

Tanzu Application Platform v1.1

VMware Tanzu Application Platform 1.1

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

Copyright © 2023 VMware, Inc. All rights reserved. [Copyright and trademark information.](#)

Contents

Tanzu Application Platform v1.1	49
Overview of Tanzu Application Platform	49
Installation profiles in Tanzu Application Platform v1.1	54
About Tanzu Application Platform package profiles	54
About installing the Tanzu Application Platform v1.1	55
Notice of telemetry collection for Tanzu Application Platform	55
Release notes	57
v1.1.2	57
Security fixes	57
Tanzu Application Platform GUI	57
Resolved issues	57
Application Live View	57
Grype scanner	57
Supply Chain Security Tools - Scan	57
Tanzu Application Platform GUI	58
Known issues	58
Grype scanner	58
Supply Chain Security Tools - Scan	58
Supply Chain Security Tools - Sign	59
v1.1.1	59
Resolved issues	59
Supply Chain Choreographer plug-in	59
Supply Chain Security Tools - Scan	59
Supply Chain Security Tools - Sign	59
Supply Chain Security Tools - Store	59
Grype Scanner	59
Tanzu Application Platform GUI	60
Known issues	60
Grype scanner	60
Supply Chain Choreographer for Tanzu	60
Supply Chain Security Tools - Scan	60
Supply Chain Security Tools - Store	60
Tanzu Application Platform GUI	60

v1.1	61
Prerequisites	61
New features	61
Installing	61
Default roles for Tanzu Application Platform	61
Application Accelerator	62
Application Live View	62
Tanzu CLI - Apps plug-in	62
Service Bindings	62
Source Controller	63
Spring Boot Conventions	63
Supply Chain Choreographer	63
Supply Chain Security Tools - Scan	63
Supply Chain Security Tools - Sign	64
Supply Chain Security Tools - Store	64
Tanzu Application Platform GUI	64
Functions (Beta)	64
Breaking changes	65
Application Accelerator	65
Supply Chain Security Tools - Scan	65
Supply Chain Security Tools - Store	65
Resolved issues	65
Application Accelerator	65
Application Live View	65
Services Toolkit	65
Supply Chain Security Tools - Scan	65
Grype Scanner	66
Supply Chain Security Tools - Store	66
Tanzu CLI - Apps plug-in	66
Tanzu Application Platform GUI	66
Known issues	66
Tanzu Application Platform	67
Tanzu Cluster Essentials	67
Application Live View	67
Grype scanner	67
Supply Chain Choreographer plug-in	67
Supply Chain Security Tools - Scan	68
Supply Chain Security Tools - Store	68
Tanzu Application Platform GUI	69

Installing Tanzu Application Platform	70
Installation process	70
Prerequisites	70
VMware Tanzu Network and container image registry requirements	70
DNS Records	71
Tanzu Application Platform GUI	71
Kubernetes cluster requirements	72
Resource requirements	73
Tools and CLI requirements	73
Accepting Tanzu Application Platform EULAs, installing Cluster Essentials and the Tanzu CLI	73
Accept the End User License Agreements	74
Example of accepting the Tanzu Application Platform EULA	74
Set the Kubernetes cluster context	76
Install Cluster Essentials for Tanzu	77
Install or update the Tanzu CLI and plug-ins	77
Install Tanzu CLI: Linux or macOS	77
Install Tanzu CLI: Windows	78
Install/Update Tanzu CLI plug-ins	79
Installing the Tanzu Application Platform package and profiles	80
Relocate images to a registry	80
Install your Tanzu Application Platform profile	83
(Optional) Configure LoadBalancer for Contour ingress	84
Full profile	85
Light Profile	87
View possible configuration settings for your package	87
Identify the values for your package	88
Install your Tanzu Application Platform package	90
Access Tanzu Application Platform GUI	91
Exclude packages from a Tanzu Application Platform profile	91
Opting out of telemetry collection	91
Turn off telemetry collection	91
Upgrading Tanzu Application Platform	93
Prerequisites	93
Add new package repository	93
Perform upgrade of Tanzu Application Platform	94

Upgrade instructions for Profile-based installation	94
Upgrade instructions for component-specific installation	94
Verify the upgrade	94
Migrate Tanzu Application Platform profiles	97
Prerequisites	97
Add new package repository	97
Edit the tap-values.yaml configuration file that was used during installation	97
Perform migration of Tanzu Application Platform profile	98
Getting started with the Tanzu Application Platform	99
Purpose	99
Getting started prerequisites	99
Section 1: Develop your first application on the Tanzu Application Platform	100
About application accelerators	100
Deploy your application	100
Add your application to Tanzu Application Platform GUI Software Catalog	102
Iterate on your application	104
Live update your application	104
Debug your application	105
Monitor your running application	105
Section 2: Create your application accelerator	105
Create an application accelerator	106
Publish the new accelerator	106
Working with accelerators	107
Updating an accelerator	107
Deleting an accelerator	107
Using an accelerator manifest	107
Section 3: Add Testing and Security Scanning to Your Application	108
Introducing a Supply Chain	108
A path to production	108
Available Supply Chains	109
1: OOTB Basic (default)	109
2: OOTB Testing	110
3: OOTB Testing+Scanning	111
Install OOTB Testing	111
Tekton pipeline config example	112
Workload update	113
Install OOTB Testing+Scanning	114

Workload update	116
Query for vulnerabilities	117
Congratulations! You have successfully deployed your application on the Tanzu Application Platform.	118
Section 4: Configure image signing and verification in your supply chain	118
Configure your supply chain to sign your image builds	118
Next steps	119
Scan and Store: Introducing vulnerability scanning and metadata storage to your Supply Chain	119
Next steps	120
Section 5: Consuming services on Tanzu Application Platform	120
Key concepts	120
Service instances	120
Service bindings	120
Resource claims	120
Services you can use with Tanzu Application Platform	121
User roles and responsibilities	121
Walkthrough	122
Prerequisites	123
Set up a service	123
Create a service instance	125
Claim a service instance	126
Bind an application workload to the service instance	127
Advanced use cases and further reading	129
Overview of multicluster Tanzu Application Platform	130
Next steps	130
Install multicluster Tanzu Application Platform profiles	131
Prerequisites	131
Multicluster Installation Order of Operations	131
Install View cluster	131
Install Build clusters	132
Install Run clusters	132
Add Build and Run clusters to Tanzu Application Platform GUI	132
Next steps	132
Getting started with multicluster Tanzu Application Platform	132
Prerequisites	133
Start the workload on the Build profile cluster	133

Build profile	135
Run profile	137
View profile	138
Troubleshooting Tanzu Application Platform	140
Troubleshoot installing Tanzu Application Platform	140
Developer cannot be verified when installing Tanzu CLI on macOS	140
Access .status.usefulErrorMessage details	140
“Unauthorized to access” error	141
“Serviceaccounts already exists” error	142
After package installation, one or more packages fails to reconcile	142
Failure to accept an End User License Agreement error	146
Troubleshoot using Tanzu Application Platform	146
Missing build logs after creating a workload	146
“Workload already exists” error after updating the workload	147
Workload creation fails due to authentication failure in Docker Registry	147
Explanation	148
Solution	148
Telemetry component logs show errors fetching the “reg-creds” secret	148
Debug convention may not apply	148
Execute bit not set for App Accelerator build scripts	149
“No live information for pod with ID” error	149
“image-policy-webhook-service not found” error	149
“Increase your cluster resources” error	150
MutatingWebhookConfiguration prevents pod admission	150
Priority class of webhook’s pods preempts less privileged pods	151
CrashLoopBackOff from password authentication fails	152
Password authentication fails	153
metadata-store-db pod fails to start	153
Missing persistent volume	154
Supply Chain Security Tools - Sign rejects images	154
Supply Chain Security Tools - Scan unable to decode CycloneDX	155
Troubleshoot Tanzu Application Platform components	155
Uninstalling Tanzu Application Platform	156
Delete the packages	156

Delete the Tanzu Application Platform package repository	156
Remove Tanzu CLI, plug-ins, and associated files	157
Component documentation	158
Installing individual packages	158
Install pages for individual Tanzu Application Platform packages	158
Verify the installed packages	159
Set up developer namespaces to use installed packages	160
Enable single user access	160
Enable additional users access with Kubernetes RBAC	162
Tanzu CLI	164
Tanzu CLI plug-ins	164
Apps CLI plug-in overview	164
About workloads	164
Command reference	165
Usage and examples	165
Install Apps CLI plug-in	165
Prerequisites	165
Install	165
Create a workload	165
Prerequisites	166
Get started with an example workload	166
Check build logs	166
Get the workload status and details	167
Create a workload from local source code	167
Bind a service to a workload	167
Next steps	168
Command reference	168
Tanzu apps	168
Options	168
See also	169
Tanzu apps workload	169
Options	169

Options inherited from parent commands	169
See also	169
Tanzu apps workload apply	170
Synopsis	170
Examples	170
Options	170
Options inherited from parent commands	171
See also	171
Tanzu apps workload create	171
Synopsis	171
Examples	172
Options	172
Options inherited from parent commands	173
See also	173
Tanzu apps workload update	173
Synopsis	173
Examples	173
Options	174
Options inherited from parent commands	175
See also	175
Tanzu apps workload get	175
Examples	175
Options	175
Options inherited from parent commands	175
See also	175
Tanzu apps workload delete	175
Examples	176
Options	176
Options inherited from parent commands	176
See also	176
Tanzu apps workload list	176
Examples	176
Options	176
Options inherited from parent commands	177
See also	177

Tanzu apps workload tail	177
Examples	177
Options	177
Options inherited from parent commands	177
See also	177
Tanzu apps cluster supply chain	178
Options	178
Options inherited from parent commands	178
See also	178
Tanzu apps cluster supply chain list	178
Examples	178
Options	178
Options inherited from parent commands	178
See also	179
Usage and examples	179
Changing clusters	179
Checking update status	179
Working with YAML files	179
Autocompletion	180
Bash	180
Zsh	180
Tanzu Insight plug-in overview	180
Install the Tanzu Insight CLI plug-in	181
Configure the Tanzu Insight CLI plug-in	181
Set the target and certificate authority certificate	181
Check the connection	182
Configure target endpoint and certificate	182
Use Ingress	182
Not use Ingress	183
Use LoadBalancer	183
Use NodePort	183
Configure port forwarding	183
Modify your /etc/hosts file	184

Configure access tokens	184
Service accounts	184
Read-only service account	184
Read-write service account	185
Getting the Access Token	186
Setting the Access Token	186
Query data	187
Supported use cases	187
Query using the Tanzu Insight CLI plug-in	187
Example #1: What packages & CVEs does a specific image contain?	187
Example #2: What packages & CVEs does my source code contain?	188
Example #3: What dependencies are affected by a specific CVE?	188
Add data	189
Add data	189
Supported formats and file types	189
Generate a CycloneDX file	189
Add data with the Tanzu Insight plug-in	190
Example #1: Add an image report	190
Example #2: Add a source report	190
Command reference	191
Synopsis	191
Options	191
See also	191
Tanzu insight config set-target	191
Tanzu insight config set-target	191
Synopsis	191
Examples	192
Options	192
See also	192
Tanzu insight config	192
Options	192
See also	192
Tanzu insight health	192
Tanzu insight health	192
Synopsis	192

Examples	193
Options	193
See also	193
Tanzu insight image	193
Options	193
See also	193
Tanzu insight image add	193
Examples	193
Options	193
See also	194
Tanzu insight image get	194
Synopsis	194
Examples	194
Options	194
See Also	194
Tanzu insight image packages	194
Synopsis	194
Examples	194
Options	195
See also	195
Tanzu insight image vulnerabilities	195
Examples	195
Options	195
See also	195
Tanzu insight package	195
Options	195
See also	195
Tanzu insight package get	196
Synopsis	196
Examples	196
Options	196
See also	196
Tanzu insight Package Images	196

Synopsis	196
Examples	196
Options	197
See also	197
Tanzu insight package sources	197
Synopsis	197
Examples	197
Options	197
See also	197
Tanzu insight Package Vulnerabilities	197
Synopsis	197
Examples	198
Options	198
See also	198
Tanzu insight source	198
Options	198
See also	198
Tanzu insight source add	198
Examples	198
Options	198
See also	199
Tanzu insight source get	199
Synopsis	199
Examples	199
Options	199
See also	199
Tanzu insight source packages	199
Synopsis	199
Examples	200
Options	200
See also	200
Tanzu insight source vulnerabilities	200
Synopsis	200
Examples	200

