Type	Namespace	Cluster	Age	Link
tanzu-java-web-app	✓ Ready	Knative Service	dev-workloads	fd1533f9-run-cluster	3 days ago	http://tanzu-java-web-app.d...
tanzu-java-web-app	✓ Ready	Knative Service	dev-workloads	6fc22c54-run-cluster	4 days ago	http://tanzu-java-web-app.d...
tanzu-java-web-app	✓ Ready	Knative Service	dev-workloads	9fc879b3-run-cluster	a day ago	http://tanzu-java-web-app.d...
tanzu-java-web-app	✓ Ready	Knative Service	dev-workloads	28f0e842-run-cluster	20 hours ago	http://tanzu-java-web-app.d...
tanzu-java-web-app-config-writer-7db5k-pod	! Ready: False	Pod	dev-workloads	fd1533f9-run-cluster	3 days ago	

Setting up a Tanzu Application Platform GUI authentication provider

Tanzu Application Platform GUI (commonly called TAP GUI) 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 Application Platform GUI currently supports the following authentication providers:

- Auth0
- Azure
- Bitbucket
- GitHub
- GitLab
- Google
- Okta
- OneLogin

You can also configure a custom OpenID Connect (OIDC) provider.

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"

tap_gui:
  service_type: ClusterIP
  app_config:
    app:
      baseUrl: http://tap-gui.INGRESS-DOMAIN
    catalog:
      locations:
        - type: url
          target: https://GIT-CATALOG-URL/catalog-info.yaml
    backend:
      baseUrl: http://tap-gui.INGRESS-DOMAIN
      cors:
        origin: http://tap-gui.INGRESS-DOMAIN
#Existing values file above
    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 Application Platform GUI 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 Application Platform GUI 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 Application Platform GUI 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 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 Application Platform GUI (commonly called TAP GUI) based on user roles and permissions for each of the visible remote clusters.

Role-based Access Control (RBAC) is currently supported for the Kubernetes cluster provider GKE (Google Kubernetes Engine) on GCP. For more information, see Enable authorization on remote GKE clusters by using Google Auth

Support for other Kubernetes cluster providers is planned for future releases of Tanzu Application Platform.

Tanzu Application Platform GUI 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.

To do so:

1. Set up the OIDC provider
2. Configure the Kubernetes cluster with the OIDC provider
3. Configure the Tanzu Application Platform GUI to view the remote cluster
4. Upgrade the Tanzu Application Platform GUI package

After following these steps, you can view your runtime resources on a remote cluster in Tanzu Application Platform GUI. For more information, see View runtime resources on authorization-enabled clusters.

View resources on remote clusters

You can control the access to Kubernetes runtime resources on Tanzu Application Platform GUI (commonly called TAP GUI) based on user roles and permissions for each of the visible remote clusters.

Role-based Access Control (RBAC) is currently supported for the Kubernetes cluster provider GKE (Google Kubernetes Engine) on GCP. For more information, see Enable authorization on remote GKE clusters by using Google Auth

Support for other Kubernetes cluster providers is planned for future releases of Tanzu Application Platform.

Tanzu Application Platform GUI 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.

To do so:

1. Set up the OIDC provider
2. Configure the Kubernetes cluster with the OIDC provider
3. Configure the Tanzu Application Platform GUI to view the remote cluster
4. Upgrade the Tanzu Application Platform GUI package

After following these steps, you can view your runtime resources on a remote cluster in Tanzu Application Platform GUI. For more information, see View runtime resources on authorization-enabled clusters.

View resources on remote GKE clusters

This topic tells you how to use Google Auth to add access-controlled visibility for a remote GKE cluster.

After the authorization is enabled, you can view your runtime resources on a remote cluster in Tanzu Application Platform GUI. For more information, see View runtime resources on remote clusters.

Use Google Auth

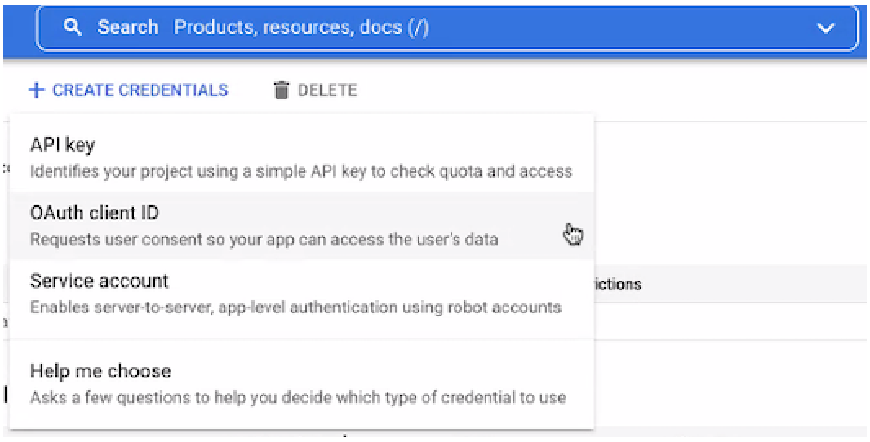
To use Google's OIDC provider to enable authorization:

1. Create OAuth credentials in Google Cloud
2. Configure the Tanzu Application Platform GUI to view the remote GKE cluster
3. Upgrade the Tanzu Application GUI package

Create OAuth credentials in Google Cloud

Follow these steps to create OAuth credentials in Google Cloud. For more information, see the Consent and Credentials topics in the Google Cloud documentation.

1. Log in to the Google Cloud console.
2. From the drop-down menu on the top bar, select the project to which your GKE cluster belongs.
3. If a consent screen has not already been configured for this project
    1. Go to **APIs & Services** > **OAuth consent screen**
    2. Add the top domain of your Tanzu Application Platform GUI server as an authorized domain
    3. For scopes, select **openid**, **auth/userinfo.email**, and **auth/userinfo.profile**
4. Go to **APIs & Services** > **Credentials**.
5. Click **Create Credentials** and select **OAuth client ID**.



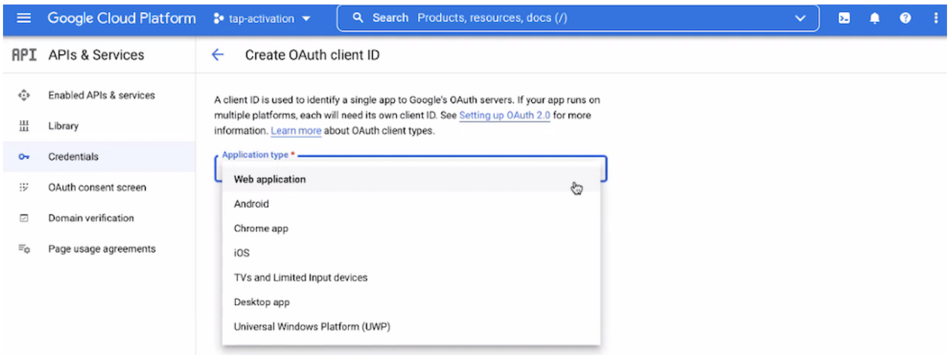
API key
Identifies your project using a simple API key to check quota and access

OAuth client ID
Requests user consent so your app can access the user's data

Service account
Enables server-to-server, app-level authentication using robot accounts

Help me choose
Asks a few questions to help you decide which type of credential to use

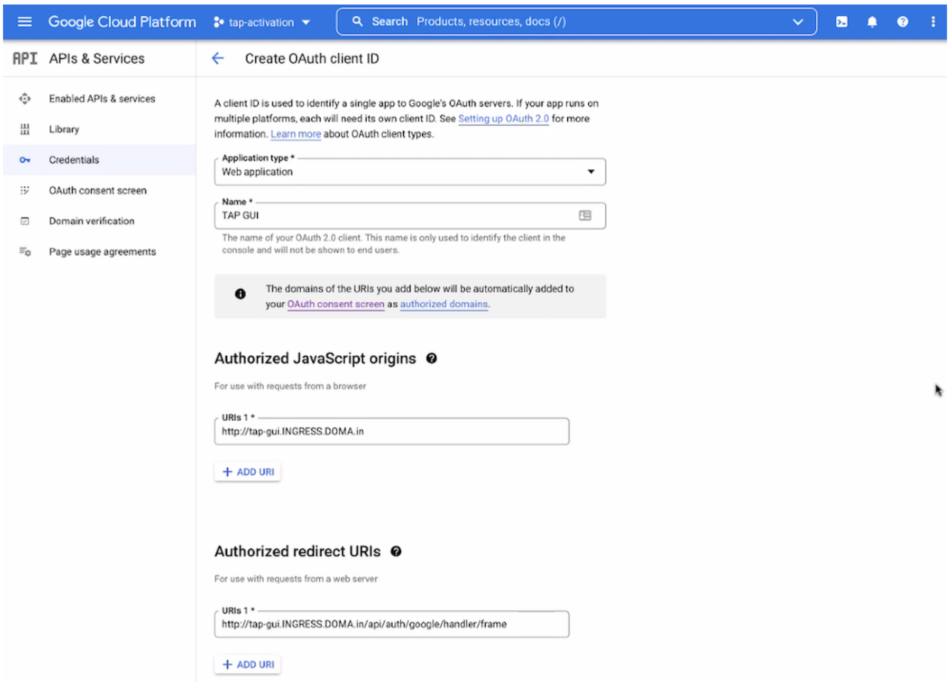
6. Select **Web Application** as the **Application Type**.



A client ID is used to identify a single app to Google's OAuth servers. If your app runs on multiple platforms, each will need its own client ID. See Setting up OAuth 2.0 for more information. Learn more about OAuth client types.

7. Populate the following dialog boxes with these settings:
   - Name: `TAP GUI` or your custom app name
   - Authorized JavaScript origins: `http://tap-gui.INGRESS-DOMAIN`
   - Authorized redirect URIs: `http://tap-gui.INGRESS-DOMAIN/api/auth/google/handler/frame`

   Where `INGRESS-DOMAIN` is the ingress domain you specified during the installation of Tanzu Application Platform GUI



A client ID is used to identify a single app to Google's OAuth servers. If your app runs on multiple platforms, each will need its own client ID. See Setting up OAuth 2.0 for more information. Learn more about OAuth client types.

Application type *: Web application

Name *: TAP GUI

The name of your OAuth 2.0 client. This name is only used to identify the client in the console and will not be shown to end users.

The domains of the URIs you add below will be automatically added to your OAuth consent screen as authorized domains.

Authorized JavaScript origins

For use with requests from a browser

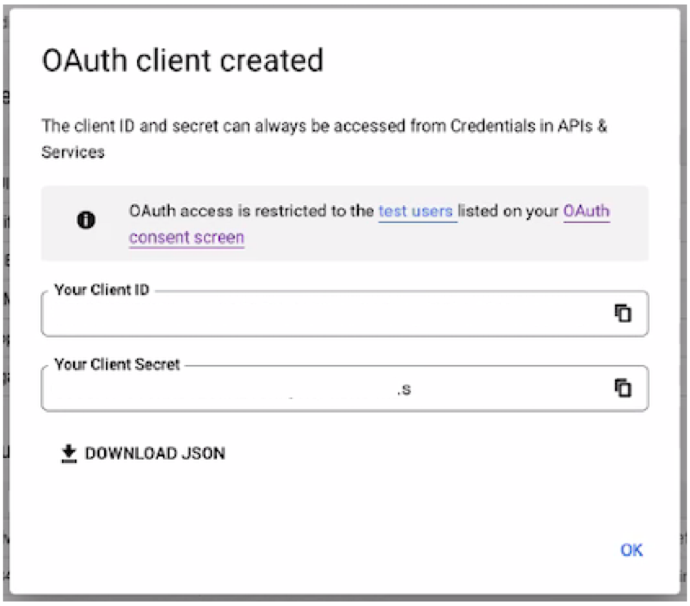
URIs 1 *: http://tap-gui.INGRESS.DOMA.in

Authorized redirect URIs

For use with requests from a web server

URIs 1 *: http://tap-gui.INGRESS.DOMA.in/api/auth/google/handler/frame

8. Click `Create` and store your credentials.



OAuth client created

The client ID and secret can always be accessed from Credentials in APIs & Services

OAuth access is restricted to the test users listed on your OAuth consent screen

Your Client ID

Your Client Secret

.s

Configure Tanzu Application Platform GUI to view the remote GKE cluster

Configure visibility of the remote GKE cluster in Tanzu Application Platform GUI:

1. Ensure you added an `auth` section to the `app_config` section that Tanzu Application Platform GUI 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

2. Record your cluster URL for later use by running:

   ```
   CLUSTER_URL=$(kubectl config view --minify -o jsonpath='{.clusters[0].cluster.s
   erver}')

   echo CLUSTER-URL: $CLUSTER_URL
   ```

3. Record the CA certificate data for later use by running:

   ```
   CA_DATA=$(kubectl config view --minify -o jsonpath='{.clusters[0].cluster.certi
   ficate-authority-data}' --raw)

   echo CA-DATA: $CA_DATA
   ```

4. You must also add a `kubernetes` section to the `app_config` section that Tanzu Application Platform GUI 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 Application Platform GUI.
- `CA-DATA` is the CA certificate data.

If there are any other clusters that you want to make visible in Tanzu Application Platform GUI, add their entries to `clusters` as well.

Upgrade the Tanzu Application Platform GUI package

To upgrade the Tanzu Application Platform GUI package:

1. Update the `tap-gui` package after the new configuration file is ready by running:

   ```
   tanzu package installed update tap-gui --values-file tap-gui-values.yaml
   ```

   Or, if using the `tap-values.yaml` file, run:

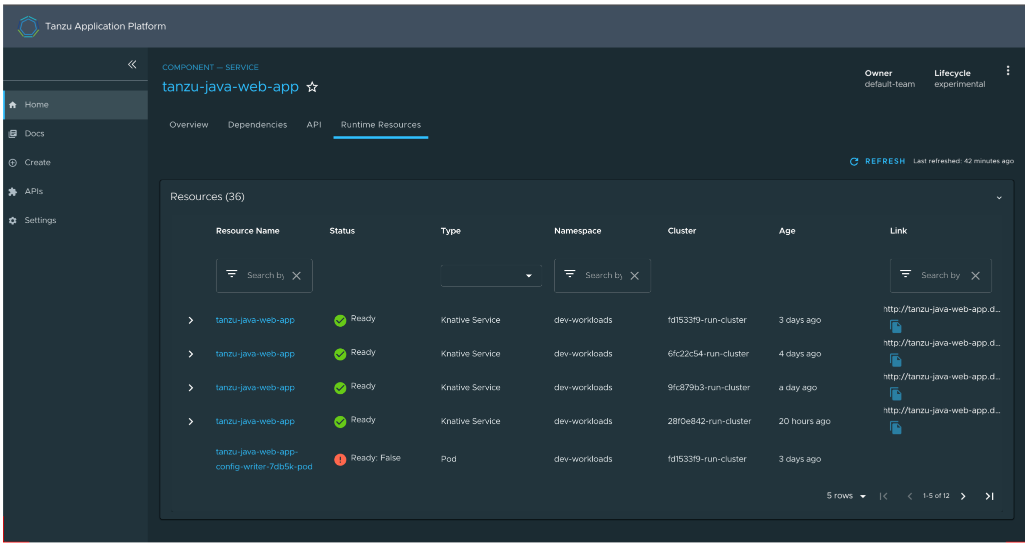
   ```
   tanzu package installed update tap --values-file tap-values.yaml
   ```

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 runtime resources on authorization-enabled clusters

To visualize runtime resources on authorization-enabled clusters in Tanzu Application Platform GUI (commonly called TAP GUI), proceed to the software catalog **Component** of choice and click the **Runtime Resources** tab at the top of the ribbon.



Resource Name	Status	Type	Namespace	Cluster	Age	Link
tanzu-java-web-app	Ready	Knative Service	dev-workloads	fd1533f9-run-cluster	3 days ago	http://tanzu-java-web-app.d...
tanzu-java-web-app	Ready	Knative Service	dev-workloads	6fc22c54-run-cluster	4 days ago	http://tanzu-java-web-app.d...
tanzu-java-web-app	Ready	Knative Service	dev-workloads	9fc879b3-run-cluster	a day ago	http://tanzu-java-web-app.d...
tanzu-java-web-app	Ready	Knative Service	dev-workloads	28f0e842-run-cluster	20 hours ago	http://tanzu-java-web-app.d...
tanzu-java-web-app-config-writer-7db5k-pod	Ready: False	Pod	dev-workloads	fd1533f9-run-cluster	3 days ago	

After you click **Runtime Resources**, Tanzu Application Platform GUI uses your credentials to query the clusters for the respective runtime resources. The system checks if 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 made visible by using the general service account of Tanzu Application Platform GUI. The visibility 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 Application Platform GUI..

When you access **Runtime Resources**, Tanzu Application Platform GUI queries all Kubernetes namespaces for runtime resources that have a matching `kubernetes-label-selector`. This usually has a `part-Of` prefix.

Assigning 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 Application Platform GUI 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
```

Adding Tanzu Application Platform GUI integrations

You can integrate Tanzu Application Platform GUI (commonly called TAP GUI) 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 Application Platform GUI prerequisites.

Add a Git-based provider integration that isn't GitHub

To enable Tanzu Application Platform GUI 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 Application Platform GUI 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 making changes to `tap-values.yaml`, update the package profile by running:

```
tanzu package installed update  tap --package-name tap.tanzu.vmware.com --version VERS
ION-NUMBER \
--values-file tap-values.yaml -n tap-install
```

Where `VERSION-NUMBER` is the Tanzu Application Platform version. For example, `1.2.2`.

For example:

```
$ tanzu package installed update  tap --package-name tap.tanzu.vmware.com --version \
1.2.2 --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'
```

Configuring the Tanzu Application Platform GUI database

The Tanzu Application Platform GUI (commonly called TAP GUI) 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 Application Platform GUI 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, VMware recommends using a PostgreSQL database.

Configure a PostgreSQL database

To use a PostgreSQL database:

1. Use the following values in `tap-values.yaml`:

   ```
     backend:
       baseUrl: http://tap-gui.INGRESS-DOMAIN
       cors:
         origin: http://tap-gui.INGRESS-DOMAIN
     # Existing tap-values.yaml above
       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.

2. Update the package profile by running:

```
tanzu package installed update  tap --package-name tap.tanzu.vmware.com --versi
on VERSION-NUMBER --values-file tap-values.yaml -n tap-install
```

Where `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.2.2`.

For example:

```
$ tanzu package installed update  tap --package-name tap.tanzu.vmware.com --ver
sion 1.2.2 --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 Application Platform GUI (commonly called TAP GUI). 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. To configure Amazon S3 access:

1. 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**.

2. 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/name

TechDocs are generated for catalogs that have markdown source files for TechDocs. To find the catalog locations and their entities' namespace/kind/name:

1. The catalogs appearing in Tanzu Application Platform GUI are listed in the config values under `app_config.catalog.locations`.

2. For a given 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 for each of the targets:

    1. Navigate 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 successfully.

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.

> **Important**
>
> 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 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
         credentials:
           accessKeyId: AWS-READONLY-ACCESS-KEY-ID
           secretAccessKey: AWS-READONLY-SECRET-ACCESS-KEY
         region: AWS-REGION
         s3ForcePathStyle: false
   ```

2. Update your installation from the Tanzu CLI.

   - If you installed Tanzu Application Platform GUI 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 \
-f VALUES-FILE
```

Where:

- `PACKAGE-VERSION` is your package version
- `VALUES-FILE` is your values file

- If you installed Tanzu Application Platform GUI 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 \
-f VALUES-FILE
```

Where:

- `PACKAGE-VERSION` is your package version
- `VALUES-FILE` is your values file

3. Verify the status of this update by running:

```
tanzu package installed list
```

4. Navigate to the **Docs** section of your catalog and view the TechDocs pages to verify the content is loaded from the S3 bucket successfully.

Overview of Tanzu Application Platform GUI plug-ins

Tanzu Application Platform GUI (commonly called TAP GUI) has many pre-integrated plug-ins. You need not configure the plug-ins. To use a plug-in, you must install the relevant Tanzu Application Platform component.

Tanzu Application Platform has the following GUI plug-ins:

- Runtime Resources Visibility
- Application Live View
- Application Accelerator
- API Documentation
- Supply Chain Choreographer

Overview of Tanzu Application Platform GUI plug-ins

Tanzu Application Platform GUI (commonly called TAP GUI) has many pre-integrated plug-ins. You need not configure the plug-ins. To use a plug-in, you must install the relevant Tanzu Application Platform component.

Tanzu Application Platform has the following GUI plug-ins:

- Runtime Resources Visibility
- Application Live View
- Application Accelerator
- API Documentation
- Supply Chain Choreographer

Runtime resources visibility in Tanzu Application Platform GUI

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 Application Platform GUI separately.

Visualize Workloads on Tanzu Application Platform GUI

In order to view your applications on Tanzu Application Platform GUI, use the following steps:

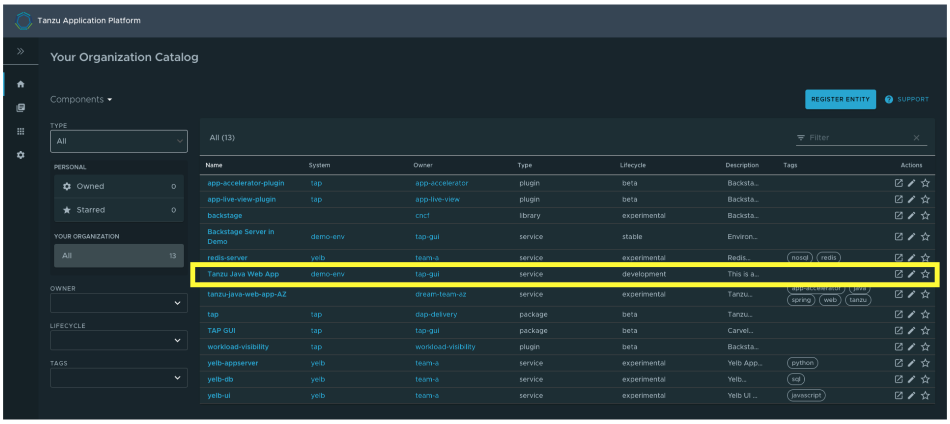
1. Deploy your first application on the Tanzu Application Platform
2. Add your application to Tanzu Application Platform GUI Software Catalog

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.

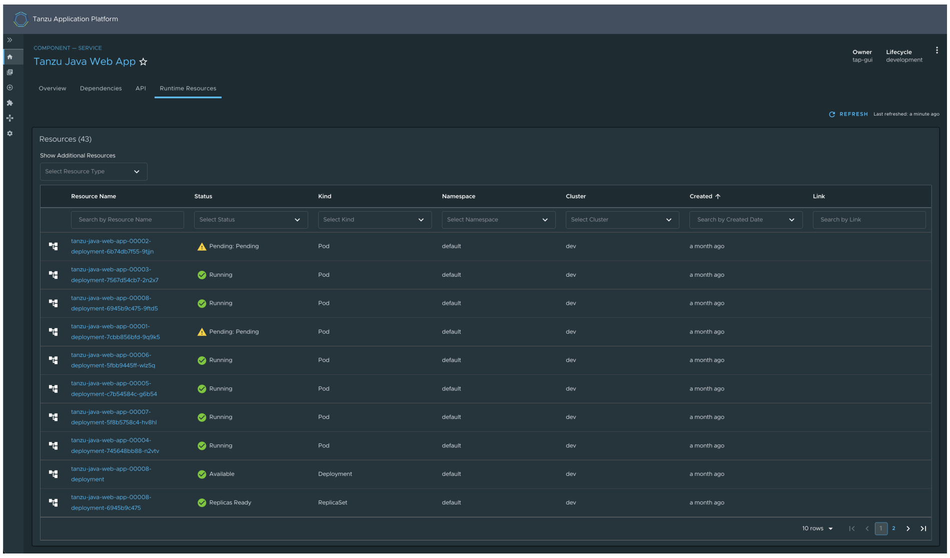
To view the list of your running resources:

1. Select your component from the Catalog index page.



Name	System	Owner	Type	Lifecycle	Description	Tags	Actions
app-accelerator-plugin	tap	app-accelerator	plugin	beta	Backsta...		
app-live-view-plugin	tap	app-live-view	plugin	beta	Backsta...		
backstage		cncf	library	experimental	Backsta...		
Backstage Server in Demo	demo-env	tap-gui	service	stable	Environ...		
redis-server	yelb	team-a	service	experimental	Redis...	nosql redis	
Tanzu Java Web App	demo-env	tap-gui	service	development	This is a...		
tanzu-java-web-app-AZ		dream-team-az	service	experimental	Tanzu...	app-accelerator java spring web tanzu	
tap	tap	dap-delivery	package	beta	Tanzu...		
TAP GUI	tap	tap-gui	package	beta	Carvel...		
workload-visibility	tap	workload-visibility	plugin	beta	Backsta...		
yelb-appserver	yelb	team-a	service	experimental	Yelb App...	python	
yelb-db	yelb	team-a	service	experimental	Yelb...	sql	
yelb-ui	yelb	team-a	service	experimental	Yelb UI ...	javascript	

2. Select the **Runtime Resources** tab.



Resource Name	Status	Kind	Namespace	Cluster	Created ↑	Link
Search by Resource Name	Select Status	Select Kind	Select Namespace	Select Cluster	Search by Created Date	Search by Link
tanzu-java-web-app-00002-deployment-6b74db7f55-9tjjn	Pending: Pending	Pod	default	dev	a month ago	
tanzu-java-web-app-00003-deployment-7567d54cb7-2n2x7	Running	Pod	default	dev	a month ago	
tanzu-java-web-app-00008-deployment-6945b9c475-9ftd5	Running	Pod	default	dev	a month ago	
tanzu-java-web-app-00001-deployment-7cbb856bfd-9q9k5	Pending: Pending	Pod	default	dev	a month ago	
tanzu-java-web-app-00006-deployment-5fbb9445ff-wlz5q	Running	Pod	default	dev	a month ago	
tanzu-java-web-app-00005-deployment-c7b54584c-g6b54	Running	Pod	default	dev	a month ago	
tanzu-java-web-app-00007-deployment-5f8b5758c4-hv8hl	Running	Pod	default	dev	a month ago	
tanzu-java-web-app-00004-deployment-745648bb88-n2vtv	Running	Pod	default	dev	a month ago	
tanzu-java-web-app-00008-deployment	Available	Deployment	default	dev	a month ago	
tanzu-java-web-app-00008-deployment-6945b9c475	Replicas Ready	ReplicaSet	default	dev	a month ago	

Resources

Built-in Kubernetes resources in this view are:

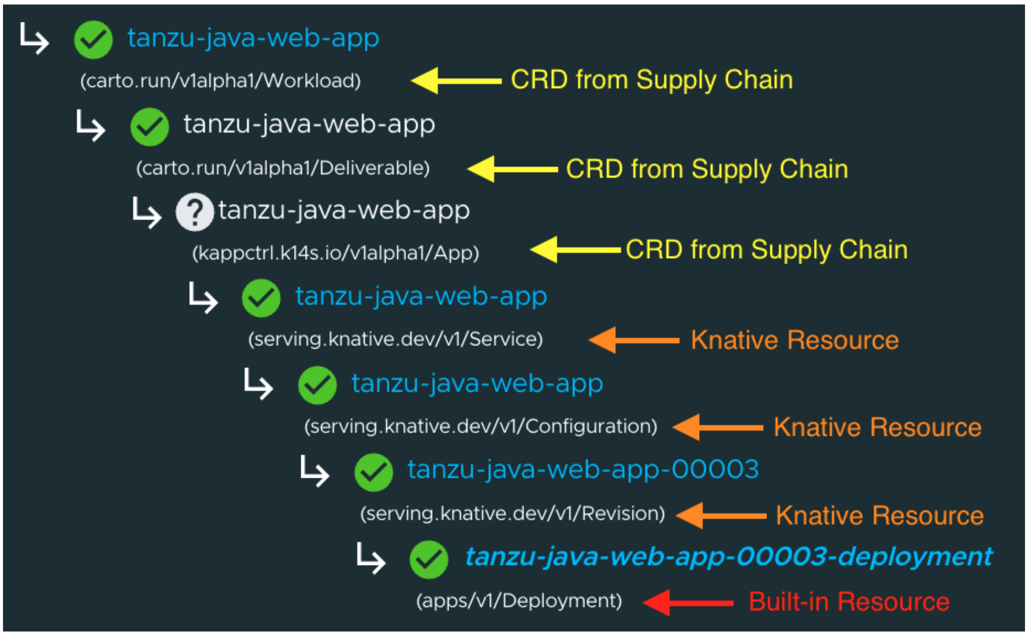
- Services
- Deployments
- ReplicaSets
- Pods

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 Application Platform GUI.

CRDs from Supply Chain are associated with Knative Resources, further down the chain, and built-in resources even further down the chain.

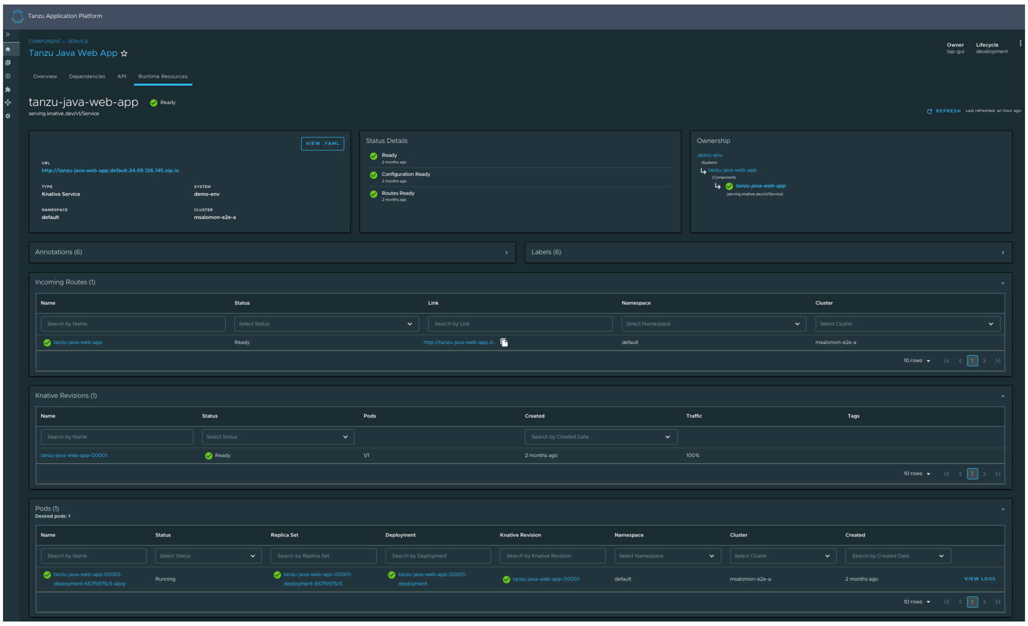


↳ ✓ tanzu-java-web-app
(carto.run/v1alpha1/Workload) ← CRD from Supply Chain
↳ ✓ tanzu-java-web-app
(carto.run/v1alpha1/Deliverable) ← CRD from Supply Chain
↳ ? tanzu-java-web-app
(kappctrl.k14s.io/v1alpha1/App) ← CRD from Supply Chain
↳ ✓ tanzu-java-web-app
(serving.knative.dev/v1/Service) ← Knative Resource
↳ ✓ tanzu-java-web-app
(serving.knative.dev/v1/Configuration) ← Knative Resource
↳ ✓ tanzu-java-web-app-00003
(serving.knative.dev/v1/Revision) ← Knative Resource
↳ ✓ *tanzu-java-web-app-00003-deployment*
(apps/v1/Deployment) ← Built-in Resource

Knative service details page

To view details about your Knative services, select any resource that has a Knative Service type. In this page, additional information is available for Knative resources, including:

- status
- an ownership hierarchy
- incoming routes
- revisions
- pod details



COMPONENT — SERVICE

Tanzu Java Web App

Owner tap-gui

Lifecycle development

tanzu-java-web-app ✓ Ready

serving.knative.dev/v1/Service

Last refreshed: an hour ago

URL http://tanzu-java-web-app.default.34.69.126.145.nip.io

TYPE Knative Service

SYSTEM demo-env

NAMESPACE default

CLUSTER msalomon-e2e-a

Status Details

- ✓ Ready 2 months ago
- ✓ Configuration Ready 2 months ago
- ✓ Routes Ready 2 months ago

Ownership

demo-env (System)
↳ tanzu-java-web-app (Component)
↳ ✓ tanzu-java-web-app (serving.knative.dev/v1/Service)

Annotations (6)

Labels (6)

Incoming Routes (1)

Name	Status	Link	Namespace	Cluster
Search by Name	Select Status	Search by Link	Select Namespace	Select Cluster
✓ tanzu-java-web-app	Ready	http://tanzu-java-web-app.d...	default	msalomon-e2e-a

Knative Revisions (1)

Name	Status	Pods	Created	Traffic	Tags
Search by Name	Select Status		Search by Created Date		
tanzu-java-web-app-00001	✓ Ready	1/1	2 months ago	100%	

Pods (1)
Desired pods: 1

Name	Status	Replica Set	Deployment	Knative Revision	Namespace	Cluster	Created	
Search by Name	Select Status	Search by Replica Set	Search by Deployment	Search by Knative Revision	Select Namespace	Select Cluster	Search by Created Date	
✓ tanzu-java-web-app-00001-deployment-657f5975c5-kljvg	Running	✓ tanzu-java-web-app-00001-deployment-657f5975c5	✓ tanzu-java-web-app-00001-deployment	✓ tanzu-java-web-app-00001	default	msalomon-e2e-a	2 months ago	VIEW LOGS

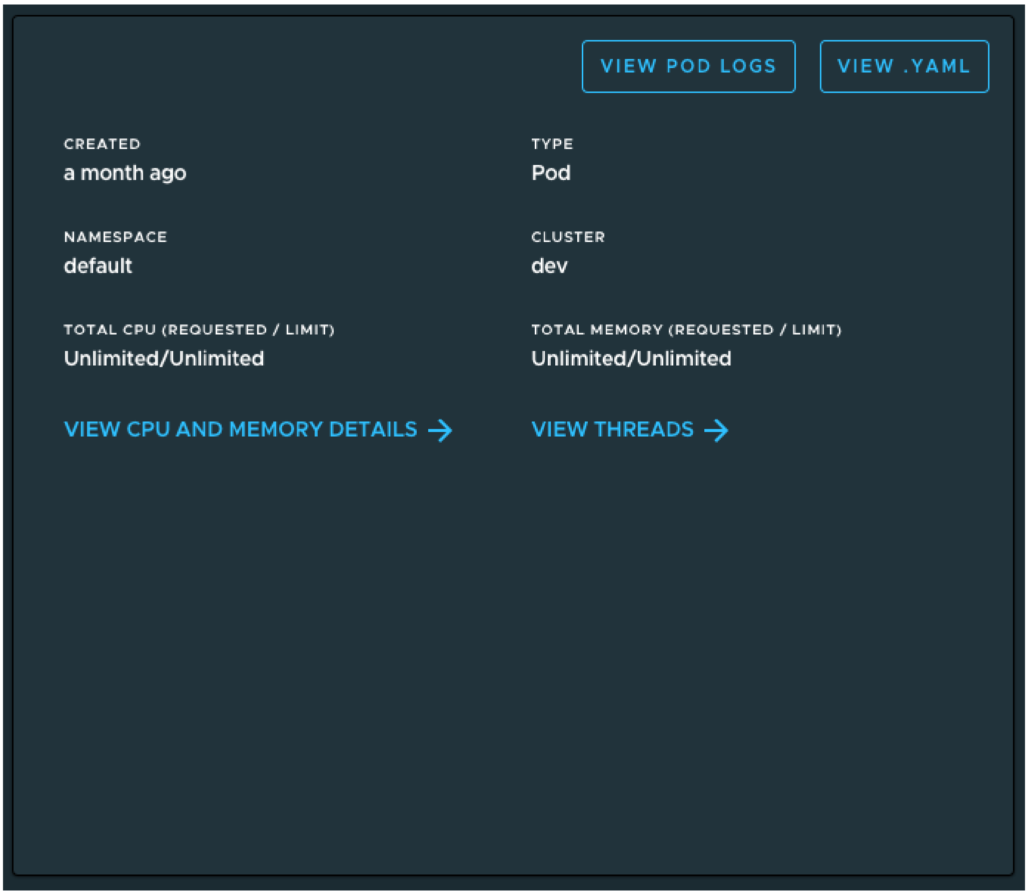
Detail pages

The Runtime Resources Visibility plug-in provides additional details of the Kubernetes resources in the Detail pages.