Options	200
See also	200
Tanzu insight version	200
Options	201
See also	201
Tanzu insight vulnerabilities	201
Options	201
See also	201
Tanzu insight vulnerabilities get	201
Synopsis	201
Examples	201
Options	201
See also	202
Tanzu insight vulnerabilities images	202
Synopsis	202
Examples	202
Options	202
See also	202
Tanzu insight vulnerabilities packages	202
Synopsis	202
Examples	202
Options	203
See also	203
Tanzu insight vulnerabilities sources	203
Synopsis	203
Examples	203
Options	203
See also	203
Overview	203
Default roles	204
Working with roles using the RBAC CLI plug-in	204
Disclaimer	204
Setting up authentication for Tanzu Application Platform	204
Tanzu Kubernetes Grid	204

Installing Pinniped on a single cluster	205
Prerequisites	205
Install Pinniped Supervisor	205
Create Certificates (letsencrypt/cert-manager)	205
Create Ingress resources	206
Create Pinniped-Supervisor configuration	207
Apply the resources	207
Install Pinniped Concierge	208
Log in to the cluster	209
Integrating Azure Active Directory	209
Integrate Azure AD with a new or existing AKS without Pinniped	209
Prerequisites	209
Set up a platform operator	209
Set up a Tanzu Application Platform default role group	210
Set up kubeconfig	210
Integrate Azure AD with Pinniped	211
Prerequisites	211
Set up the Azure AD app	211
Set up the Tanzu Application Platform default role group	212
Set up kubeconfig	213
Role descriptions	213
app-editor	213
app-viewer	214
app-operator	214
workload	214
deliverable	214
Detailed role permissions breakdown	214
Native Kubernetes Resources	214
apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"	214
apps.tanzu.vmware.com/aggregate-to-app-operator: "true"	215
App Accelerator	215
apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"	215
apps.tanzu.vmware.com/aggregate-to-app-operator: "true"	215
Cartographer	215
apps.tanzu.vmware.com/aggregate-to-app-editor: "true"	215
apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"	215

apps.tanzu.vmware.com/aggregate-to-app-viewer-cluster-access: "true"	216
apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access	216
Cloud Native Runtimes	216
apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"	216
apps.tanzu.vmware.com/aggregate-to-app-operator: "true"	216
Convention Service	216
apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"	216
apps.tanzu.vmware.com/aggregate-to-app-viewer-cluster-access: "true"	217
apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access	217
Developer Conventions	217
apps.tanzu.vmware.com/aggregate-to-app-editor: "true"	217
OOTB Templates	217
apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"	217
apps.tanzu.vmware.com/aggregate-to-workload: "true"	218
apps.tanzu.vmware.com/aggregate-to-deliverable: "true"	218
Service Bindings	219
apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"	219
Services Toolkit	219
apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"	219
apps.tanzu.vmware.com/aggregate-to-app-viewer-cluster-access: "true"	219
apps.tanzu.vmware.com/aggregate-to-app-operator: "true"	219
apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access	219
Source Controller	219
apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"	220
Supply Chain Security Tools — Store	220
apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"	220
apps.tanzu.vmware.com/aggregate-to-app-operator: "true"	220
Tanzu Build Service	220
apps.tanzu.vmware.com/aggregate-to-app-editor: "true"	220
apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"	220
apps.tanzu.vmware.com/aggregate-to-app-viewer-cluster-access: "true"	220
apps.tanzu.vmware.com/aggregate-to-app-operator: "true"	220
apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access	220
Tekton	221
apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"	221
apps.tanzu.vmware.com/aggregate-to-app-viewer-cluster-access: "true"	221
apps.tanzu.vmware.com/aggregate-to-app-operator: "true"	221
apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access	221

Bind a user or group to a default role	221
Prerequisites	221
Install the Tanzu Application Platform RBAC CLI plug-in	222
Use a different kubeconfig location	222
Add the specified user or group to a role	222
Get a list of users and groups from a role	223
Remove the specified user or group from a role	223
Error logs	223
Troubleshooting	225
Login using Pinniped	225
Generate and distribute kubeconfig to users	225
Login with provided kubeconfig	225
Additional resources	226
Install	226
Install default roles independently	226
Prerequisites	226
Install	226
Application Accelerator for VMware Tanzu	227
Install Application Accelerator	227
Prerequisites	227
Configure properties and resource usage	228
Install	229
Application Live View for VMware Tanzu	230
Install Application Live View	231
Prerequisites	231
Install Application Live View	231
Install Application Live View Backend	231
Install Application Live View Connector	233
Install Application Live View Conventions	235
Convention Service	236
Overview	236
About applying conventions	237
Applying conventions by using image metadata	237

Applying conventions without using image metadata	237
Install Convention Service	237
Prerequisites	238
Install	238
Creating conventions	240
Introduction	240
Convention server	240
Convention controller	241
Getting started	241
Prerequisites	241
Define convention criteria	242
Define the convention behavior	245
Matching criteria by labels or annotations	245
Matching criteria by environment variables	246
Matching criteria by image metadata	246
Configure and install the convention server	247
Deploy a convention server	249
Next Steps	252
Troubleshoot Convention Service	252
No server in the cluster	252
Symptoms	252
Cause	252
Solution	252
Server with wrong certificates configured	253
Symptoms	253
Cause	253
Solution	253
Server fails when processing a request	253
Symptoms	253
Cause	254
Solution	254
Connection refused due to unsecured connection	255
Symptoms	255
Cause	256
Solution	256
Convention Resources	256

Convention Service Resources	256
API Structure	256
Template Status	257
Chaining Multiple Conventions	257
Collecting Logs from the Controller	257
References	257
ImageConfig	258
PodConventionContextSpec	259
PodConventionContextStatus	260
PodConventionContext	260
PodConventionContext Structure	261
ClusterPodConvention	262
PodIntent	262
BOM	262
cert-manager, Contour, and FluxCD Source Controller	263
Install cert-manager, Contour, and FluxCD Source Controller	263
Prerequisites	263
Install cert-manager	264
Install Contour	265
Install FluxCD source-controller	269
Cloud Native Runtimes	271
Install Cloud Native Runtimes	271
Prerequisites	271
Install	271
Spring Boot conventions	274
Overview	274
Install Spring Boot conventions	275
Prerequisites	275
Install Spring Boot conventions	276

Conventions	276
Set a JAVA_TOOL_OPTIONS property for a workload	277
Spring Boot convention	278
Spring boot graceful shut down convention	279
Spring Boot web convention	280
Spring Boot Actuator convention	281
Spring Boot Actuator Probes convention	282
Service intent conventions	283
Example	284
Troubleshoot Spring Boot Conventions	286
Collect logs	286
Service Bindings for Kubernetes	286
Install Service Bindings	287
Prerequisites	287
Install Service Bindings	287
Troubleshoot Service Bindings	288
Collect logs	288
Resources	290
ServiceBinding (servicebinding.io/v1alpha3)	290
Services Toolkit	291
Install Services Toolkit	291
Prerequisites	291
Install Services Toolkit	291
Source Controller	292
Install Source Controller	292
Prerequisites	292
Install	293
Troubleshoot Source Controller	295
Collecting Logs from Source Controller Manager	295
Source Controller Reference	295
ImageRepository	295

Developer Conventions for Tanzu Application Platform	296
Overview	296
Features	296
Enabling Live Updates	296
Enabling debugging	297
Next steps	297
Install Developer Conventions	297
Prerequisites	298
Install	298
Resource limits	299
Uninstall	299
Learning Center for Tanzu Application Platform	299
Overview	299
Use cases	299
Use case requirements	300
Platform architectural overview	301
Next steps	302
Install Learning Center	302
Prerequisites	302
Install	303
Procedure to install the Self-Guided Tour Training Portal and Workshop	305
Supported Learning Center Values Configuration	305
Learning Center workshops	307
Getting started with Learning Center	309
Learning Center operator	309
Installing and setting up Learning Center operator	309
Cluster pod security policies	310
Specifying the ingress domain	310
Set the environment variable manually	311
Enforcing secure connections	311
Configuration YAML	312
Create the TLS secret manually	312
Specifying the ingress class	312
Configuration YAML	313

Set the environment variable manually	313
Trusting unsecured registries	313
Deleting Learning Center	314
Learning Center Workshops	314
Creating the workshop environment	315
Requesting a workshop instance	315
Deleting the workshop instance	316
Deleting the workshop environment	317
TrainingPortal	317
Working with multiple workshops	317
Loading the workshop definition	317
Creating the workshop training portal	318
Accessing workshops via the web portal	320
Deleting the workshop training portal	322
Learning Center local install guides	322
Installing on Kind	322
Prerequisites	323
Kind cluster creation	323
Ingress controller with DNS	323
Install carvel tools	324
Install Tanzu package repository	324
Create a configuration YAML file for Learning Center package	325
Using a nip.io DNS address	326
Install Learning Center package onto a Kubernetes cluster	327
Install workshop tutorial package onto a Kubernetes cluster	327
Run the workshop	327
Trusting insecure registries	327
Installing on Minikube	329
Trusting insecure registries	329
Prerequisites	330
Ingress controller with DNS	330
Install carvel tools	330
Install Tanzu package repository	331
Create a configuration YAML file for the Learning Center package	331
Using a nip.io DNS address	333

Install Learning Center package onto a minikube cluster	333
Install workshop tutorial package onto a minikube cluster	333
Run the workshop	334
Working with large images	334
Limited resource availability	334
Storage provisioner issue	334
Creating Learning Center workshops	335
Workshop configuration	335
Specifying structure of the content	335
Specifying the runtime configuration	337
Next steps	338
Workshop images	338
Templates for creating a workshop	338
Workshop content directory layout	339
Directory for workshop exercises	339
Workshop content	340
Deactivating reserved sessions	340
Live updates to the content	340
Custom workshop image changes	341
Custom workshop image overlay	342
Changes to workshop definition	343
Local build of workshop image	343
Building an image	344
Structure of the Dockerfile	344
Base images and version tags	344
Custom workshop base images	345
Installing extra system packages	345
Installing third-party packages	346
Workshop instructions	346
Annotation of executable commands	346
Annotation of text to be copied	347
Extensible clickable actions	348
Clickable actions for the dashboard	350
Clickable actions for the editor	352
Clickable actions for file download	354

Clickable actions for the examiner	354
Clickable actions for sections	356
Overriding title and description	357
Escaping of code block content	358
Interpolation of data variables	358
Adding custom data variables	359
Passing environment variables	360
Handling embedded URL links	360
Conditional rendering of content	361
Embedding custom HTML content	361
Workshop runtime	362
Predefined environment variables	362
Running steps on container start	363
Running background applications	363
Terminal user shell environment	364
Overriding terminal shell command	364
Presenter slides	365
Using reveal.js presentation tool	365
Using a PDF file for presenter slides	365
Learning Center runtime environment	365
Custom resources	366
Workshop definition resource	366
Workshop environment resource	367
Workshop request resource	367
Workshop session resource	368
Training portal resource	368
System profile resource	368
Loading the workshop CRDs	369
Workshop resource	369
Workshop title and description	369
Downloading workshop content	371
Container image for the workshop	373
Setting environment variables	375
Overriding the memory available	375
Mounting a persistent volume	376

Resource budget for namespaces	376
Patching workshop deployment	379
Creation of session resources	380
Overriding default role-based access control (RBAC) rules	382
Running user containers as root	383
Creating additional namespaces	384
Shared workshop resources	387
Workshop pod security policy	388
Custom security policies for user containers	390
Defining additional ingress points	391
External workshop instructions	393
Disabling workshop instructions	395
Enabling the Kubernetes console	395
Enabling the integrated editor	396
Enabling workshop downloads	396
Enabling the test examiner	397
Enabling session image registry	398
Enabling ability to use Docker	400
Enabling WebDAV access to files	401
Customizing the terminal layout	402
Adding custom dashboard tabs	403
WorkshopEnvironment resource	404
Specifying the workshop definition	404
Overriding environment variables	404
Overriding the ingress domain	405
Controlling access to the workshop	407
Overriding the login credentials	408
Additional workshop resources	408
Creation of workshop instances	409
WorkshopRequest resource	410
Specifying workshop environment	410
Specifying required access token	411
TrainingPortal resource	411
Specifying the workshop definitions	411
Limit the number of sessions	412
Capacity of individual workshops	412
Set reserved workshop instances	413