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



VIEW POD LOGS
VIEW .YAML

CREATED
a month ago

TYPE
Pod

NAMESPACE
default

CLUSTER
dev

TOTAL CPU (REQUESTED / LIMIT)
Unlimited/Unlimited

TOTAL MEMORY (REQUESTED / LIMIT)
Unlimited/Unlimited

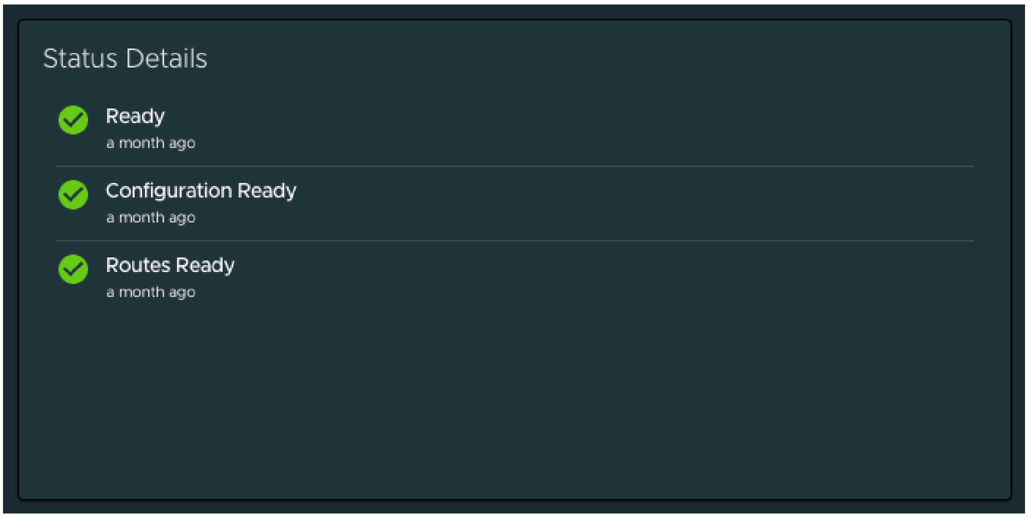
VIEW CPU AND MEMORY DETAILS →
VIEW THREADS →

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.



Status Details

✓ Ready
a month ago

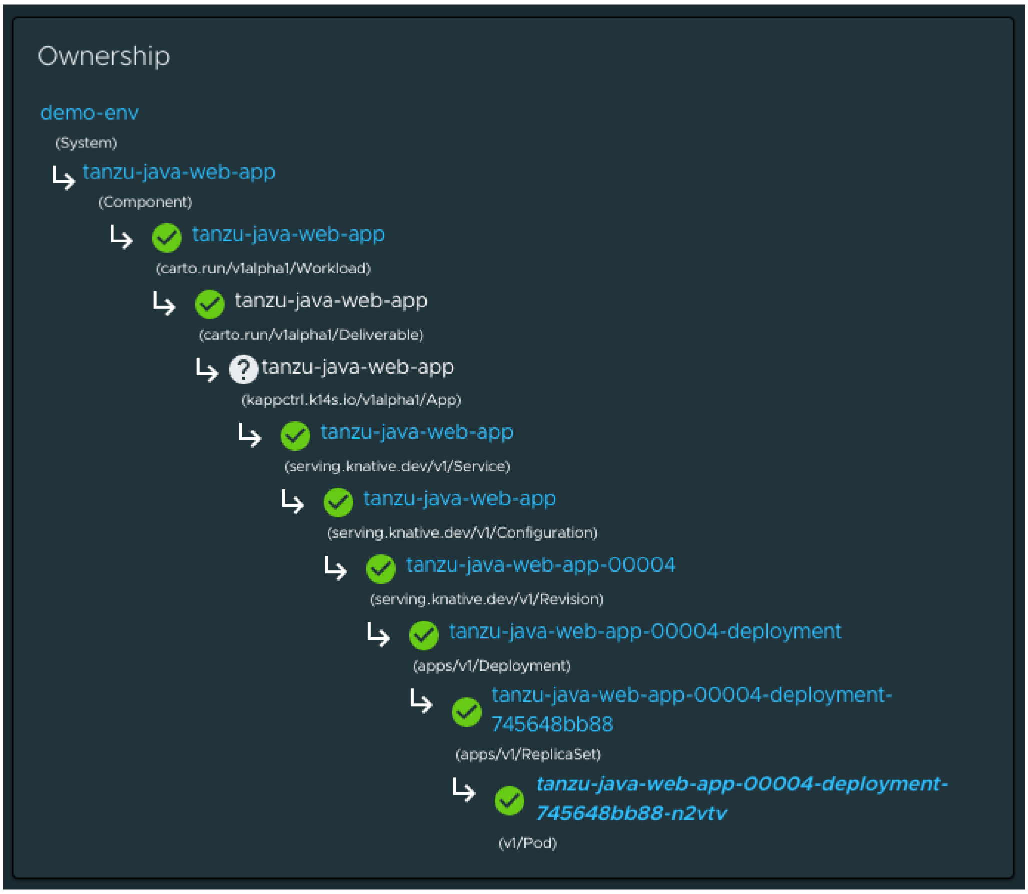
✓ Configuration Ready
a month ago

✓ Routes Ready
a month ago

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.

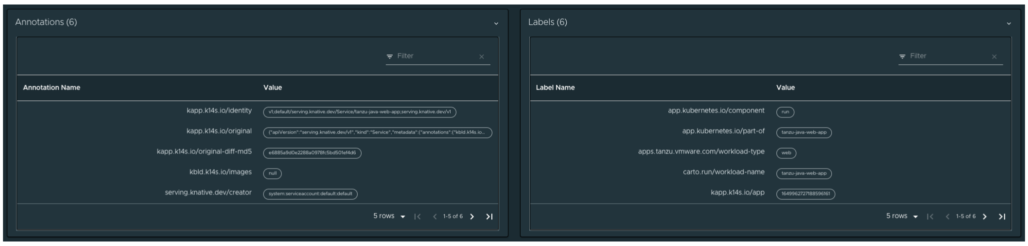


Ownership

- demo-env (System)
  - ↳ tanzu-java-web-app (Component)
    - ↳ ✓ tanzu-java-web-app (carto.run/v1alpha1/Workload)
      - ↳ ✓ tanzu-java-web-app (carto.run/v1alpha1/Deliverable)
        - ↳ ? tanzu-java-web-app (kappctrl.k14s.io/v1alpha1/App)
          - ↳ ✓ tanzu-java-web-app (serving.knative.dev/v1/Service)
            - ↳ ✓ tanzu-java-web-app (serving.knative.dev/v1/Configuration)
              - ↳ ✓ tanzu-java-web-app-00004 (serving.knative.dev/v1/Revision)
                - ↳ ✓ tanzu-java-web-app-00004-deployment (apps/v1/Deployment)
                  - ↳ ✓ tanzu-java-web-app-00004-deployment-745648bb88 (apps/v1/ReplicaSet)
                    - ↳ ✓ *tanzu-java-web-app-00004-deployment-745648bb88-n2vtv* (v1/Pod)

Annotations and Labels

The Annotations and Labels card displays information about `metadata.annotations` and `metadata.labels`.



Annotations (6)

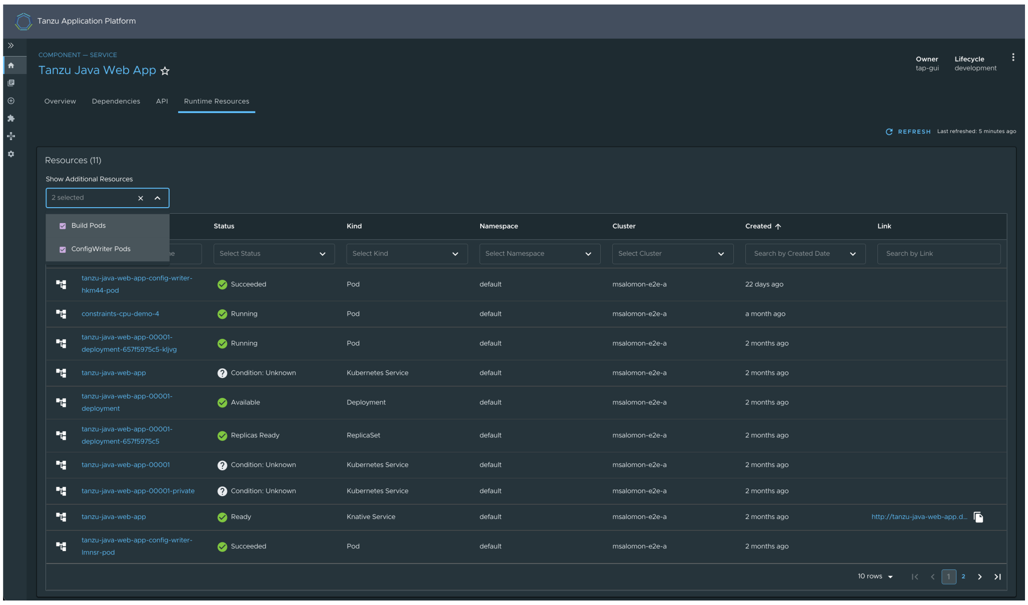
Annotation Name	Value
kapp.k14s.io/identity	v1;default/serving.knative.dev/Service/tanzu-java-web-app;serving.knative.dev/v1
kapp.k14s.io/original	{"apiVersion":"serving.knative.dev/v1","kind":"Service","metadata":{"annotations":{"kbld.k14s.io...
kapp.k14s.io/original-diff-md5	e6885a9d0e2288a0978fc5bd501ef4d6
kbld.k14s.io/images	null
serving.knative.dev/creator	system:serviceaccount:default:default

Labels (6)

Label Name	Value
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	tanzu-java-web-app
kapp.k14s.io/app	1649962727188596161

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.



COMPONENT — SERVICE

Tanzu Java Web App ☆

Owner: tap-gui

Lifecycle: development

Last refreshed: 5 minutes ago

Resources (11)

Show Additional Resources

2 selected

☑ Build Pods

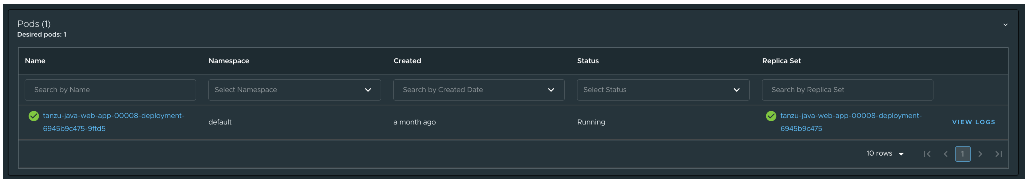
☑ ConfigWriter Pods

		Status	Kind	Namespace	Cluster	Created ↑	Link
		Select Status	Select Kind	Select Namespace	Select Cluster	Search by Created Date	Search by Link
	tanzu-java-web-app-config-writer-hkm44-pod	✓ Succeeded	Pod	default	msalomon-e2e-a	22 days ago	
	constraints-cpu-demo-4	✓ Running	Pod	default	msalomon-e2e-a	a month ago	
	tanzu-java-web-app-00001-deployment-657f5975c5-kljvg	✓ Running	Pod	default	msalomon-e2e-a	2 months ago	
	tanzu-java-web-app	? Condition: Unknown	Kubernetes Service	default	msalomon-e2e-a	2 months ago	
	tanzu-java-web-app-00001-deployment	✓ Available	Deployment	default	msalomon-e2e-a	2 months ago	
	tanzu-java-web-app-00001-deployment-657f5975c5	✓ Replicas Ready	ReplicaSet	default	msalomon-e2e-a	2 months ago	
	tanzu-java-web-app-00001	? Condition: Unknown	Kubernetes Service	default	msalomon-e2e-a	2 months ago	
	tanzu-java-web-app-00001-private	? Condition: Unknown	Kubernetes Service	default	msalomon-e2e-a	2 months ago	
	tanzu-java-web-app	✓ Ready	Knative Service	default	msalomon-e2e-a	2 months ago	http://tanzu-java-web-app.d...
	tanzu-java-web-app-config-writer-lmnsr-pod	✓ Succeeded	Pod	default	msalomon-e2e-a	2 months ago	

10 rows 1 2

Navigating to the pod Details page

Users can see the pod table in each resource details page.



Pods (1)

Desired pods: 1

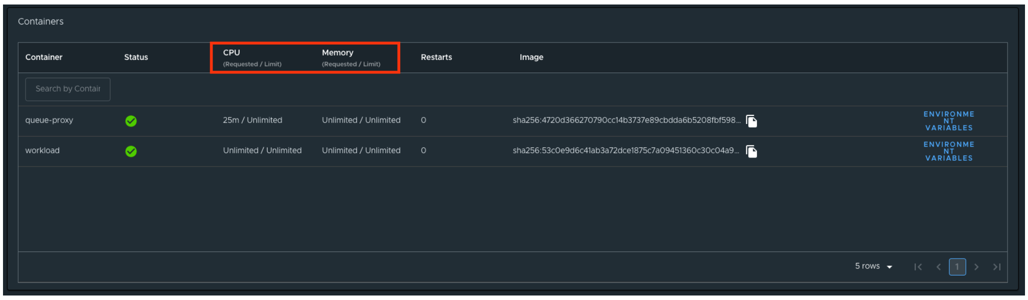
Name	Namespace	Created	Status	Replica Set	
Search by Name	Select Namespace	Search by Created Date	Select Status	Search by Replica Set	
✓ tanzu-java-web-app-00008-deployment-6945b9c475-9ftd5	default	a month ago	Running	✓ tanzu-java-web-app-00008-deployment-6945b9c475	VIEW LOGS

10 rows 1

Overview of pod metrics

The overview card displays the user-configured resource limits on the pod, defined in accordance with the Kubernetes documentation. These limits do not represent actual real-time resource use. To monitor actual real-time resource use, see Application Live View for Spring Boot Applications in Tanzu Application Platform GUI.

Each container displays its resource limits, if defined.

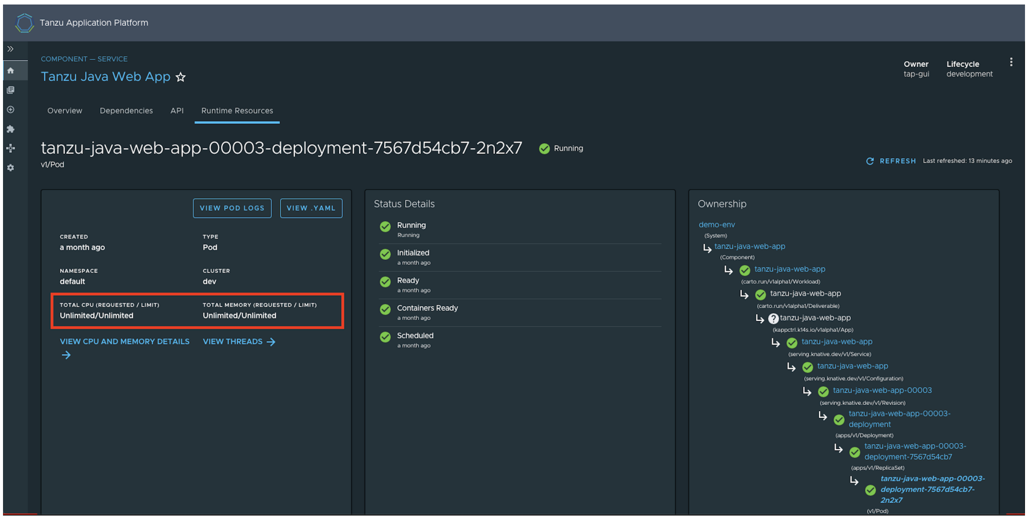


Containers

Container	Status	CPU (Requested / Limit)	Memory (Requested / Limit)	Restarts	Image	
Search by Contair						
queue-proxy	✓	25m / Unlimited	Unlimited / Unlimited	0	sha256:4720d366270790cc14b3737e89cbdda6b5208fbf598...	ENVIRONMENT VARIABLES
workload	✓	Unlimited / Unlimited	Unlimited / Unlimited	0	sha256:53c0e9d6c41ab3a72dce1875c7a09451360c30c04a9...	ENVIRONMENT VARIABLES

5 rows 1

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.



Owner: tap-gui
Lifecycle: development

tanzu-java-web-app-00003-deployment-7567d54cb7-2n2x7 ✓ Running
v1/Pod

Last refreshed: 13 minutes ago

CREATED: a month ago
TYPE: Pod
NAMESPACE: default
CLUSTER: dev
TOTAL CPU (REQUESTED / LIMIT): Unlimited/Unlimited
TOTAL MEMORY (REQUESTED / LIMIT): Unlimited/Unlimited

Status Details

- ✓ Running — Running
- ✓ Initialized — a month ago
- ✓ Ready — a month ago
- ✓ Containers Ready — a month ago
- ✓ Scheduled — a month ago

Ownership

```
demo-env
(System)
↳ tanzu-java-web-app
  (Component)
  ↳ ✓ tanzu-java-web-app
    (carto.run/v1alpha1/Workload)
    ↳ ✓ tanzu-java-web-app
      (carto.run/v1alpha1/Deliverable)
      ↳ ? tanzu-java-web-app
        (kappctrl.k14s.io/v1alpha1/App)
        ↳ ✓ tanzu-java-web-app
          (serving.knative.dev/v1/Service)
          ↳ ✓ tanzu-java-web-app
            (serving.knative.dev/v1/Configuration)
            ↳ ✓ tanzu-java-web-app-00003
              (serving.knative.dev/v1/Revision)
              ↳ ✓ tanzu-java-web-app-00003-deployment
                (apps/v1/Deployment)
                ↳ ✓ tanzu-java-web-app-00003-deployment-7567d54cb7
                  (apps/v1/ReplicaSet)
                  ↳ ✓ tanzu-java-web-app-00003-deployment-7567d54cb7-2n2x7
                    (v1/Pod)
```

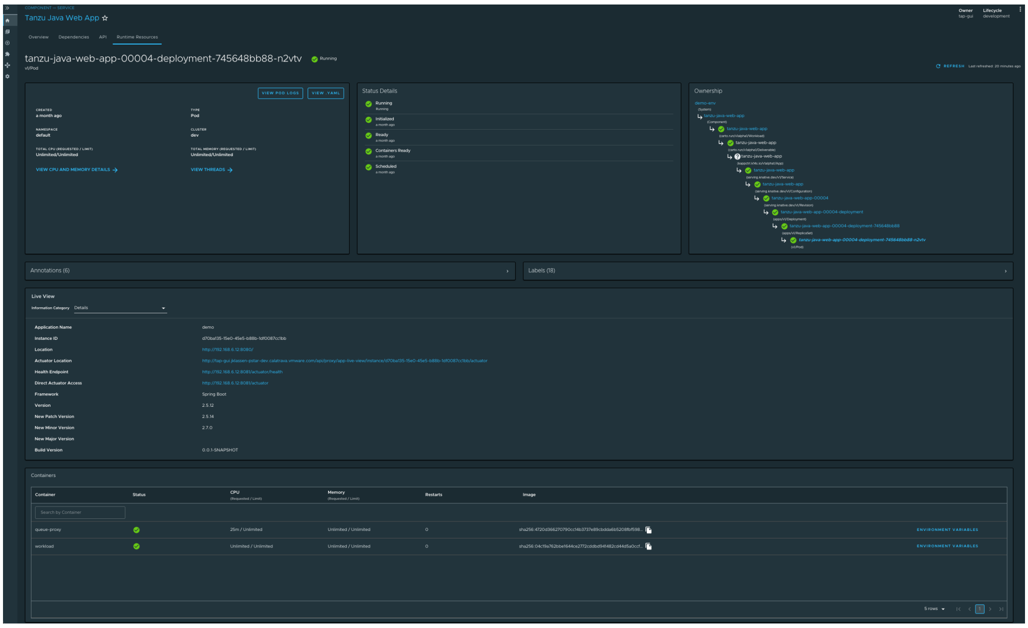
Namespace-level resource limits, such as default memory limits and default CPU limits, are not considered as part of these calculations.

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.



Owner: tap-gui
Lifecycle: development

tanzu-java-web-app-00004-deployment-745648bb88-n2vtv ✓ Running
v1/Pod

Last refreshed: 20 minutes ago

CREATED: a month ago
TYPE: Pod
NAMESPACE: default
CLUSTER: dev
TOTAL CPU (REQUESTED / LIMIT): Unlimited/Unlimited
TOTAL MEMORY (REQUESTED / LIMIT): Unlimited/Unlimited

Status Details

- ✓ Running — Running
- ✓ Initialized — a month ago
- ✓ Ready — a month ago
- ✓ Containers Ready — a month ago
- ✓ Scheduled — a month ago

Ownership

```
demo-env
(System)
↳ tanzu-java-web-app
  (Component)
  ↳ ✓ tanzu-java-web-app
    (carto.run/v1alpha1/Workload)
    ↳ ✓ tanzu-java-web-app
      (carto.run/v1alpha1/Deliverable)
      ↳ ? tanzu-java-web-app
        (kappctrl.k14s.io/v1alpha1/App)
        ↳ ✓ tanzu-java-web-app
          (serving.knative.dev/v1/Service)
          ↳ ✓ tanzu-java-web-app
            (serving.knative.dev/v1/Configuration)
            ↳ ✓ tanzu-java-web-app-00004
              (serving.knative.dev/v1/Revision)
              ↳ ✓ tanzu-java-web-app-00004-deployment
                (apps/v1/Deployment)
                ↳ ✓ tanzu-java-web-app-00004-deployment-745648bb88
                  (apps/v1/ReplicaSet)
                  ↳ ✓ tanzu-java-web-app-00004-deployment-745648bb88-n2vtv
                    (v1/Pod)
```

Live View

Information Category: Details

Application Name	demo
Instance ID	d70ba135-15e0-45e5-b88b-1df0087cc1bb
Location	http://192.168.6.12:8080/
Actuator Location	http://tap-gui.jklassen-pstar-dev.calatrava.vmware.com/api/proxy/app-live-view/instance/d70ba135-15e0-45e5-b88b-1df0087cc1bb/actuator
Health Endpoint	http://192.168.6.12:8081/actuator/health
Direct Actuator Access	http://192.168.6.12:8081/actuator
Framework	Spring Boot
Version	2.5.12
New Patch Version	2.5.14
New Minor Version	2.7.0
New Major Version	
Build Version	0.0.1-SNAPSHOT

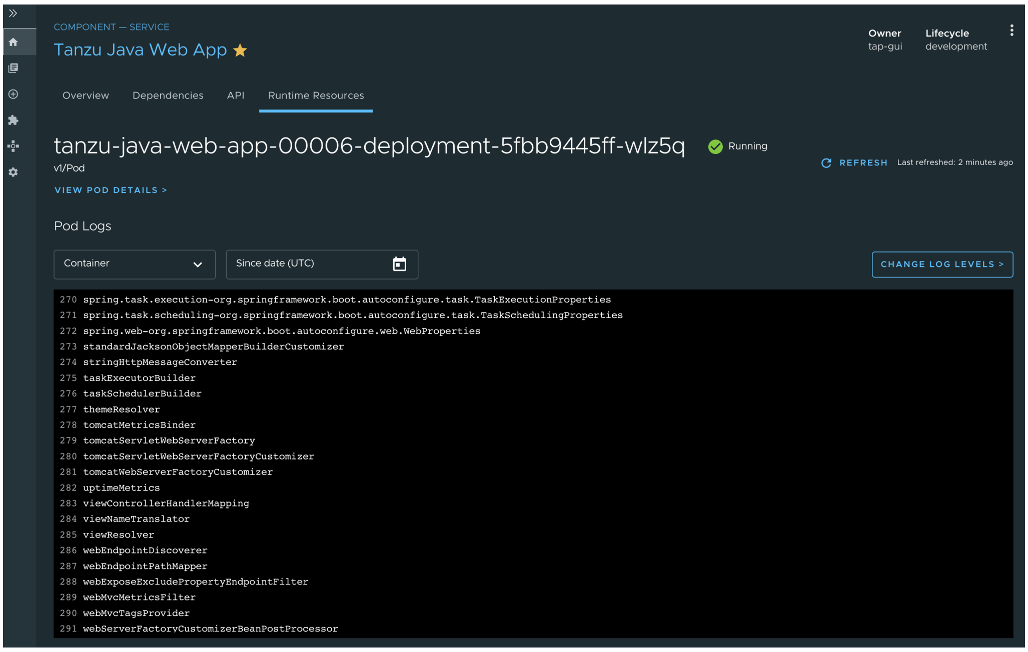
Containers

Container	Status	CPU (Requested / Limit)	Memory (Requested / Limit)	Restarts	Image	
Search by Container						
queue-proxy	✓	25m / Unlimited	Unlimited / Unlimited	0	sha256:4720d366270790cc14b3737e89cbdda6b5208fbf598...	ENVIRONMENT VARIABLES
workload	✓	Unlimited / Unlimited	Unlimited / Unlimited	0	sha256:04c19a762bbe1644ce2772cddbd941482cd44d5a0ccf...	ENVIRONMENT VARIABLES

Viewing pod logs

To view logs for a pod, click **View Pod Logs** from the **Pod Details** page. By default, you see all the logs for the pod since its creation for all the pod's containers.

The logs displayed are not streamed in real time. To fetch new log entries, click **Refresh** in the upper right corner of the page.



```
spring.task.execution-org.springframework.boot.autoconfigure.task.TaskExecutionProperties
spring.task.scheduling-org.springframework.boot.autoconfigure.task.TaskSchedulingProperties
spring.web-org.springframework.boot.autoconfigure.web.WebProperties
standardJacksonObjectMapperBuilderCustomizer
stringHttpMessageConverter
taskExecutorBuilder
taskSchedulerBuilder
themeResolver
tomcatMetricsBinder
tomcatServletWebServerFactory
tomcatServletWebServerFactoryCustomizer
tomcatWebServerFactoryCustomizer
uptimeMetrics
viewControllerHandlerMapping
viewNameTranslator
viewResolver
webEndpointDiscoverer
webEndpointPathMapper
webExposeExcludePropertyEndpointFilter
webMvcMetricsFilter
webMvcTagsProvider
webServerFactoryCustomizerBeanPostProcessor
```

Filtering by container

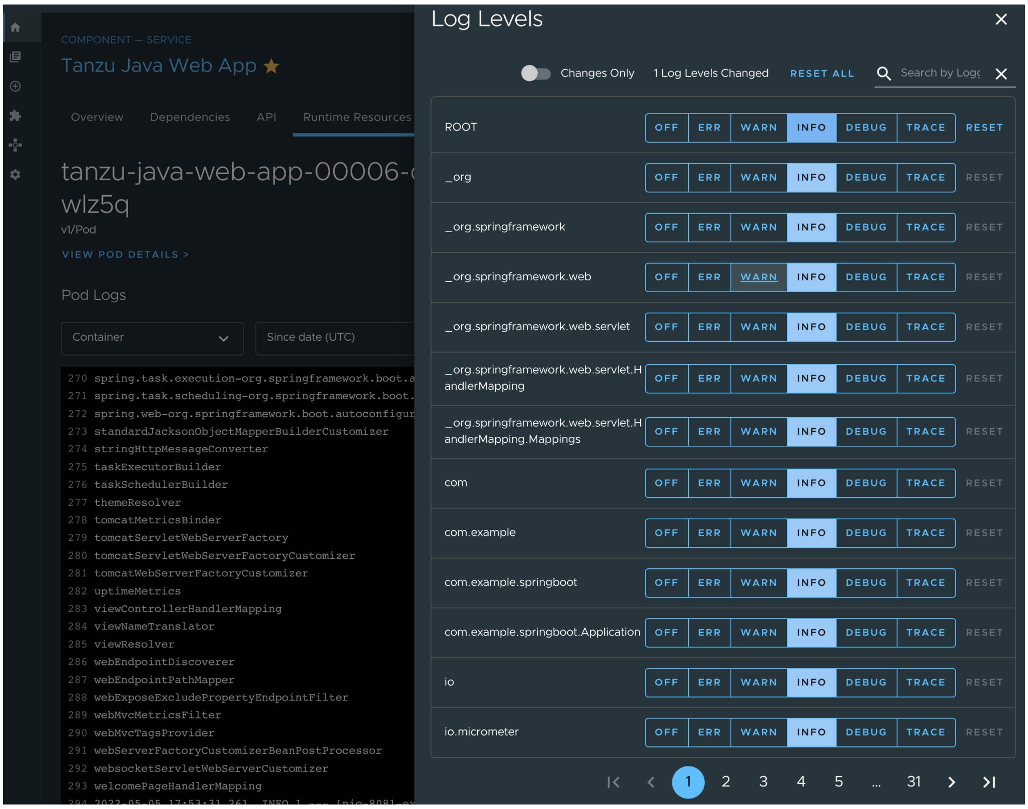
To display logs for a specific container only, select the desired container from the **Container** drop-down menu. Deselecting this drop-down menu causes logs for all containers within the pod to appear.

Filtering by date and time

To see all 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. Deselect this field to see all logs created since the pod was created.

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 allows you to select levels for each logger associated with your application.



tanzu-java-web-app-00006-
wlz5q
v1/Pod

```
spring.task.execution-org.springframework.boot.a
spring.task.scheduling-org.springframework.boot.
spring.web-org.springframework.boot.autoconfigur
standardJacksonObjectMapperBuilderCustomizer
stringHttpMessageConverter
taskExecutorBuilder
taskSchedulerBuilder
themeResolver
tomcatMetricsBinder
tomcatServletWebServerFactory
tomcatServletWebServerFactoryCustomizer
tomcatWebServerFactoryCustomizer
uptimeMetrics
viewControllerHandlerMapping
viewNameTranslator
viewResolver
webEndpointDiscoverer
webEndpointPathMapper
webExposeExcludePropertyEndpointFilter
webMvcMetricsFilter
webMvcTagsProvider
webServerFactoryCustomizerBeanPostProcessor
websocketServletWebServerCustomizer
welcomePageHandlerMapping
2022-05-05 17:53:31.261  INFO 1 --- [nio-8081-ex
```

ROOT	OFF	ERR	WARN	[✓ INFO]	DEBUG	TRACE	RESET
_org	OFF	ERR	WARN	[✓ INFO]	DEBUG	TRACE	RESET
_org.springframework	OFF	ERR	WARN	[✓ INFO]	DEBUG	TRACE	RESET
_org.springframework.web	OFF	ERR	WARN	[✓ INFO]	DEBUG	TRACE	RESET
_org.springframework.web.servlet	OFF	ERR	WARN	[✓ INFO]	DEBUG	TRACE	RESET
_org.springframework.web.servlet.HandlerMapping	OFF	ERR	WARN	[✓ INFO]	DEBUG	TRACE	RESET
_org.springframework.web.servlet.HandlerMapping.Mappings	OFF	ERR	WARN	[✓ INFO]	DEBUG	TRACE	RESET
com	OFF	ERR	WARN	[✓ INFO]	DEBUG	TRACE	RESET
com.example	OFF	ERR	WARN	[✓ INFO]	DEBUG	TRACE	RESET
com.example.springboot	OFF	ERR	WARN	[✓ INFO]	DEBUG	TRACE	RESET
com.example.springboot.Application	OFF	ERR	WARN	[✓ INFO]	DEBUG	TRACE	RESET
io	OFF	ERR	WARN	[✓ INFO]	DEBUG	TRACE	RESET
io.micrometer	OFF	ERR	WARN	[✓ INFO]	DEBUG	TRACE	RESET

To change the levels for your application, select the desired level for each logger presented and then close the panel by clicking **X** in the upper right corner of the panel or by pressing the Escape key on your keyboard.

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. Click **Refresh** in the upper right corner of the page to fetch new messages after changing log levels.

After refreshing, 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 producing logs at the expected levels. Your application might be idling or otherwise not running a code path that starts the desired logger.

Application Live View in Tanzu Application Platform GUI

This topic tells you about Application Live View in Tanzu Application Platform GUI (commonly called TAP GUI).

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: only running processes are recognized by Application Live View. 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 Application Platform GUI. To use the Application Live View plug-in:

1. Select the relevant component under the **Organization Catalog** in Tanzu Application Platform GUI.
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 Application Platform GUI

This topic tells you about Application Live View in Tanzu Application Platform GUI (commonly called TAP GUI).

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: only running processes are recognized by Application Live View. 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 Application Platform GUI. To use the Application Live View plug-in:

1. Select the relevant component under the **Organization Catalog** in Tanzu Application Platform GUI.
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 Application Platform GUI

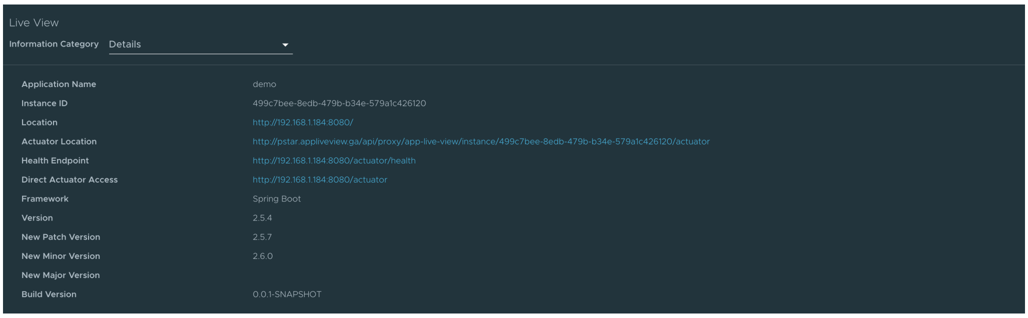
This topic tells you about the Application Live View pages for Spring Boot Applications in Tanzu Application Platform GUI (commonly called TAP GUI).

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

The user can navigate between **Information Categories** by selecting from the drop-down menu on the top right corner of the page.

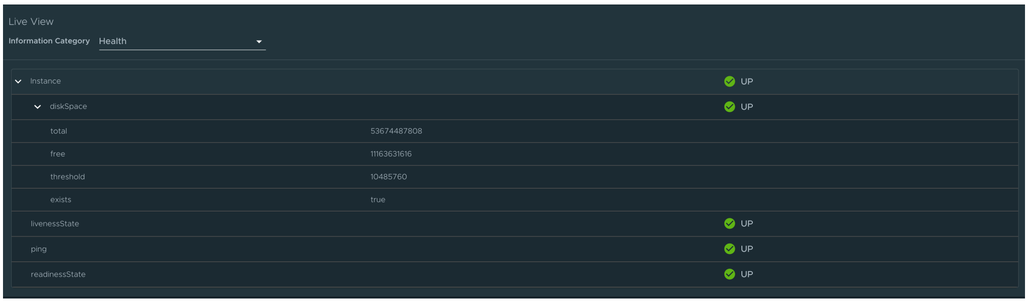


Information Category: Details

Application Name	demo
Instance ID	499c7bee-8edb-479b-b34e-579a1c426120
Location	http://192.168.1.184:8080/
Actuator Location	http://pstar.appliveview.ga/api/proxy/app-live-view/instance/499c7bee-8edb-479b-b34e-579a1c426120/actuator
Health Endpoint	http://192.168.1.184:8080/actuator/health
Direct Actuator Access	http://192.168.1.184:8080/actuator
Framework	Spring Boot
Version	2.5.4
New Patch Version	2.5.7
New Minor Version	2.6.0
New Major Version	
Build Version	0.0.1-SNAPSHOT

Health page

To navigate to the health page, the user can 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, details associated with each of the components.



Information Category: Health

Instance		✓ UP
diskSpace		✓ UP
total	53674487808	
free	11163631616	
threshold	10485760	
exists	true	
livenessState		✓ UP
ping		✓ UP
readinessState		✓ UP

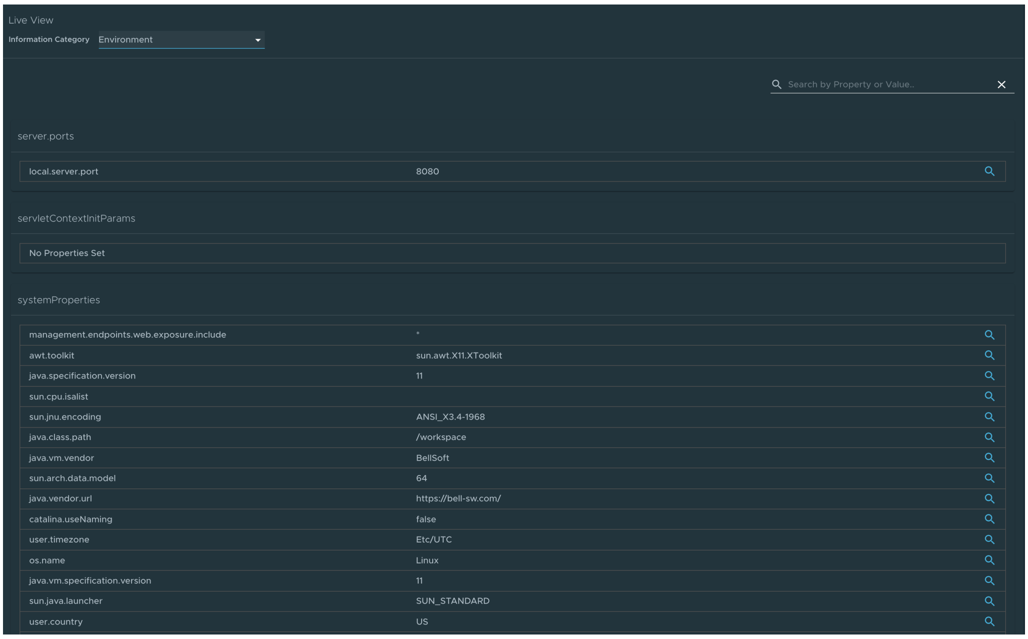
Environment page

To navigate to the **Environment** page, the user can 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 features:

- The UI has search feature that enables the user to search for a property or values.
- Each property has a search icon at the right corner which helps the user quickly see all the occurrences of a specific property key without manually typing in the search field. Clicking the search button trims down the page to that property name.
- The **Refresh Scope** on the top right corner of the page probes the application to refresh all the environment properties.
- The user can edit existing property by clicking the **Override** in the row and editing the value. After the value is saved, the user can see the updated property in the Applied overrides section at the top of the page.
- The **Reset** resets the environment property to the original state
- The user can edit or remove the overridden environment variables in the **Applied Overrides** section.
- The **Applied Overrides** section also enables the user to add new environment properties to the application.

The `management.endpoint.env.post.enabled=true` has to be set in the application config properties of the application and a corresponding, editable Environment has to be present in the application.



server.ports

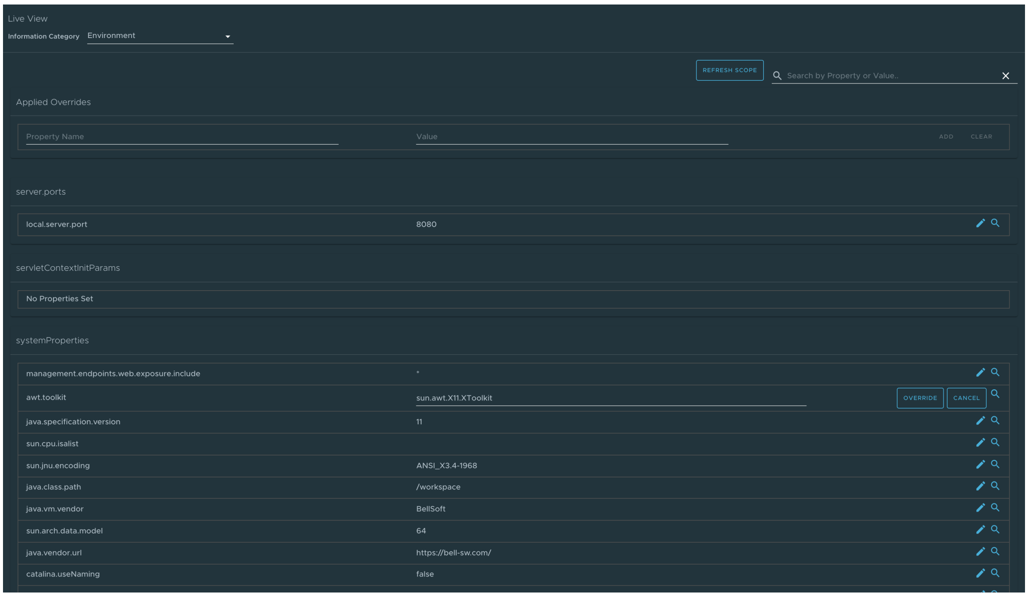
local.server.port	8080

servletContextInitParams

No Properties Set

systemProperties

management.endpoints.web.exposure.include	*
awt.toolkit	sun.awt.X11.XToolkit
java.specification.version	11
sun.cpu.isalist	
sun.jnu.encoding	ANSI_X3.4-1968
java.class.path	/workspace
java.vm.vendor	BellSoft
sun.arch.data.model	64
java.vendor.url	https://bell-sw.com/
catalina.useNaming	false
user.timezone	Etc/UTC
os.name	Linux
java.vm.specification.version	11
sun.java.launcher	SUN_STANDARD
user.country	US



Live View

Information Category: Environment

Applied Overrides

server.ports

local.server.port	8080

servletContextInitParams

No Properties Set

systemProperties

management.endpoints.web.exposure.include	*
awt.toolkit	sun.awt.X11.XToolkit
java.specification.version	11
sun.cpu.isalist	
sun.jnu.encoding	ANSI_X3.4-1968
java.class.path	/workspace
java.vm.vendor	BellSoft
sun.arch.data.model	64
java.vendor.url	https://bell-sw.com/
catalina.useNaming	false

Log Levels page

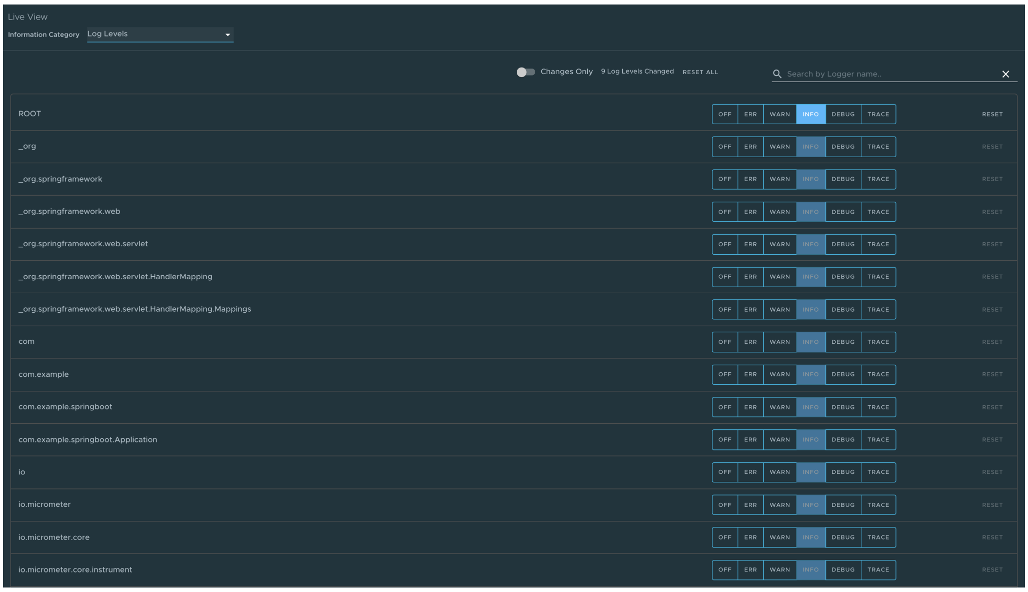
To navigate to the **Log Levels** page, the user can select the **Log Levels** option from the **Information Category** drop-down menu. The log levels page provides access to the application's loggers and the configuration of their levels.

The user can configure the log levels such as INFO, DEBUG, and TRACE in real time from the UI. The user can search for a package and edit its respective log level. The user 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. The search feature enables the user 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

The UI allows the user to change the log levels and see the live changes on the application. These changes are temporary and will go away if the underlying pod gets restarted.



Live View

Information Category: Log Levels

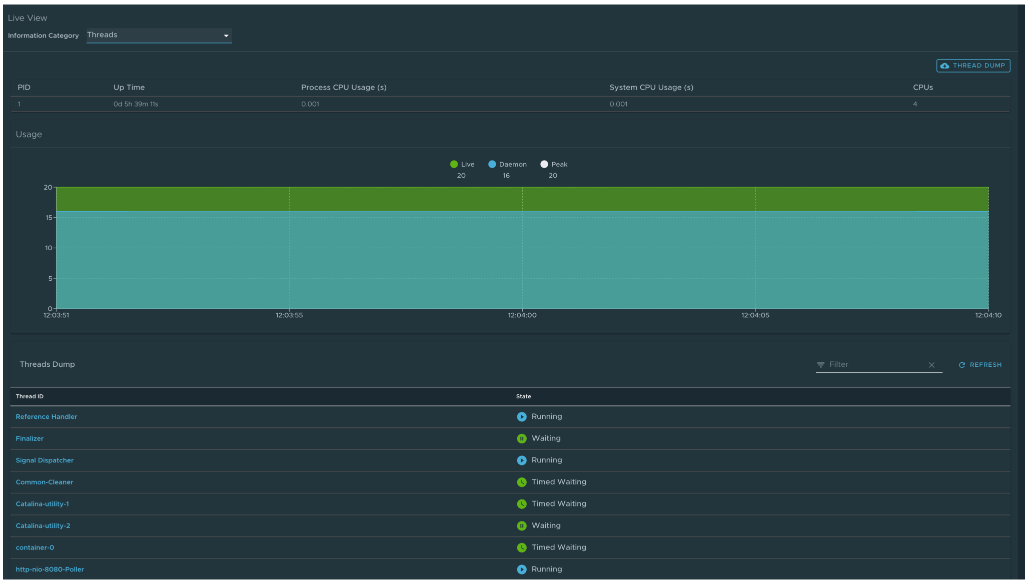
ROOT	OFF ERR WARN INFO DEBUG TRACE RESET
_org	OFF ERR WARN INFO DEBUG TRACE RESET
_org.springframework	OFF ERR WARN INFO DEBUG TRACE RESET
_org.springframework.web	OFF ERR WARN INFO DEBUG TRACE RESET
_org.springframework.web.servlet	OFF ERR WARN INFO DEBUG TRACE RESET
_org.springframework.web.servlet.HandlerMapping	OFF ERR WARN INFO DEBUG TRACE RESET
_org.springframework.web.servlet.HandlerMapping.Mappings	OFF ERR WARN INFO DEBUG TRACE RESET
com	OFF ERR WARN INFO DEBUG TRACE RESET
com.example	OFF ERR WARN INFO DEBUG TRACE RESET
com.example.springboot	OFF ERR WARN INFO DEBUG TRACE RESET
com.example.springboot.Application	OFF ERR WARN INFO DEBUG TRACE RESET
io	OFF ERR WARN INFO DEBUG TRACE RESET
io.micrometer	OFF ERR WARN INFO DEBUG TRACE RESET
io.micrometer.core	OFF ERR WARN INFO DEBUG TRACE RESET
io.micrometer.core.instrument	OFF ERR WARN INFO DEBUG TRACE RESET

Threads page

To navigate to the **Threads** page, the user can select the **Threads** option from the **Information Category** drop-down menu.

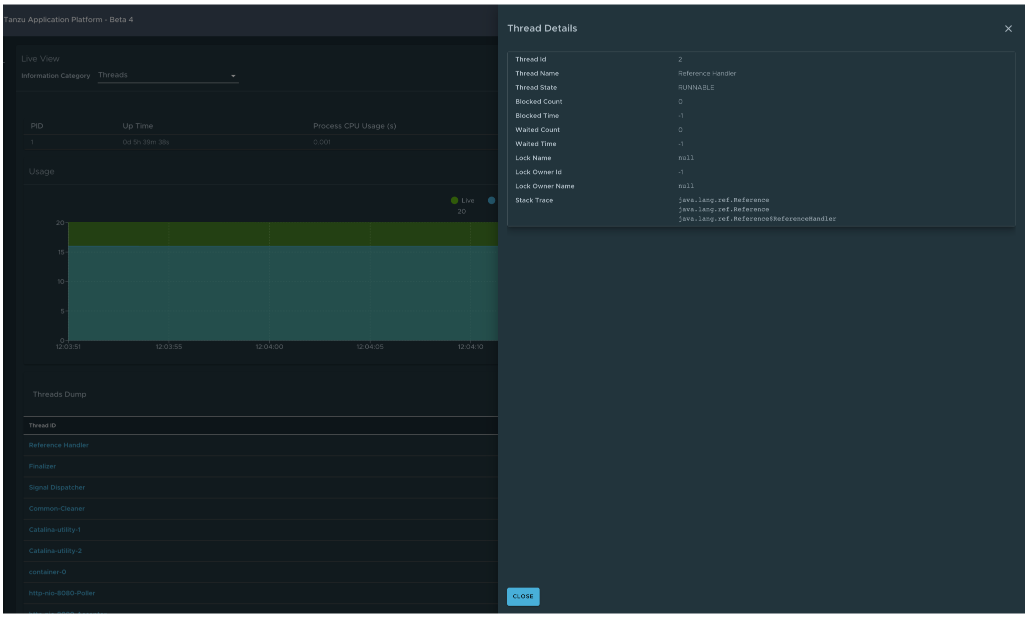
This page displays all details related to JVM threads and running processes of the application. This tracks live threads and daemon threads real-time. It is a snapshot of different thread states. Navigating to a thread state displays all the information about a particular thread and its stack trace.

The search feature enables the user to search for threads by thread ID or state. The refresh icon refreshes to the latest state of the threads. The user can view more thread details by clicking on the Thread ID. The page also has a feature to download thread dump for analysis purposes.



PID	Up Time	Process CPU Usage (s)	System CPU Usage (s)	CPUs
1	0d 5h 39m 11s	0.001	0.001	4

Thread ID	State
Reference Handler	Running
Finalizer	Waiting
Signal Dispatcher	Running
Common-Cleaner	Timed Waiting
Catalina-utility-1	Timed Waiting
Catalina-utility-2	Waiting
container-0	Timed Waiting
http-nio-8080-Poller	Running



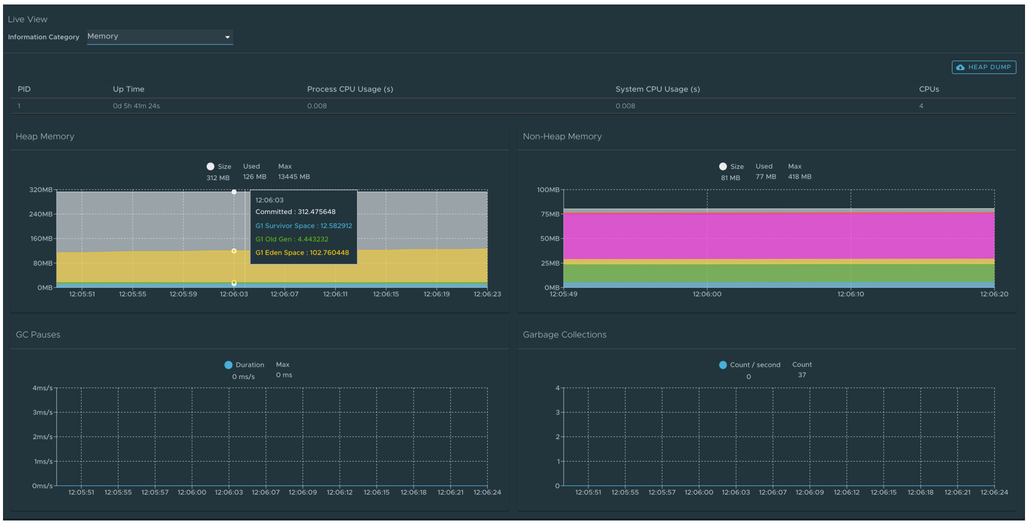
Thread Id	2
Thread Name	Reference Handler
Thread State	RUNNABLE
Blocked Count	0
Blocked Time	-1
Waited Count	0
Waited Time	-1
Lock Name	null
Lock Owner Id	-1
Lock Owner Name	null
Stack Trace	java.lang.ref.Reference java.lang.ref.Reference java.lang.ref.Reference$ReferenceHandler

Memory page

To navigate to the **Memory** page, the user can 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** on top right corner allows the user to download heap dump data.



Live View

Information Category: Memory

PID	Up Time	Process CPU Usage (s)	System CPU Usage (s)	CPUs
1	0d 5h 41m 24s	0.008	0.008	4

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.

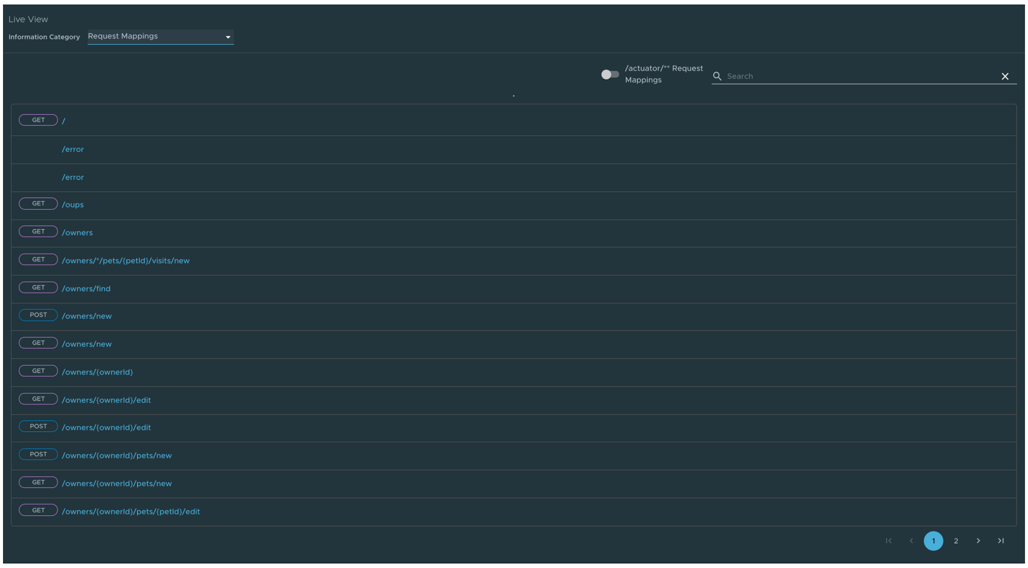
Request Mappings page

To navigate to the Request Mappings page, the user should select the **Request Mappings** option from the **Information Category** drop-down menu.

This page provides information about the application's request mappings. For each of the mapping, it displays the request handler method. The user can view more details of the request mapping such as header metadata of the application. That is, it produces, consumes and HTTP method by clicking on the mapping.

The search feature enables the user to search on the request mapping or the method. The toggle **/actuator/**** **Request Mappings** displays the actuator related mappings of the application.

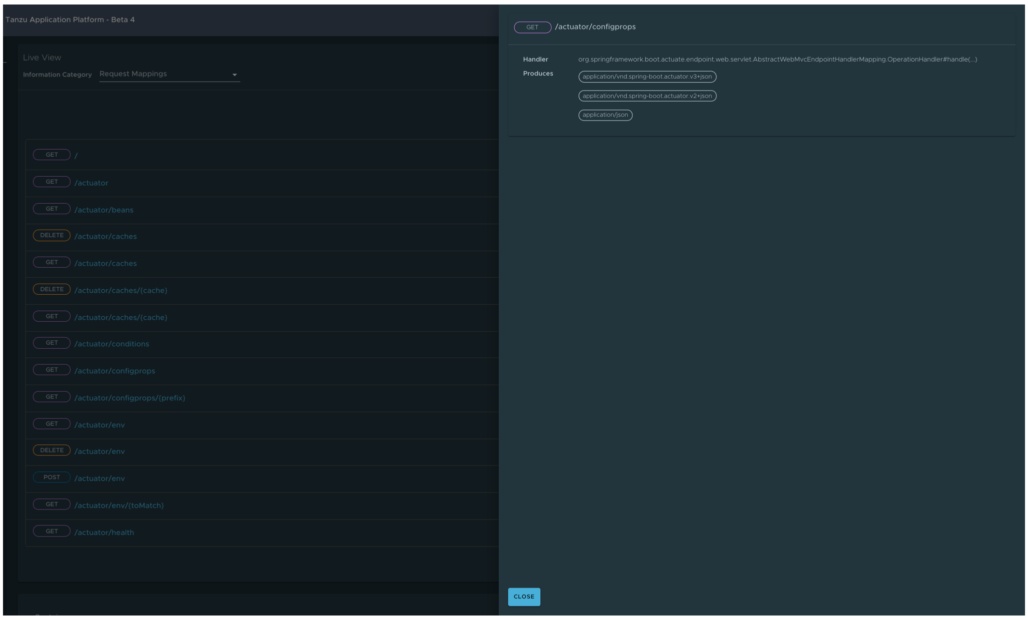
When the application actuator endpoint is exposed on `management.server.port`, the application does not return any actuator request mappings data in the context. The application displays a message when the actuator toggle is enabled.



Live View

Information Category: Request Mappings

GET	/
	/error
	/error
GET	/oups
GET	/owners
GET	/owners/*/pets/{petId}/visits/new
GET	/owners/find
POST	/owners/new
GET	/owners/new
GET	/owners/{ownerId}
GET	/owners/{ownerId}/edit
POST	/owners/{ownerId}/edit
POST	/owners/{ownerId}/pets/new
GET	/owners/{ownerId}/pets/new
GET	/owners/{ownerId}/pets/{petId}/edit



Tanzu Application Platform - Beta 4

Live View

Information Category: Request Mappings

GET	/
GET	/actuator
GET	/actuator/beans
DELETE	/actuator/caches
GET	/actuator/caches
DELETE	/actuator/caches/{cache}
GET	/actuator/caches/{cache}
GET	/actuator/conditions
GET	/actuator/configprops
GET	/actuator/configprops/{prefix}
GET	/actuator/env
DELETE	/actuator/env
POST	/actuator/env
GET	/actuator/env/{toMatch}
GET	/actuator/health

GET /actuator/configprops

Handler: org.springframework.boot.actuate.endpoint.web.servlet.AbstractWebMvcEndpointHandlerMapping.OperationHandler#handle(...)

Produces: application/vnd.spring-boot.actuator.v3+json
application/vnd.spring-boot.actuator.v2+json
application/json

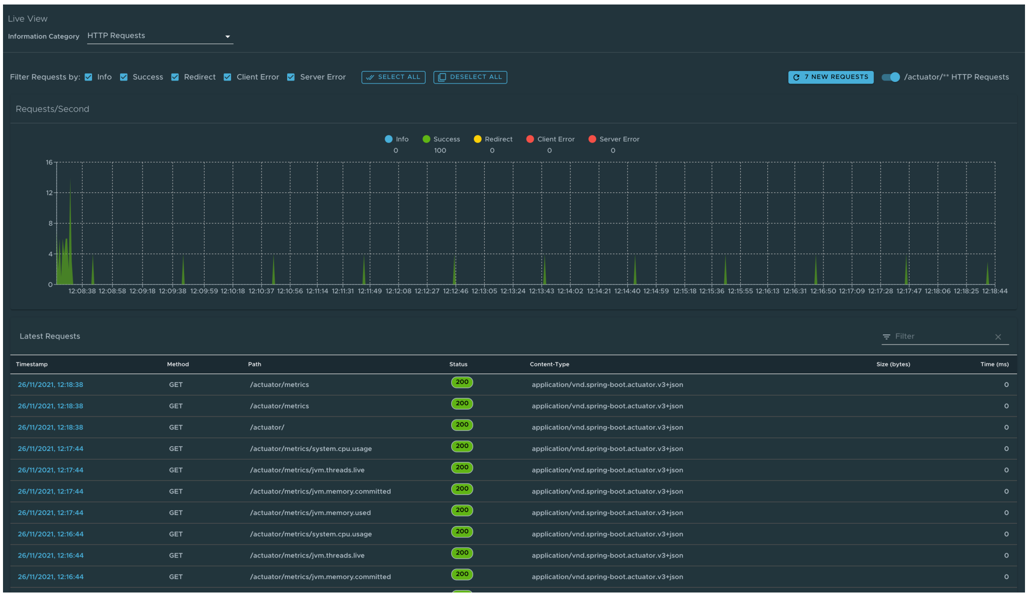
HTTP Requests page

To navigate to the HTTP Requests page, the user should 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 indicating the response status of all the requests. The user can filter on the response statuses which include info, success, redirects, client-errors, 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. The user can view more details of the request such as method, headers, 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.

When the 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.



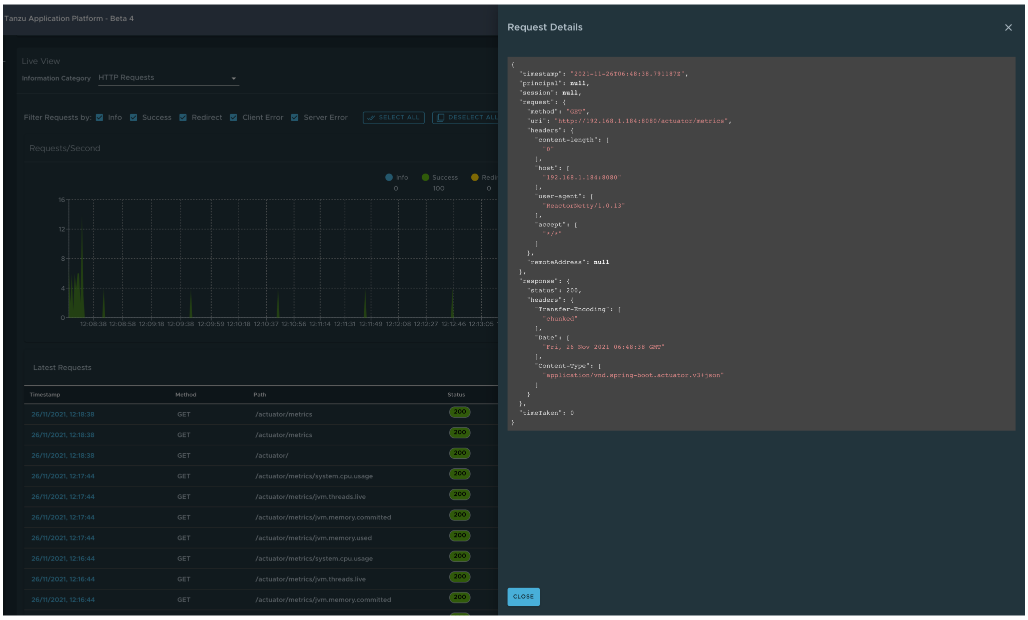
Live View

Information Category: HTTP Requests

Filter Requests by: ☑ Info ☑ Success ☑ Redirect ☑ Client Error ☑ Server Error

Latest Requests

Timestamp	Method	Path	Status	Content-Type	Size (bytes)	Time (ms)
26/11/2021, 12:18:38	GET	/actuator/metrics	200	application/vnd.spring-boot.actuator.v3+json		0
26/11/2021, 12:18:38	GET	/actuator/metrics	200	application/vnd.spring-boot.actuator.v3+json		0
26/11/2021, 12:18:38	GET	/actuator/	200	application/vnd.spring-boot.actuator.v3+json		0
26/11/2021, 12:17:44	GET	/actuator/metrics/system.cpu.usage	200	application/vnd.spring-boot.actuator.v3+json		0
26/11/2021, 12:17:44	GET	/actuator/metrics/jvm.threads.live	200	application/vnd.spring-boot.actuator.v3+json		0
26/11/2021, 12:17:44	GET	/actuator/metrics/jvm.memory.committed	200	application/vnd.spring-boot.actuator.v3+json		0
26/11/2021, 12:17:44	GET	/actuator/metrics/jvm.memory.used	200	application/vnd.spring-boot.actuator.v3+json		0
26/11/2021, 12:16:44	GET	/actuator/metrics/system.cpu.usage	200	application/vnd.spring-boot.actuator.v3+json		0
26/11/2021, 12:16:44	GET	/actuator/metrics/jvm.threads.live	200	application/vnd.spring-boot.actuator.v3+json		0
26/11/2021, 12:16:44	GET	/actuator/metrics/jvm.memory.committed	200	application/vnd.spring-boot.actuator.v3+json		0



Timestamp	Method	Path	Status
26/11/2021, 12:18:38	GET	/actuator/metrics	200
26/11/2021, 12:18:38	GET	/actuator/metrics	200
26/11/2021, 12:18:38	GET	/actuator/	200
26/11/2021, 12:17:44	GET	/actuator/metrics/system.cpu.usage	200
26/11/2021, 12:17:44	GET	/actuator/metrics/jvm.threads.live	200
26/11/2021, 12:17:44	GET	/actuator/metrics/jvm.memory.committed	200
26/11/2021, 12:17:44	GET	/actuator/metrics/jvm.memory.used	200
26/11/2021, 12:16:44	GET	/actuator/metrics/system.cpu.usage	200
26/11/2021, 12:16:44	GET	/actuator/metrics/jvm.threads.live	200
26/11/2021, 12:16:44	GET	/actuator/metrics/jvm.memory.committed	200

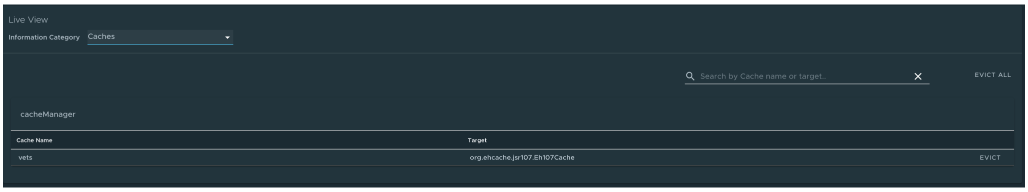
```
{
  "timestamp": "2021-11-26T06:48:38.791187Z",
  "principal": null,
  "session": null,
  "request": {
    "method": "GET",
    "uri": "http://192.168.1.184:8080/actuator/metrics",
    "headers": {
      "content-length": [
        "0"
      ],
      "host": [
        "192.168.1.184:8080"
      ],
      "user-agent": [
        "ReactorNetty/1.0.13"
      ],
      "accept": [
        "*/*"
      ]
    },
    "remoteAddress": null
  },
  "response": {
    "status": 200,
    "headers": {
      "Transfer-Encoding": [
        "chunked"
      ],
      "Date": [
        "Fri, 26 Nov 2021 06:48:38 GMT"
      ],
      "Content-Type": [
        "application/vnd.spring-boot.actuator.v3+json"
      ]
    }
  },
  "timeTaken": 0
}
```

Caches page

To navigate to the **Caches** page, the user can 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.

The search feature in the Caches Page enables the user to search for a specific cache/cache manager. The user can clear individual caches by clicking **Evict**. The user 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.

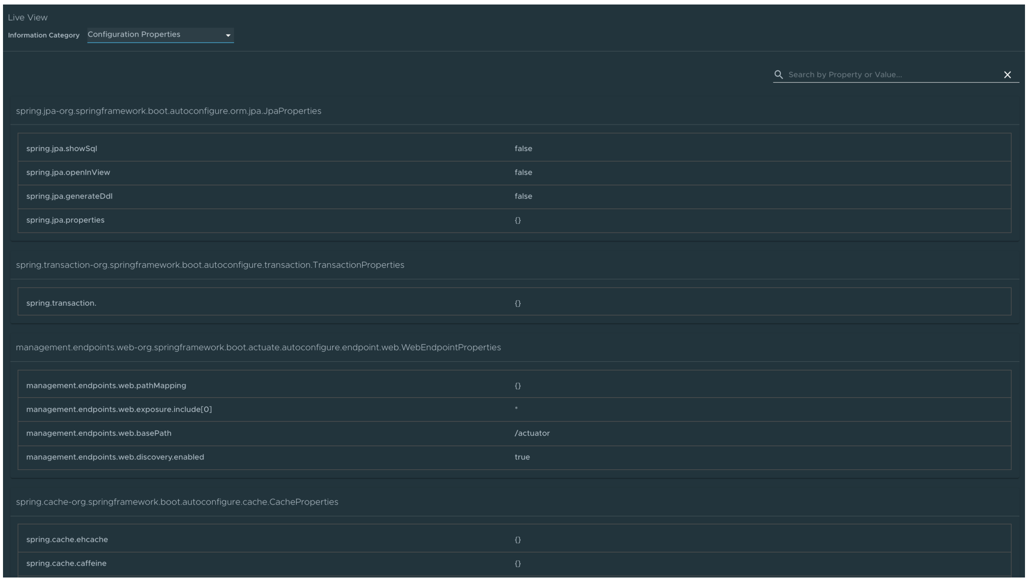


Cache Name	Target	
vets	org.ehcache.jsr107.Eh107Cache	EVICT

Configuration Properties page

To navigate to the **Configuration Properties** page, the user can 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. The search feature allows the user to look up for property's key/value or the bean name.



spring.jpa-org.springframework.boot.autoconfigure.orm.jpa.JpaProperties

spring.jpa.showSql	false
spring.jpa.openInView	false
spring.jpa.generateDdl	false
spring.jpa.properties	{}

spring.transaction-org.springframework.boot.autoconfigure.transaction.TransactionProperties

spring.transaction.	{}

management.endpoints.web-org.springframework.boot.actuate.autoconfigure.endpoint.web.WebEndpointProperties

management.endpoints.web.pathMapping	{}
management.endpoints.web.exposure.include[0]	*
management.endpoints.web.basePath	/actuator
management.endpoints.web.discovery.enabled	true

spring.cache-org.springframework.boot.autoconfigure.cache.CacheProperties

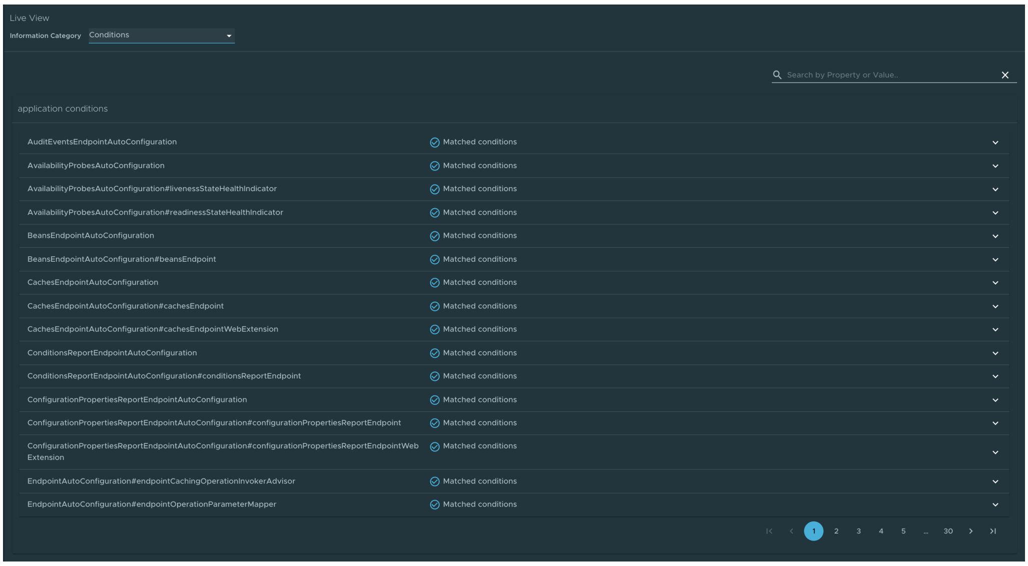
spring.cache.ehcache	{}
spring.cache.caffeine	{}

Conditions page

To navigate to the **Conditions** page, the user can 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 case of Spring Boot, this gives the user a view of all the beans configured in the application. When the user clicks on the bean name, the conditions and the reason for the conditional match is displayed.