Override initial number of sessions	414
Setting defaults for all workshops	414
Set caps on individual users	415
Expiration of workshop sessions	415
Updates to workshop environments	417
Override the ingress domain	417
Override the portal host name	419
Set extra environment variables	419
Override portal credentials	420
Control registration type	421
Specify an event access code	422
Make a list of workshops public	423
Use an external list of workshops	423
Override portal title and logo	424
Allow the portal in an iframe	424
Collect analytics on workshops	425
Track using Google Analytics	426
SystemProfile resource	427
Operator default system profile	427
Defining configuration for ingress	428
Defining container image registry pull secrets	428
Defining storage class for volumes	429
Defining storage group for volumes	429
Restricting network access	431
Running Docker daemon rootless	431
Overriding network packet size	432
Image registry pull through cache	432
Setting default access credentials	434
Overriding the workshop images	434
Tracking using Google Analytics	435
Overriding styling of the workshop	436
Additional custom system profiles	437
WorkshopSession resource	437
Specifying the session identity	437
Specifying the login credentials	438
Specifying the ingress domain	438
Setting the environment variables	440

Learning Center Portal Rest API	441
Anonymous access	441
Enabling anonymous access	441
Triggering workshop creation	441
Workshop catalog	442
Listing available workshops	442
Session management	444
Disabling portal user registration	444
Requesting a workshop session	445
Associating sessions with a user	446
Listing all workshop sessions	447
Client authentication	448
Querying the credentials	449
Requesting an access token	449
Refreshing the access token	450
Troubleshoot Learning Center	450
Training portal stays in pending state	450
image-policy-webhook-service not found	451
Cannot update parameters	451
Increase your cluster's resources	451
Supply Chain Choreographer for Tanzu	452
Overview	452
Out of the Box Supply Chains	452
Install Supply Chain Choreographer	452
Prerequisites	453
Install	453
Out of the Box Delivery Basic	453
Prerequisites	453
Usage	453
Install Out of the Box Delivery Basic	454
Prerequisites	454
Install	455

Out of the Box Supply Chain Basic	455
Prerequisites	456
Developer Namespace	456
Registries Secrets	456
ServiceAccount	457
RoleBinding	458
Developer workload	459
Install Out of the Box Supply Chain Basic	459
Prerequisites	459
Install	460
Out of the Box Supply Chain with Testing	462
Prerequisites	462
Developer Namespace	463
Updates to the developer Namespace	463
Tekton/Pipeline	463
Allow multiple Tekton pipelines in a namespace	464
Developer Workload	465
Install Out of the Box Supply Chain with Testing	466
Prerequisites	466
Install	467
Out of the Box Supply Chain with Testing and Scanning	469
Prerequisites	470
Developer Namespace	470
Updates to the developer Namespace	471
ScanPolicy	471
ScanTemplate	472
Allow multiple Tekton pipelines in a namespace	473
Developer workload	474
Install Out of the Box Supply Chain with Testing and Scanning	475
Prerequisites	475
Install	475
Out of the Box Templates	478
Install Out of the Box Templates	478

Prerequisites	479
Install	479
Building from source	480
Git source	480
Private GitRepository	481
HTTP(S) Basic-auth / Token-based authentication	482
SSH auth	483
How it works	484
Workload parameters	484
Local source	485
Authentication	486
Developer	486
Supply chain components	486
How it works	486
Using a prebuilt image	487
Requirements for prebuilt images	487
Configure your workload to use a prebuilt image	488
Examples	489
Using a Dockerfile	489
Using Spring Boot's build-image Maven target	490
About Out of the Box Supply Chains	491
Understanding the supply chain for a prebuilt image	492
Git authentication	493
HTTP	494
SSH	495
GitOps vs. RegistryOps	496
GitOps	497
Authentication	498
HTTP(S) Basic-auth / Token-based authentication	498
SSH	499
GitOps workload parameters	499
RegistryOps	501
Authoring supply chains	501
Providing your own supply chain	501
Providing your own templates	503
Modifying an Out of the Box Supply Chain	504

Example	504
Modifying an Out of the Box Supply template	505
Example	506
Live modification of supply chains and templates	507
Supply Chain Security Tools - Scan	508
Overview	508
Use cases	508
Supply Chain Security Tools - Scan features	509
A Note on Vulnerability Scanners	509
Missed CVEs	509
False positives	509
Install Supply Chain Security Tools - Scan	510
Prerequisites	511
Scanner support	511
Install	511
Spec reference	514
About source and image scans	514
About policy enforcement around vulnerabilities found	515
Scan samples	515
Sample public image scan with compliance check	515
Public image scan	516
Define the ScanPolicy and ImageScan	516
(Optional) Set up a watch	517
Deploy the resources	517
View the scan results	517
Edit the ScanPolicy	517
Clean up	517
Sample public source code scan with compliance check	518
Public source scan	518
Run an example public source scan	518
Sample private image scan	520
Define the resources	520
Set up target image pull secret	520
Create the private image scan	521

(Optional) Set up a watch	521
Deploy the resources	522
View the scan results	522
Clean up	522
View vulnerability reports	522
Sample private source scan	522
Define the resources	522
(Optional) Set up a watch	524
Deploy the resources	524
View the scan status	524
Clean up	525
View vulnerability reports	525
Sample public source scan of a blob	525
Define the resources	525
(Optional) Set up a watch	525
Deploy the resources	525
View the scan results	526
Clean up	526
View vulnerability reports	526
Observe Supply Chain Security Tools - Scan	526
Watching in-flight jobs	526
Troubleshooting Supply Chain Security Tools - Scan	526
Missing target image pull secret	527
Disable Supply Chain Security Tools - Store	527
Resolving Incompatible Syft Schema Version	527
Resolving “Unable to decode cyclonedx”	528
Blob Source Scan is reporting wrong source URL	528
Additional resources	529
Configure code repositories and image artifacts to be scanned	529
Prerequisite	529
Deploy scan custom resources	529
SourceScan	529
ImageScan	531
Enforce compliance policy using Open Policy Agent	532

Writing a policy template	532
Rego file contract	533
Define a Rego file for policy enforcement	533
Create a ScanTemplate	534
Structure	534
Pod requirements	534
Best practices	535
View scan status conditions	535
Viewing scan status	535
Understanding conditions	535
Condition types for the scans	535
Scanning	535
Succeeded	535
SendingResults	536
PolicySucceeded	536
Understanding CVECount	536
Understanding MetadataURL	536
Understanding Phase	536
Understanding ScannedBy	537
Understanding ScannedAt	537
Supply Chain Security Tools for VMware Tanzu - Sign	537
Install Supply Chain Security Tools - Sign	538
Prerequisites	538
Install	538
Configure	542
Known issues	543
Configuring Supply Chain Security Tools - Sign	543
Create a ClusterImagePolicy resource	543
Provide credentials for the package	545
Provide secrets for authentication in your policy	546
Provide secrets for authentication in the image-policy-registry-credentials service account	547
Image name patterns	548
Verify your configuration	548
Logs messages and reasons	549

Supply Chain Security Tools for Tanzu – Store	552
Using the Tanzu Insight CLI plug-in	553
Multicluster configuration	553
Additional documentation	553
Install Supply Chain Security Tools - Store independent from Tanzu Application Platform profiles	553
Prerequisites	553
Install	553
Configure target endpoint and certificate	556
Use Ingress	557
Not use Ingress	557
Use LoadBalancer	557
Use NodePort	558
Configure port forwarding	558
Modify your /etc/hosts file	558
Configure access tokens	558
Service accounts	559
Read-only service account	559
Read-write service account	560
Getting the Access Token	561
Setting the Access Token	561
Security details	561
Application security	561
TLS encryption	561
Cryptographic algorithms:	562
Access controls	562
Authentication	562
Authorization	562
Container security	562
Non-root user	563
Security scanning	563
Static Application Security Testing (SAST)	563
Software Composition Analysis (SCA)	563
Additional documentation	563
API details	564

Information	564
Version	564
Content negotiation	564
URI Schemes	564
Consumes	564
Produces	564
All endpoints	564
images	564
Operations	564
Packages	564
Sources	565
Vulnerabilities	565
Paths	565
Create a new image report. Related packages and vulnerabilities are also created. (CreateImageReport)	565
Parameters	566
All responses	566
Responses	566
200 - Image	566
Schema	566
Default Response	566
Schema	566
Create a new source report. Related packages and vulnerabilities are also created. (CreateSourceReport)	566
Parameters	566
All responses	566
Responses	567
200 - Source	567
Schema	567
Default Response	567
Schema	567
List the packages in an image. (GetImagePackages)	567
Parameters	567
All responses	567
Responses	567
200 - Package	567
Schema	567
Default Response	568
Schema	568
List vulnerabilities from the given image. (GetImageVulnerabilities)	568

Parameters	568
All responses	568
Responses	568
200 - Vulnerability	568
Schema	568
Default Response	568
Schema	568
Search image by id or digest. (GetImages)	568
Parameters	569
responses	569
Responses	569
200 - Image	569
Schema	569
Default Response	569
Schema	569
List the images that contain the given package. (GetPackagelImages)	569
Parameters	569
All responses	569
Responses	570
200 - Image	570
Schema	570
Default Response	570
Schema	570
List the sources containing the given package. (GetPackageSources)	570
Parameters	570
All responses	570
Responses	570
200 - Source	570
Schema	570
Default Response	571
Schema	571
List vulnerabilities from the given package. (GetPackageVulnerabilities)	571
Parameters	571
All responses	571
Responses	571
200 - Vulnerability	571
Schema	571
Default Response	571
Schema	571

Search packages by id, name and/or version. (GetPackages)	571
Parameters	572
All responses	572
Responses	572
200 - Package	572
Schema	572
Default Response	572
Schema	572
get source packages (GetSourcePackages)	572
Parameters	572
All responses	573
Responses	573
200 - Package	573
Schema	573
Default Response	573
Schema	573
List packages of the given source. (GetSourcePackagesQuery)	573
Parameters	573
All responses	573
Responses	573
200 - Package	574
Schema	574
Default Response	574
Schema	574
get source vulnerabilities (GetSourceVulnerabilities)	574
Parameters	574
All responses	574
Responses	574
200 - Vulnerability	574
Schema	574
Default Response	574
Schema	575
List vulnerabilities of the given source. (GetSourceVulnerabilitiesQuery)	575
Parameters	575
All responses	575
Responses	575
200 - Vulnerability	575
Schema	575
Default Response	575

Schema	575
Search for sources by ID, repository, commit sha and/or organization. (GetSources)	575
All responses	576
Responses	576
200 - Source	576
Schema	576
Default Response	576
Schema	576
Search for vulnerabilities by CVE id. (GetVulnerabilities)	576
Parameters	576
All responses	576
Responses	576
200 - Vulnerability	576
Schema	577
Default Response	577
Schema	577
List the images that contain the given vulnerability. (GetVulnerabilityImages)	577
Parameters	577
All responses	577
Responses	577
200 - Image	577
Schema	577
Default Response	577
Schema	577
List packages that contain the given CVE id. (GetVulnerabilityPackages)	578
Parameters	578
All responses	578
Responses	578
200 - Package	578
Schema	578
Default Response	578
Schema	578
List sources that contain the given vulnerability. (GetVulnerabilitySources)	578
Parameters	578
All responses	579
Responses	579
200 - Source	579
Schema	579
Default Response	579

Schema	579
health check (HealthCheck)	579
All responses	579
Responses	579
200	579
Schema	579
Default Response	579
Schema	580
Models	580
DeletedAt	580
ErrorMessage	580
Image	580
MethodType	580
Model	581
NullTime	581
Package	581
Rating	582
Source	582
StringArray	582
Vulnerability	582
API walkthrough	583
Using CURL to POST an image report	583
Deployment details and configuration	584
What is deployed	585
Deployment configuration	585
Database configuration	585
Using AWS RDS postgres database	585
Custom database password	585
App service type	586
Service accounts	586
Exporting certificates	586
Ingress support	586
Install Supply Chain Security Tools - Store independent from Tanzu Application Platform profiles	586
Prerequisites	587
Install	587

AWS RDS Postgres configuration	590
Prerequisites	590
AWS RDS	590
Database backup recommendations	590
Backup	591
Restore	591
Log configuration and usage	592
Log levels	592
Error Logs	592
Obtaining logs	592
API endpoint log output	593
Format	594
Log header	594
Name	594
Key-value pairs	594
Common to all logs	594
Logging query and path parameter values	595
API payload log output	595
SQL Query log output	595
Format	596
Troubleshooting	596
Persistent volume retains data	596
Symptom	596
Solution	596
Missing persistent volume	597
Symptom	597
Solution	597
Multicenter Support: Error sending results to SCST - Store running in a different cluster	597
Symptom	597
Solution	597
Certificate Expiries	597
Symptom	597
Explanation	598
Solution	598
Troubleshooting upgrading	598
Database deployment does not exist	598

Invalid checkpoint record	599
Upgraded pod hanging	599
Failover, redundancy, and backups	599
API Server	599
Database	600
Ingress and multicluster support	600
Multicluster setup	601
TLS CA certificate	601
RBAC Auth token	601
Supply Chain Security Tools - Scan installation	602
Overview of VMware Tanzu Developer Tools for Visual Studio Code	603
Extension Features	603
Installing Tanzu Developer Tools for Visual Studio Code	603
Prerequisites	603
Install	604
Configure	604
Uninstall	605
Next steps	605
Getting started with Tanzu Developer Tools for Visual Studio Code	605
Prerequisite	605
Create the workload.yaml file	605
Create the catalog-info.yaml file	607
Create the Tiltfile file	607
Example project	609
Next steps	609
Using Tanzu Developer Tools for Visual Studio Code	610
Configure for multiple projects in the workspace	610
Debugging on the cluster	610
Start debugging on the cluster	610
Stop Debugging on the cluster	611
Live Update	612
Start Live Update	612
Stop Live Update	612
Deactivate Live Update	614
Live Update status	614

Switch Namespace	614
Pinniped compatibility	615
Oauth	615
LDAP	615
Tanzu API portal	615
Install Tanzu API portal	615
Prerequisites	616
Install	616
Tanzu Application Platform GUI	617
Overview of Tanzu Application Platform GUI	617
Install Tanzu Application Platform GUI	618
Prerequisites	619
Procedure	619
Accessing Tanzu Application Platform GUI	621
Access with the LoadBalancer method (default)	621
Access with the shared Ingress method	621
Catalog operations	622
Adding catalog entities	623
Users and groups	623
Systems	624
Components	624
Update software catalogs	625
Register components	625
Deregister components	625
Add or change organization catalog locations	625
Install demo apps and their catalogs	626
Yelb system	626
Install Yelb	626
Install the Yelb catalog	626
Viewing resources on multiple clusters in Tanzu Application Platform GUI	627
Set up a Service Account to view resources on a cluster	627
Update Tanzu Application Platform GUI to view resources on multiple clusters	629

View resources on multiple clusters in the Runtime Resources Visibility plug-in	630
Setting up a Tanzu Application Platform GUI authentication provider	630
Configure an authentication provider	631
(Optional) Allow guest access	632
(Optional) Customize the login page	632
Support menu customization	633
Overview	633
Customizing	633
Structure of the support configuration	633
URL	633
Items	634
Title	634
Icon	634
Links	634
Adding Tanzu Application Platform GUI integrations	635
Add a GitHub provider integration	635
Add a Git-based provider integration that isn't GitHub	635
Add a non-Git provider integration	636
Update the package profile	636
Configuring the Tanzu Application Platform GUI database	636
Configure a PostgreSQL database	637
TechDocs	638
Create an Amazon S3 bucket	638
Configure Amazon S3 access	638
Find the catalog locations and their entities' namespace/kind/name	639
Use the TechDocs CLI to generate and publish TechDocs	639
Update techdocs section in app-config.yaml to point to the Amazon S3 bucket	640
Tanzu Application Platform GUI plug-ins	641
Overview	641
Runtime resources visibility	642
Introduction	642
Prerequisite	642
Visualize Workloads on Tanzu Application Platform GUI	642
Navigate to the Runtime Resources Visibility screen	642

Knative service details page	643
View details for a specific resource	643
Detail pages	644
Overview card	644
Status card	645
Ownership card	645
Annotations and Labels	646
Navigating to Pod Details Page	646
Navigating to Application Live View	647
Application Live View in Tanzu Application Platform GUI	648
Overview	648
Entry point to Application Live View plug-in	649
Application Live View pages	649
Details page	649
Health page	650
Environment page	650
Log Levels page	651
Threads page	652
Memory page	653
Request Mappings page	654
HTTP Requests page	655
Caches page	656
Configuration Properties page	657
Conditions page	657
Scheduled Tasks page	658
Beans page	659
Metrics page	659
Actuator page	660
Troubleshooting	660
Install Application Live View	660
Prerequisites	661
Install Application Live View	661
Install Application Live View Backend	661
Install Application Live View Connector	663
Install Application Live View Conventions	665
Application Accelerator in Tanzu Application Platform GUI	666
Overview	666

Access Application Accelerator	666
Configure project generation	667
Create the project	668
Develop your code	669
Next steps	669
Install Application Accelerator	670
Prerequisites	670
Configure properties and resource usage	670
Install	671
API documentation plug-in in Tanzu Application Platform GUI	673
Overview	673
Use the API documentation plug-in	673
Create a new API entry	676
Getting started with API documentation plug-in	677
Add your API entry to the Tanzu Application Platform GUI software catalog	677
About API entities	678
Add a demo API entity to Tanzu Application Platform GUI software catalog	678
Update your demo API entry	682
Supply Chain Choreographer in Tanzu Application Platform GUI	682
Overview	682
Prerequisites	683
Supply Chain Visibility	683
Upgrade Tanzu Application Platform GUI	684
Considerations	684
Upgrade within a Tanzu Application Platform profile	684
Upgrade Tanzu Application Platform GUI individually	684
Troubleshoot Tanzu Application Platform GUI	685
Tanzu Application Platform GUI does not work in Safari	685
Symptom	685
Solution	685
Catalog not found	685
Symptom	685
Cause	685
Solution	686
Issues updating the values file	686

Symptom	686
Solution	686
Pull logs from Tanzu Application Platform GUI	687
Symptom	687
Solution	687
Runtime Resources tab	687
Error communicating with Tanzu Application Platform web server	687
Symptom	687
Causes	688
Solution	688
No data available	688
Symptom	688
Cause	688
Solution	688
Errors retrieving resources	688
Symptom	688
Accelerators page	689
No accelerators	689
Symptom	689
Cause	689
Solution	689
Tanzu Build Service	690
Tanzu Build Service Dependencies	690
Configuration	690
Descriptors	690
Automatic Dependency Updates	691
Manual Control of Dependency Updates	691
Install Tanzu Build Service	691
Prerequisites	691
Install Tanzu Build Service by using the Tanzu CLI	692
Install Tanzu Build Service using the Tanzu CLI air-gapped	694
Installation using Secret references for registry credentials	695
Descriptors	696
About descriptors	696
Lite descriptor	697
Full descriptor	697
Descriptor comparison	697

Tekton	698
Install Tekton	698
Prerequisites	698
Install Tekton Pipelines	698
Configure a namespace to use Tekton Pipelines	699
Workload types	701
Using web workloads	701
Using TCP workloads (Beta)	701
Overview	701
Prerequisites	702
Create a tcp SupplyChain	702
Create supply chain templates	702
Add RBAC permissions	707
Define the ClusterSupplyChain	707
Use the tcp workload type	709
Using queue workloads (Beta)	710
Overview	710
Prerequisites	710
Create a queue SupplyChain	710
Create supply chain templates	711
Add RBAC permissions	714
Define the ClusterSupplyChain	715
Use the queue workload type	717
Functions (Beta)	717
Using functions (Beta)	717
Overview	717
Prerequisites	718
Adding function buildpacks	718
Add accelerators to Tanzu Application Platform GUI	720
Create a function project from an accelerator	721
Create a function project using the Tanzu CLI	722
Deploy your function	723
Iterating on your function	723

Prerequisites	724
Configure the Tanzu Developer Tools extension	724
Live update your application	724
Debug your application	725

Tanzu Application Platform v1.1

Overview of Tanzu Application Platform

VMware Tanzu Application Platform is an application development platform that provides 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 delivers a superior developer experience for enterprises building and deploying cloud-native applications on Kubernetes. It enables application teams to get to production faster by automating source-to-production pipelines. It clearly defines the roles of developers and operators so they can work collaboratively and integrate their efforts.

Tanzu Application Platform includes elements that enable developers to quickly begin building and testing applications regardless of their familiarity with Kubernetes.

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. And connecting to other applications and data, regardless of how they're built or what kind of infrastructure they run on, has never been easier, thanks to an internal API management portal.



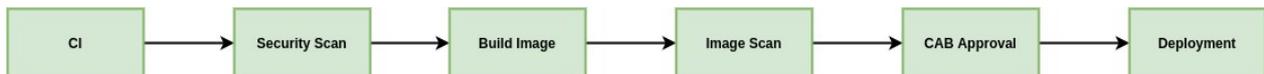
Customers can simplify workflows in both the inner loop and outer loop of Kubernetes-based app development with Tanzu Application Platform while creating supply chains.

- **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, and configuring continuous integration and continuous delivery (CI/CD) pipelines.
 - ◊ Outer loop activities are challenging in a Kubernetes-based development environment due to app delivery platforms being 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 continuous integration and continuous deployment (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 you can add any and every step that is necessary for an application to reach production or a lower environment.

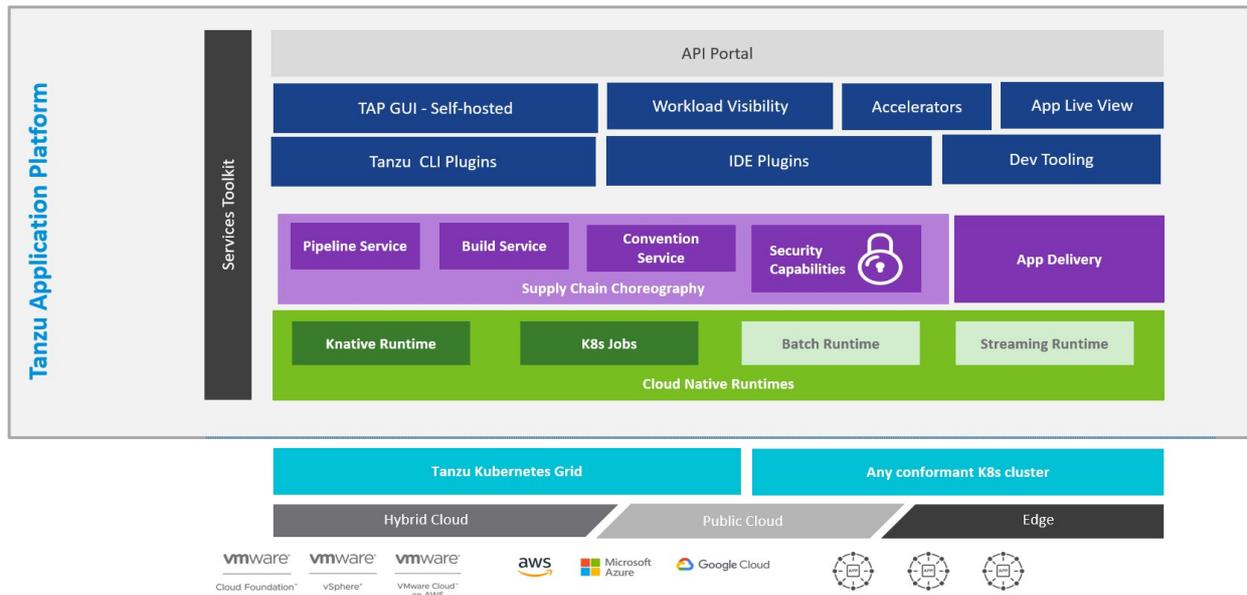


To address the developer experience gap, the path to production allows users to create a unified access point for all of the tools required for their applications to reach a customer-facing environment.

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 the 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



The following packages are part of the Tanzu Application Platform:

- **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 be displayed in a single instance.

- **Application Accelerator for VMware Tanzu**

The Application Accelerator component helps app developers and app operators through the creation and generation of 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, decreasing operator concerns about whether developers are implementing their desired 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. Fundamentally, 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.

- **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**

The convention service provides a means for people in operational roles to express their hard-won knowledge and opinions about how apps should 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 operator and developer time.