In case of not 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. The user can filter out on the beans and the conditions using the search feature.



application conditions

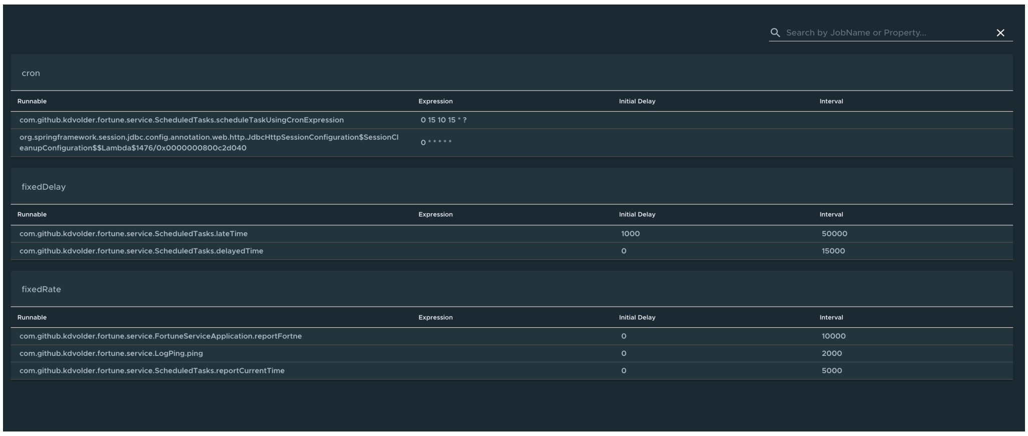
AuditEventsEndpointAutoConfiguration	✓ Matched conditions
AvailabilityProbesAutoConfiguration	✓ Matched conditions
AvailabilityProbesAutoConfiguration#livenessStateHealthIndicator	✓ Matched conditions
AvailabilityProbesAutoConfiguration#readinessStateHealthIndicator	✓ Matched conditions
BeansEndpointAutoConfiguration	✓ Matched conditions
BeansEndpointAutoConfiguration#beansEndpoint	✓ Matched conditions
CachesEndpointAutoConfiguration	✓ Matched conditions
CachesEndpointAutoConfiguration#cachesEndpoint	✓ Matched conditions
CachesEndpointAutoConfiguration#cachesEndpointWebExtension	✓ Matched conditions
ConditionsReportEndpointAutoConfiguration	✓ Matched conditions
ConditionsReportEndpointAutoConfiguration#conditionsReportEndpoint	✓ Matched conditions
ConfigurationPropertiesReportEndpointAutoConfiguration	✓ Matched conditions
ConfigurationPropertiesReportEndpointAutoConfiguration#configurationPropertiesReportEndpoint	✓ Matched conditions
ConfigurationPropertiesReportEndpointAutoConfiguration#configurationPropertiesReportEndpointWebExtension	✓ Matched conditions
EndpointAutoConfiguration#endpointCachingOperationInvokerAdvisor	✓ Matched conditions
EndpointAutoConfiguration#endpointOperationParameterMapper	✓ Matched conditions

Scheduled Tasks page

To navigate to the **Scheduled Tasks** page, the user can 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.

The user can search for a particular property or a task in the search bar to retrieve the task or property details.



cron

Runnable	Expression	Initial Delay	Interval
com.github.kdvolder.fortune.service.ScheduledTasks.scheduleTaskUsingCronExpression	0 15 10 15 * ?		
org.springframework.session.jdbc.config.annotation.web.http.JdbcHttpSessionConfiguration$SessionCleanupConfiguration$$Lambda$1476/0x0000000800c2d040	0 * * * * *		

fixedDelay

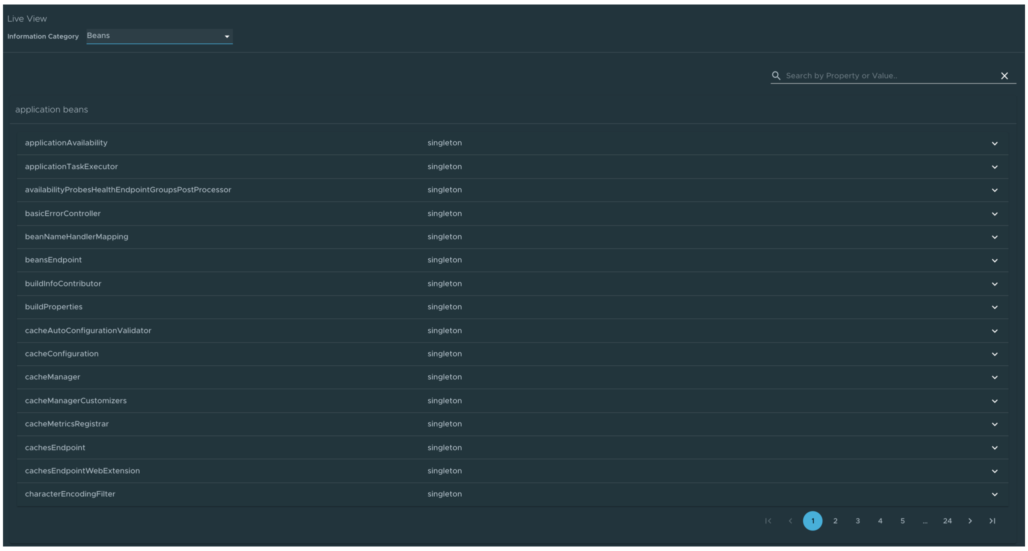
Runnable	Expression	Initial Delay	Interval
com.github.kdvolder.fortune.service.ScheduledTasks.lateTime		1000	50000
com.github.kdvolder.fortune.service.ScheduledTasks.delayedTime		0	15000

fixedRate

Runnable	Expression	Initial Delay	Interval
com.github.kdvolder.fortune.service.FortuneServiceApplication.reportFortne		0	10000
com.github.kdvolder.fortune.service.LogPing.ping		0	2000
com.github.kdvolder.fortune.service.ScheduledTasks.reportCurrentTime		0	5000

Beans page

To navigate to the **Beans** page, the user can 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. The user can search by the bean name or its corresponding fields.



application beans

applicationAvailability	singleton
applicationTaskExecutor	singleton
availabilityProbesHealthEndpointGroupsPostProcessor	singleton
basicErrorController	singleton
beanNameHandlerMapping	singleton
beansEndpoint	singleton
buildInfoContributor	singleton
buildProperties	singleton
cacheAutoConfigurationValidator	singleton
cacheConfiguration	singleton
cacheManager	singleton
cacheManagerCustomizers	singleton
cacheMetricsRegistrar	singleton
cachesEndpoint	singleton
cachesEndpointWebExtension	singleton
characterEncodingFilter	singleton

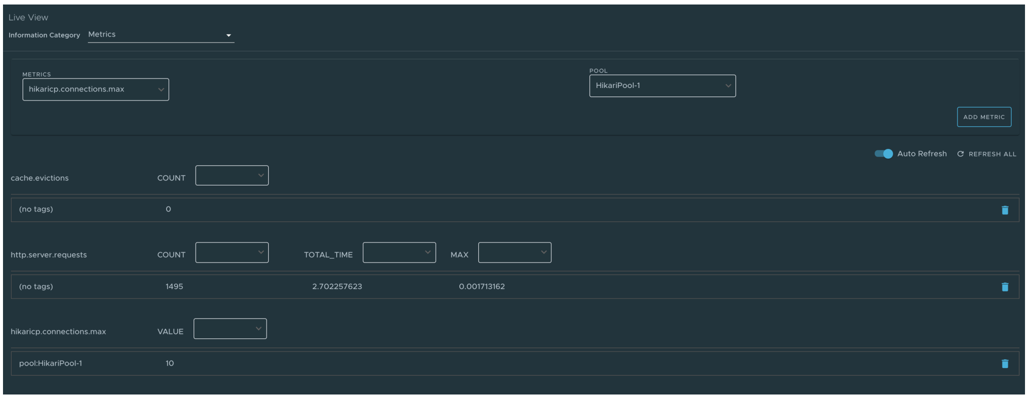
Metrics page

To navigate to the **Metrics** page, the user can select the **Metrics** option from the **Information Category** drop-down menu.

The metrics page provides access to application metrics information. The user 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, the user can view the associated tags. The user can choose the value of each of the tags based on filtering criteria. Clicking **Add Metric** adds the metric to the page which is refreshed every 5 seconds by default.

The user can pause the auto refresh feature by deactivating the **Auto Refresh** toggle. The user can also refresh the metrics manually by clicking **Refresh All**. The format of the metric value can be changed according to the user’s needs. They can delete a particular metric by clicking the minus symbol in the same row.



Information Category: Metrics

METRICS: hikaricp.connections.max

POOL: HikariPool-1

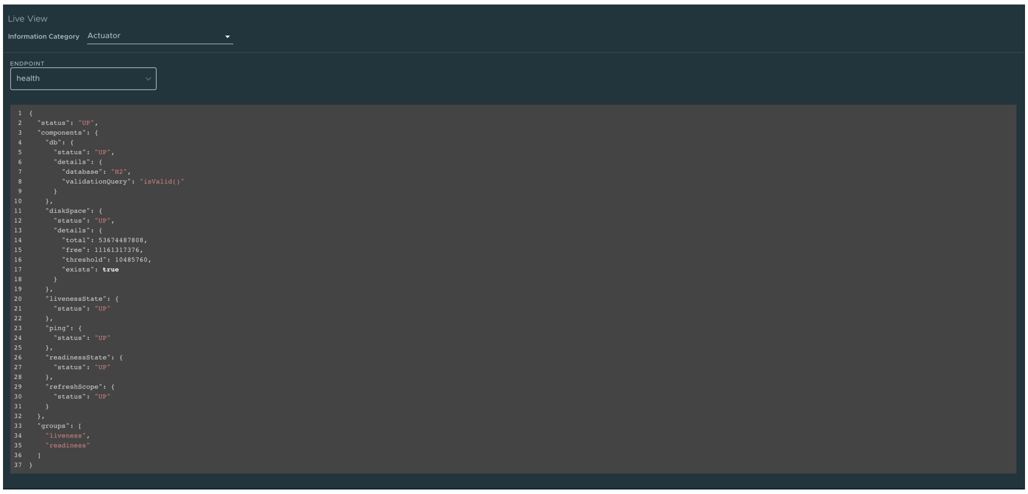
cache.evictions	COUNT		
(no tags)	0		

http.server.requests	COUNT	TOTAL_TIME	MAX
(no tags)	1495	2.702257623	0.001713162

hikaricp.connections.max	VALUE
pool:HikariPool-1	10

Actuator page

To navigate to the **Actuator** page, the user can select the **Actuator** option from the **Information Category** drop-down menu. The actuator page provides a tree view of the actuator data. The user can choose from a list of actuator endpoints and parse through the raw actuator data.



Information Category: Actuator

ENDPOINT: health

```
{
  "status": "UP",
  "components": {
    "db": {
      "status": "UP",
      "details": {
        "database": "H2",
        "validationQuery": "isValid()"
      }
    },
    "diskSpace": {
      "status": "UP",
      "details": {
        "total": 53674487808,
        "free": 11161317376,
        "threshold": 10485760,
        "exists": true
      }
    },
    "livenessState": {
      "status": "UP"
    },
    "ping": {
      "status": "UP"
    },
    "readinessState": {
      "status": "UP"
    },
    "refreshScope": {
      "status": "UP"
    }
  },
  "groups": [
    "liveness",
    "readiness"
  ]
}
```

Troubleshooting

You might run into cases where a workload running on your cluster does not show up in the Application Live View overview, the detail pages do not load any information while running, or similar issues. See Troubleshooting in the Application Live View documentation.

Application Live View for Spring Cloud Gateway Applications in Tanzu Application Platform GUI

This topic tells you about the Application Live View pages for Spring Cloud Gateway Applications in Tanzu Application Platform GUI (commonly called TAP GUI).

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 of the routes.

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 show up in the Application Live View overview, or the detail pages do not load any information while running, or other similar issues. For more information, see Troubleshooting in the Application Live View documentation.

Application Live View for Steeltoe Applications in Tanzu Application Platform GUI

This topic tells you about the Application Live View pages for Steeltoe Applications in Tanzu Application Platform GUI (commonly called TAP GUI).

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 of the components.

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 features:

- The UI has a search feature that allows you to search for a property or values.
- Each property has a search icon at the right corner that helps you quickly see all occurrences of a specific property key without manually typing in the search field. Clicking the search button trims down the page to that property name.
- The **Refresh Scope** on the top-right corner of the page probes the application to refresh all the environment properties.
- You can edit an existing property by clicking **Override** in the row and editing the value. After the value is saved, you can see the updated property in the Applied overrides section at the top of the page.
- **Reset** resets the environment property to the original state.
- You can edit or remove the overridden environment variables in the **Applied Overrides** section.
- The **Applied Overrides** section also allows you to add new environment properties to the application.

The `management.endpoint.env.post.enabled=true` has to be set in the application config properties of the application, and a corresponding editable Environment has to be present in the application.

Log Levels page

To access 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.
- 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 modifying the log level of root logger to OFF.

The UI on the Log Levels page includes the following features:

- Toggle **Changes Only** to display the changed log levels.
- The search feature allows you to search by logger name.
- **Reset** resets the log levels to the original state.
- **Reset All** on the top-right corner of the page resets all the loggers to the default state.

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 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 also has a feature to download thread dump for analysis.

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 heap dump for analysis.

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, which is refreshed every 5 seconds by default.

The UI on the Metrics page includes the features that allow 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 symbol in the same 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 show up in the Application Live View overview, or the detail pages do not load any information while running, or other similar issues. For more information, see Troubleshooting.

Application Accelerator in Tanzu Application Platform GUI

This topic tells you how to use Application Accelerator in Tanzu Application Platform GUI (commonly called TAP GUI).

Overview

Application Accelerator for VMware Tanzu helps you bootstrap developing and deploying your applications in a discoverable and repeatable way.

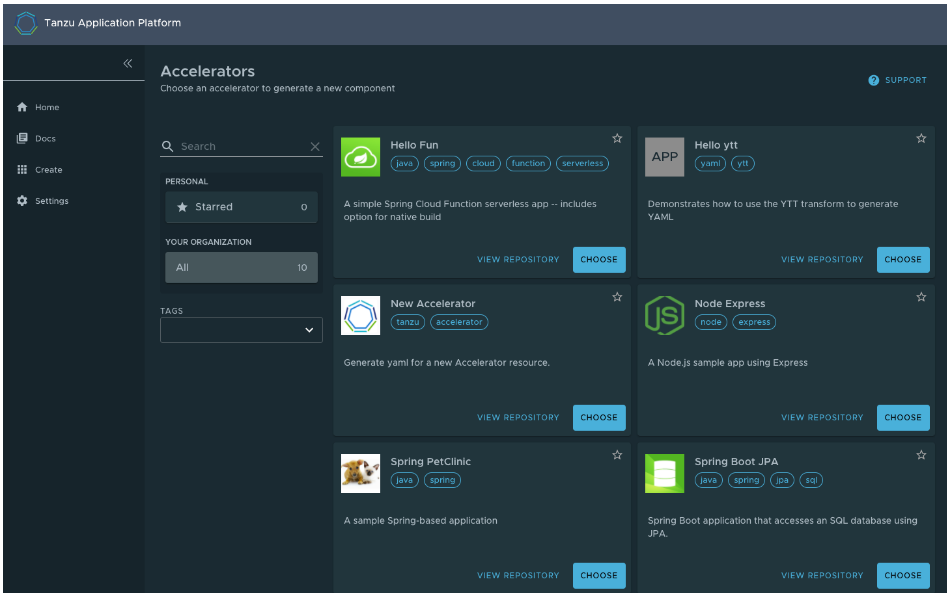
Enterprise architects author and publish accelerator projects that provide developers and operators with ready-made, enterprise-conforming code and configurations. You can then use Application Accelerator to create new projects based on those accelerator projects.

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.



Accelerators

Choose an accelerator to generate a new component

Hello Fun: A simple Spring Cloud Function serverless app -- includes option for native build

Hello ytt: Demonstrates how to use the YTT transform to generate YAML

New Accelerator: Generate yaml for a new Accelerator resource.

Node Express: A Node.js sample app using Express

Spring PetClinic: A sample Spring-based application

Spring Boot JPA: Spring Boot application that accesses an SQL database using JPA.

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.

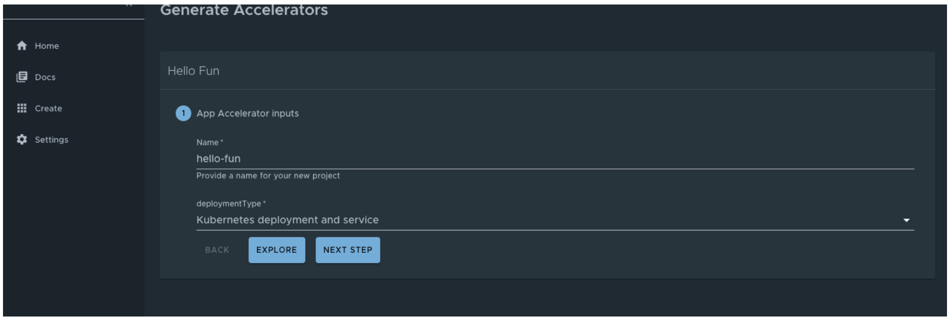
3. Search and filter based on text and tags associated with the accelerators to find the accelerator representing the project you want to create.

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.



Generate Accelerators

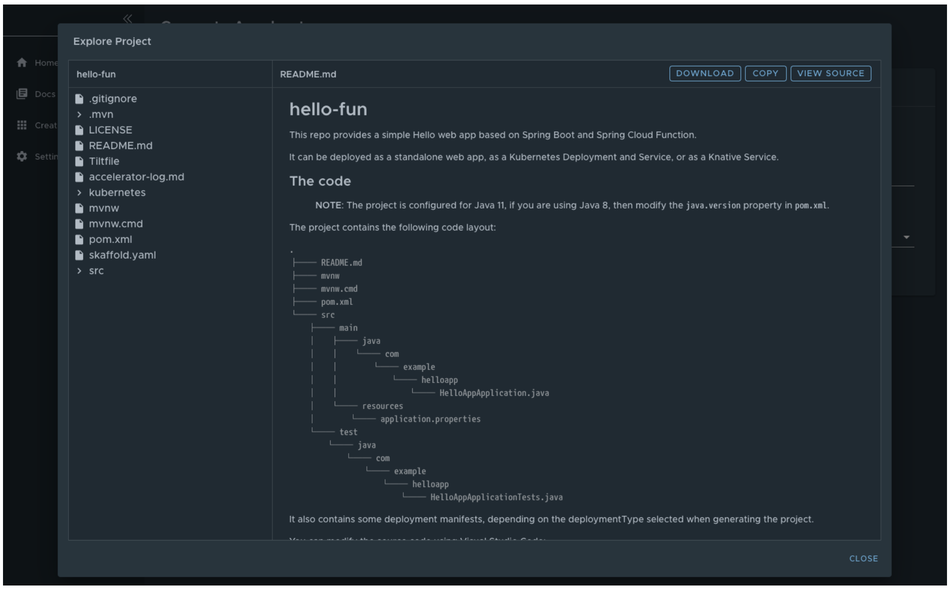
Hello Fun

1 App Accelerator inputs

Name *
hello-fun
Provide a name for your new project

deploymentType *
Kubernetes deployment and service

2. Click **EXPLORE** to open the **Explore Project** page and view the project before it is generated.



hello-fun

- .gitignore
- .mvn
- LICENSE
- README.md
- Tiltfile
- accelerator-log.md
- kubernetes
- mvnw
- mvnw.cmd
- pom.xml
- skaffold.yaml
- src

README.md

hello-fun

This repo provides a simple Hello web app based on Spring Boot and Spring Cloud Function.

It can be deployed as a standalone web app, as a Kubernetes Deployment and Service, or as a Knative Service.

The code

> **NOTE**: The project is configured for Java 11, if you are using Java 8, then modify the `java.version` property in `pom.xml`.

The project contains the following code layout:

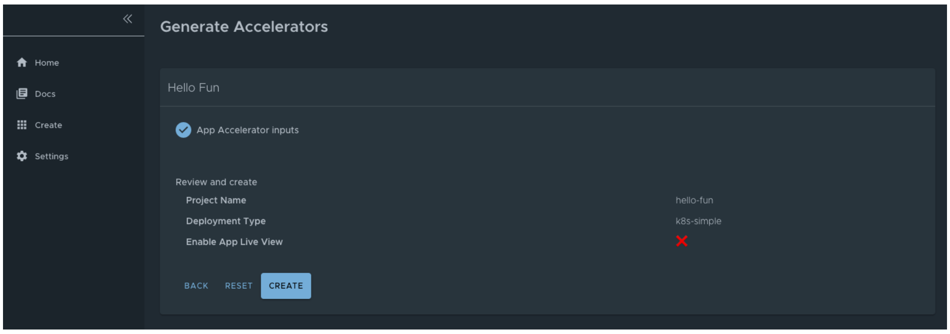
```
.
├── README.md
├── mvnw
├── mvnw.cmd
├── pom.xml
└── src
    ├── main
    │   ├── java
    │   │   └── com
    │   │       └── example
    │   │           └── helloapp
    │   │               └── HelloAppApplication.java
    │   └── resources
    │       └── application.properties
    └── test
        └── java
            └── com
                └── example
                    └── helloapp
                        └── HelloAppApplicationTests.java
```

It also contains some deployment manifests, depending on the deploymentType selected when generating the project.

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.



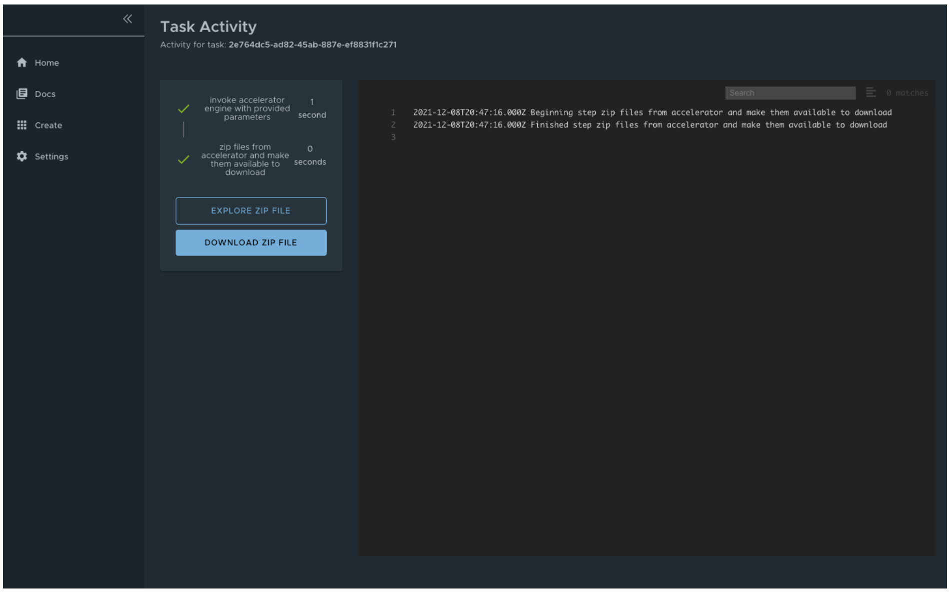
Review and create

Project Name	hello-fun
Deployment Type	k8s-simple
Enable App Live View	✗

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.



Activity for task: 2e764dc5-ad82-45ab-887e-ef8831f1c271

✓ invoke accelerator engine with provided parameters	1 second
✓ zip files from accelerator and make them available to download	0 seconds

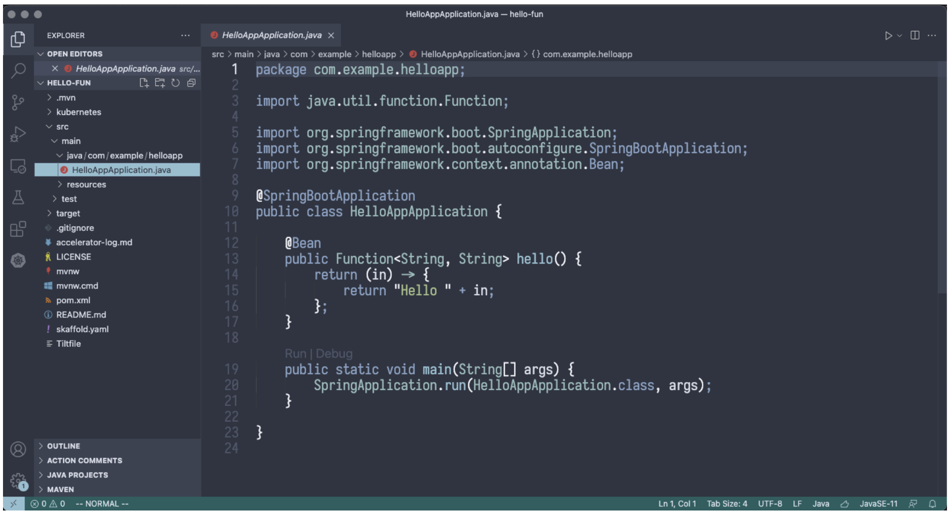
```
2021-12-08T20:47:16.000Z Beginning step zip files from accelerator and make them available to download
2021-12-08T20:47:16.000Z Finished step zip files from accelerator and make them available to download
```

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).



```
package com.example.helloapp;

import java.util.function.Function;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
public class HelloAppApplication {

    @Bean
    public Function<String, String> hello() {
        return (in) -> {
            return "Hello " + in;
        };
    }

    public static void main(String[] args) {
        SpringApplication.run(HelloAppApplication.class, args);
    }

}
```

Next steps

To learn more about Application Accelerator for VMware Tanzu, see the Application Accelerator documentation.

Application Accelerator in Tanzu Application Platform GUI

This topic tells you how to use Application Accelerator in Tanzu Application Platform GUI (commonly called TAP GUI).

Overview

Application Accelerator for VMware Tanzu helps you bootstrap developing and deploying your applications in a discoverable and repeatable way.

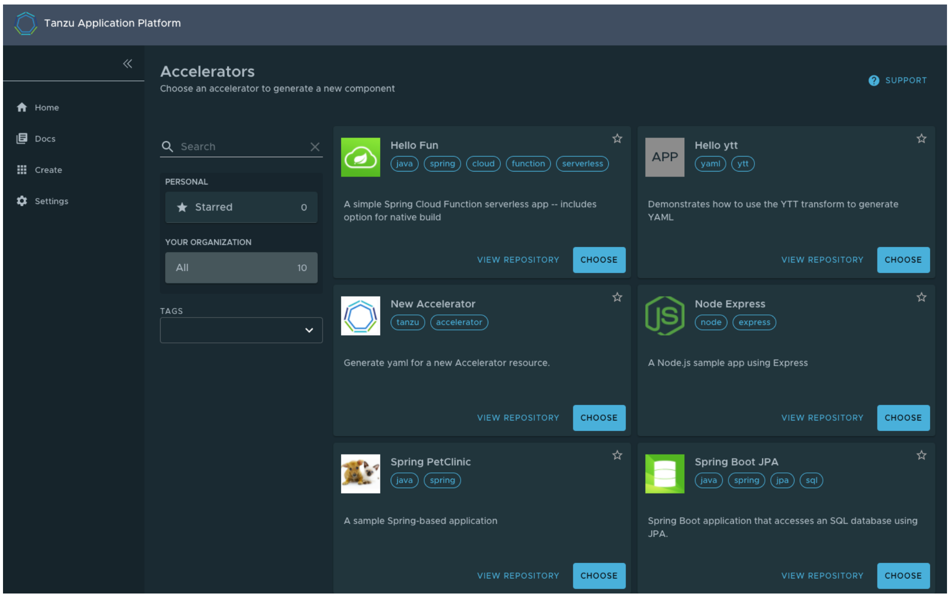
Enterprise architects author and publish accelerator projects that provide developers and operators with ready-made, enterprise-conforming code and configurations. You can then use Application Accelerator to create new projects based on those accelerator projects.

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.



A simple Spring Cloud Function serverless app -- includes option for native build

Demonstrates how to use the YTT transform to generate YAML

Generate yaml for a new Accelerator resource.

A Node.js sample app using Express

A sample Spring-based application

Spring Boot application that accesses an SQL database using JPA.

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.

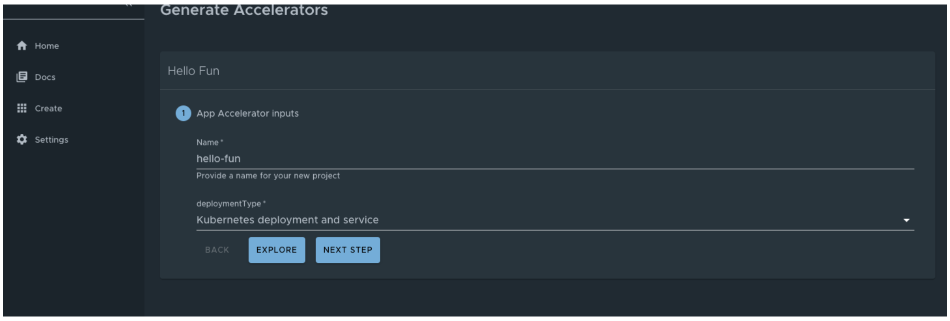
3. Search and filter based on text and tags associated with the accelerators to find the accelerator representing the project you want to create.

4. Click **CHOOSE** for the accelerator you want. This opens the **Generate Accelerators** page.

Configure project generation

To configure how projects are generated:

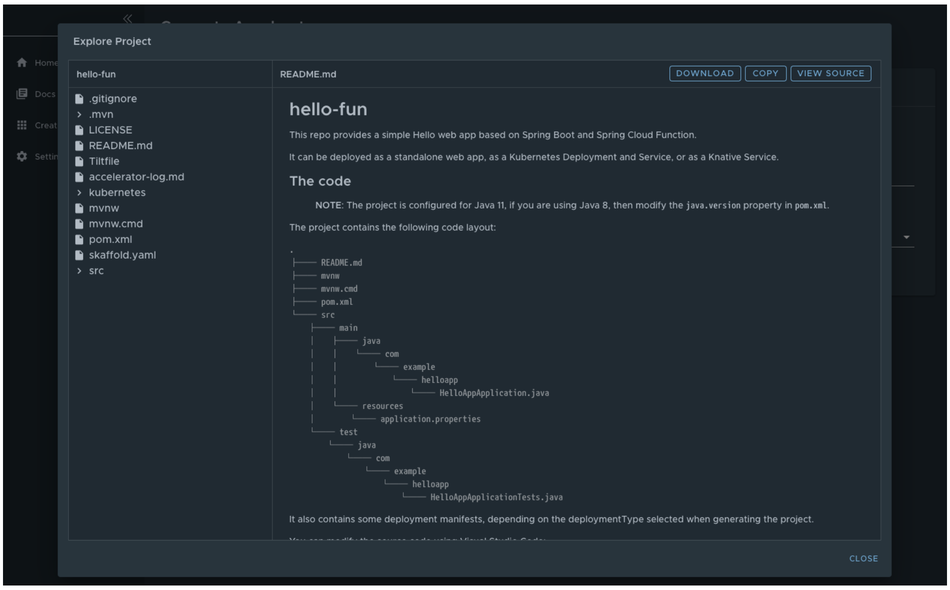
1. On the **Generate Accelerators** page, add any configuration values needed to generate the project. The application architect defined these values in `accelerator.yaml` in the accelerator definition. Filling some text boxes can cause other text boxes to appear. Fill them all in.



Name *
hello-fun
Provide a name for your new project

deploymentType *
Kubernetes deployment and service

2. Click **EXPLORE** to open the **Explore Project** page and view the project before it is generated.



hello-fun

- .gitignore
- › .mvn
- LICENSE
- README.md
- Tiltfile
- accelerator-log.md
- › kubernetes
- mvnw
- mvnw.cmd
- pom.xml
- skaffold.yaml
- › src

README.md

hello-fun

This repo provides a simple Hello web app based on Spring Boot and Spring Cloud Function.

It can be deployed as a standalone web app, as a Kubernetes Deployment and Service, or as a Knative Service.

The code

> **NOTE**: The project is configured for Java 11, if you are using Java 8, then modify the `java.version` property in `pom.xml`.

The project contains the following code layout:

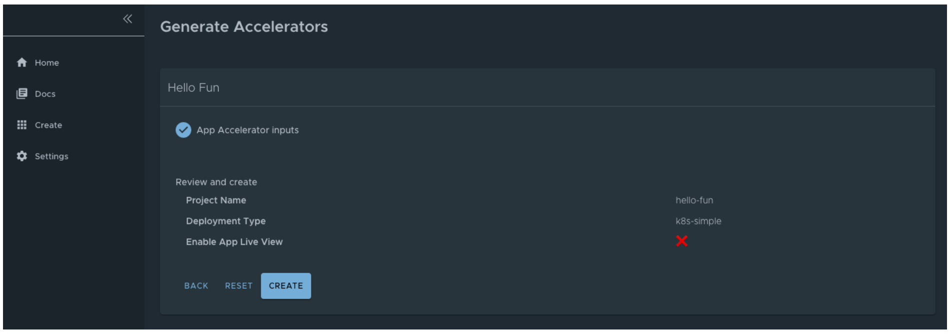
```
.
├── README.md
├── mvnw
├── mvnw.cmd
├── pom.xml
└── src
    ├── main
    │   ├── java
    │   │   └── com
    │   │       └── example
    │   │           └── helloapp
    │   │               └── HelloAppApplication.java
    │   └── resources
    │       └── application.properties
    └── test
        └── java
            └── com
                └── example
                    └── helloapp
                        └── HelloAppApplicationTests.java
```

It also contains some deployment manifests, depending on the deploymentType selected when generating the project.

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.