- **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 the permissions that 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 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 VS Code IDE Extension or Tanzu CLI Apps plug-in, which rely on the Developer Conventions to modify the workload to enable inner loop capabilities.

- **Flux Source Controller**

The main role of the 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**

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

- **Supply Chain Choreographer for VMware Tanzu**

Supply Chain Choreographer is based on open-source [Cartographer](#). It enables app operators to create pre-approved 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 - test, build, scan, and deploy - enabling developers to focus on delivering value to their users while also providing app operators with the peace of mind that all code in production has passed through all the steps of an approved workflow.

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

With Supply Chain Security Tools for VMware Tanzu - Scan, Tanzu customers 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 gatekeeping 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 - Sign**

Supply Chain Security Tools - Sign provides an admission controller that allows a cluster operator to specify a policy that allows or denies images from running based on signature verification against public keys. It works with [cosign signature format](#) and allows for fine-tuned configuration 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 Supply Chain Security Tools - 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 Buildpacks 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 Buildpacks 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 Visual Studio Code**

Tanzu Developer Tools for Visual Studio Code is the official VMware Tanzu IDE extension for VS Code to help you develop code using the Tanzu Application Platform. The VSCode 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**

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

Installation profiles in Tanzu Application Platform v1.1

Tanzu Application Platform can be deployed through predefined profiles or individual packages. The profiles are designed to allow the Tanzu Application Platform to scale across an organization's multicluster, multicloud, or hybrid cloud infrastructure. These profiles are not meant to cover all customer's use cases, but rather serve as a starting point to allow for further customization.

The following profiles are available in Tanzu Application Platform:

- **Full:** This profile contains all of the Tanzu Application Platform packages.
- **Iterate:** This profile is intended for iterative application development.
- **Build:** This profile is intended for the transformation of source revisions to workload revisions. Specifically, hosting Workloads and SupplyChains.
- **Run:** This profile is intended for the transformation of workload revisions to running Pods. Specifically, hosting Deliveries and Deliverables.
- **View:** This profile is intended for instances of applications related to centralized developer experiences. Specifically, Tanzu Application Platform GUI and Metadata Store.

About Tanzu Application Platform package profiles

Tanzu Application Platform can be installed through predefined profiles or through individual packages. This section explains how to install a profile.

Tanzu Application Platform contains the following five profiles:

- Full (`full`)
- Iterate (`iterate`)
- Build (`build`)
- Run (`run`)
- View (`view`)

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	✓				✓
Cloud Native Runtimes	✓	✓		✓	
Convention Controller	✓	✓	✓		
Default Roles	✓	✓	✓	✓	✓
Developer Conventions	✓	✓			
Flux Source Controller	✓	✓	✓	✓	✓
Grype	✓		✓		
Image Policy Webhook	✓	✓		✓	
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	✓	✓	✓	✓	
Service Bindings	✓	✓		✓	
Services Toolkit	✓	✓		✓	
Source Controller	✓	✓	✓	✓	✓
Spring Boot Convention	✓	✓	✓		
Supply Chain Choreographer	✓	✓	✓	✓	
Supply Chain Security Tools - Scan	✓		✓		
Supply Chain Security Tools - Store	✓				✓
Tanzu Build Service	✓	✓	✓		
Tanzu Application Platform GUI	✓				✓
Tekton Pipelines	✓	✓	✓		
Telemetry	✓	✓	✓	✓	✓

* Only one supply chain should be installed at any given time. For information on switching from one supply chain to another, see [Getting Started with Tanzu Application Platform](#).

About installing the Tanzu Application Platform v1.1

To install the Tanzu Application Platform profiles, see [Installing Tanzu Application Platform](#).

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](#).

Release notes

This topic contains release notes for Tanzu Application Platform v1.1

v1.1.2

Release Date: June 14, 2022

Security fixes

Tanzu Application Platform GUI

- [CVE-2022-1664](#): Improper Limitation of a Pathname to a Restricted Directory
- [CVE-2022-1292](#): Improper Neutralization of Special Elements used in an OS Command
- [CVE-2022-25878](#): Improperly Controlled Modification of Object Prototype Attributes

Resolved issues

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

Application Live View

- Application Live View Connector package now supports values without quotes in `sslDisabled` boolean flag.
- Application Live View Convention Service sets `tanzu.app.live.view.application.name` to `carto.run/workload-name` if not set in workload yaml.
- Application Live View now supports environment editing for newer Spring Boot apps.

Grype scanner

- Added useful error message when the Syft schema version embedded in images is not compatible with the current Syft schema version supported by the Grype version.
- `zlib` has been updated to `1.2.11-2.ph3`
- `subversion` has been updated to `1.10.8-2.ph3`

Supply Chain Security Tools - Scan

- Fixed `SourceScans` failing for a blob scan without `sourceScan.spec.revision` set and without a `.git` folder in the source code.

- Fixed the values schema to include an `importFromNamespace` key for the authentication token needed to communicate to the Metadata Store.

Tanzu Application Platform GUI

- CVE fixes
- Various styling and bug fixes

Known issues

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

Grype scanner

Scanning Java source code may not reveal vulnerabilities: Source Code Scanning only scans files present in the source code repository. 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, dependency lock files are not guaranteed, so Grype uses the dependencies present in the built binaries (`.jar` or `.war` files) instead.

Because VMware discourages 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 - Scan

Blob Source Scan is reporting wrong source URL: - When running a Source Scan of a blob compressed file, SCST - Scan looks for a `.git` directory present in the files to extract useful information for the report sent to the SCST - Store deployment.

- Workaround - The following workarounds fix this issue:
 1. This problem is resolved in SCST - Scan `v1.2.0`. Upgrade your SCST - Scan and Grype Scanner deployment to version `v1.2.0` or later.
 2. Configure your SourceScan or Workload to connect using HTTPS to the repository instead of using SSH.
 3. Edit the FluxCD GitRepository resource to not include the `.git` directory.

Error: Unable to decode cyclonedx: Supply Chain Security Tools - Scan has a known issue where it will set the phase to `Error` and show an `unable to decode cyclonedx` error in the `Succeeded` condition. The root cause of the problem is not known, but it is an intermittent issue that cuts the CycloneDX XML stream to the logs and then the scan controller can't process the results properly.

Workaround: As this is an intermittent issue, if you're applying the scan manually, you can delete the scan and re-apply it again to retry the scan. If this problem happened while running a Supply Chain from the OOTB Supply Chains, you can run `kubectl get imagescans -n <workload namespace>` or `kubectl get sourcescans -n <workload namespace>` to get the scan name, delete it running `kubectl delete <imagescan or sourcescan> <scan name> -n <workload namespace>` and the Choreographer controller will recreate it for you.

Supply Chain Security Tools - Sign

Supply Chain Security Tools - Sign rejects images from private registries when the image is deployed to a non-default namespace. For a workaround, see [Supply Chain Security Tools - Sign rejects images](#).

v1.1.1

Release Date: May 10, 2022

Resolved issues

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

Supply Chain Choreographer plug-in

- ImageScan stage shows incorrect status
- Workloads page does not show errors
- Build stage shows error while building

Supply Chain Security Tools - Scan

- Resolved edge case for scan phase to correctly indicate `Error` when error occurs during scanning
- Added missing `SecretImport` for the RBAC Auth token `store-auth-token` for multicluster
- Resolved race condition involving reading Store secrets and exporting to the Scan Controller namespace

Supply Chain Security Tools - Sign

- Updated golang to v1.17.9 to address [CVE-2022-27191](#)

Supply Chain Security Tools - Store

- Updated `containerd` version to `v1.5.10` to resolve [GHSA-crp2-qrr5-8pq7](#) in Github
- Updated postgres image to resolve [CVE-2018-25032](#)
- Updated `brancz/kube-rbac-proxy` image to `0.12.0` to resolve [GHSA-c3h9-896r-86jm](#) in Github
- Fixed the issue where new vulnerabilities are not appended to the existing packages
- Fixed the issue where Insight CLI plug-in fails to start on Windows platforms

Grype Scanner

- Removed package `gnutls` to address [CVE-2021-20232](#) and [CVE-2021-20231](#)

- Removed package `lua` to address [CVE-2022-28805](#)
- Updated module `golang.org/x/crypto` to `v0.0.0-20210220033148-5ea612d1eb83` to address [CVE-2022-27191](#)

Tanzu Application Platform GUI

- CVE fixes
- Various styling fixes
- TLS Certificate and Ingress bug fix
- Supply Chain plug-in upgrade

Known issues

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

Grype scanner

Scanning Java source code may not reveal vulnerabilities: Source Code Scanning only scans files present in the source code repository. 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, dependency lock files are not guaranteed, so Grype uses the dependencies present in the built binaries (`.jar` or `.war` files) instead.

Because VMware discourages 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 Choreographer for Tanzu

- **SourceScan error when deploying Java functions workloads:** Using Out of the Box Supply Chain with Testing and Scanning to deploy Java functions workloads causes a `SourceScan` error. The workload deployed shows an error for `SourceScan` because it cannot find the scan template. You can receive enhanced scanning coverage for Java and Node.js workloads, including application runtime layer dependencies, by using both Tanzu Build Service and Grype in your Tanzu Application Platform supply chain. Python workloads are not supported.

Supply Chain Security Tools - Scan

The Supply Change Security Tools - Scan has the following CVEs at high severity from the `kube-rbac-proxy v0.11.0-vmware.1` image:

Supply Chain Security Tools - Store

The Supply Change Security Tools - Store has [CVE-2022-21698](#) at high severity from `brancz/kube-rbac-proxy:0.12.0` image.

Tanzu Application Platform GUI

- **Accelerators not appearing on Accelerator page:** If the `app_config.backend.reading.allow` section is configured during the `tap-gui` package installation, no accelerators show on the accelerator page. For more information, see [Troubleshooting](#).

v1.1

Release Date: April 12, 2022

Prerequisites

Installation requires Kubernetes clusters v1.21, v1.22, or v1.23. See [prerequisites](#) for supported Kubernetes platforms.

New features

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

Installing

There are four new profiles available, and additions to the Full profile. The inclusion of new profiles supports a multicluster deployment architecture.

- **Tanzu Application Platform profile - Iterate** is intended for iterative development in contrast to the path to production.
- **Tanzu Application Platform profile - Build** is intended for the transformation of source revisions to workload revisions. Specifically, it's for hosting workloads and SupplyChains.
- **Tanzu Application Platform profile - Run** is intended for the transformation of workload revisions to running pods. Specifically, it's for hosting deliveries and deliverables.
- **Tanzu Application Platform profile - View** is intended for instances of applications related to centralized developer experiences, such as Tanzu Application Platform GUI and Metadata Store.
- **Tanzu Application Platform profile - Full** contains all of the Tanzu Application Platform packages. New packages in the Full profile:
 - ◊ Application Live View (Build)
 - ◊ Application Live View (Run)
 - ◊ Application Live View (GUI)
 - ◊ Default Roles
 - ◊ Telemetry

Default roles for Tanzu Application Platform

There are five new default roles and related permissions that apply to Kubernetes resources. These roles help operators set up common sets of permissions to limit the access that users and service accounts have on a cluster that runs Tanzu Application Platform.

Three roles are for users, including `app-editor`, `app-viewer` and `app-operator`. Two roles are for “robot” or system permissions, including `workload` and `deliverable`.

For more information, see [Overview of Default Roles](#).

Application Accelerator

- Option values can now be validated using regex
- TLS for ingress is enabled using `ingress.enable_tls` flag during package install

Application Live View

- Application Live View supports a multicluster setup now
- Application Live View components are split into three bundles with new package reference names (backend, connector, conventions)
- Application Live View Convention Service is compatible with cert-manager v1.7.1
- Application Live View Convention takes the management port setting from the Spring Boot Convention into account
- Structured JSON logging is integrated into Application Live View Backend and Application Live View Convention
- Updated Spring Native v0.10.5 to v0.10.6

Tanzu CLI - Apps plug-in

- `workload create/update/apply`:
 - ◊ Accept `workload.yaml` from stdin (through `--file -`).
 - ◊ Enable providing `spec.build.env` values (through new `-build.env` flag).
 - ◊ When `--git-repo` and `--git-tag` are provided, `git-branch` is not required.
 - ◊ Add new `--annotations` flag. Annotations provided are propagated to the running pod for the workload.
- `workload list`:
 - ◊ Shorthand `-A` can be passed in for `--all-namespaces`.
- `workload get`:
 - ◊ Service Claim details are returned in command output.
 - ◊ The existing STATUS value in the pods table in the output reflects when a pod is terminating.
- Deprecation
 - ◊ The `namespace` value you can pass for the `--service-ref` flag is deprecated.
 - ◊ A deprecation warning message is added to the `workload create/update/apply...` when user specifies a namespace in the `--service-ref` object.