Generate Accelerators

Hello Fun

✓ App Accelerator inputs

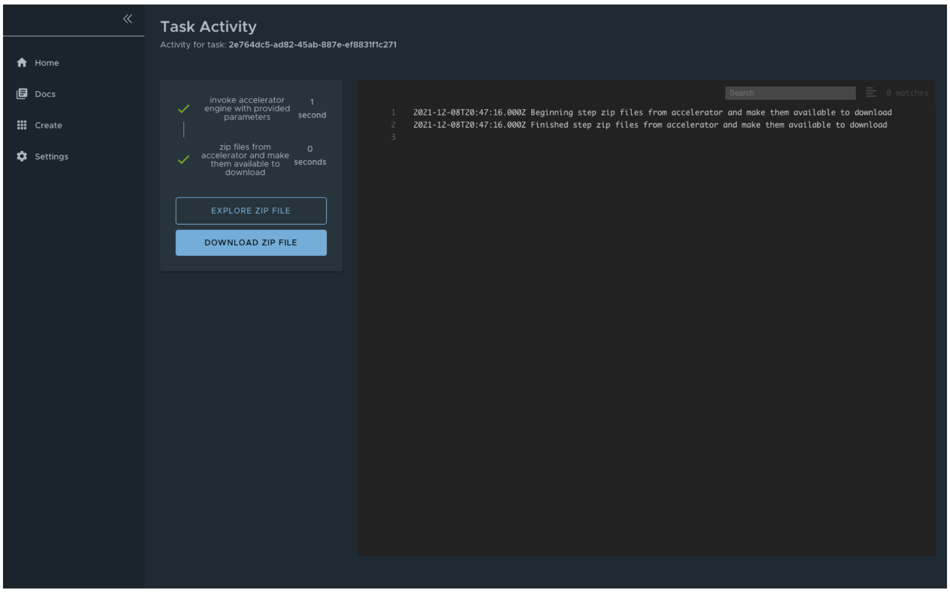
Review and create

Project Name	hello-fun
Deployment Type	k8s-simple
Enable App Live View	✗

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.



Task Activity

Activity for task: **2e764dc5-ad82-45ab-887e-ef8831f1c271**

✓ invoke accelerator engine with provided parameters — 1 second

✓ zip files from accelerator and make them available to download — 0 seconds

```
2021-12-08T20:47:16.000Z Beginning step zip files from accelerator and make them available to download
2021-12-08T20:47:16.000Z Finished step zip files from accelerator and make them available to download
```

2. After the project is generated, click **EXPLORE ZIP FILE** to open the **Explore Project** page to verify configuration.

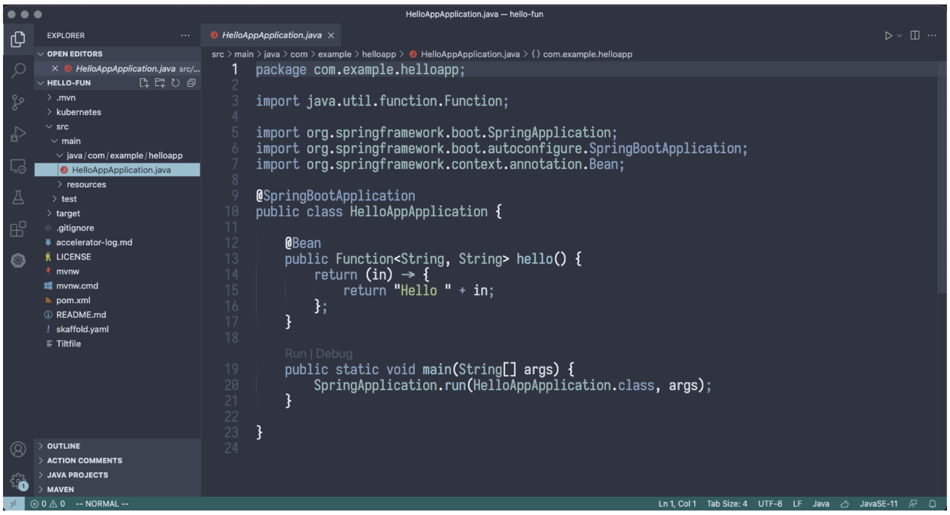
3. Click **DOWNLOAD ZIP FILE** to download the project in a ZIP file.

Develop your code

To develop your code:

1. Expand the ZIP file.

2. Open the project in your integrated development environment (IDE).



```
package com.example.helloapp;

import java.util.function.Function;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
public class HelloAppApplication {

    @Bean
    public Function<String, String> hello() {
        return (in) -> {
            return "Hello " + in;
        };
    }

    Run | Debug
    public static void main(String[] args) {
        SpringApplication.run(HelloAppApplication.class, args);
    }

}
```

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 cert-manager, Contour, and Flux CD Source Controller.

- Install Source Controller on the cluster. See Install Source Controller.

Configure properties and resource usage

When you install the Application Accelerator, you can configure the following optional properties:

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, defaults to `shared.ingress_domain`
tls.secret_n_ame	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	Should the system record each engine invocation when generating files for an accelerator?

VMware recommends that you do not override the defaults 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

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 ta
p-install
- Retrieving package versions for accelerator.apps.tanzu.vmware.com...
  NAME                               VERSION  RELEASED-AT
  accelerator.apps.tanzu.vmware.com  1.2.1    2022-06-22 13:00:00 -0400 EDT
```

2. (Optional) To make changes to the default installation settings, run:

   ```
   tanzu package available get accelerator.apps.tanzu.vmware.com/VERSION-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 accelerator.apps.tanzu.vmware.com/1.2.1 --values-sc
   hema --namespace tap-install
   ```

   For more information about values schema options, see the properties listed earlier.

3. Create an `app-accelerator-values.yaml` using the following example code:

   ```
   server:
     service_type: "LoadBalancer"
     watched_namespace: "accelerator-system"
   samples:
     include: true
   ```

   Edit the values if needed or leave the default values.

   For clusters that do not support the `LoadBalancer` service type, override the default value for `server.service_type`.

4. Install the package by running:

   ```
   tanzu package install app-accelerator -p accelerator.apps.tanzu.vmware.com -v V
   ERSION-NUMBER -n tap-install -f app-accelerator-values.yaml
   ```

   Where `VERSION-NUMBER` is the version included in the Tanzu Application Platform installation.

   For example:

   ```
   $ tanzu package install app-accelerator -p accelerator.apps.tanzu.vmware.com -v
   1.2.1 -n tap-install -f 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'
   ```

5. 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:         1.2.1
   STATUS:                  Reconcile succeeded
   ```

```
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`.

6. To see the IP address for the Application Accelerator API when the `server.service_type` is set to `LoadBalancer`, run the following command:

   ```
   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` command.

API documentation plug-in in Tanzu Application Platform GUI

This topic gives you an overview of the API documentation plug-in of Tanzu Application Platform GUI (commonly called TAP GUI). 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 Application Platform GUI 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 to systems and show up on the system diagram. To show such 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.

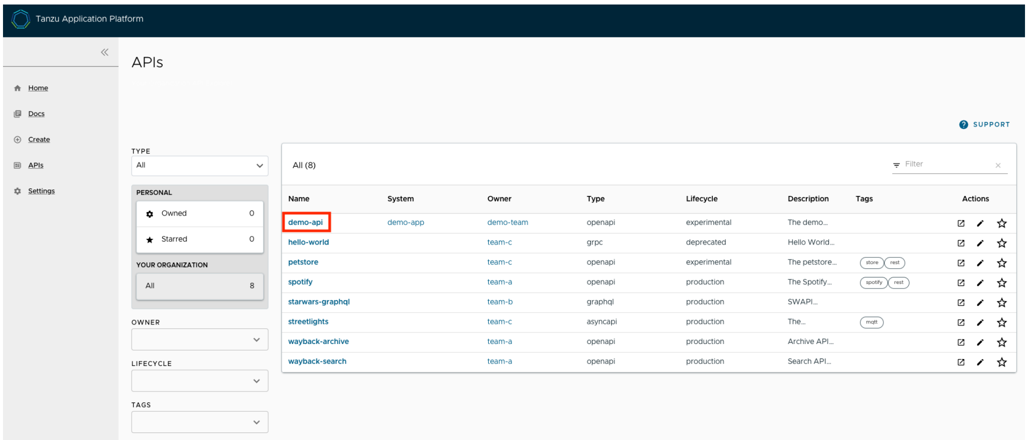
```
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: example-component
  description: Example Component
spec:
  type: service
  lifecycle: experimental
  owner: team-a
  system: example-system
  providesApis: # list of APIs provided by the Component
    - example-api-1
  consumesApis: # 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 Application Platform GUI.

The first way to use the API documentation plug-in is API-first. Click **APIs** in the left-hand navigation sidebar of Tanzu Application Platform GUI. This opens the **API catalog page**.



Tanzu Application Platform

Home
Docs
Create
APIs
Settings

APIs

SUPPORT

TYPE
All

PERSONAL
Owned 0
Starred 0

YOUR ORGANIZATION
All 8

OWNER

LIFECYCLE

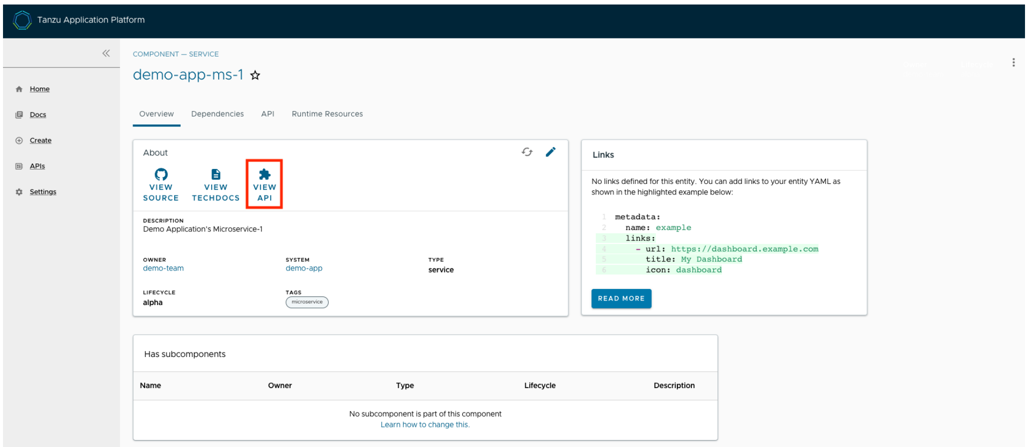
TAGS

All (8)

Name	System	Owner	Type	Lifecycle	Description	Tags	Actions
demo-api	demo-app	demo-team	openapi	experimental	The demo...		
hello-world		team-c	grpc	deprecated	Hello World...		
petstore		team-c	openapi	experimental	The petstore...	store rest	
spotify		team-a	openapi	production	The Spotify...	spotify rest	
starwars-graphql		team-b	graphql	production	SWAPI...		
streetlights		team-c	asyncapi	production	The...	mqtt	
wayback-archive		team-a	openapi	production	Archive API...		
wayback-search		team-a	openapi	production	Search API...		

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 by using components and systems of the software catalog, listed on the home page of Tanzu Application Platform GUI. If there is an API entity associated with the selected component or system, the **VIEW API** icon is active.



Tanzu Application Platform

Home
Docs
Create
APIs
Settings

COMPONENT — SERVICE

demo-app-ms-1

Overview Dependencies API Runtime Resources

About

VIEW SOURCE VIEW TECHDOCS VIEW API

DESCRIPTION
Demo Application's Microservice-1

OWNER
demo-team

SYSTEM
demo-app

TYPE
service

LIFECYCLE
alpha

TAGS
microservice

Links

No links defined for this entity. You can add links to your entity YAML as shown in the highlighted example below:

```
metadata:
  name: example
  links:
    - url: https://dashboard.example.com
      title: My Dashboard
      icon: dashboard
```

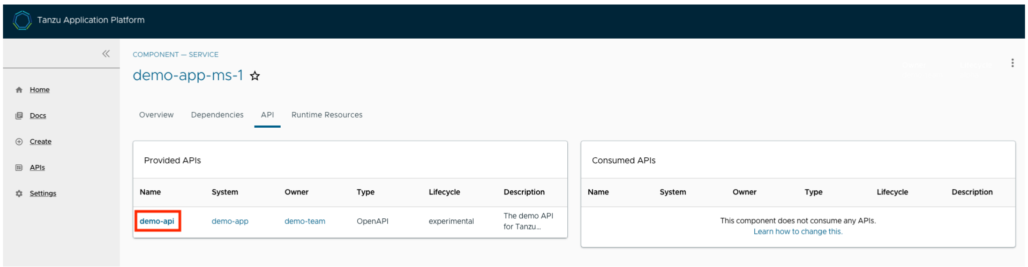
READ MORE

Has subcomponents

Name	Owner	Type	Lifecycle	Description

No subcomponent is part of this component
Learn how to change this.

The **VIEW API** tab displays which APIs are being consumed by a component and which APIs are being provided by the component.



Tanzu Application Platform

Home
Docs
Create
APIs
Settings

COMPONENT — SERVICE

demo-app-ms-1

Overview Dependencies API Runtime Resources

Provided APIs

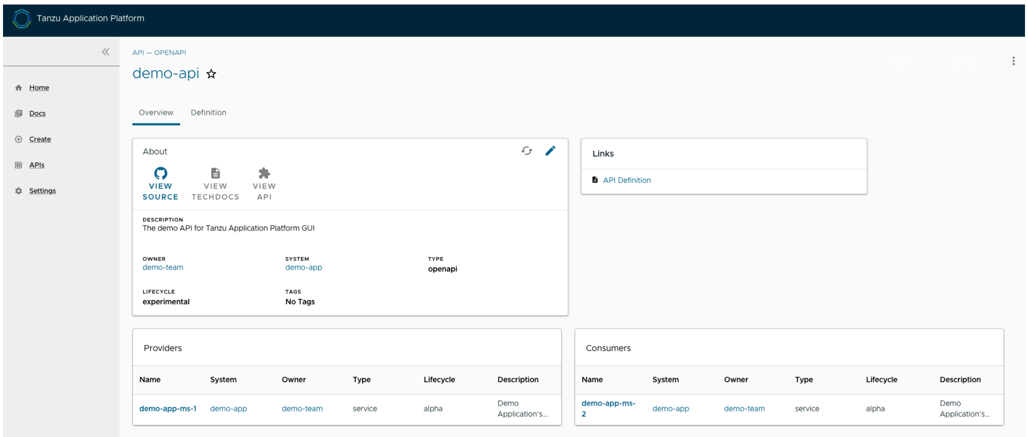
Name	System	Owner	Type	Lifecycle	Description
demo-api	demo-app	demo-team	OpenAPI	experimental	The demo API for Tanzu...

Consumed APIs

Name	System	Owner	Type	Lifecycle	Description

This component does not consume any APIs.
Learn how to change this.

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.



DESCRIPTION
The demo API for Tanzu Application Platform GUI

OWNER
demo-team
SYSTEM
demo-app
TYPE
openapi
LIFECYCLE
experimental
TAGS
No Tags

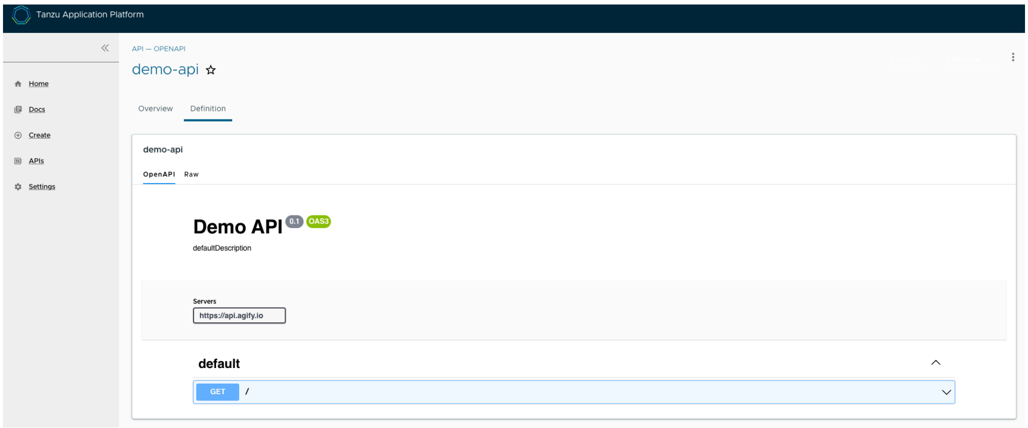
Providers

Name	System	Owner	Type	Lifecycle	Description
demo-app-ms-1	demo-app	demo-team	service	alpha	Demo Application's...

Consumers

Name	System	Owner	Type	Lifecycle	Description
demo-app-ms-2	demo-app	demo-team	service	alpha	Demo Application's...

By choosing the **Definition** tab on the top of the API page, you can see the definition of that API in human-readable and machine-readable format.



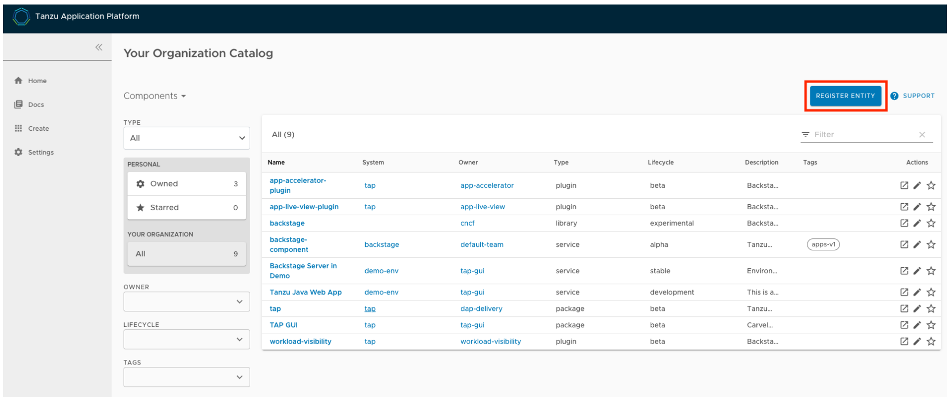
The API documentation plug-in supports the following API formats:

- OpenAPI 2 & 3
- AsyncAPI
- GraphQL
- Plain (to support any other format)

Create a new API entry

To create a new API entity, you must follow the same steps as if you were registering any other software catalog entity:

1. Click the **Home** icon located on the left-side navigation bar to access the home page of Tanzu Application Platform GUI.
2. Click **REGISTER ENTITY**.

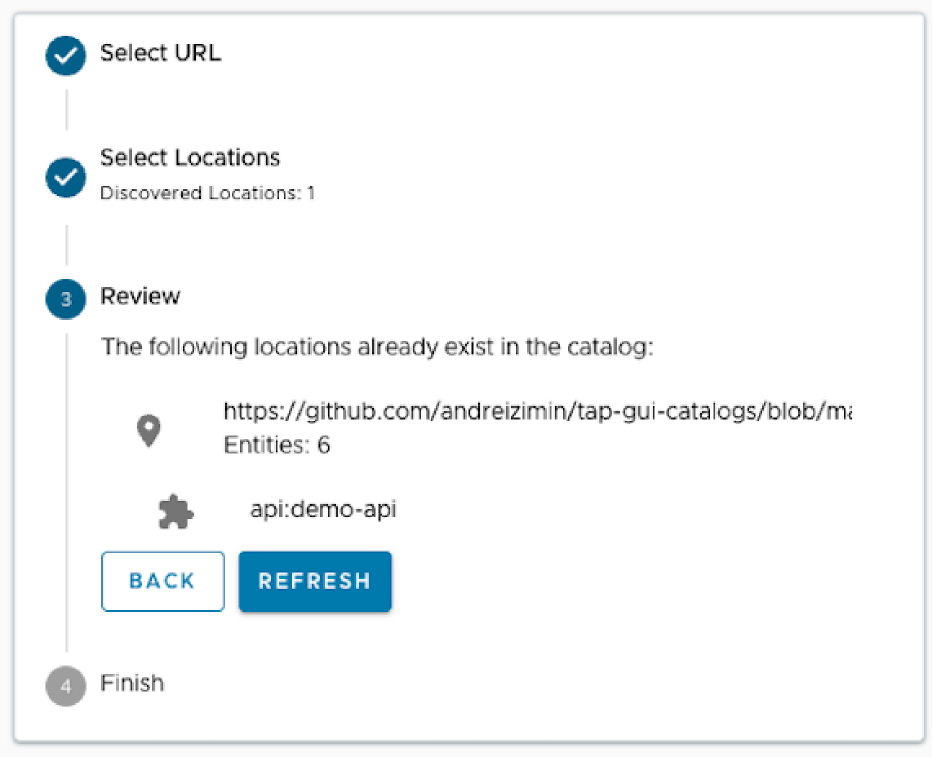


Name	System	Owner	Type	Lifecycle	Description	Tags	Actions
app-accelerator-plugin	tap	app-accelerator	plugin	beta	Backsta...		
app-live-view-plugin	tap	app-live-view	plugin	beta	Backsta...		
backstage		cncf	library	experimental	Backsta...		
backstage-component	backstage	default-team	service	alpha	Tanzu...	apps-v1	
Backstage Server in Demo	demo-env	tap-gui	service	stable	Environ...		
Tanzu Java Web App	demo-env	tap-gui	service	development	This is a...		
tap	tap	dap-delivery	package	beta	Tanzu...		
TAP GUI	tap	tap-gui	package	beta	Carvel...		
workload-visibility	tap	workload-visibility	plugin	beta	Backsta...		

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 Application Platform GUI
  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.



Select URL

Select Locations

Discovered Locations: 1

3 Review

The following locations already exist in the catalog:

https://github.com/andreizimin/tap-gui-catalogs/blob/ma

Entities: 6

api:demo-api

4 Finish

5. Click **IMPORT**.
6. Click **APIs** on the left-hand side navigation panel to view entries on the **API** page.

API documentation plug-in in Tanzu Application Platform GUI

This topic gives you an overview of the API documentation plug-in of Tanzu Application Platform GUI (commonly called TAP GUI). 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 Application Platform GUI 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 to systems and show up on the system diagram. To show such 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
```

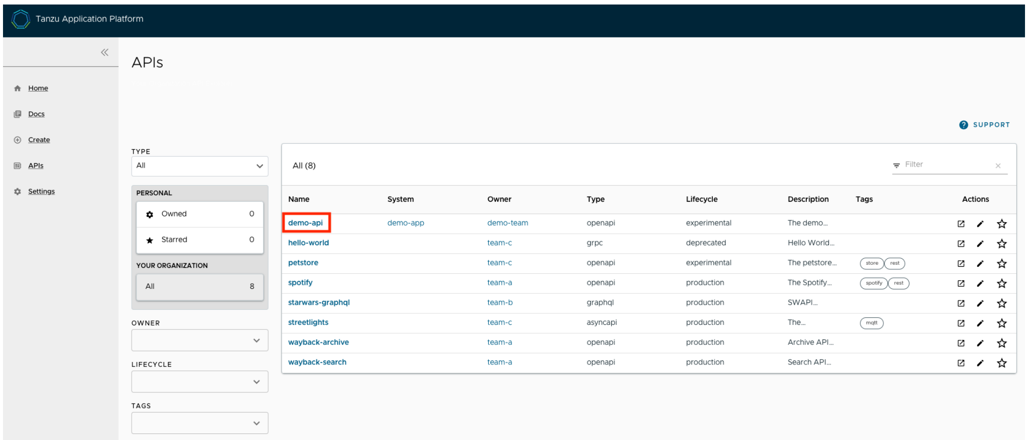
```
consumesApis: # list of APIs consumed by the Component
  - example-api-2
```

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 Application Platform GUI.

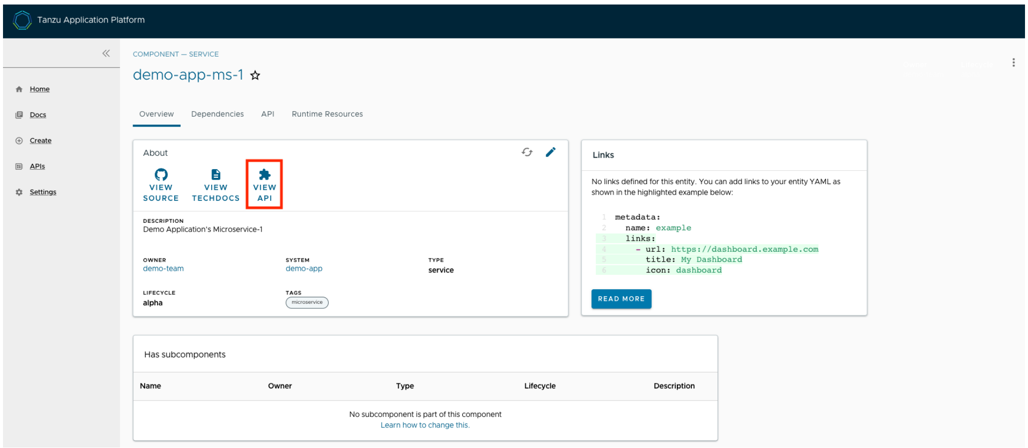
The first way to use the API documentation plug-in is API-first. Click **APIs** in the left-hand navigation sidebar of Tanzu Application Platform GUI. This opens the **API catalog page**.



Name	System	Owner	Type	Lifecycle	Description	Tags	Actions
demo-api	demo-app	demo-team	openapi	experimental	The demo...		
hello-world		team-c	grpc	deprecated	Hello World...		
petstore		team-c	openapi	experimental	The petstore...	store rest	
spotify		team-a	openapi	production	The Spotify...	spotify rest	
starwars-graphql		team-b	graphql	production	SWAPI...		
streetlights		team-c	asyncapi	production	The...	mqtt	
wayback-archive		team-a	openapi	production	Archive API...		
wayback-search		team-a	openapi	production	Search API...		

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 by using components and systems of the software catalog, listed on the home page of Tanzu Application Platform GUI. If there is an API entity associated with the selected component or system, the **VIEW API** icon is active.



DESCRIPTION
Demo Application's Microservice-1

OWNER
demo-team

SYSTEM
demo-app

TYPE
service

LIFECYCLE
alpha

TAGS
microservice

No links defined for this entity. You can add links to your entity YAML as shown in the highlighted example below:

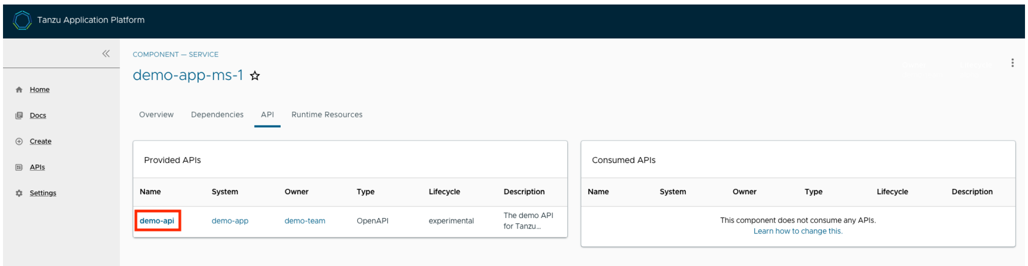
```
metadata:
  name: example
  links:
    - url: https://dashboard.example.com
      title: My Dashboard
      icon: dashboard
```

Has subcomponents

Name	Owner	Type	Lifecycle	Description

No subcomponent is part of this component
Learn how to change this.

The **VIEW API** tab displays which APIs are being consumed by a component and which APIs are being provided by the component.



Provided APIs

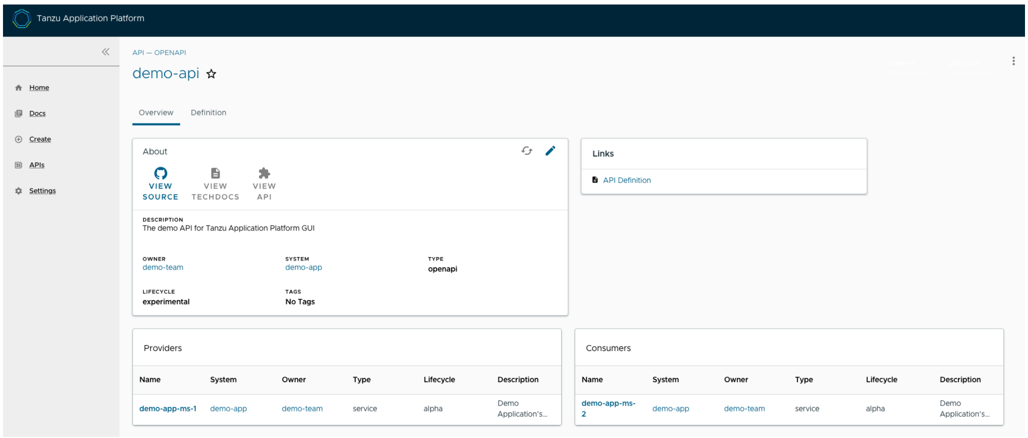
Name	System	Owner	Type	Lifecycle	Description
demo-api	demo-app	demo-team	OpenAPI	experimental	The demo API for Tanzu...

Consumed APIs

Name	System	Owner	Type	Lifecycle	Description

This component does not consume any APIs.
Learn how to change this.

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.



DESCRIPTION
The demo API for Tanzu Application Platform GUI

OWNER demo-team
SYSTEM demo-app
TYPE openapi
LIFECYCLE experimental
TAGS No Tags

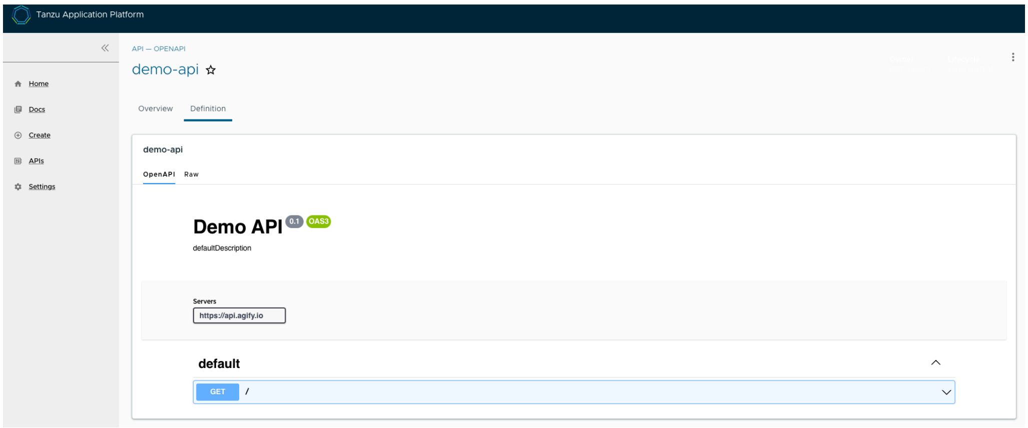
Providers

Name	System	Owner	Type	Lifecycle	Description
demo-app-ms-1	demo-app	demo-team	service	alpha	Demo Application's...

Consumers

Name	System	Owner	Type	Lifecycle	Description
demo-app-ms-2	demo-app	demo-team	service	alpha	Demo Application's...

By choosing the **Definition** tab on the top of the API page, you can see the definition of that API in human-readable and machine-readable format.



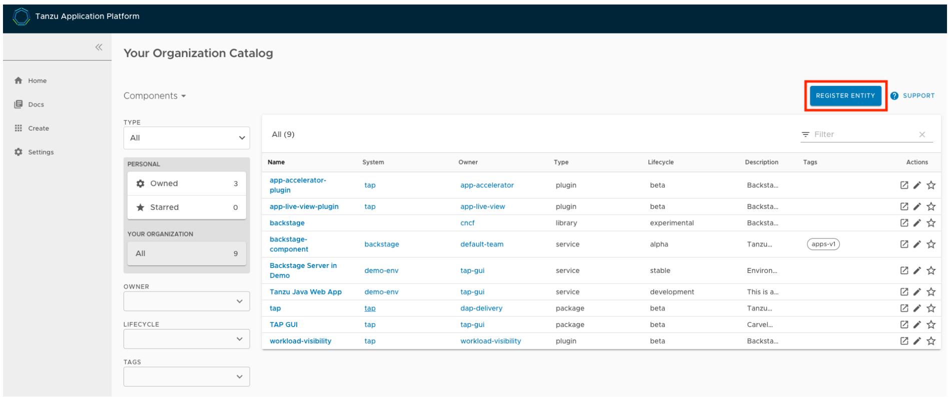
The API documentation plug-in supports the following API formats:

- OpenAPI 2 & 3
- AsyncAPI
- GraphQL
- Plain (to support any other format)

Create a new API entry

To create a new API entity, you must follow the same steps as if you were registering any other software catalog entity:

1. Click the **Home** icon located on the left-side navigation bar to access the home page of Tanzu Application Platform GUI.
2. Click **REGISTER ENTITY**.

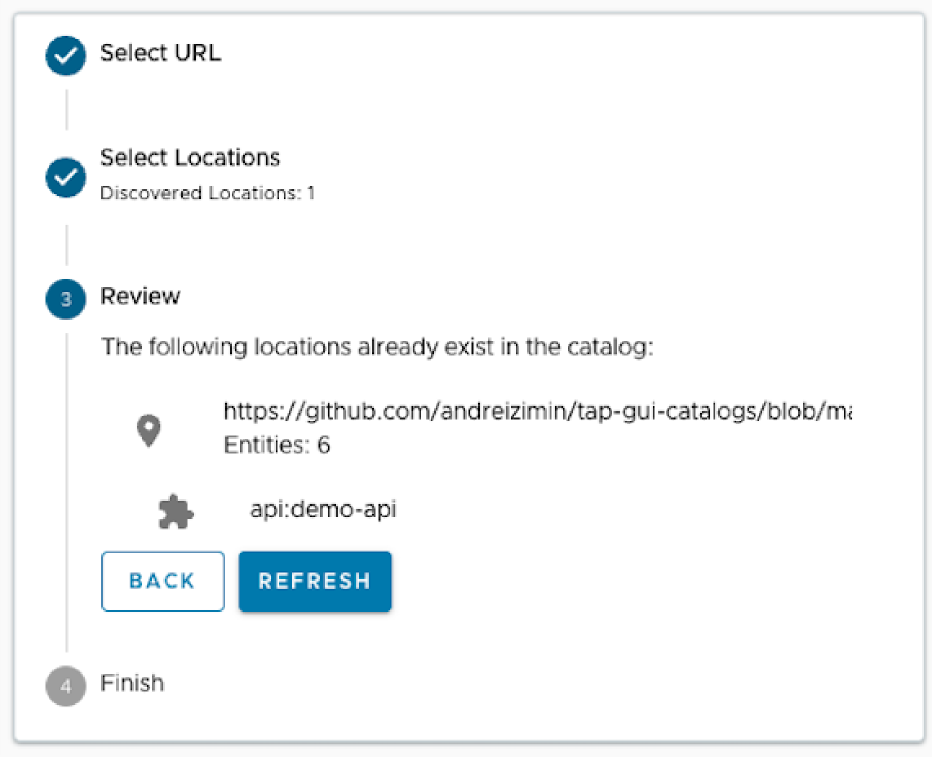


Name	System	Owner	Type	Lifecycle	Description	Tags	Actions
app-accelerator-plugin	tap	app-accelerator	plugin	beta	Backsta...		
app-live-view-plugin	tap	app-live-view	plugin	beta	Backsta...		
backstage		cncf	library	experimental	Backsta...		
backstage-component	backstage	default-team	service	alpha	Tanzu...	apps-v1	
Backstage Server in Demo	demo-env	tap-gui	service	stable	Environ...		
Tanzu Java Web App	demo-env	tap-gui	service	development	This is a...		
tap	tap	dap-delivery	package	beta	Tanzu...		
TAP GUI	tap	tap-gui	package	beta	Carvel...		
workload-visibility	tap	workload-visibility	plugin	beta	Backsta...		

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 Application Platform GUI
  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.