Service Bindings

- Applied [RFC-3339](#) timestamps to service binding logs.
- Added Tanzu Application Platform aggregate roles to support Tanzu Application Platform Authentication and Authorization (new feature referenced above).
- Added support for `servicebinding.io/v1beta1`
- Corrected Postgres resource pluralization error.

Source Controller

- Enable Source Controller to connect to image registries that use self-signed or private certificate authorities to support air-gapped installations. This is an optional configuration. See [Source Controller Installation](#) for details.
- Applied [RFC-3339](#) timestamps to source controller logs.
- Added Tanzu Application Platform aggregate roles to support Tanzu Application Platform Authentication and Authorization (new feature referenced above).

Spring Boot Conventions

- **Set the management port to 8081:** This is instead of the default port 8080.
 - ◊ This change increases the security of Spring Boot apps running on the platform by preventing access to actuator endpoints. Actuator endpoints can leak sensitive information and allow access to trigger actions that can impact the app.
 - ◊ If the app explicitly sets the management port using the `JAVA_TOOL_OPTIONS` in the `workload.yaml`, the Spring Boot conventions respect that setting and do not set the management port to 8081. For more information, see [Set the `JAVA_TOOL_OPTIONS` property for a workload](#).
 - ◊ The convention overrides other common management port configuration methods such as `application.properties/yml` and `config server`.
- **RFC-3339 timestamps:** Applied [RFC-3339](#) timestamps to service binding logs.
- **Added Kubernetes liveness and readiness probes by using Spring Boot health endpoints:**
 - ◊ This convention is applied to Spring Boot v2.6 and later apps.
 - ◊ The probes are exposed on the main serving port for the app, which is port 8080 by default.

Supply Chain Choreographer

- All Supply Chains provided by Tanzu Application Platform support pre-built images for workloads
- Supply Chains can select workloads by fields and expressions in addition to labels
- Supply Chains can select which template to stamp out based on optional criteria
- Workloads include stamped resource references in their status

Supply Chain Security Tools - Scan

Support for configuring Supply Chain Security Tools - Scan to remotely connect to Supply Chain Security Tools - Store in a different cluster.

Supply Chain Security Tools - Sign

- Support configuring Webhook resources
- Support configuring Namespace where webhook is installed
- Support for registries with self-signed certificates

Supply Chain Security Tools - Store

- Added Contour Ingress support with custom domain name
- Created Tanzu CLI plug-in called `insight`. Currently, `insight` plug-in only supports macOS and Linux.

Tanzu Application Platform GUI

- Added improvements to the information presentation and filtering mechanisms of the Runtime Resources tab
- Added the new Supply Chain plug-in
- Added the Backstage API Documentation plug-in
- Updated overall theme to Clarity City
- Added compatibility with v1beta3 Backstage Templates
- Small security fixes
- Various accessibility and styling fixes

Plug-in improvements and additions include:

- **Runtime Resources Visibility plug-in:**
 - ◊ Textual and enumerated table column filters for ease of search
 - ◊ Meaningful error messages and paths forward to troubleshoot issues
 - ◊ Tags in Knative revision table on the details page
 - ◊ Kubernetes Service on the resources page to provide more insights into Kubernetes service details
 - ◊ Improved UI components for a more accessible user experience
- **Supply Chain Choreographer plug-in:**
 - ◊ Added a graphical representation of the execution of a workload by using an installed supply chain. This includes CRDs in the supply chain, the source results of each stage, and details to facilitate the troubleshooting of workloads on their path to production.

Functions (Beta)

Tanzu Application Platform enables developers to deploy functions, use starter templates to bootstrap their functions 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. This feature is in beta and subject to changes based on user feedback. It is intended for evaluation and test purposes only.

For more information, see [Functions](#).

Breaking changes

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

Application Accelerator

When enabling ingress, the TLS support must now be explicitly enabled using `ingress.tls_enable`.

Supply Chain Security Tools - Scan

API version `scanning.apps.tanzu.vmware.com/v1alpha1` is deprecated.

Supply Chain Security Tools - Store

- The independent `insight` CLI is deprecated. You can now use the Tanzu CLI plug-in Tanzu Insight, which currently supports macOS and Linux only.
- Renamed all instances of the `create` verb as `add` for all CLI commands.

Resolved issues

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

Application Accelerator

Accelerator engine no longer fails with `java.lang.OutOfMemoryError: Direct buffer memory` when processing very large Git repositories.

Application Live View

Updated Spring Boot to v2.5.12 to address [CVE-2022-22965](#) and [CVE-2020-36518](#)

Services Toolkit

- Resolved an issue with the `tanzu services` CLI plug-in that meant it was not compatible with Kubernetes clusters running on GKE.
- Fixed a potential race condition during reconciliation of ResourceClaims which might have caused the Services Toolkit manager to stop responding.
- Updated configuration of the Services Toolkit carvel Package to prevent an unwanted build up of ConfigMap resources.

Supply Chain Security Tools - Scan

- Resolved two scan jobs and two scan pods that are created when reconciling `ScanTemplates` and `ScanPolicies`
- Updated package `client_golang` to v1.11.1 to address [CVE-2022-21698](#)

Grype Scanner

- Updated golang to v1.17.8 to address [CVE-2022-24921](#)
- Updated photon to address [CVE-2022-23308](#) and [CVE-2022-0778](#)

Supply Chain Security Tools - Store

- Fixed an issue where querying a source report with local path name returned the following error: `{ "message": "Not found" }`.
- Return related packages when querying image and source vulnerabilities.
- Ratings are updated when updating vulnerabilities.
- Fixed [CVE-2022-24407](#) and [CVE-2022-0778](#) found in the PostgreSQL image.
- Updated package `client_golang` to v1.17.8 to address [CVE-2022-24921](#).

Tanzu CLI - Apps plug-in

- Apps plug-in no longer fails when `KUBECONFIG` includes the colon (`:`) config file delimiter.
- `tanzu apps workload get`: Passing in `--output json` and `--export` flags together exports the workload in JSON rather than YAML.
- `tanzu apps workload tail`: Duplicate log entries created for init containers are removed.
- `tanzu apps workload create/update/apply`
 - ◊ When the `--wait` flag passed and the dialog box “Do you want to create this workload?” is declined, the command immediately exits 0, rather than hanging and continuing to wait.
 - ◊ Workload name is now validated when the workload values are passed in by using `--file workload.yaml`.
 - ◊ When creating or applying a workload from `-local-path`, if user answers “No” to the prompt “Are you sure you want to publish your local code to [registry name] where others may be able to access it?”, the command now exits 0 immediately rather than showing the workload diff and prompting to continue with workload creation.
 - ◊ `.spec.build.env` in workload YAML definition file is being removed when using `Tanzu apps workload apply` command.

Tanzu Application Platform GUI

Applied a fix for [CVE-2021-3918](#) from the `json-schema` package

Known issues

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

Tanzu Application Platform

Deprecated profile: Tanzu Application Platform light profile is deprecated.

Tanzu Cluster Essentials

- **Feature Disabled error:** When adding Tanzu Application Platform clusters with pre-installed Tanzu Cluster Essentials to a Tanzu Mission Control instance, the `tanzunet-secret` export shows `Feature Disabled`.
- **-export-all-namespaces not properly observed:** When deploying Tanzu Application Platform on Google Kubernetes Engine (GKE) v1.23.5-gke.200 and running `tanzu secret registry add tanzunet-creds`, the `--export-all-namespaces` is not properly observed.

Application Live View

- **Application Live View Connector sometimes does not connect to the back end:** Search the Application Live View Connector pod logs for issues with rsocket connection to the back end. If you find any issues, delete the connector pod to recreate it:

```
kubectl -n app-live-view delete pods -l=name=application-live-view-connector
```

- **Application Live View Convention auto-exposes all actuators:** Application Live View Convention exposes all Spring Boot actuator endpoints by default to whatever is configured using the Spring Boot Convention for the management port. You can change this configuration if it does not suit your needs. For more information, see [Convention Server](#).
- **Frequent Application Live View Connector restarts:** In some cases, the Application Live View Connector component restarts frequently. This usually doesn't cause problems when using Application Live View.
- **No structured JSON logging on the connector:** The format of the log output of the Application Live View Connector component is not currently aligned with the standard Tanzu Application Platform logging format. A fix is planned for Tanzu Application Platform v1.1.1.

Grype scanner

Scanning Java source code may not reveal vulnerabilities: Source Code Scanning only scans files present in the source code repository. 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, dependency lock files are not guaranteed, so Grype instead uses the dependencies present in the built binaries (`.jar` or `.war` files).

Because VMware discourages 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 Choreographer plug-in

- **Details for ConfigMap CRD not appearing:** The error `Unable to retrieve conditions for ConfigMap...` appears in the details section after clicking on the ConfigMap stage in the graph view of a supply chain. This error does not necessarily mean that the workload failed its execution through the supply chain.
- **Scan results not shown:** Current CVEs found during Image or Build scanning do not appear. However, results are still present in the metadata store and are available by using the Tanzu CLI.

Supply Chain Security Tools - Scan

- **Scan Phase indicates Scanning incorrectly:** Scans have an edge case that when an error occurs during scanning, the `Scan Phase` field is not updated to `Error` and remains in the `Scanning` phase. Read the scan pod logs to verify the existence of an error.
- **Multicluster Support: Error sending results to SCST - Store running in a different cluster:** During installation, Supply Chain Security Tools - Scan (Scan) creates the `SecretImport` for ingesting the TLS CA certificate secret, but misses the `SecretImport` for the RBAC Auth token. See, [Troubleshooting Supply Chain Security Tools - Store](#).
- **User sees error message indicating Supply Chain Security Tools - Store (Store) is not configured even though configuration values were supplied:** The Scan Controller experiences a race-condition when deploying Store in the same cluster, that shows Store as not configured, even when it is present and properly configured. This happens when the Scan Controller is deployed and reconciled before the Store is reconciled and the corresponding secrets are exported to the Scan Controller namespace. As a workaround, after your Store is successfully reconciled, restart your Supply Chain Security Tools - Scan deployment by running:

```
kubectl rollout restart deployment.apps/scan-link-controller-manager -n scan-link-system
```

Note: If you deploy Supply Chain Security Tools - Scan to a different namespace than the default one, replace `-n scan-link-system` with `-n <my_custom_namespace>`.

Supply Chain Security Tools - Store

- **Tanzu Insight CLI plug-in does not support Windows:**
Currently, the Tanzu Insight plug-in only supports macOS and Linux.
- **Existing packages with new vulnerabilities not updated:**
It is a known issue that Supply Chain Security Tools - Store does not correctly save new vulnerabilities for a package that was already submitted in a previous report. This issue causes new vulnerabilities not saved to the database.
- **Persistent volume retains data:**
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 cause is the persistent volume that PostgreSQL uses retaining old data, even though the

retention policy is set to `DELETE`.

To resolve this issue, see [solution](#).

- **Missing persistent volume:**

After Supply Chain Security Tools - Store is deployed, `metadata-store-db` pod might fail for missing volume while `postgres-db-pv-claim` pvc is in the `PENDING` state. This issue might occur if 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`. To resolve this issue, see [solution](#).

- **No support for installing in custom namespaces:**

Supply Chain Security Tools — Store is deployed to the `metadata-store` namespace. There is no support for configuring the namespace.

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.
- **Runtime Resources errors:** The **Runtime Resources** tab shows cluster query errors when attempting to retrieve Kubernetes object details from clusters that are not on the Full profile. For more information, see [Troubleshooting](#)

- **Supply Chain displays incorrect data if there are workloads with same name and namespace:** When there are two workloads that have the same name and namespace, but exist on different clusters, clicking either of them in the supply chain page always shows the details for the first one. There is no way to access details for the second.

- **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.

Installing Tanzu Application Platform

This topic provides an overview to installing Tanzu Application Platform.