Discovered Locations: 1

3 Review

The following locations already exist in the catalog:

https://github.com/andreizimin/tap-gui-catalogs/blob/ma
Entities: 6

api:demo-api

5. Click **IMPORT**.
6. Click **APIs** on the left-hand side navigation panel to view entries on the **API** page.

Get started with the API documentation plug-in

This topic tells you how to get started with the API documentation plug-in in Tanzu Application Platform GUI (commonly called TAP GUI).

Add your API entry to the Tanzu Application Platform GUI software catalog

In this section, you will:

- Learn about API entities of the Software Catalog
- Add a demo API entity and its related Catalog objects to Tanzu Application Platform GUI
- Update your demo API entry

About API entities

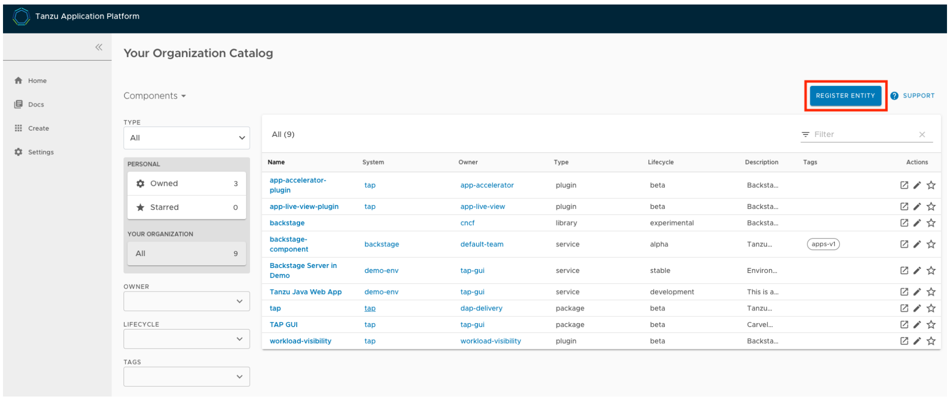
The list of API entities is visible on the left-hand side navigation panel of Tanzu Application Platform GUI. It is also visible on the overview page of specific components on the home page. APIs are a definition of the interface between components.

Their definition is provided in machine-readable (“raw”) and human-readable formats. For more information, see API plugin documentation.

Add a demo API entity to Tanzu Application Platform GUI software catalog

To add a demo API entity and its related Catalog objects, follow the same steps as registering any other software catalog entity:

1. Navigate to the home page of Tanzu Application Platform GUI. Click **Home** on the left-side navigation bar. Click **REGISTER ENTITY**.



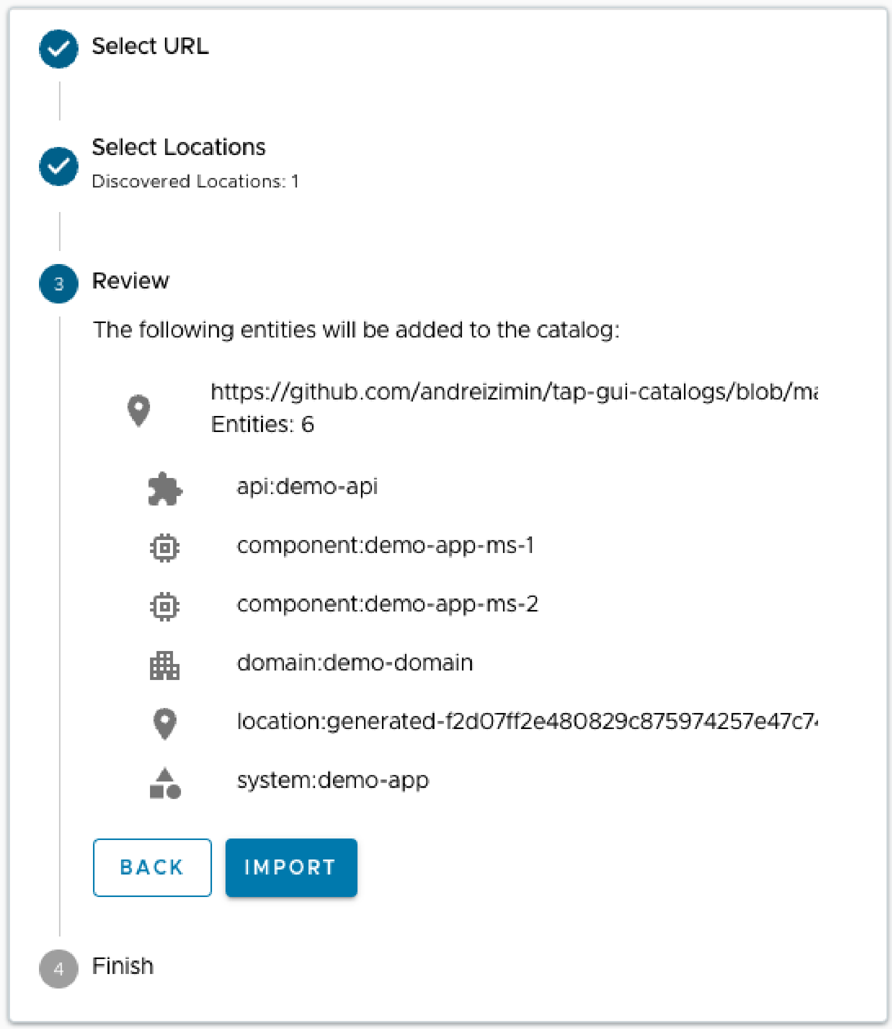
Name	System	Owner	Type	Lifecycle	Description	Tags	Actions
app-accelerator-plugin	tap	app-accelerator	plugin	beta	Backsta...		
app-live-view-plugin	tap	app-live-view	plugin	beta	Backsta...		
backstage		cncf	library	experimental	Backsta...		
backstage-component	backstage	default-team	service	alpha	Tanzu...	apps-v1	
Backstage Server in Demo	demo-env	tap-gui	service	stable	Environ...		
Tanzu Java Web App	demo-env	tap-gui	service	development	This is a...		
tap	tap	dap-delivery	package	beta	Tanzu...		
TAP GUI	tap	tap-gui	package	beta	Carvel...		
workload-visibility	tap	workload-visibility	plugin	beta	Backsta...		

2. **Register an existing component** prompts you to type a repository URL. Type the link to the `catalog-info.yaml` file of your choice or use the following sample definition. Save this code block as `catalog-info.yaml`, upload it to the Git repository of your choice, and copy the link to `catalog-info.yaml`.

   This demo setup includes a domain called `demo-domain` with a single system called `demo-system`. This systems consists of two microservices - `demo-app-ms-1` and `demo-app-ms-1` - and one API called `demo-api` that `demo-app-ms-1` provides and `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 Application Platform GUI
  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: {}
```

3. Paste the link to the `catalog-info.yaml` and click **ANALYZE**. Review the catalog entities and click **IMPORT**.



Select URL

Select Locations
Discovered Locations: 1

3 Review

The following entities will be added to the catalog:

https://github.com/andreizimin/tap-gui-catalogs/blob/ma
Entities: 6

api:demo-api

component:demo-app-ms-1

component:demo-app-ms-2

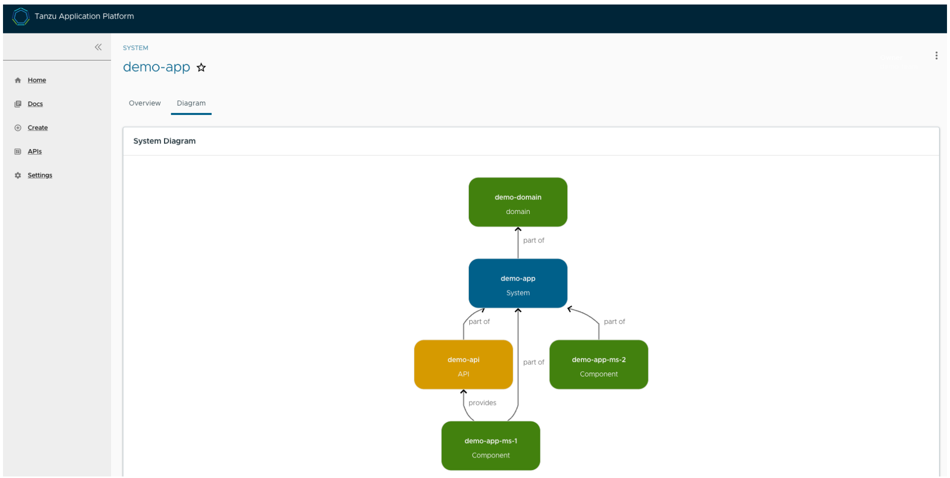
domain:demo-domain

location:generated-f2d07ff2e480829c875974257e47c7

system:demo-app

4 Finish

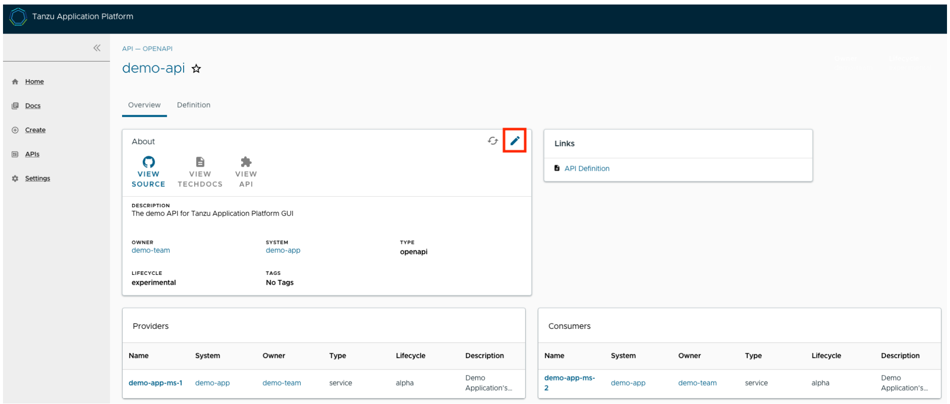
4. Navigate to the **API** page by clicking **APIs** on the left-hand side navigation panel. The catalog changes and entries are visible for further inspection. If you select the system **demo-app**, the diagram appears as follows:



Update your demo API entry

To update your demo API entry:

1. To update your demo API entity, select **demo-api** from the list of available APIs in your software catalog and click the **Edit** icon on the **Overview** page.



DESCRIPTION
The demo API for Tanzu Application Platform GUI

OWNER
demo-team

SYSTEM
demo-app

TYPE
openapi

LIFECYCLE
experimental

TAGS
No Tags

Providers

Name	System	Owner	Type	Lifecycle	Description
demo-app-ms-1	demo-app	demo-team	service	alpha	Demo Application's...

Consumers

Name	System	Owner	Type	Lifecycle	Description
demo-app-ms-2	demo-app	demo-team	service	alpha	Demo Application's...

It opens the source `catalog-info.yaml` file that you can edit. For example, change the `spec.paths.parameters.example` from `type_any_name` to `Tanzu` and save your changes.

2. After you made the edits, Tanzu Application Platform GUI re-renders the API entry with the next refresh cycle.

Supply Chain Choreographer in Tanzu Application Platform GUI

This topic tells you about Supply Chain Choreographer in Tanzu Application Platform GUI (commonly called TAP GUI).

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 Application Platform GUI you must have:

- One of the following installed on your cluster:
  - Tanzu Application Platform Full profile
  - Tanzu Application Platform View profile
  - Tanzu Application Platform GUI 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 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
        headers:
          Authorization: "Bearer ACCESS-TOKEN"
          X-Custom-Source: project-star
```

Where `ACCESS-TOKEN` is the token you obtained after creating a read-only service account.

Important

The `Authorization` value must start with the word `Bearer`.

Enable View Approvals

To enable the supply chain box-and-line diagram to show **View Approvals**, set up for GitOps and pull requests. For more information, see GitOps vs. RegistryOps.

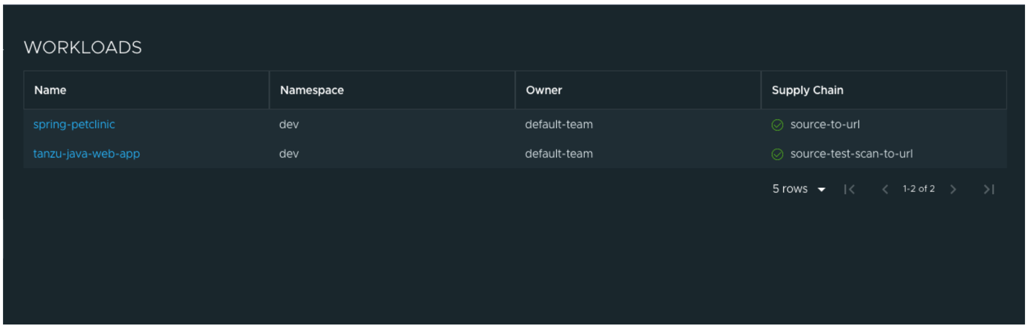
Supply Chain Visibility

Before using the 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.

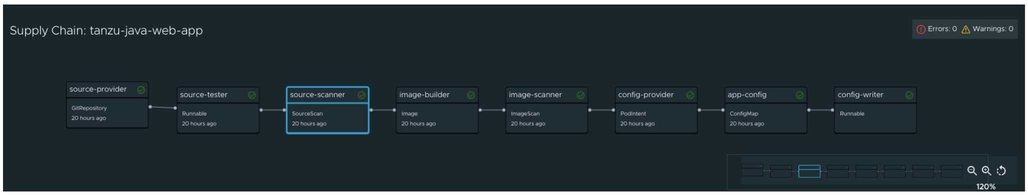
The example workload described in this topic is named `tanzu-java-web-app`.



WORKLOADS

Name	Namespace	Owner	Supply Chain
spring-petclinic	dev	default-team	✓ source-to-url
tanzu-java-web-app	dev	default-team	✓ source-test-scan-to-url

Click **tanzu-java-web-app** in the **WORKLOADS** table to navigate to the visualization of the supply chain.

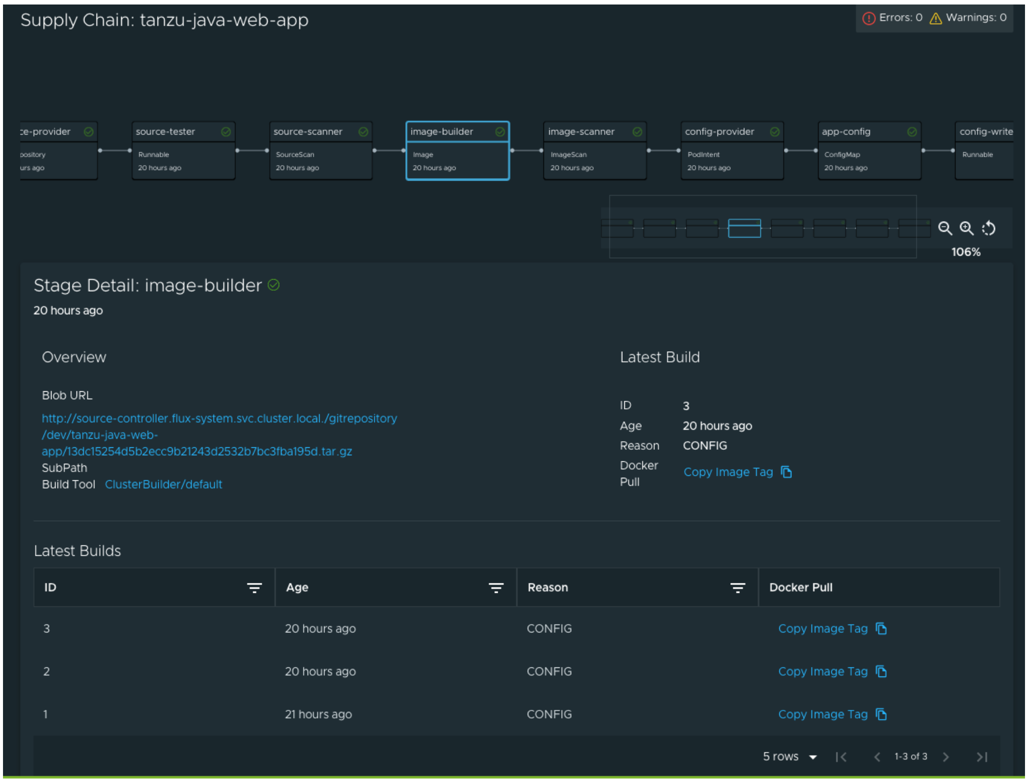


Supply Chain: tanzu-java-web-app

There are two sections within this view:

- The box-and-line diagram at the top shows all the configured 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

This is a sample result of the Build stage for the `tanzu-java-web-app` from using Tanzu Build Service:



Supply Chain: tanzu-java-web-app

Errors: 0 Warnings: 0

Stage Detail: image-builder

20 hours ago

Overview

Blob URL

http://source-controller.flux-system.svc.cluster.local./gitrepository/dev/tanzu-java-web-app/13dc15254d5b2ecc9b21243d2532b7bc3fba195d.tar.gz

SubPath

Build Tool ClusterBuilder/default

Latest Build

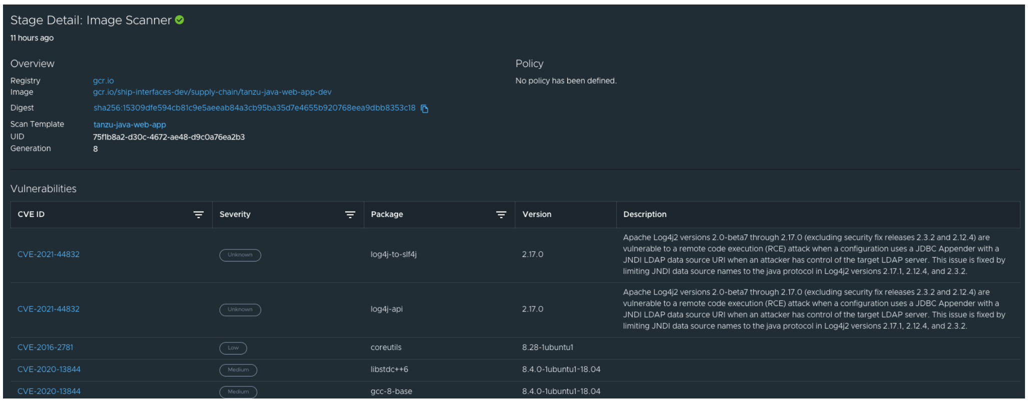
ID	3
Age	20 hours ago
Reason	CONFIG
Docker Pull	Copy Image Tag

Latest Builds

ID	Age	Reason	Docker Pull
3	20 hours ago	CONFIG	Copy Image Tag
2	20 hours ago	CONFIG	Copy Image Tag
1	21 hours ago	CONFIG	Copy Image Tag

5 rows 1-3 of 3

This is a sample result of the **Image Scan** stage using Grype, which is only available in the `test-scan` OOTB supply chain. For more information, see the View Vulnerability Scan Results section.



Stage Detail: Image Scanner

11 hours ago

Overview

Registry	gcr.io
Image	gcr.io/ship-interfaces-dev/supply-chain/tanzu-java-web-app-dev
Digest	sha256:15309dfe594cb81c9e5aeeab84a3cb95ba35d7e4655b920768eea9dbb8353c18
Scan Template	tanzu-java-web-app
UID	75f1b8a2-d30c-4672-ae48-d9c0a76ea2b3
Generation	8

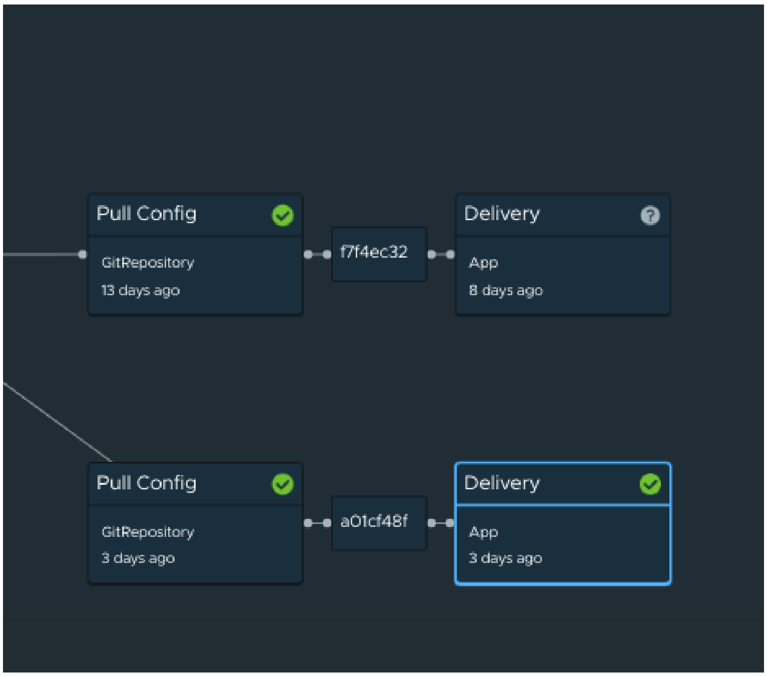
Policy

No policy has been defined.

Vulnerabilities

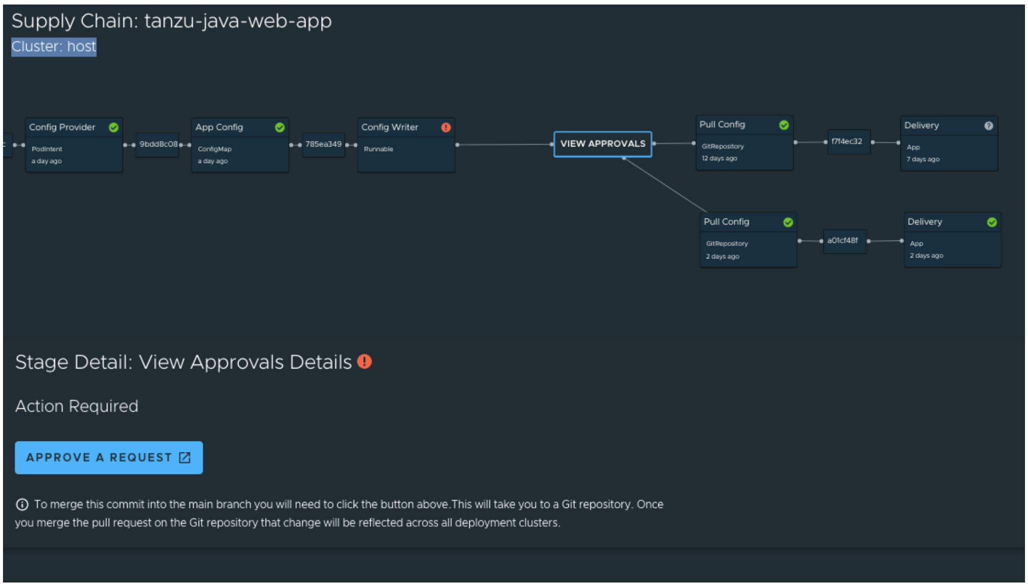
CVE ID	Severity	Package	Version	Description
CVE-2021-44832	Unknown	log4j-to-slf4j	2.17.0	Apache Log4j2 versions 2.0-beta7 through 2.17.0 (excluding security fix releases 2.3.2 and 2.12.4) are vulnerable to a remote code execution (RCE) attack when a configuration uses a JDBC Appender with a JNDI LDAP data source URI when an attacker has control of the target LDAP server. This issue is fixed by limiting JNDI data source names to the java protocol in Log4j2 versions 2.17.1, 2.12.4, and 2.3.2.
CVE-2021-44832	Unknown	log4j-api	2.17.0	Apache Log4j2 versions 2.0-beta7 through 2.17.0 (excluding security fix releases 2.3.2 and 2.12.4) are vulnerable to a remote code execution (RCE) attack when a configuration uses a JDBC Appender with a JNDI LDAP data source URI when an attacker has control of the target LDAP server. This issue is fixed by limiting JNDI data source names to the java protocol in Log4j2 versions 2.17.1, 2.12.4, and 2.3.2.
CVE-2016-2781	Low	coreutils	8.28-1ubuntu1	
CVE-2020-13844	Medium	libstdc++6	8.4.0-1ubuntu1-18.04	
CVE-2020-13844	Medium	gcc-8-base	8.4.0-1ubuntu1-18.04	

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 **View Approvals** in the supply chain diagram.

In the following example, the merge request is approved, which causes **Pull Config** and **Delivery** boxes to appear in the supply chain diagram.


Supply Chain: tanzu-java-web-app

Cluster: host

Stage Detail: View Approvals Details

Action Required

To merge this commit into the main branch you will need to click the button above.This will take you to a Git repository. Once you merge the pull request on the Git repository that change will be reflected across all deployment clusters.

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.

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.

Upgrade Tanzu Application Platform GUI

This topic tells you how to upgrade Tanzu Application Platform GUI (commonly called TAP GUI) 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 Application Platform GUI 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 Application Platform GUI as part of a Tanzu Application Platform profile, see Upgrading Tanzu Application Platform.

Upgrade Tanzu Application Platform GUI individually

These steps only apply to installing Tanzu Application Platform GUI individually, not as part of a Tanzu Application Platform profile.

To upgrade Tanzu Application Platform GUI 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 Application Platform GUI that you want.
   - `TAP-GUI-VALUES` is the configuration values file that contains the configuration used when you installed Tanzu Application Platform GUI.

3. Verify that you upgraded your application by running:

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

Troubleshoot Tanzu Application Platform GUI

This topic tells you how to troubleshoot issues encountered when installing Tanzu Application Platform GUI (commonly called TAP GUI).

Tanzu Application Platform GUI does not work in Safari

Symptom

Tanzu Application Platform GUI does not work in the Safari web browser.

Solution

Currently there is no way to use Tanzu Application Platform GUI in Safari. Please use a different web browser.

Catalog not found

Symptom

When you pull up Tanzu Application Platform GUI, 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 Application Platform GUI 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 Application Platform GUI 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.

Issues updating the values file

Symptom

After updating the configuration of Tanzu Application Platform GUI, 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 Application Platform GUI 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 Application Platform GUI

Symptom

You have a problem with Tanzu Application Platform GUI, 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.

Runtime Resources tab

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:

- `Access error when querying cluster CLUSTER_NAME for resource KUBERNETES_RESOURCE_PATH (status: 401). Contact your administrator.`
  - **Cause:** There is a problem with the cluster configuration.
  - **Solution:** Confirm the access token used to request information in the cluster.
- `Access error when querying cluster CLUSTER_NAME for resource KUBERNETES_RESOURCE_PATH (status: 403). Contact your administrator.`
  - **Cause:** The service account used doesn't have access to the specific resource type in the cluster.
  - **Solution:** If the cluster is the same where **Tanzu Application Platform** is running, review the version installed to confirm it contains the desired resource. If the error is in a watched cluster, review the process to grant access to it in Viewing resources on multiple clusters in Tanzu Application Platform GUI.
- `Knative is not installed on CLUSTER_NAME (status: 404). Contact your administrator.`
  - **Cause:** The cluster does not have Cloud Native Runtimes installed.
  - **Solution:** Install the Knative components by following the instructions in Install Cloud Native Runtimes.

- `Error when querying cluster CLUSTER_NAME for resource KUBERNETES_RESOURCE_PATH (status: 404). Contact your administrator.`
    - **Cause:** The package that contains the resource is not installed.
    - **Solution:** Install the missing package.

Accelerators page

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 Application Platform GUI 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
```

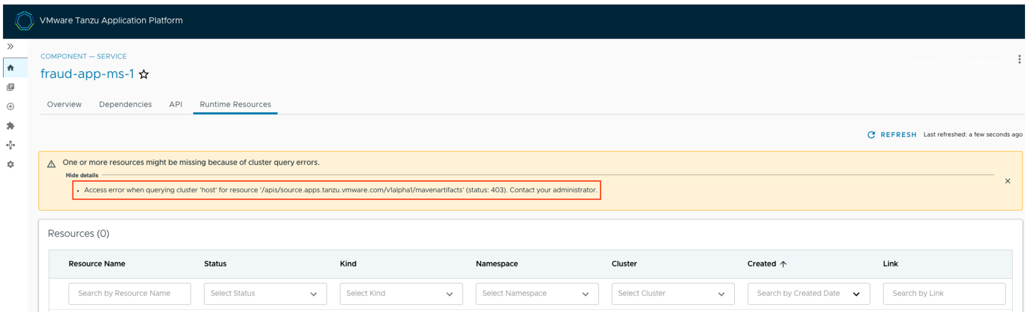
Maven artifacts access error

Symptom

When accessing the **Runtime Resources** tab from the **Component** view, the following warning appears:

```
Access error when querying cluster 'host' for resource '/apis/source.apps.tanzu.vmwar
e.com/v1alpha1/mavenartifacts' (status: 403). Contact your administrator.
```

In most cases, this issue only affects the `full` profile and only when multicluster visibility is not set up for Tanzu Application Platform GUI. It only appears in v1.2.0 and is resolved in v1.2.1.



One or more resources might be missing because of cluster query errors.

- Access error when querying cluster 'host' for resource '/apis/source.apps.tanzu.vmware.com/v1alpha1/mavenartifacts' (status: 403). Contact your administrator.

Resource Name	Status	Kind	Namespace	Cluster	Created ↑	Link
Search by Resource Name	Select Status	Select Kind	Select Namespace	Select Cluster	Search by Created Date	Search by Link

Solution

Register Maven artifacts with the service account of Tanzu Application Platform GUI by using the `package_overlays` key in the Tanzu Application Platform values file.

For instructions, see Customizing Package Installation.

The following is the overlay for adding Maven artifacts to the Tanzu Application Platform GUI service account:

```
#@ load("@ytt:overlay", "overlay")

#@overlay/match by=overlay.subset({"kind": "ClusterRole", "metadata": {"name": "k8s-re
ader"}}), expects="1+"
---
rules:
  #@overlay/match by=overlay.subset({"apiGroups": ["source.apps.tanzu.vmware.com"]})
  - resources: ['mavenartifacts']
```

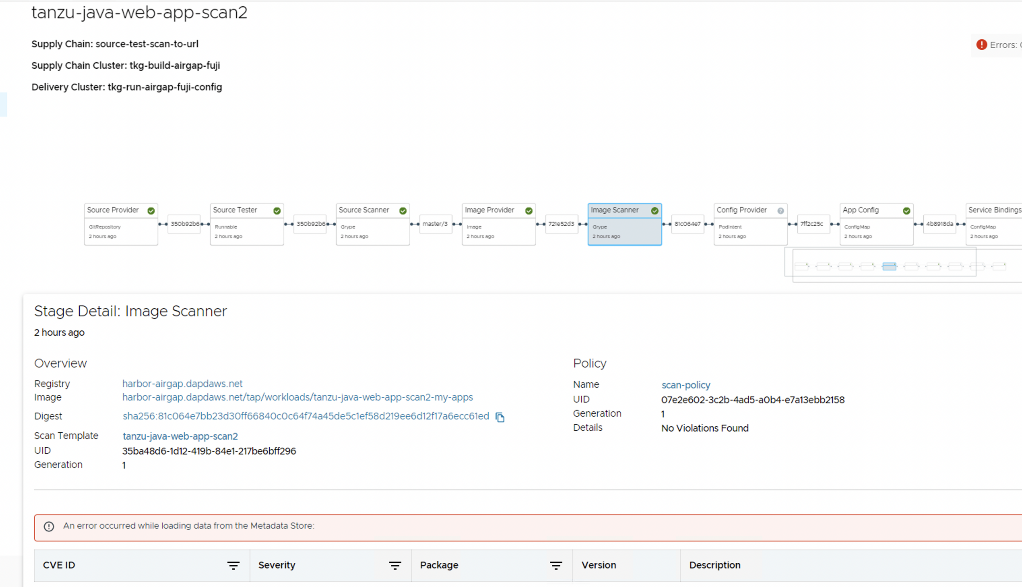
Supply Chain Choreographer plug-in

These are troubleshooting steps 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`.



tanzu-java-web-app-scan2

Supply Chain: source-test-scan-to-url

Supply Chain Cluster: tkg-build-airgap-fuji

Delivery Cluster: tkg-run-airgap-fuji-config

Stage Detail: Image Scanner

2 hours ago

Overview

Registry harbor-airgap.dapdaws.net
Image harbor-airgap.dapdaws.net/tap/workloads/tanzu-java-web-app-scan2-my-apps
Digest sha256:81c064e7bb23d30ff66840c0c64f74a45de5c1ef58d219ee6d12f17a6ecc61ed
Scan Template tanzu-java-web-app-scan2
UID 35ba48d6-1d12-419b-84e1-217be6bff296
Generation 1

Policy

Name scan-policy
UID 07e2e602-3c2b-4ad5-a0b4-e7a13ebb2158
Generation 1
Details No Violations Found

An error occurred while loading data from the Metadata Store:

CVE ID	Severity	Package	Version	Description

Cause

There are multiple potential causes. The most common cause is `tap-values.yaml` missing the configuration that enables Tanzu Application Platform GUI 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 Application Platform GUI deployment with the new values.

Configuring TLS Certificate for Tanzu Application Platform GUI

To configure a TLS certificate for Tanzu Application Platform GUI:

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.

2. Add the Issuer and Certificate to your cluster by running:

   ```
   kubectl apply -f certificate.yaml
   ```

3. Update your `tap-gui` values to include:

   - a top-level `tls` key with subkeys for `namespace` and `secretName`
   - a namespace referring to the namespace containing the above `Certificate` object
   - a secret name referring to the `secretName` value defined in your `Certificate` resource earlier

   Example:

   ```
   tls:
    namespace: tap-gui
    secretName: tap-gui-cert
   ```

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.

For more information about Tanzu Build Service, see the Tanzu Build Service documentation.

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.

For more information about Tanzu Build Service, see the Tanzu Build Service documentation.

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.

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 About Tanzu Application Platform components and profiles.

Note: 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.

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 in the previous step.

3. Create a `tbs-values.yaml` file using the following template:

   ```
   ---
   kp_default_repository: "REPO-NAME"
   kp_default_repository_username: "REPO-USERNAME"
   kp_default_repository_password: "REPO-PASSWORD"
   ```

   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"`.
   - `REPO-USERNAME` and `REPO-PASSWORD` are the username and password for the user that can write to `REPO-NAME`. For Google Cloud Registry, use `_json_key` as the username and the contents of the service account JSON file for the password.

> **Note:** If you do not want to use plaintext for these credentials, you can configure them by using a secret reference or by using AWS IAM authentication. For more information, see Use Secret References for registry credentials or Use AWS IAM authentication for registry credentials.

4. (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_username: "REPO-USERNAME"
   kp_default_repository_password: "REPO-PASSWORD"
   exclude_dependencies: true
   ```

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

5. Install the Tanzu Build Service package by running:

   ```
   tanzu package install tbs -p buildservice.tanzu.vmware.com -v VERSION -n tap-in
   stall -f tbs-values.yaml
   ```

   Where `VERSION` is the version of the Tanzu Build Service package you retrieved earlier.

   For example:

   ```
   $ tanzu package install tbs -p buildservice.tanzu.vmware.com -v VERSION -n tap-
   install -f 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'
   ```

6. (Optional) Verify the cluster builders that the Tanzu Build Service installation created by running:

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

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

(Optional) Alternatives to plaintext registry credentials

Tanzu Build Service requires credentials for the `kp_default_repository` and the Tanzu Network registry.

You can apply them directly in-line in plaintext in the `tbs-values.yaml` or `tap-values.yaml` configuration by using the `kp_default_repository_username`, `kp_default_repository_password`, `tanzunet_username`, and `tanzunet_password` fields.

If you do not want credentials saved in plaintext, you can use existing secrets or IAM roles by using secret references or AWS IAM authentication in your `tbs-values.yaml` or `tap-values.yaml`.

Use Secret references for registry credentials

You might not want to install Tanzu Build Service with passwords saved in plaintext in the `tbs-values.yaml`.

To store these credentials in `Secrets` and reference them in the `tbs-values.yaml`:

1. Using the Tanzu CLI, create a secret of type `kubernetes.io/dockerconfigjson` containing credentials for the writable repository in your registry (`kp_default_repository`):

   ```
   tanzu secret registry add kp-default-repository-creds \
     --username "${USERNAME}" \
     --password "${PASSWORD}" \
     --server "${SERVER-NAME}" \
     --namespace tap-install
   ```

   Where:

   - `USERNAME` and `PASSWORD` are the user name and password for the user that can write to the `kp_default_repository`. For Google Cloud Registry, use `_json_key` as the user name, and the contents of the service account JSON file for the password.
   - `SERVER-NAME` is the host name of the registry server for the `kp_default_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"`.

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: "KP-DEFAULT-REPOSITORY"
   kp_default_repository_secret:
     name: kp-default-repository-creds
     namespace: tap-install
   ```

   Where:

   - `KP-DEFAULT-REPOSITORY` 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"`

3. To apply this configuration, continue the installation steps.

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 in the previous step. 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 location where workload images will be stored.
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. If you have not done so already, add the key-value pair `exclude_dependencies: true` to your `tbs-values.yaml` file. For example:

   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_username: "REPO-USERNAME"
     kp_default_repository_password: "REPO-PASSWORD"
     exclude_dependencies: true
   ```

2. Get the latest version of the Tanzu Build Service package by running:

   ```
   tanzu package available list buildservice.tanzu.vmware.com --namespace tap-inst
   all
   ```

3. Relocate the Tanzu Build Service `full` dependencies package repository by running:

   ```
   imgpkg copy -b registry.tanzu.vmware.com/tanzu-application-platform/full-tbs-de
   ps-package-repo:VERSION \
   --to-repo INSTALL-REGISTRY-HOSTNAME/TARGET-REPOSITORY/tbs-full-deps
   ```

   Where:

   - `VERSION` is the version of the Tanzu Build Service package you retrieved in the previous step.
   - `INSTALL-REGISTRY-HOSTNAME` is your container image registry.
   - `TARGET-REPOSITORY` is your target repository.

4. Add the TBS `full` dependencies package repository by running:

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

Where:

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

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

```
tanzu package install full-tbs-deps -p full-tbs-deps.tanzu.vmware.com -v VERSIO
N -n tap-install
```

Where `VERSION` is the version of the Tanzu Build Service package you retrieved earlier.

(Optional) Configure automatic dependency updates

Important: The automatic updates feature is being deprecated. The recommended way to patch dependencies is by upgrading Tanzu Application Platform to the latest patch version. For upgrade instructions, see Upgrading Tanzu Application Platform.

You can configure Tanzu Build Service to update dependencies in the background as they are released. This enables workloads to keep up to date automatically. For more information about automatic dependency updates, see About automatic dependency updates (deprecated).

To configure automatic dependency updates, add the following to the contents of your `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.

```
tanzunet_username: TANZU-NET-USERNAME
tanzunet_password: TANZU-NET-PASSWORD
descriptor_name: DESCRIPTOR-NAME
enable_automatic_dependency_updates: true
```

Where:

- `TANZU-NET-USERNAME` and `TANZU-NET-PASSWORD` are the email address and password to log in to VMware Tanzu Network. You can also configure these credentials by using a secret reference. For more information, see Use Secret references for registry credentials.
- `DESCRIPTOR-NAME` is the name of the descriptor to import. For more information, see Descriptors. Available options are:
  - `lite` is the default if not set. It has a smaller footprint, which enables faster installations.
  - `full` is optimized to speed up builds and includes dependencies for all supported workload types.

Install Tanzu Build Service on an air-gapped environment

This topic describes how to install Tanzu Build Service on a Kubernetes cluster and registry that are air-gapped from external traffic.

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 About Tanzu Application Platform components and profiles.

To install Tanzu Build Service on an air-gapped environment, you must:

1. Install the Tanzu Build Service package
2. Install the Tanzu Build Service dependencies

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.

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_username: REGISTRY-USERNAME
   kp_default_repository_password: REGISTRY-PASSWORD
   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`
   - `REPO-USERNAME` and `REPO-PASSWORD` are the user name and password for the user that can write to `REPO-NAME`.

     Note: If you do not want to use plaintext for these credentials, you can instead configure these credentials by using a Secret reference. For more information, see Use Secret references for registry credentials.

   - `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 -p buildservice.tanzu.vmware.com -v VERSION -n tap-in
   stall -f tbs-values.yaml
   ```

   Where `VERSION` is the version of the Tanzu Build Service package you retrieved earlier.

For example:

```
$ tanzu package install tbs -p buildservice.tanzu.vmware.com -v VERSION -n tap-
install -f 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

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

When installing Tanzu Build Service on an air-gapped environment, the `lite` dependencies cannot be used as they require Internet access. You must install the `full` dependencies.

To install `full` dependencies:

1. Relocate the Tanzu Build Service `full` dependencies package repository by running:

```
imgpkg copy -b registry.tanzu.vmware.com/tanzu-application-platform/full-tbs-de
ps-package-repo:VERSION \
  --to-tar=tbs-full-deps.tar
# move tbs-full-deps.tar to environment with registry access
imgpkg copy --tar tbs-full-deps.tar \
  --to-repo=INSTALL-REGISTRY-HOSTNAME/TARGET-REPOSITORY/tbs-full-deps
```

   Where:

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

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

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

   Where:

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

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

```
tanzu package install full-tbs-deps -p full-tbs-deps.tanzu.vmware.com -v VERSIO
N -n tap-install
```

   Where `VERSION` is the version of the Tanzu Build Service package you retrieved earlier.

Configure Tanzu Build Service properties on a workload

This topic tells you how to configure your workload with Tanzu Build Service properties.

Tanzu Build Service builds registry images from source code for Tanzu Application Platform. You can configure these build configurations by using a workload.

Note: 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` for a secret as follows:

   ```
   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.

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": "Secret"}]' \
     ...
   ```

   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"`.

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 kpack tutorial, this topic builds on 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 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 key pair 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 successfully, 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
   ```

   Note: For more information about configuring Cosign key pairs, see 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 key pair 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"`

   Note: If you are using Out of the Box Supply Chains, modify the respective `ClusterImageTemplate` to enable signing in your supply chain. For more information, see Authoring supply chains.

   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.

8. Apply the image resource to the cluster by running:

   ```
   kubectl apply -f image-cosign.yaml
   ```

9. 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
```

10. Verify image signature by running:

    ```
    cosign verify --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...`

    The expected output is similar to the following:

    ```
    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.
    ```

11. 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.

Tanzu Build Service Dependencies

This topic tells you about Tanzu Build Service dependencies.

Tanzu Build Service requires dependencies in the form of Cloud Native Buildpacks and Stacks to build OCI images.

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 Upgrading Tanzu Application Platform.

To upgrade Tanzu Build Service dependencies outside of Tanzu Application Platform releases, use the `kpack` CLI. This enables you to consume new versions of buildpacks and stacks and remediate vulnerabilities more quickly. For more information, see Updating Build Service Dependencies.

By default, Tanzu Build Service is installed with the `lite` set of dependencies, which are 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 later in this topic.

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.

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`
- `default` (identical to `base`)

For more information, see Stacks in the VMware Tanzu Buildpacks documentation.

Lite dependencies: buildpacks

The `lite` dependencies contain the following buildpacks:

Buildpack	Version included in Tanzu Application Platform v1.2
Java Buildpack for VMware Tanzu (Lite)	6.27.0
Java Native Image Buildpack for Tanzu (Lite)	6.18.1
.NET Core Buildpack for VMware Tanzu (Lite)	1.14.3
Node.js Buildpack for VMware Tanzu (Lite)	1.14.2
Python Buildpack for VMware Tanzu (Lite)	2.0.0
Go Buildpack for VMware Tanzu (Lite)	1.12.0
NGINX Buildpack for VMware Tanzu (Lite)	0.6.0
Procfile Buildpack for VMware Tanzu (Lite)	5.2.1
Base Stack of Ubuntu Bionic for VMware Tanzu	1.3.54

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`
- `default` (identical to `base`)
- `full`

- `tiny`

For more information, see Stacks in the VMware Tanzu Buildpacks documentation.

Full dependencies: buildpacks

The `full` dependencies contain the following buildpacks:

Buildpack	Version included in Tanzu Application Platform v1.2
Java Buildpack for VMware Tanzu	6.27.0
Java Native Image Buildpack for Tanzu	6.18.1
.NET Core Buildpack for VMware Tanzu	1.14.3
Node.js Buildpack for VMware Tanzu	1.14.2
Python Buildpack for VMware Tanzu	2.0.0
Go Buildpack for VMware Tanzu	1.12.0
PHP Buildpack for VMware Tanzu	0.1.0
Web Servers Buildpack for VMware Tanzu	1.0.1
NGINX Buildpack for VMware Tanzu	0.6.0
Procfile Buildpack for VMware Tanzu	5.2.1
Tiny Stack of Ubuntu Bionic for VMware Tanzu	1.1.72
Base Stack of Ubuntu Bionic for VMware Tanzu	1.3.54
Full Stack of Ubuntu Bionic for VMware Tanzu	1.3.62

Dependency comparison

The following table compares the contents of the `lite` and `full` dependencies.

	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
Supports Java workloads	Yes	Yes
Supports Node.js workloads	Yes	Yes
Supports Go workloads	Yes	Yes
Supports Python workloads	Yes	Yes
Supports .NET Core workloads	Yes	Yes
Supports PHP workloads	No	Yes
Supports static workloads	Yes	Yes
Supports binary workloads	Yes	Yes
Supports web servers buildpack	No	Yes

About automatic dependency updates (deprecated)

Important: The automatic updates feature is being deprecated. The recommended way to patch dependencies is by upgrading Tanzu Application Platform to the latest patch version. For upgrade instructions, see Upgrading Tanzu Application Platform.

You can configure Tanzu Build Service to update dependencies in the background as they are released. This enables workloads to keep up to date automatically.

Descriptors (deprecated)

Tanzu Build Service descriptors are curated sets of dependencies that include stacks and buildpacks. Descriptors are only used if Tanzu Build Service is configured for automatic dependency updates. Descriptors are imported into Tanzu Build Service to update the entire cluster.

Descriptors are continuously released on the VMware Tanzu Network Build Service Dependencies page to provide updated buildpack dependencies and updated stack images. This allows the use of dependencies that have patched CVEs. For more information about buildpacks and stacks, see the VMware Tanzu Buildpacks documentation.

There are two types of descriptor, `lite` and `full`. The different descriptors can apply to different use cases and workload types. The differences between the `full` and `lite` descriptors are the same as the the differences between `full` and `lite` dependencies. For a comparison of the `lite` and `full` descriptors, see About lite and full dependencies.

Troubleshooting 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 after upgrading to Tanzu Application Platform v1.2

Symptom

After upgrading to Tanzu Application Platform v1.2, you see failing builds.

Explanation

After the upgrade, Tanzu Build Service image resources automatically run a build that fails due to a missing dependency.

This error does not persist and any subsequent builds will resolve this error.

Solution

You can safely wait for the next build of the workloads, which is triggered by new source code changes.

If you do not want to wait for subsequent builds to run automatically, you can use the open source kp CLI to re-run failing builds:

1. List the image resources in the developer namespace by running:

   ```
   kp image list -n DEVELOPER-NAMESPACE
   ```

   Where `DEVELOPER-NAMESPACE` is the namespace where workloads are created.

2. Manually trigger the image resources to re-run builds for each failing image by running:

   ```
   kp image trigger IMAGE-NAME -n DEVELOPER-NAMESPACE
   ```

   Where:

   - `IMAGE-NAME` is the name of the failing image.
   - `DEVELOPER-NAMESPACE` is the namespace where workloads are created.

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'
```

Explanation

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. 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.

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-premise 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-premise 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.

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>
```

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.

⚠ **Caution**

Windows workloads are deactivated and cause an error if any Tasks try to use Windows scripts.

Workload types

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 Command Reference.

Tanzu Application Platform supports a range of workload types, including scalable web applications (`web`), traditional application servers (`tcp`), background applications (`queue`), and serverless functions. You can use a collection of workloads of different types to deploy microservices that function as a logical application, or deploy your entire application as a single monolith.

Workload types

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 Command Reference.

Tanzu Application Platform supports a range of workload types, including scalable web applications (`web`), traditional application servers (`tcp`), background applications (`queue`), and serverless functions. You can use a collection of workloads of different types to deploy microservices that function as a logical application, or deploy your entire application as a single monolith.

Using web workloads

This topic tells you how to use the `web` workload type in Tanzu Application Platform (commonly known as TAP).

The `web` workload is a good match for modern web applications that store state in external databases and follow the 12-factor principles.

The out of the box (OOTB) supply chains include definitions for the `web` workload type which leverage Cloud Native Runtimes to provide:

- 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 `tanzu apps workload create`, you can use the `--type=web` argument to select the `web` workload type. You can also use the `apps.tanzu.vmware.com/workload-type:web` label in the YAML workload description to support this deployment type.

Using TCP workloads (Beta)

This topic describes how to create and install a supply chain for the `tcp` workload type.

Overview

The `tcp` 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 `tcp` workload is a good match for traditional applications, including HTTP applications, that are implemented as follows:

- 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

Applications using the `tcp` workload type have the following features:

- Do not natively autoscale, but can be used with the Kubernetes Horizontal Pod Autoscaler
- By default are exposed only within the cluster using a `ClusterIP` Service
- Use health checks if defined by a convention
- Use a rolling update pattern by default

When creating a workload with `tanzu apps workload create`, you can use the `--type=tcp` argument to select the `tcp` workload type. For more information, see Use the `tcp` Workload Type later in this topic. You can also use the `apps.tanzu.vmware.com/workload-type:tcp` annotation in the YAML workload description to support this deployment type.

Important: Beta features have been tested for functionality, but not performance. Features enter the beta stage so that customers can gain early access, and give feedback on the design and behavior. Beta features might undergo changes based on this feedback before the end of the beta stage. VMware discourages running beta features in production. VMware cannot guarantee that you can upgrade any beta feature in the future.

Prerequisites

Before using `tcp` workloads on Tanzu Application Platform, you must:

- Follow all instructions in Installing Tanzu Application Platform.
- Follow all instructions in Set up developer namespaces to use installed packages.

Create a `tcp` SupplyChain

This section describes how to create a supply chain for the `tcp` workload type.

Create supply chain templates

The `tcp` supply chain replaces the `config-template` from the existing out of the box (OOTB) supply chain with two new templates:

- The `deployment-and-service-template` defines Kubernetes Deployment and Service objects that represent the workload, instead of a Knative Service.
- The `apply-bindings` template extends the `deployment-and-service-template` with requested ServiceBindings and ResourceClaims.

To create supply chain templates:

1. Create a file using the following YAML manifests:

```
apiVersion: carto.run/v1alpha1
kind: ClusterConfigTemplate
metadata:
  name: deployment-and-service-template
spec:
  configPath: .data
  ytt: |
    #@ load("@ytt:data", "data")
    #@ load("@ytt:yaml", "yaml")

    #@ def merge_labels(fixed_values):
    #@   labels = {}
    #@   if hasattr(data.values.workload.metadata, "labels"):
    #@     labels.update(data.values.workload.metadata.labels)
    #@   end
    #@   labels.update(fixed_values)
    #@   return labels
    #@ end

    #@ def delivery():
    apiVersion: apps/v1
    kind: Deployment
    metadata:
      name: #@ data.values.workload.metadata.name
      annotations:
        kapp.k14s.io/update-strategy: "fallback-on-replace"
      labels: #@ merge_labels({ "app.kubernetes.io/component": "run", "carto.ru
n/workload-name": data.values.workload.metadata.name })
    spec:
      selector:
        matchLabels: #@ data.values.config.metadata.labels
      template: #@ data.values.config
    #@ end

    #@ def merge_ports(ports_spec, containers):
    #@   ports = {}
    #@   for c in containers:
    #@     for p in getattr(c, "ports", []):
    #@       ports[p.containerPort] = {"targetPort": p.containerPort, "port":
p.containerPort, "name": getattr(p, "name", str(p.containerPort))}
    #@     end
    #@   end
    #@   for p in ports_spec:
    #@     ports[p.port] = {"targetPort": getattr(p, "containerPort", p.port),
"port": p.port, "name": getattr(p, "name", str(p.port))}
    #@   end
    #@   return ports.values()
    #@ end


    #@ def services():
    ---
    apiVersion: v1
    kind: Service
    metadata:
      name: #@ data.values.workload.metadata.name
      labels: #@ merge_labels({ "app.kubernetes.io/component": "run", "carto.ru
n/workload-name": data.values.workload.metadata.name })
    spec:
      selector: #@ data.values.config.metadata.labels
      ports:
      #@ declared_ports = {}
      #@ if "ports" in data.values.params:
      #@   declared_ports = data.values.params.ports
      #@ end
      #@ for p in merge_ports(declared_ports, data.values.config.spec.container
s):
      - #@ p
      #@ end
    #@ end

    ---
    apiVersion: v1
    kind: ConfigMap
    metadata:
      name: #@ data.values.workload.metadata.name + "-base"
      labels: #@ merge_labels({ "app.kubernetes.io/component": "config" })
    data:
      delivery.yml: #@ yaml.encode(delivery())
      service.yaml: #@ yaml.encode(services())
---
apiVersion: carto.run/v1alpha1
kind: ClusterConfigTemplate
metadata:
  name: apply-bindings
spec:
  configPath: .data
  ytt: |
    #@ load("@ytt:data", "data")
    #@ load("@ytt:yaml", "yaml")
    #@ load("@ytt:json", "json")
    #@ load("@ytt:struct", "struct")

    #@ def get_claims_extension():
    #@   claims_extension_key = "serviceclaims.supplychain.apps.x-tanzu.vmware.
com/extensions"
    #@   if not hasattr(data.values.workload.metadata, "annotations") or not ha
sattr(data.values.workload.metadata.annotations, claims_extension_key):
    #@     return None
    #@   end
    #@
    #@   extension = json.decode(data.values.workload.metadata.annotations[clai
ms_extension_key])
    #@
    #@   spec_extension = extension.get('spec')
    #@   if spec_extension == None:
    #@     return None
    #@   end
    #@
    #@   return spec_extension.get('serviceClaims')
    #@ end

    #@ def merge_claims_extension(claim, claims_extension):
    #@   if claims_extension == None:
    #@     return claim.ref
    #@   end
    #@   extension = claims_extension.get(claim.name)
    #@   if extension == None:
    #@      return claim.ref
    #@   end
    #@   extension.update(claim.ref)
    #@   return extension
    #@ end

    #@ def param(key):
    #@   if not key in data.values.params:
    #@     return None
    #@   end
    #@   return data.values.params[key]
    #@ end

    #@ def merge_labels(fixed_values):
```

```
    #@   labels = {}
    #@   if hasattr(data.values.workload.metadata, "labels"):
    #@     labels.update(data.values.workload.metadata.labels)
    #@   end
    #@   labels.update(fixed_values)
    #@   return labels
    #@ end

    #@ def merge_annotations(fixed_values):
    #@   annotations = {}
    #@   if hasattr(data.values.workload.metadata, "annotations"):
    #@     # DEPRECATED: remove in a future release
    #@     annotations.update(data.values.workload.metadata.annotations)
    #@   end
    #@   if type(param("annotations")) == "dict" or type(param("annotations"))
== "struct":
    #@     annotations.update(param("annotations"))
    #@   end
    #@   annotations.update(fixed_values)
    #@   return annotations
    #@ end

    #@ def claims():
    #@ claims_extension = get_claims_extension()
    #@ workload = struct.encode(yaml.decode(data.values.configs.app_def.config
["delivery.yml"]))
    #@ for s in data.values.workload.spec.serviceClaims:
    #@ if claims_extension == None or claims_extension.get(s.name) == None:
    ---
    apiVersion: servicebinding.io/v1alpha3
    kind: ServiceBinding
    metadata:
      name: #@ data.values.workload.metadata.name + '-' + s.name
      annotations: #@ merge_annotations({})
      labels: #@ merge_labels({ "app.kubernetes.io/component": "run", "carto.ru
n/workload-name": data.values.workload.metadata.name })
    spec:
      name: #@ s.name
      service: #@ s.ref
      workload:
        apiVersion: #@ workload.apiVersion
        kind: #@ workload.kind
        name: #@ workload.metadata.name
    #@ else:
    ---
    apiVersion: services.apps.tanzu.vmware.com/v1alpha1
    kind: ResourceClaim
    metadata:
      name: #@ data.values.workload.metadata.name + '-' + s.name
      annotations: #@ merge_annotations({})
      labels: #@ merge_labels({ "app.kubernetes.io/component": "run", "carto.ru
n/workload-name": data.values.workload.metadata.name })
    spec:
      ref: #@ merge_claims_extension(s, claims_extension)
    ---
    apiVersion: servicebinding.io/v1alpha3
    kind: ServiceBinding
    metadata:
      name: #@ data.values.workload.metadata.name + '-' + s.name
      annotations: #@ merge_annotations({})
      labels: #@ merge_labels({ "app.kubernetes.io/component": "run", "carto.ru
n/workload-name": data.values.workload.metadata.name })
    spec:
      name: #@ s.name
      service:
        apiVersion: services.apps.tanzu.vmware.com/v1alpha1
        kind: ResourceClaim
        name: #@ data.values.workload.metadata.name + '-' + s.name
      workload:
        apiVersion: #@ workload.apiVersion
        kind: #@ workload.kind
        name: #@ workload.metadata.name
    #@ end
    #@ end
    #@ end

    #@ def add_claims():
    #@ if hasattr(data.values.workload.spec, "serviceClaims") and len(data.valu
es.workload.spec.serviceClaims):
    #@   new_data = struct.decode(data.values.configs.app_def.config)
    #@   new_data.update({"serviceclaims.yml":yaml.encode(claims())})
    #@   return new_data
    #@ else:
    #@   return struct.decode(data.values.configs.app_def.config)
    #@ end
    #@ end

    ---
    apiVersion: v1
    kind: ConfigMap
    metadata:
      name: #@ data.values.workload.metadata.name + "-claims"
      labels: #@ merge_labels({ "app.kubernetes.io/component": "config" })
    data: #@ add_claims()
```

2. Apply the YAML file by running the command:

   ```
   kubectl apply -f FILENAME
   ```

   Where `FILENAME` is the name of the file you created in the previous step.

Add RBAC permissions

Because the `queue` workload deployment creates different resources, you must extend the `deliverable` ClusterRole.

To add the additional role to the cluster:

1. Create a file using the following YAML:

   ```
   apiVersion: rbac.authorization.k8s.io/v1
   kind: ClusterRole
   metadata:
     name: additional-k8s-deliverable
     labels:
       apps.tanzu.vmware.com/aggregate-to-deliverable: "true"
   rules:
   - apiGroups: [""]
     resources: ["services"]
     verbs: ["get", "list", "watch", "create", "patch", "update", "delete", "delet
   ecollection"]
   - apiGroups: ["apps"]
     resources: ["deployments"]
     verbs: ["get", "list", "watch", "create", "patch", "update", "delete", "delet
   ecollection"]
   ```

2. Apply the YAML file by running the command:

   ```
   kubectl apply -f FILENAME
   ```

   Where `FILENAME` is the name of the file you created in the previous step.

Define the ClusterSupplyChain

To define the ClusterSupplyChain:

1. Create a file using the following YAML and substitute in your `registry` values from your `tap-values.yaml` file:

   ```
   apiVersion: carto.run/v1alpha1
   kind: ClusterSupplyChain
   metadata:
     name: tcp
   ```

```
spec:
  params:
  - default: main
    name: gitops_branch
  - default: supplychain
    name: gitops_user_name
  - default: supplychain
    name: gitops_user_email
  - default: supplychain@cluster.local
    name: gitops_commit_message
  - default: DEFAULT-GIT-SECRET
    name: gitops_ssh_secret
  - default:
    - containerPort: 8080
      port: 8080
      name: http
    name: ports
  resources:
  - name: source-provider
    params:
    - name: serviceAccount
      value: default
    - name: gitImplementation
      value: go-git
    templateRef:
      kind: ClusterSourceTemplate
      name: source-template
  - name: deliverable
    params:
    - name: registry
      value:
        repository: REGISTRY-REPO
        server: REGISTRY-SERVER
        # Add the following key if you have set ca_cert_data in tap-values.yaml
        - name: ca_cert_data
          value: CERT-AS-STRING
    templateRef:
      kind: ClusterTemplate
      name: deliverable-template
  - name: image-builder
    params:
    - name: serviceAccount
      value: default
    - name: clusterBuilder
      value: default
    - name: registry
      value:
        repository: REGISTRY-REPO
        server: REGISTRY-SERVER
        # Add the following key if you have set ca_cert_data in tap-values.yaml
        - name: ca_cert_data
          value: CERT-AS-STRING
    sources:
    - name: source
      resource: source-provider
    templateRef:
      kind: ClusterImageTemplate
      name: kpack-template
  - images:
    - name: image
      resource: image-builder
    name: config-provider
    params:
    - name: serviceAccount
      value: default
    templateRef:
      kind: ClusterConfigTemplate
      name: convention-template
  - configs:
    - name: config
      resource: config-provider
    name: app-config
    templateRef:
      kind: ClusterConfigTemplate
      name: deployment-and-service-template
  - configs:
    - name: app_def
      resource: app-config
    name: apply-bindings
    templateRef:
      kind: ClusterConfigTemplate
      name: apply-bindings
  - configs:
    - name: config
      resource: apply-bindings
    name: config-writer
    params:
    - name: serviceAccount
      value: default
    - name: registry
      value:
        repository: REGISTRY-REPO
        server: REGISTRY-SERVER
        # Add the following key if you have set ca_cert_data in tap-values.yaml
        - name: ca_cert_data
          value: CERT-AS-STRING
    templateRef:
      kind: ClusterTemplate
      name: config-writer-template
  selector:
    apps.tanzu.vmware.com/workload-type: tcp
```

Where:

- `DEFAULT-GIT-SECRET` is the value from `gitops.ssh_secret` in your `tap-values.yaml` file, or `""` to deactivate SSH authentication.
- `REGISTRY-SERVER` is the registry server from your `tap-values.yaml` file.
- `REGISTRY-REPO` is the registry repository from your `tap-values.yaml` file.
- `CERT-AS-STRING` is the value you added to `tap-values.yaml` file. Only add this if you set `ca_cert_data` in your `tap-values.yaml` file.

2. Apply the YAML file by running the command:

```
kubectl apply -f FILENAME
```

Where `FILENAME` is the name of the file you created in the previous step.

Use the `tcp` workload type

The `spring-sensors-consumer-web` workload in the getting started example using Service Toolkit claims is a good match for the `tcp` 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.

If you have followed the Services Toolkit example, you can update the `spring-sensors-consumer-web` to use the `tcp` supply chain by changing the workload type by running:

```
tanzu apps workload update spring-sensors-consumer-web --type=tcp
```

This shows the change in the workload label, and prompts you to accept the change. After the workload completes the new deployment, you'll notice a few differences:

- The workload no longer advertises 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 also set up a Kubernetes ingress rule to direct traffic from outside the cluster to the workload. Using an ingress rule, you can 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.

Using queue workloads (Beta)

This topic describes how to create and install a supply chain for the `queue` workload type.

Overview

The `queue` 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 `queue` workload is a good match for applications that manage their own work by reading from a queue or a background scheduled time source, and don't expose any network interfaces.

Applications using the `queue` workload type have the following features:

- Do not natively autoscale, but can be used with the Kubernetes Horizontal Pod Autoscaler
- Do not expose any network services
- Use health checks if defined by a convention
- Use a rolling update pattern by default

When creating a workload with `tanzu apps workload create`, you can use the `--type=queue` argument to select the `queue` workload type. For more information, see Use the `queue` Workload Type later in this topic. You can also use the `apps.tanzu.vmware.com/workload-type:queue` annotation in the YAML workload description to support this deployment type.

Important: Beta features have been tested for functionality, but not performance. Features enter the beta stage so that customers can gain early access, and give feedback on the design and behavior. Beta features might undergo changes based on this feedback before the end of the beta stage. VMware discourages running beta features in production. VMware cannot guarantee that you can upgrade any beta feature in the future.

Prerequisites

Before using `queue` workloads on Tanzu Application Platform, you must:

- Follow all instructions in Installing Tanzu Application Platform.
- Follow all instructions in Set up developer namespaces to use installed packages.

Create a `queue` SupplyChain

This section describes how to create a supply chain for the `queue` workload type.

Create supply chain templates

The `queue` supply chain replaces the `config-template` from the existing Out of the Box (OOTB) supply chain with two new templates:

- The `deployment-template` defines Kubernetes Deployment and Service objects that represent the workload, instead of a Knative Service.
- The `apply-bindings` template extends the `deployment-and-service-template` with requested ServiceBindings and ResourceClaims.

To create supply chain templates:

1. Create a file using the following YAML manifests:

```
apiVersion: carto.run/v1alpha1
kind: ClusterConfigTemplate
metadata:
  name: deployment-template
spec:
  configPath: .data
  ytt: |
    #@ load("@ytt:data", "data")
    #@ load("@ytt:yaml", "yaml")

    #@ def merge_labels(fixed_values):
    #@   labels = {}
    #@   if hasattr(data.values.workload.metadata, "labels"):
    #@     labels.update(data.values.workload.metadata.labels)
    #@   end
    #@   labels.update(fixed_values)
    #@   return labels
    #@ end

    #@ def delivery():
    apiVersion: apps/v1
    kind: Deployment
    metadata:
      name: #@ data.values.workload.metadata.name
      annotations:
        kapp.k14s.io/update-strategy: "fallback-on-replace"
      labels: #@ merge_labels({ "app.kubernetes.io/component": "run", "carto.ru
n/workload-name": data.values.workload.metadata.name })
    spec:
      selector:
        matchLabels: #@ data.values.config.metadata.labels
      template: #@ data.values.config
    #@ end

    ---
    apiVersion: v1
    kind: ConfigMap
    metadata:
      name: #@ data.values.workload.metadata.name + "-base"
      labels: #@ merge_labels({ "app.kubernetes.io/component": "config" })
    data:
      delivery.yml: #@ yaml.encode(delivery())
---
apiVersion: carto.run/v1alpha1
kind: ClusterConfigTemplate
metadata:
  name: apply-bindings
spec:
  configPath: .data
  ytt: |
    #@ load("@ytt:data", "data")
    #@ load("@ytt:yaml", "yaml")
    #@ load("@ytt:json", "json")
    #@ load("@ytt:struct", "struct")

    #@ def get_claims_extension():
    #@   claims_extension_key = "serviceclaims.supplychain.apps.x-tanzu.vmware.
com/extensions"
    #@   if not hasattr(data.values.workload.metadata, "annotations") or not ha
sattr(data.values.workload.metadata.annotations, claims_extension_key):
    #@     return None
    #@   end
    #@
    #@   extension = json.decode(data.values.workload.metadata.annotations[clai
ms_extension_key])
    #@
    #@   spec_extension = extension.get('spec')
    #@   if spec_extension == None:
    #@     return None
    #@   end
    #@
    #@   return spec_extension.get('serviceClaims')
    #@ end

    #@ def merge_claims_extension(claim, claims_extension):
    #@   if claims_extension == None:
    #@     return claim.ref
    #@   end
    #@   extension = claims_extension.get(claim.name)
    #@   if extension == None:
    #@      return claim.ref
    #@   end
    #@   extension.update(claim.ref)
    #@   return extension
    #@ end

    #@ def param(key):
    #@   if not key in data.values.params:
    #@     return None
    #@   end
    #@   return data.values.params[key]
    #@ end

    #@ def merge_labels(fixed_values):
    #@   labels = {}
    #@   if hasattr(data.values.workload.metadata, "labels"):
    #@     labels.update(data.values.workload.metadata.labels)
    #@   end
    #@   labels.update(fixed_values)
    #@   return labels
    #@ end

    #@ def merge_annotations(fixed_values):
    #@   annotations = {}
    #@   if hasattr(data.values.workload.metadata, "annotations"):
    #@     # DEPRECATED: remove in a future release
    #@     annotations.update(data.values.workload.metadata.annotations)
    #@   end
    #@   if type(param("annotations")) == "dict" or type(param("annotations"))
== "struct":
    #@     annotations.update(param("annotations"))
    #@   end
    #@   annotations.update(fixed_values)
    #@   return annotations
    #@ end

    #@ def claims():
    #@ claims_extension = get_claims_extension()
    #@ workload = struct.encode(yaml.decode(data.values.configs.app_def.config
["delivery.yml"]))
    #@ for s in data.values.workload.spec.serviceClaims:
    #@ if claims_extension == None or claims_extension.get(s.name) == None:
    ---
    apiVersion: servicebinding.io/v1alpha3
    kind: ServiceBinding
    metadata:
      name: #@ data.values.workload.metadata.name + '-' + s.name
      annotations: #@ merge_annotations({})
      labels: #@ merge_labels({ "app.kubernetes.io/component": "run", "carto.ru
n/workload-name": data.values.workload.metadata.name })
    spec:
      name: #@ s.name
      service: #@ s.ref
      workload:
        apiVersion: #@ workload.apiVersion
        kind: #@ workload.kind
        name: #@ workload.metadata.name
    #@ else:
    ---
    apiVersion: services.apps.tanzu.vmware.com/v1alpha1
    kind: ResourceClaim
    metadata:
      name: #@ data.values.workload.metadata.name + '-' + s.name
      annotations: #@ merge_annotations({})
      labels: #@ merge_labels({ "app.kubernetes.io/component": "run", "carto.ru
n/workload-name": data.values.workload.metadata.name })
    spec:
      ref: #@ merge_claims_extension(s, claims_extension)
    ---
    apiVersion: servicebinding.io/v1alpha3
    kind: ServiceBinding
    metadata:
      name: #@ data.values.workload.metadata.name + '-' + s.name
      annotations: #@ merge_annotations({})
      labels: #@ merge_labels({ "app.kubernetes.io/component": "run", "carto.ru
n/workload-name": data.values.workload.metadata.name })
    spec:
      name: #@ s.name
      service:
        apiVersion: services.apps.tanzu.vmware.com/v1alpha1
        kind: ResourceClaim
        name: #@ data.values.workload.metadata.name + '-' + s.name
      workload:
        apiVersion: #@ workload.apiVersion
        kind: #@ workload.kind
        name: #@ workload.metadata.name
    #@ end
    #@ end
    #@ end

    #@ def add_claims():
    #@ if hasattr(data.values.workload.spec, "serviceClaims") and len(data.valu
es.workload.spec.serviceClaims):
    #@   new_data = struct.decode(data.values.configs.app_def.config)
    #@   new_data.update({"serviceclaims.yml":yaml.encode(claims())})
    #@   return new_data
    #@ else:
    #@   return struct.decode(data.values.configs.app_def.config)
    #@ end
    #@ end

    ---
    apiVersion: v1
    kind: ConfigMap
    metadata:
      name: #@ data.values.workload.metadata.name + "-claims"
      labels: #@ merge_labels({ "app.kubernetes.io/component": "config" })
    data: #@ add_claims()
```

2. Apply the YAML file by running the command:

   ```
   kubectl apply -f FILENAME
   ```

   Where `FILENAME` is the name of the file you created in the previous step.

Add RBAC permissions

Because the `queue` workload deployment creates different resources, you must extend the `deliverable` ClusterRole.