Installation process

The process of installing Tanzu Application Platform includes the following tasks:

Step	Task	Link
1.	Review the prerequisites to ensure that you have set up everything required before beginning the installation	Prerequisites
2.	Accept the end-user license agreements	Accept the EULAs
3.	Install the Tanzu command line interface (CLI) and plug-ins for the Tanzu CLI	Install the Tanzu CLI and plug-ins
4.	Create a namespace, add a secret, and add the Tanzu Application Platform package repository	Add the Tanzu Application Platform Package Repository
5.	Prepare your Tanzu Application Platform profile	Prepare to install your Tanzu Application Platform profile
6.	Install the profile to the cluster	Install your Tanzu Application Platform package
7.	(Optional) Install any additional packages that were not in the profile	Installing Individual Packages
8.	Install developer tools into your integrated development environment (IDE)	Installing Tanzu Developer Tools for Visual Studio Code

Prerequisites

The following are required to install Tanzu Application Platform:

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.

- ◆ If installing using the `lite` descriptor for Tanzu Build Service, 1 GB of available storage is recommended.
- ◆ If installing using the `full` descriptor for Tanzu Build Service, which is suitable for offline environments, 10 GB of available storage is recommended.

Note: For production environments, the `full` descriptor is recommended to optimize security and performance. For more information about Tanzu Build Service descriptors, see [About descriptors](#).

- Registry credentials with read and write access made 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 `cnrs.domain_name` 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 specified in the `learningcenter.ingressDomain` 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 an `IngressDomain` of your choice. For example, `tap-gui.example.com`.

Tanzu Application Platform GUI

- 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 the `Using accelerator.yaml` section in [Getting started with the Tanzu Application Platform](#). 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 towards storing 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, and 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 v1.21, 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 the [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 following [resource requirements](#)
 - ◊ 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 --clusterrole=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, v1.20, v1.21 or v1.22, installed and authenticated with admin rights for your target cluster. See [Install Tools](#) in the Kubernetes documentation.

Accepting Tanzu Application Platform EULAs, installing Cluster Essentials and the Tanzu CLI

This topic describes how to:

- [Accept Tanzu Application Platform EULAs](#)
- [Set the Kubernetes cluster context](#)
- [Install Cluster Essentials for Tanzu](#)
- [Install or update the Tanzu CLI and plug-ins](#)

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:
 - ◊ [Cluster Essentials for VMware Tanzu](#)
 - ◊ [Tanzu Application Platform](#)
 - ◊ Tanzu Build Service associated components:
 - [Tanzu Build Service Dependencies](#)
 - [Buildpacks for VMware Tanzu](#)
 - [Stacks 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

GET EMAIL ALERTS

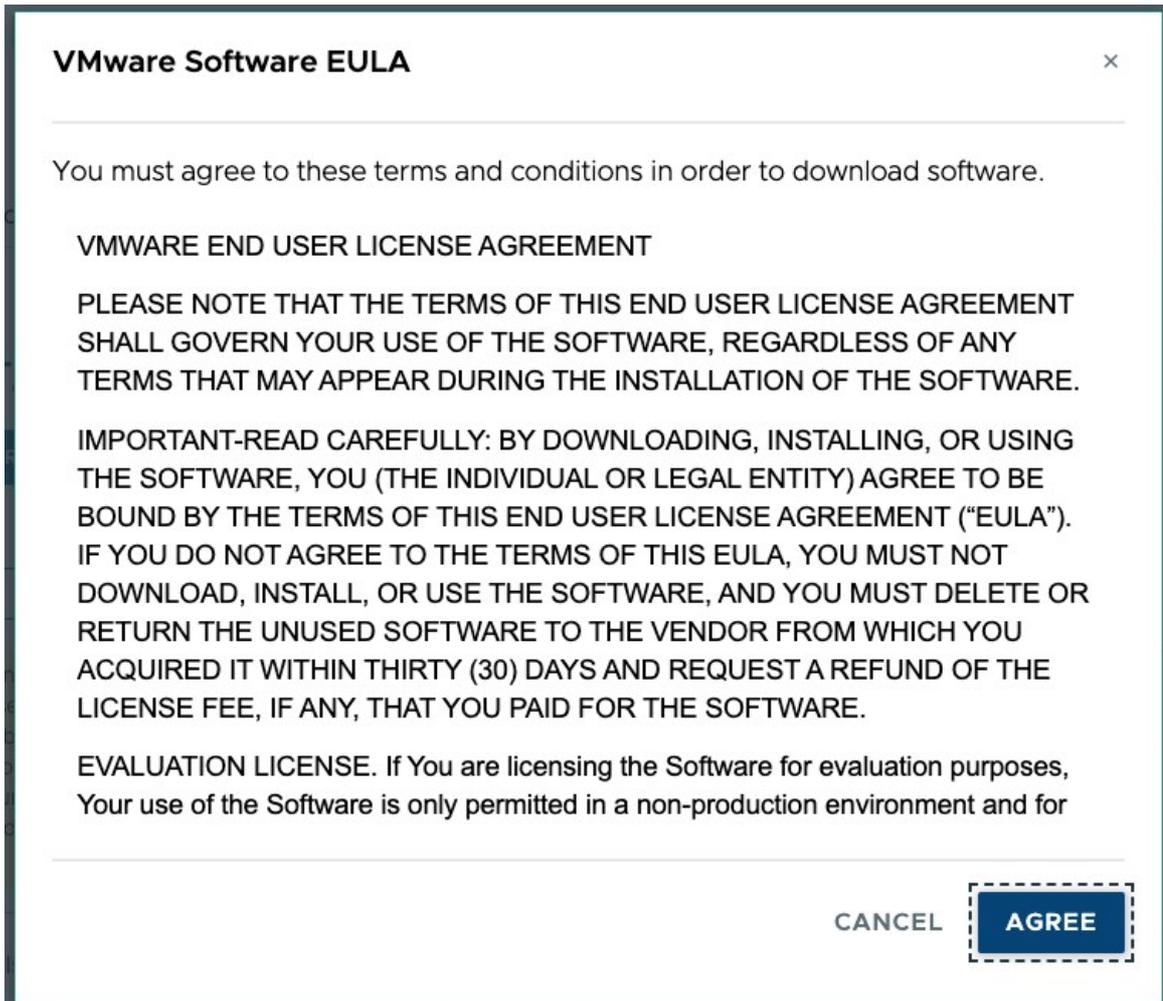
RELEASE: 1.1.2

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.](#)

	learning-center-workshop-samples.zip	
111 KB	1.0.1	
	Tanzu Developer Tools for Visual Studio Code	
16 MB	0.6.0	
	tanzu-cli-v0.11.6	>
	3 Files	
	tap-gui-catalogs-latest	>
	2 Files	
	Artifact References	>
	679 Artifacts	

Release Details	
Release Date	2022-06-14
Release Type	Patch Release
End of General Support	2023-06-30
Release Description	
Patch release includes multiple CVE fixes.	
Depends On	
Products in the "Depends On" section must be installed prior to installing or upgrading to VMware Tanzu Application Platform 1.1.2. Please install or upgrade these products to one of the listed versions.	
Cluster Essentials for VMware Tanzu 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 1.1.2. If your current version of VMware Tanzu Application Platform is not on this list, please contact Tanzu Customer Service for assistance.	
1.0.*, 1.1.0 or 1.1.1	
For any assistance with upgrades please use the Tanzu Upgrade Planner Tool .	
License Files	
VMWARE TANZU APPLICATION PLATFORM 1.1.2 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.



This example shows that you have now accepted the EULAs for Tanzu Application Platform. In addition, you must accept the EULAs for Cluster Essentials for VMware Tanzu and for Tanzu Build Services and its associated components as stated above.

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
          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 Cluster Essentials for Tanzu

[Cluster Essentials for VMware Tanzu](#) simplifies the process of installing the open-source [Carvel](#) tools on your cluster. It includes a script to download and install supported versions of `kapp-controller` and `secretgen-controller` on the target cluster. It also installs the `kapp`, `imgpkg`, `ytt`, and `kbls` CLIs on your local machine. Currently, Cluster Essentials only supports macOS and Linux.

When you are using 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.

To install Cluster Essentials, see [Deploying Cluster Essentials](#).

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 1.1.0** from the release drop-down menu.
4. Click and download the Tanzu framework bundle for your operating system.
5. (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.

To install the Tanzu CLI and plug-ins:

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.

- 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
```

- 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
```

- Install or update the CLI core by running:

Note: Replace `v0.11.2` with the version you've downloaded.

- For Linux:

```
cd $HOME/tanzu
export VERSION=v0.11.2
sudo install cli/core/$VERSION/tanzu-core-linux_amd64 /usr/local/bin/tanzu
```

- For macOS:

```
cd $HOME/tanzu
export VERSION=v0.11.2
install cli/core/$VERSION/tanzu-core-darwin_amd64 /usr/local/bin/tanzu
```

- Confirm the installation by running:

```
tanzu version
```

Expected outcome:

```
version: v0.11.4
...
```

- Proceed to [Install/Update Tanzu CLI plug-ins](#)

Install Tanzu CLI: Windows

- Open the Windows file browser.
- Create a `Program Files\tanzu` directory on your local machine.
- 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**.
- From the `Program Files\tanzu` directory, move and rename; the executable file from

```
Program Files\tanzu\cli\core\v0.11.2\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: v0.11.2
...
```

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	VERSION	STATUS
login	SCOPE DISCOVERY Login to the platform		

management-cluster	Standalone default	Kubernetes management-cluster operations	v0.11.6	not installed
package	Standalone default	Tanzu package management	v0.11.6	not installed
pinniped-auth	Standalone default	Pinniped authentication operations (usually not directly invoked)	v0.11.6	installed
secret	Standalone default	Tanzu secret management	v0.11.6	not installed
services	Standalone default	Discover Service Types, Service Instances and manage Resource Claims (ALPHA)	v0.11.6	installed
accelerator	Standalone	Manage accelerators in a Kubernetes cluster	v0.3.0-rc.2	installed
lled	Standalone		v1.2.0-build.1	insta
apps	Standalone	Applications on Kubernetes	v0.7.0-build.1	insta
lled				
insight	Standalone	post & query image, package, source, and vulnerability data	v1.2.1	installed

Note: Currently, `insight` plug-in only supports macOS and Linux.

You can now proceed with [installing the Tanzu Application Platform Package and Profiles](#).

Installing the Tanzu Application Platform package and profiles

This topic describes how to install Tanzu Application Platform packages from the Tanzu Application Platform package repository.

Before installing the packages, ensure that you have completed the prerequisites, configured and verified the cluster, accepted the EULA, and installed the Tanzu CLI with any required plug-ins. See [Accepting Tanzu Application Platform EULAs, installing Cluster Essentials and the Tanzu CLI](#) for more information.

Relocate images to a registry

VMware recommends relocating the images to your registry from VMware Tanzu Network registry before attempting installation.

If you choose not to relocate images, Tanzu Application Platform depends directly on VMware Tanzu Network for continued operation. VMware recommends relocating images because there are no uptime guarantees for installations that depend directly on VMware Tanzu Network. The option to skip relocation is documented for evaluation and proof-of-concept only.

The supported container 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. Log in to your image registry by running:

```
docker login MY-REGISTRY
```

Where `MY-REGISTRY` is your own container registry.

2. Log in to the VMware Tanzu Network registry with your VMware Tanzu Network credentials by running:

```
docker login registry.tanzu.vmware.com
```

3. 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
```

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, `1.1.0`.

4. Relocate the images with the Carvel tool `imgpkg` by running:

```
imgpkg copy -b registry.tanzu.vmware.com/tanzu-application-platform/tap-package
s:${TAP_VERSION} --to-repo ${INSTALL_REGISTRY_HOSTNAME}/TARGET-REPOSITORY/tap-p
ackages
```

Where `TARGET-REPOSITORY` is your target repository

5. 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.

6. 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
```

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

```
tanzu package repository add tanzu-tap-repository \
  --url ${INSTALL_REGISTRY_HOSTNAME}/TARGET-REPOSITORY/tap-packages:${TAP_VERSIO
N} \
  --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
```

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:           1.0.2
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.