To add the additional role to the cluster:

1. Create a file using the following YAML:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: additional-k8s-deliverable
  labels:
    apps.tanzu.vmware.com/aggregate-to-deliverable: "true"
rules:
- apiGroups: [""]
  resources: ["services"]
  verbs: ["get", "list", "watch", "create", "patch", "update", "delete", "delet
```

```
ecollection"]
- apiGroups: ["apps"]
  resources: ["deployments"]
  verbs: ["get", "list", "watch", "create", "patch", "update", "delete", "delet
ecollection"]
```

2. Apply the YAML file by running the command:

```
kubectl apply -f FILENAME
```

   Where `FILENAME` is the name of the file you created in the previous step.

Define the ClusterSupplyChain

To define the ClusterSupplyChain:

1. Create a file using the following YAML and substitute in your `registry` values from your `tap-values.yaml` file:

```
apiVersion: carto.run/v1alpha1
kind: ClusterSupplyChain
metadata:
  name: queue
spec:
  params:
  - default: main
    name: gitops_branch
  - default: supplychain
    name: gitops_user_name
  - default: supplychain
    name: gitops_user_email
  - default: supplychain@cluster.local
    name: gitops_commit_message
  - default: DEFAULT-GIT-SECRET
    name: gitops_ssh_secret
  resources:
  - name: source-provider
    params:
    - name: serviceAccount
      value: default
    - name: gitImplementation
      value: go-git
    templateRef:
      kind: ClusterSourceTemplate
      name: source-template
  - name: deliverable
    params:
    - name: registry
      value:
        repository: REGISTRY-REPO
        server: REGISTRY-SERVER
        # Add the following key if you have set ca_cert_data in tap-values.yaml
        - name: ca_cert_data
          value: CERT-AS-STRING
    templateRef:
      kind: ClusterTemplate
      name: deliverable-template
  - name: image-builder
    params:
    - name: serviceAccount
      value: default
    - name: clusterBuilder
      value: default
    - name: registry
      value:
        repository: REGISTRY-REPO
        server: REGISTRY-SERVER
        # Add the following key if you have set ca_cert_data in tap-values.yaml
        - name: ca_cert_data
          value: CERT-AS-STRING
    sources:
```

```
    - name: source
      resource: source-provider
    templateRef:
      kind: ClusterImageTemplate
      name: kpack-template
  - images:
    - name: image
      resource: image-builder
    name: config-provider
    params:
    - name: serviceAccount
      value: default
    templateRef:
      kind: ClusterConfigTemplate
      name: convention-template
  - configs:
    - name: config
      resource: config-provider
    name: app-config
    templateRef:
      kind: ClusterConfigTemplate
      name: deployment-template
  - configs:
    - name: app_def
      resource: app-config
    name: apply-bindings
    templateRef:
      kind: ClusterConfigTemplate
      name: apply-bindings
  - configs:
    - name: config
      resource: apply-bindings
    name: config-writer
    params:
    - name: serviceAccount
      value: default
    - name: registry
      value:
        repository: REGISTRY-REPO
        server: REGISTRY-SERVER
        # Add the following key if you have set ca_cert_data in tap-values.yaml
        - name: ca_cert_data
          value: CERT-AS-STRING
    templateRef:
      kind: ClusterTemplate
      name: config-writer-template
  selector:
    apps.tanzu.vmware.com/workload-type: queue
```

Where:

- `DEFAULT-GIT-SECRET` is the value from `gitops.ssh_secret` in your `tap-values.yaml` file, or `""` to deactivate SSH authentication.
- `REGISTRY-SERVER` is the registry server from your `tap-values.yaml` file.
- `REGISTRY-REPO` is the registry repository from your `tap-values.yaml` file.
- `CERT-AS-STRING` is the value you added to `tap-values.yaml` file. Only add this if you set `ca_cert_data` in your `tap-values.yaml` file.

2. Apply the YAML file by running the command:

```
kubectl apply -f FILENAME
```

Where `FILENAME` is the name of the file you created in the previous step.

Use the `queue` workload type

The `spring-sensors-producer` workload in the getting started example using Service Toolkit claims is a good match for the `queue` workload type. This is because it runs continuously without a UI to report sensor information to a RabbitMQ topic.

If you have followed the Services Toolkit example, you can update the `spring-sensors-producer` to use the `queue` supply chain by changing the workload type by running:

```
tanzu apps workload update spring-sensors-producer --type=queue
```

This shows a diff in the workload label, and prompts you to accept the change. After the workload completes the new deployment, you'll notice 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 autoscales 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.

Use functions (Beta)

This topic tells you how to create and deploy a HTTP or CloudEvent function from an Application Accelerator starter template in an online or air-gapped environment on Tanzu Application Platform (commonly known as TAP).

Overview

The function experience on Tanzu Application Platform enables developers to deploy functions, use starter templates to bootstrap their function and write only the code that matters to your business. Developers can run a single CLI command to deploy their functions to an auto-scaled cluster.

Functions provide a quick way to get started writing an application. Compared with a traditional application:

- Functions have a single entry-point and perform a single task. This means that functions can be easier to understand and monitor.
- The initial webserver and application boilerplate are managed by the function supply chain. This means that you can update the webserver and application boilerplate without needing to update each function application.
- A traditional webserver application might be a better fit if you want to implement an entire website or API in a single container

Important: Beta features have been tested for functionality, but not performance. Features enter the beta stage so that customers can gain early access, and give feedback on the design and behavior. Beta features might undergo changes based on this feedback before the end of the beta stage. VMware discourages running beta features in production. VMware cannot guarantee that you can upgrade any beta feature in the future.

Prerequisites

Before using function workloads on Tanzu Application Platform, complete the following prerequisites:

- Follow all instructions in Installing Tanzu Application Platform.
- Download and install the kp CLI for your operating system from the Tanzu Build Service page on Tanzu Network. For more information, see the kp CLI help text on GitHub.
- Follow all instructions in Set up developer namespaces to use installed packages.

Adding function buildpacks

To use the function `buildpacks`, you must upload their buildpackages to Build Service stores.

1. Add the function's buildpackages to the default ClusterStore by running:

```
kp clusterstore add default \
-b registry.tanzu.vmware.com/python-function-buildpack-for-vmware-tanzu/python-
buildpack-with-deps:0.0.11 \
-b registry.tanzu.vmware.com/java-function-buildpack-for-vmware-tanzu/java-buil
dpack-with-deps:0.0.6
```

2. Create and save a new ClusterBuilder. Run one of the following commands depending on the dependencies you used in the `buildservice` section of your `tap-values.yaml` file:

   - For the **full dependencies**, run:

     ```
     kp clusterbuilder save function --store default -o - <<EOF
     ---
     - group:
       - id: tanzu-buildpacks/python
       - id: kn-fn/python-function
     - group:
       - id: tanzu-buildpacks/java-native-image
       - id: kn-fn/java-function
     - group:
       - id: tanzu-buildpacks/java
       - id: kn-fn/java-function

     EOF
     ```

     If you still want to use default Java and Python buildpacks for non-functions workloads, add `optional: true` flags for cluster builder groups. This does not enable the full capability of non-function workloads provided by the default ClusterBuilder. For example:

     ```
     kp clusterbuilder save function --store default -o - <<EOF
     ---
     - group:
       - id: tanzu-buildpacks/python
       - id: kn-fn/python-function
       optional: true
     - group:
       - id: tanzu-buildpacks/java-native-image
       - id: kn-fn/java-function
       optional: true
     - group:
       - id: tanzu-buildpacks/java
       - id: kn-fn/java-function
       optional: true

     EOF
     ```

   - For the **lite dependencies**, run:

     ```
     kp clusterbuilder save function --store default -o - <<EOF
     ---
     - group:
       - id: tanzu-buildpacks/python-lite
       - id: kn-fn/python-function
     - group:
       - id: tanzu-buildpacks/java-native-image-lite
       - id: kn-fn/java-function
     - group:
       - id: tanzu-buildpacks/java-lite
       - id: kn-fn/java-function

     EOF
     ```

     If you still want to use default Java and Python buildpacks for non-functions workloads, add `optional: true` flags for cluster builder groups. This does not enable the full capability of non-function workloads provided by the default ClusterBuilder. For example:

```
kp clusterbuilder save function --store default -o - <<EOF
---
- group:
  - id: tanzu-buildpacks/python-lite
  - id: kn-fn/python-function
  optional: true
- group:
  - id: tanzu-buildpacks/java-native-image-lite
  - id: kn-fn/java-function
  optional: true
- group:
  - id: tanzu-buildpacks/java-lite
  - id: kn-fn/java-function
  optional: true

EOF
```

3. After creating the ClusterBuilder, update your `tap-values.yaml` configuration to use the cluster builder you created. See the following example:

   ```
   ootb_supply_chain_basic:
    cluster_builder: function
    registry:
      server: "SERVER"
      repository: "REPO"
   ```

   Where:

   - `SERVER` is your server. For example, `index.docker.io`.
   - `REPO` is your repository.

4. Apply the update by going to the directory containing `tap-values.yaml` and running:

   ```
   tanzu package installed update tap -p tap.tanzu.vmware.com -v VERSION --values-
   file tap-values.yaml -n tap-install
   ```

   Where `VERSION` is the version of Tanzu Application Platform GUI you have installed. For example, `1.0.2`.

Add accelerators to Tanzu Application Platform GUI

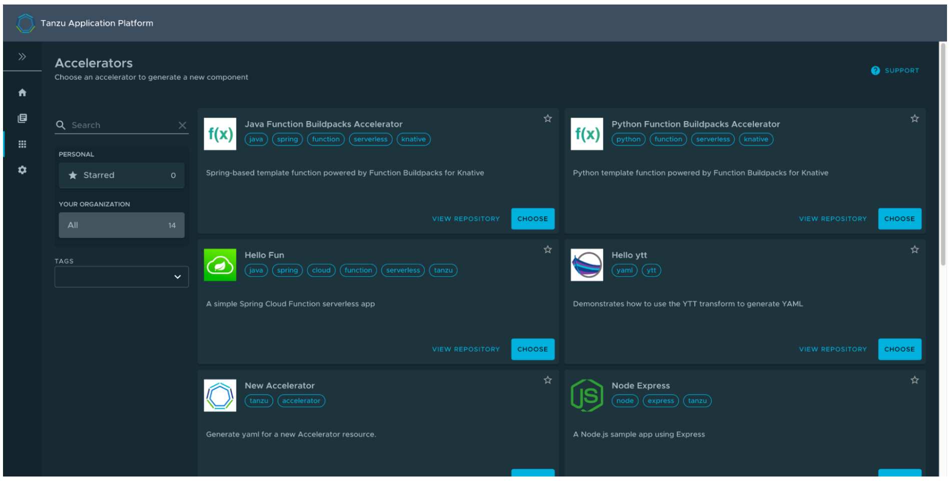
Application Accelerator is a component of Tanzu Application Platform. An accelerator contains your enterprise-conformant code and configurations that developers can use to create new projects that automatically follow the standards defined in your accelerators.

The accelerator ZIP file contains a file called k8s-resource.yaml. This file contains the resource manifest for the function accelerator.

1. Download the ZIP file for the appropriate accelerator:
   - Python HTTP Function on GitHub.
   - Java HTTP Function on GitHub.
2. Expand the accelerator ZIP file in your target cluster with Tanzu Application Platform GUI installed.
3. To update the Application Accelerator templates in Tanzu Application Platform GUI, you must apply the k8s-resource.yaml. Run the following command in your terminal in the folder where you expanded the ZIP file:

   ```
   kubectl apply -f k8s-resource.yaml --namespace accelerator-system
   ```

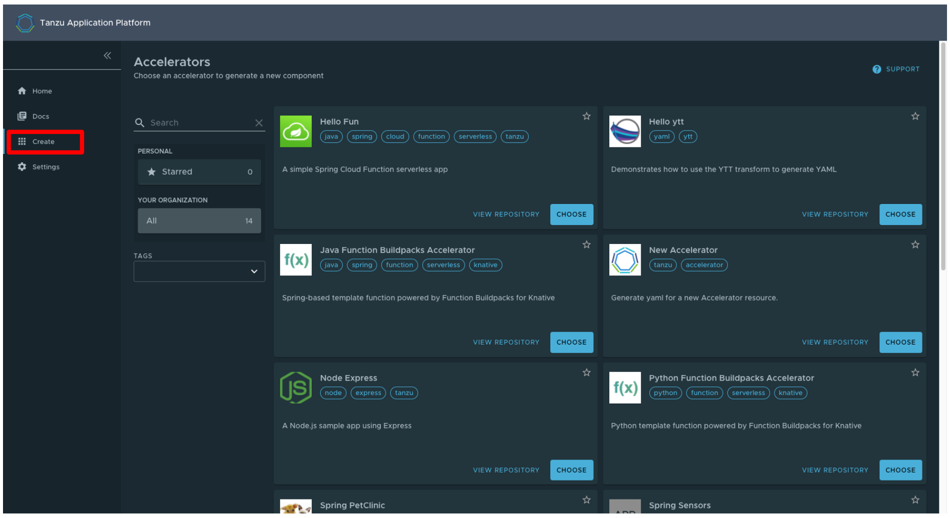
4. Refresh Tanzu Application Platform GUI to reveal function accelerator(s).


Tanzu Application Platform
Accelerators
Choose an accelerator to generate a new component
SUPPORT
Search
PERSONAL
Starred 0
YOUR ORGANIZATION
All 14
TAGS
Java Function Buildpacks Accelerator
java spring function serverless knative
Spring-based template function powered by Function Buildpacks for Knative
VIEW REPOSITORY
CHOOSE
Python Function Buildpacks Accelerator
python function serverless knative
Python template function powered by Function Buildpacks for Knative
VIEW REPOSITORY
CHOOSE
Hello Fun
java spring cloud function serverless tanzu
A simple Spring Cloud Function serverless app
VIEW REPOSITORY
CHOOSE
Hello ytt
yaml ytt
Demonstrates how to use the YTT transform to generate YAML
VIEW REPOSITORY
CHOOSE
New Accelerator
tanzu accelerator
Generate yaml for a new Accelerator resource.
Node Express
node express tanzu
A Node.js sample app using Express

It might take time for Tanzu Application Platform GUI to refresh the catalog to see your added function accelerators.

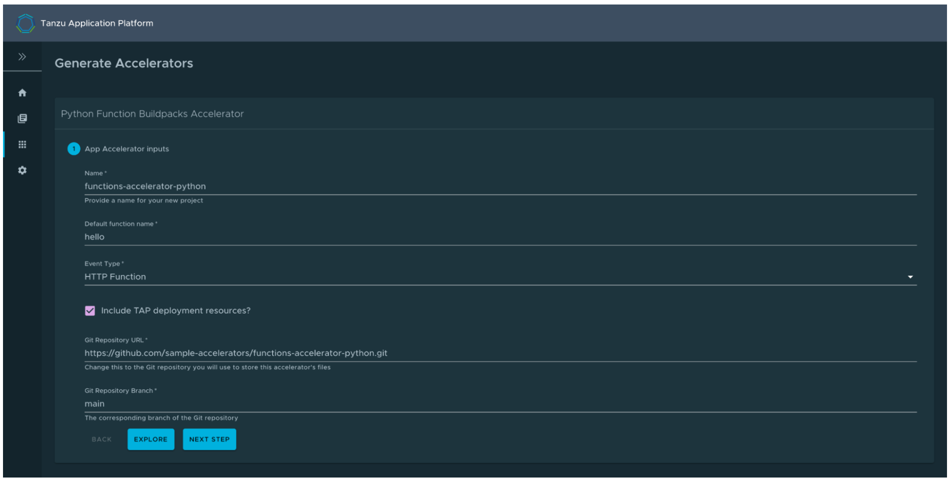
Create a function project from an accelerator

1. From the Tanzu Application Platform GUI portal, click **Create** on the left navigation bar to see the list of available accelerators.


Tanzu Application Platform
Home
Docs
Create
Settings
Accelerators
Choose an accelerator to generate a new component
SUPPORT
Search
PERSONAL
Starred 0
YOUR ORGANIZATION
All 14
TAGS
Hello Fun
java spring cloud function serverless tanzu
A simple Spring Cloud Function serverless app
VIEW REPOSITORY
CHOOSE
Hello ytt
yaml ytt
Demonstrates how to use the YTT transform to generate YAML
VIEW REPOSITORY
CHOOSE
Java Function Buildpacks Accelerator
java spring function serverless knative
Spring-based template function powered by Function Buildpacks for Knative
VIEW REPOSITORY
CHOOSE
New Accelerator
tanzu accelerator
Generate yaml for a new Accelerator resource.
VIEW REPOSITORY
CHOOSE
Node Express
node express tanzu
A Node.js sample app using Express
VIEW REPOSITORY
CHOOSE
Python Function Buildpacks Accelerator
python function serverless knative
Python template function powered by Function Buildpacks for Knative
VIEW REPOSITORY
CHOOSE
Spring PetClinic
Spring Sensors

2. Locate the Function Buildpacks accelerator and click **CHOOSE**.

3. Provide a name for your function project and function. If creating a Java function, select a project type*. Select HTTP for your event type. Provide a Git repository to store this accelerator’s files. Click **NEXT STEP**, verify the provided information, and click **CREATE**.



Python Function Buildpacks Accelerator

1 App Accelerator inputs

Name *
functions-accelerator-python
Provide a name for your new project

Default function name *
hello

Event Type *
HTTP Function

☑ Include TAP deployment resources?

Git Repository URL *
https://github.com/sample-accelerators/functions-accelerator-python.git
Change this to the Git repository you will use to store this accelerator's files

Git Repository Branch *
main
The corresponding branch of the Git repository

4. After the Task Activity processes complete, click **DOWNLOAD ZIP FILE**.
5. After downloading the ZIP file, expand it in a workspace directory and follow your preferred procedure for uploading the generated project files to a Git repository for your new project.

Create a function project using the Tanzu CLI

From the CLI, you can generate a function project using an accelerator template, then download the project artifacts as a ZIP file.

1. Validate that you have added the function accelerator template to the application accelerator server by running:

   ```
   tanzu accelerator list
   ```

2. Get the `server-url` for the Application Accelerator server. The URL depends on the configuration settings for Application Accelerator:

   - For installations configured with a shared ingress, use `https://accelerator.DOMAIN` where `DOMAIN` is provided in the values file for the accelerator configuration.
   - For installations using a LoadBalancer, look up the External IP address by running:

     ```
     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 running:

     ```
     kubectl port-forward service/acc-server -n accelerator-system 8877:80
     ```

     Use `http://localhost:8877` as the URL.

3. Generate a function project from an accelerator template by running:

   ```
   tanzu accelerator generate ACCELERATOR-NAME \
   --options '{"projectName": "FUNCTION-NAME", "interfaceType": "TYPE"}' \
   --server-url APPLICATION-ACCELERATOR-URL
   ```

   Where:

   - `ACCELERATOR-NAME` is the name of the function accelerator template you want to use.
   - `FUNCTION-NAME` is the name of your function project.
   - `TYPE` is the interface you want to use for your function. Available options are `http` or `cloudevents`. CloudEvents is experimental.
   - `APPLICATION-ACCELERATOR-URL` is the URL for the Application Accelerator server that you retrieved in the previous step.

   For example:

   ```
   tanzu accelerator generate java-function \
   --options '{"projectName": "my-func", "interfaceType": "http"}' \
   --server-url http://localhost:8877
   ```

4. After generating the ZIP file, expand it in your directory and follow your preferred procedure for uploading the generated project files to a Git repository for your new project.

Deploy your function

1. Deploy the function accelerator by running the `tanzu apps workload` create command:

   ```
   tanzu apps workload create functions-accelerator-python \
   --local-path . \
   --source-image REGISTRY/IMAGE:TAG \
   ```

```
--type web \
--yes
```

Where:

- `--source-image` is a writable repository in your registry.

Harbor has the form: “my-harbor.io/my-project/functions-accelerator-python”.

Docker Hub has the form: “my-dockerhub-user/functions-accelerator-python”.

Google Cloud Registry has the form: “gcr.io/my-project/functions-accelerator-python”.

2. View the build and runtime logs for your application by running the tail command:

   ```
   tanzu apps workload tail functions-accelerator-python --since 10m --timestamp
   ```

3. After the workload is built and running, you can view the web application in your browser. To view the URL of the web application, run the following command and then ctrl-click the Workload Knative Services URL at the bottom of the command output.

   ```
   tanzu apps workload get functions-accelerator-python
   ```

Use functions (Beta)

This topic tells you how to create and deploy a HTTP or CloudEvent function from an Application Accelerator starter template in an online or air-gapped environment on Tanzu Application Platform (commonly known as TAP).

Overview

The function experience on Tanzu Application Platform enables developers to deploy functions, use starter templates to bootstrap their function and write only the code that matters to your business. Developers can run a single CLI command to deploy their functions to an auto-scaled cluster.

Functions provide a quick way to get started writing an application. Compared with a traditional application:

- Functions have a single entry-point and perform a single task. This means that functions can be easier to understand and monitor.
- The initial webserver and application boilerplate are managed by the function supply chain. This means that you can update the webserver and application boilerplate without needing to update each function application.
- A traditional webserver application might be a better fit if you want to implement an entire website or API in a single container

Important: Beta features have been tested for functionality, but not performance. Features enter the beta stage so that customers can gain early access, and give feedback on the design and behavior. Beta features might undergo changes based on this feedback before the end of the beta stage. VMware discourages running beta features in production. VMware cannot guarantee that you can upgrade any beta feature in the future.

Prerequisites

Before using function workloads on Tanzu Application Platform, complete the following prerequisites:

- Follow all instructions in Installing Tanzu Application Platform.
- Download and install the kp CLI for your operating system from the Tanzu Build Service page on Tanzu Network. For more information, see the kp CLI help text on GitHub.
- Follow all instructions in Set up developer namespaces to use installed packages.

Adding function buildpacks

To use the function `buildpacks`, you must upload their buildpackages to Build Service stores.

1. Add the function's buildpackages to the default ClusterStore by running:

   ```
   kp clusterstore add default \
   -b registry.tanzu.vmware.com/python-function-buildpack-for-vmware-tanzu/python-buildpack-with-deps:0.0.11 \
   -b registry.tanzu.vmware.com/java-function-buildpack-for-vmware-tanzu/java-buildpack-with-deps:0.0.6
   ```

2. Create and save a new ClusterBuilder. Run one of the following commands depending on the dependencies you used in the `buildservice` section of your `tap-values.yaml` file:

   - For the **full dependencies**, run:

     ```
     kp clusterbuilder save function --store default -o - <<EOF
     ---
     - group:
       - id: tanzu-buildpacks/python
       - id: kn-fn/python-function
     - group:
       - id: tanzu-buildpacks/java-native-image
       - id: kn-fn/java-function
     - group:
       - id: tanzu-buildpacks/java
       - id: kn-fn/java-function

     EOF
     ```

     If you still want to use default Java and Python buildpacks for non-functions workloads, add `optional: true` flags for cluster builder groups. This does not enable the full capability of non-function workloads provided by the default ClusterBuilder. For example:

     ```
     kp clusterbuilder save function --store default -o - <<EOF
     ---
     - group:
       - id: tanzu-buildpacks/python
       - id: kn-fn/python-function
       optional: true
     - group:
       - id: tanzu-buildpacks/java-native-image
       - id: kn-fn/java-function
       optional: true
     - group:
       - id: tanzu-buildpacks/java
       - id: kn-fn/java-function
       optional: true

     EOF
     ```

   - For the **lite dependencies**, run:

     ```
     kp clusterbuilder save function --store default -o - <<EOF
     ---
     - group:
       - id: tanzu-buildpacks/python-lite
       - id: kn-fn/python-function
     - group:
       - id: tanzu-buildpacks/java-native-image-lite
       - id: kn-fn/java-function
     - group:
       - id: tanzu-buildpacks/java-lite
       - id: kn-fn/java-function

     EOF
     ```

If you still want to use default Java and Python buildpacks for non-functions workloads, add `optional: true` flags for cluster builder groups. This does not enable the full capability of non-function workloads provided by the default ClusterBuilder. For example:

```
kp clusterbuilder save function --store default -o - <<EOF
---
- group:
  - id: tanzu-buildpacks/python-lite
  - id: kn-fn/python-function
  optional: true
- group:
  - id: tanzu-buildpacks/java-native-image-lite
  - id: kn-fn/java-function
  optional: true
- group:
  - id: tanzu-buildpacks/java-lite
  - id: kn-fn/java-function
  optional: true

EOF
```

3. After creating the ClusterBuilder, update your `tap-values.yaml` configuration to use the cluster builder you created. See the following example:

```
ootb_supply_chain_basic:
 cluster_builder: function
 registry:
   server: "SERVER"
   repository: "REPO"
```

Where:

- `SERVER` is your server. For example, `index.docker.io`.
- `REPO` is your repository.

4. Apply the update by going to the directory containing `tap-values.yaml` and running:

```
tanzu package installed update tap -p tap.tanzu.vmware.com -v VERSION --values-file tap-values.yaml -n tap-install
```

Where `VERSION` is the version of Tanzu Application Platform GUI you have installed. For example, `1.0.2`.

Add accelerators to Tanzu Application Platform GUI

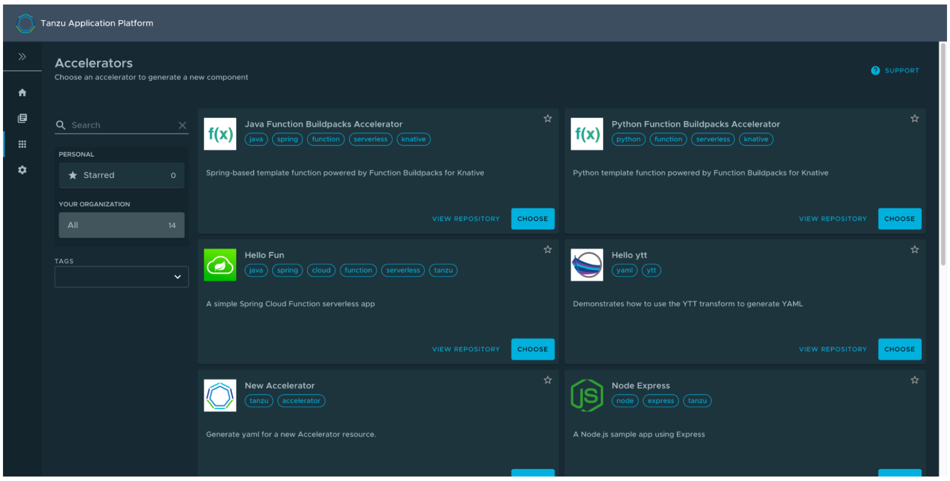
Application Accelerator is a component of Tanzu Application Platform. An accelerator contains your enterprise-conformant code and configurations that developers can use to create new projects that automatically follow the standards defined in your accelerators.

The accelerator ZIP file contains a file called k8s-resource.yaml. This file contains the resource manifest for the function accelerator.

1. Download the ZIP file for the appropriate accelerator:
   - Python HTTP Function on GitHub.
   - Java HTTP Function on GitHub.
2. Expand the accelerator ZIP file in your target cluster with Tanzu Application Platform GUI installed.
3. To update the Application Accelerator templates in Tanzu Application Platform GUI, you must apply the k8s-resource.yaml. Run the following command in your terminal in the folder where you expanded the ZIP file:

```
kubectl apply -f k8s-resource.yaml --namespace accelerator-system
```

4. Refresh Tanzu Application Platform GUI to reveal function accelerator(s).



Tanzu Application Platform
Accelerators
Choose an accelerator to generate a new component
SUPPORT
Search
PERSONAL
Starred 0
YOUR ORGANIZATION
All 14
TAGS
Java Function Buildpacks Accelerator
java spring function serverless knative
VIEW REPOSITORY
CHOOSE
Python Function Buildpacks Accelerator
python function serverless knative
Hello Fun
java spring cloud function serverless tanzu
Hello ytt
yaml ytt
New Accelerator
tanzu accelerator
Node Express
node express tanzu

Spring-based template function powered by Function Buildpacks for Knative

Python template function powered by Function Buildpacks for Knative

A simple Spring Cloud Function serverless app

Demonstrates how to use the YTT transform to generate YAML

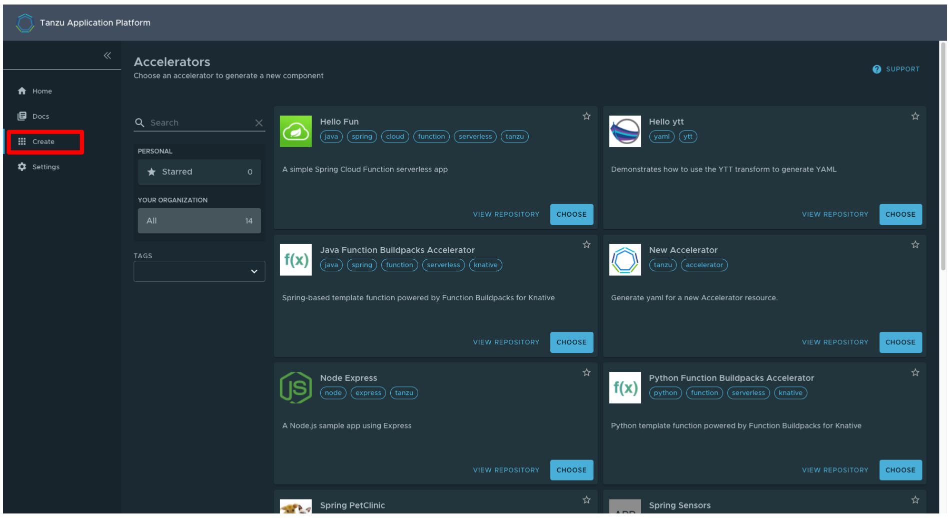
Generate yaml for a new Accelerator resource.

A Node.js sample app using Express

It might take time for Tanzu Application Platform GUI to refresh the catalog to see your added function accelerators.

Create a function project from an accelerator

1. From the Tanzu Application Platform GUI portal, click **Create** on the left navigation bar to see the list of available accelerators.



Tanzu Application Platform
Home
Docs
Create
Settings
Accelerators
Choose an accelerator to generate a new component
SUPPORT
Search
PERSONAL
Starred 0
YOUR ORGANIZATION
All 14
TAGS
Hello Fun
java spring cloud function serverless tanzu
VIEW REPOSITORY
CHOOSE
Hello ytt
yaml ytt
Java Function Buildpacks Accelerator
java spring function serverless knative
New Accelerator
tanzu accelerator
Node Express
node express tanzu
Python Function Buildpacks Accelerator
python function serverless knative
Spring PetClinic
Spring Sensors

A simple Spring Cloud Function serverless app

Demonstrates how to use the YTT transform to generate YAML

Spring-based template function powered by Function Buildpacks for Knative

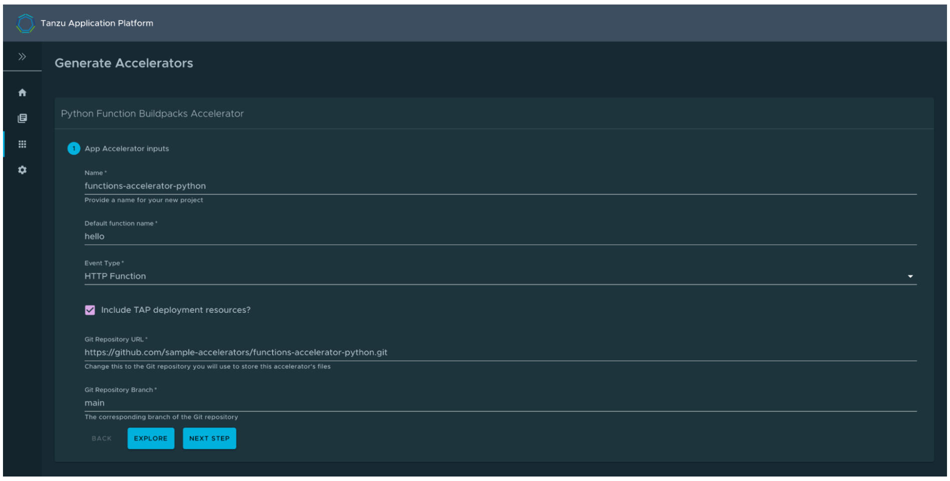
Generate yaml for a new Accelerator resource.

A Node.js sample app using Express

Python template function powered by Function Buildpacks for Knative

2. Locate the Function Buildpacks accelerator and click **CHOOSE**.

3. Provide a name for your function project and function. If creating a Java function, select a project type*. Select HTTP for your event type. Provide a Git repository to store this accelerator's files. Click **NEXT STEP**, verify the provided information, and click **CREATE**.



Generate Accelerators

Python Function Buildpacks Accelerator

1 App Accelerator inputs

Name *
functions-accelerator-python
Provide a name for your new project

Default function name *
hello

Event Type *
HTTP Function

☑ Include TAP deployment resources?

Git Repository URL *
https://github.com/sample-accelerators/functions-accelerator-python.git
Change this to the Git repository you will use to store this accelerator's files

Git Repository Branch *
main
The corresponding branch of the Git repository

4. After the Task Activity processes complete, click **DOWNLOAD ZIP FILE**.

5. After downloading the ZIP file, expand it in a workspace directory and follow your preferred procedure for uploading the generated project files to a Git repository for your new project.

Create a function project using the Tanzu CLI

From the CLI, you can generate a function project using an accelerator template, then download the project artifacts as a ZIP file.

1. Validate that you have added the function accelerator template to the application accelerator server by running:

```
tanzu accelerator list
```

2. Get the `server-url` for the Application Accelerator server. The URL depends on the configuration settings for Application Accelerator:

   - For installations configured with a shared ingress, use `https://accelerator.DOMAIN` where `DOMAIN` is provided in the values file for the accelerator configuration.

   - For installations using a LoadBalancer, look up the External IP address by running:

     ```
     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 running:

     ```
     kubectl port-forward service/acc-server -n accelerator-system 8877:80
     ```

     Use `http://localhost:8877` as the URL.

3. Generate a function project from an accelerator template by running:

```
tanzu accelerator generate ACCELERATOR-NAME \
--options '{"projectName": "FUNCTION-NAME", "interfaceType": "TYPE"}' \
--server-url APPLICATION-ACCELERATOR-URL
```

   Where:

   - `ACCELERATOR-NAME` is the name of the function accelerator template you want to use.

   - `FUNCTION-NAME` is the name of your function project.

   - `TYPE` is the interface you want to use for your function. Available options are `http` or `cloudevents`. CloudEvents is experimental.

   - `APPLICATION-ACCELERATOR-URL` is the URL for the Application Accelerator server that you retrieved in the previous step.

   For example:

```
tanzu accelerator generate java-function \
--options '{"projectName": "my-func", "interfaceType": "http"}' \
--server-url http://localhost:8877
```

4. After generating the ZIP file, expand it in your directory and follow your preferred procedure for uploading the generated project files to a Git repository for your new project.

Deploy your function

1. Deploy the function accelerator by running the `tanzu apps workload` create command:

   ```
   tanzu apps workload create functions-accelerator-python \
   --local-path . \
   --source-image REGISTRY/IMAGE:TAG \
   --type web \
   --yes
   ```

   Where:

   - `--source-image` is a writable repository in your registry.

   Harbor has the form: “my-harbor.io/my-project/functions-accelerator-python”.

   Docker Hub has the form: “my-dockerhub-user/functions-accelerator-python”.

   Google Cloud Registry has the form: “gcr.io/my-project/functions-accelerator-python”.

2. View the build and runtime logs for your application by running the tail command:

   ```
   tanzu apps workload tail functions-accelerator-python --since 10m --timestamp
   ```

3. After the workload is built and running, you can view the web application in your browser. To view the URL of the web application, run the following command and then ctrl-click the Workload Knative Services URL at the bottom of the command output.

   ```
   tanzu apps workload get functions-accelerator-python
   ```