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 to e
nable search, discovery and try-out of API endpoints at ease.
backend.appliveview.tanzu.vmware.com     Application Live View fo
r VMware Tanzu                         App for monitoring and troubl
eshooting running apps
connector.appliveview.tanzu.vmware.com   Application Live View Co
nector for VMware Tanzu                 App for discovering and regis
tering running apps
conventions.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 t
he building and automation of containerized software workflows securely and at
scale.
cartographer.tanzu.vmware.com            Cartographer
Kubernetes native Supply Chai
```

n Choreographer. cnrs.tanzu.vmware.com	Cloud Native Runtimes Cloud Native Runtimes is a se
rverless runtime based on Knative controller.conventions.apps.tanzu.vmware.com	Convention Service for V Convention Service enables ap
Mware Tanzu p operators to consistently apply desired runtime configurations to fleets of w orkloads.	orkloads.
controller.source.apps.tanzu.vmware.com	Tanzu Source Controller Tanzu Source Controller enabl
es workload create/update from source code. developer-conventions.tanzu.vmware.com	Tanzu App Platform Devel Developer Conventions
oper Conventions grype.scanning.apps.tanzu.vmware.com	Grype Scanner for Supply Default scan templates using
Chain Security Tools - Scan Anchore Grype image-policy-webhook.signing.apps.tanzu.vmware.com	Image Policy Webhook The Image Policy Webhook allo
ws platform operators to define a policy that will use cosign to verify signatu res of container images learningcenter.tanzu.vmware.com	Learning Center for Tanz Guided technical workshops
u Application Platform ootb-supply-chain-basic.tanzu.vmware.com	Tanzu App Platform Out o Out of The Box Supply Chain B
f The Box Supply Chain Basic asic. ootb-supply-chain-testing-scanning.tanzu.vmware.com	Tanzu App Platform Out o Out of The Box Supply Chain w
ith Testing and Scanning. ootb-supply-chain-testing.tanzu.vmware.com	Tanzu App Platform Out o Out of The Box Supply Chain w
ith Testing. ootb-templates.tanzu.vmware.com	Tanzu App Platform Out o Out of The Box Templates.
f The Box Templates scanning.apps.tanzu.vmware.com	Supply Chain Security To Scan for vulnerabilities and
ols - Scan enforce policies directly within Kubernetes native Supply Chains. metadata-store.apps.tanzu.vmware.com	Tanzu Supply Chain Secur The Metadata Store enables sa
ity Tools - Store saving and querying image, package, and vulnerability data. service-bindings.labs.vmware.com	Service Bindings for Kub Service Bindings for Kubernet
ernetes 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 Resource s (databases, message queues, DNS records, etc.). spring-boot-conventions.tanzu.vmware.com	Tanzu Spring Boot Conven Default Spring Boot conventio
tions Server n server. tap-gui.tanzu.vmware.com	Tanzu Application Platfo web app graphical user interf
rm GUI ace for Tanzu Application Platform tap.tanzu.vmware.com	Tanzu Application Platfo Package to install a set of T
rm AP components to get you started based on your use case. workshops.learningcenter.tanzu.vmware.com	Workshop Building Tutori Workshop Building Tutorial
al	

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 [Installation profiles in Tanzu Application Platform](#).

The following profiles are available for Tanzu Application Platform:

- **Full:** This profile contains all of the Tanzu Application Platform packages.
- **Iterate:** This profile is intended for iterative application development.
- **Build:** This profile is intended for the transformation of source revisions to workload revisions. Specifically, hosting workloads and SupplyChains.
- **Run:** This profile is intended for the transformation of workload revisions to running pods. Specifically, hosting deliveries and deliverables.
- **View:** This profile is intended for instances of applications related to centralized developer experiences. Specifically, the TAP GUI and Metadata Store.

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

Important: Keep the values file for future configuration use.

3. [\(Optional\) Configure LoadBalancer for Contour ingress](#)
4. Proceed to the [View possible configuration settings for your package](#) section.

(Optional) Configure LoadBalancer for Contour ingress

Important: This section only applies when you use Tanzu Application Platform to deploy its own shared Contour ingress controller in `tanzu-system-ingress`. It is not applicable when you use your existing ingress.

Before defining other parameters for your Tanzu Application Platform installation, VMware recommends defining your ingress because several components, including Tanzu Application Platform GUI, rely on it.

You can share this ingress across Cloud Native Runtimes (`cnrs`), Tanzu Application Platform GUI (`tap_gui`), and Learning Center (`learningcenter`).

By default, Contour uses `NodePort` as the service type. To set the service type to `LoadBalancer`, add the following to your `tap-values.yaml`:

```

contour:
  envoy:
    service:
      type: LoadBalancer

```

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

```

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

```

Full profile

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

```

profile: full

contour:
  envoy:
    service:
      type: LoadBalancer

shared:
  ingress_domain: INGRESS-DOMAIN

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"
  tanzunet_username: "TANZUNET-USERNAME"
  tanzunet_password: "TANZUNET-PASSWORD"
  descriptor_name: "DESCRIPTOR-NAME"

supply_chain: basic

cnrs:
  domain_name: INGRESS-DOMAIN

ootb_supply_chain_basic:
  registry:
    server: "SERVER-NAME"
    repository: "REPO-NAME"
  gitops:
    ssh_secret: "SSH-SECRET-KEY"

learningcenter:
  ingressDomain: "INGRESS-DOMAIN"

tap_gui:
  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

metadata_store:
  app_service_type: LoadBalancer # (optional) Defaults to LoadBalancer. Change to Node
  Port for distributions that don't support LoadBalancer

grype:
  namespace: "MY-DEV-NAMESPACE" # (optional) Defaults to default namespace.
  targetImagePullSecret: "TARGET-REGISTRY-CREDENTIALS-SECRET"

```

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 username 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. You can also configure this credential by using a secret reference. See [Install Tanzu Build Service](#) for details.
 - ◊ For Google Cloud Registry, use the contents of the service account JSON file.
- `TANZUNET-USERNAME` and `TANZUNET-PASSWORD` are the email address and password to log in to VMware Tanzu Network. Your VMware Tanzu Network credentials enable you to configure the dependencies updater. This resource accesses and installs the build dependencies (buildpacks and stacks) that Tanzu Build Service requires on your cluster. It can also optionally keep these dependencies up to date as new versions are released on VMware Tanzu Network. You can also configure this credential by using a secret reference. For more information, see [Install Tanzu Build Service](#).
- `DESCRIPTOR-NAME` is the name of the descriptor to import. For more information, see [Overview of Tanzu Build Service](#). 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.
- `SERVER-NAME` is the hostname 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.
- `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, you can use a Backstage-compliant catalog you've already built and posted on the Git infrastructure.
- `MY-DEV-NAMESPACE` is the namespace where you want to deploy the `ScanTemplates`. This is the namespace where the scanning feature runs.
- `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the secret that contains the credentials to pull an image from the registry for scanning. 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](#).

Note: When you install Tanzu Application Platform, it is bootstrapped with a set of dependencies (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 in the descriptor file on [Tanzu Network](#). The current installed version of the descriptor is `100.0.293`. Sometimes the dependencies require updates. You can use a [manual process in a CI/CD context](#), or an [automatic update process](#) in the background by Tanzu Application Platform.

Light Profile

The Light profile is deprecated. Although existing values files might still refer to the Light profile, VMware recommends to migrate to one of the new profiles described in [Install your Tanzu Application Platform profile](#) by following the procedures in [Migrate Tanzu Application Platform profiles](#).

View possible configuration settings for your package

To view possible configuration settings for a package, run:

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

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

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/1.0.3 --values-schema` for Cloud Native Runtimes.

```
profile: full

# ...

# e.g. CNRs specific values go under its name
cnrs:
  provider: local

# e.g. App Accelerator specific values go under its name
accelerator:
  server:
    service_type: "ClusterIP"
```

Identify the values for your package

You can identify the values for your Tanzu package by running:

```
tanzu package available get PACKAGE-NAME.tanzu.vmware.com/VERSION --values-schema -n tap-install
```

Where:

- `VERSION` is the version number of the package. For example, `0.5.1` for Supply Chain Basic package.
- `PACKAGE-NAME` is the value of `Top-level Key` for package-specific configuration within your `tap-values.yaml`, as summarized in the following table:

Package	Top-level Key
<i>see table below</i>	<code>shared</code>
API portal	<code>api_portal</code>
Application Accelerator	<code>accelerator</code>
Application Live View	<code>appliveview</code>
Application Live View Connector	<code>appliveview_connector</code>
Application Live View Conventions	<code>appliveview-conventions</code>

Package	Top-level Key
Cartographer	<code>cartographer</code>
Cloud Native Runtimes	<code>cnrs</code>
Convention Controller	<code>convention_controller</code>
Source Controller	<code>source_controller</code>
Supply Chain	<code>supply_chain</code>
Supply Chain Basic	<code>ootb_supply_chain_basic</code>
Supply Chain Testing	<code>ootb_supply_chain_testing</code>
Supply Chain Testing Scanning	<code>ootb_supply_chain_testing_scanning</code>
Supply Chain Security Tools - Scan	<code>scanning</code>
Supply Chain Security Tools - Scan (Grype Scanner)	<code>grype</code>
Supply Chain Security Tools - Store	<code>metadata_store</code>
Image Policy Webhook	<code>image_policy_webhook</code>
Build Service	<code>buildservice</code>
Tanzu Application Platform GUI	<code>tap_gui</code>
Learning Center	<code>learningcenter</code>

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
<code>ca_cert_data</code>	<code>convention_controller,</code> <code>source_controller</code>	Optional: PEM Encoded certificate data to trust TLS connections with a private CA.

For information about package-specific configuration, see [Installing individual packages](#).

For example, to identify the SSH secret keys for Supply Chain Basic, you can run:

```
tanzu package available get ootb-supply-chain-basic.tanzu.vmware.com/0.5.1 --values-schema -n tap-install
```

Expect to see the following outputs that list the all the SSH secret keys and the descriptions applicable to the package:

KEY	DEFAULT	TYPE	DESCRIPTION
<code>cluster_builder</code>	<code>default</code>	<code>string</code>	Name of the Tanzu Build Service (TBS) ClusterBuilder to use by default on image objects managed by the supply chain.
<code>gitops.branch</code>	<code>main</code>	<code>string</code>	Default branch to use for pushing Kubernetes configuration files produced by the supply chain.
<code>gitops.commit_message</code>	<code>bump configuration</code>	<code>string</code>	Default git commit message

e to write when publishing Kubernetes configuration files produces by the supply chain to git.

```

gitops.email          supplychain@cluster.local  string  Default user email to be
used for the commits produced by the supply chain.

gitops.repository_prefix <nil>                string  Default prefix to be used
for forming Git SSH URLs for pushing Kubernetes configuration produced by the supply
chain.

gitops.ssh_secret     git-ssh                string  Name of the default Secret
containing SSH credentials to lookup in the developer namespace for the supply chain
to fetch source code from and push configuration to.

gitops.username       supplychain            string  Default user name to be u
sed for the commits produced by the supply chain.

registry.repository   <nil>                string  Name of the repository in
the image registry server where the application images from the workloads should be p
ushed to (required).

registry.server       index.docker.io        string  Name of the registry serv
er where application images should be pushed to (required).

service_account       default                string  Name of the service accou
nt in the namespace where the Workload is submitted to utilize for providing registry
credentials to Tanzu Build Service (TBS) Image objects as well as deploying the applic
ation.

```

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
```

4. (Optional) [Install any additional packages](#) not included in your profile.

Important: Ensure you have [set up developer namespaces to use your installed packages](#).

After installing Full Profile on to your cluster, you can install the Tanzu Developer Tools for VSCode extension to help you develop against it. For instructions, see [Installing Tanzu Developer Tools for](#)

[VS Code](#).

Access Tanzu Application Platform GUI

To access Tanzu Application Platform GUI, you can use the hostname that you configured earlier. This hostname 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`.

Warning: You can break package dependencies by removing a package. Allow 20 minutes to verify that all packages have reconciled correctly while troubleshooting.

Opting out of telemetry collection

This topic describes how to opt out of the VMware Customer Experience Improvement Program (CEIP). By default, when you install Tanzu Application Platform, you are opted into telemetry collection. To turn off telemetry collection, complete following the instructions.

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.

Upgrading Tanzu Application Platform

This document describes how to upgrade Tanzu Application Platform.

You can perform 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`

Add new package repository

Follow these steps to update to the new package repository:

1. Relocate the latest version of TAP images which you want to update with the Carvel tool `imgpkg` by following the steps provided in [document](#) from Step 1 - Step 4.

```
tanzu package repository update tanzu-tap-repository \
  --url registry.tanzu.vmware.com/tanzu-application-platform/tap-packages:VER
SION \
  --namespace tap-install
```

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

Note: Make sure to Update the export `TAP_VERSION=VERSION-NUMBER` which you want to update to

2. Add the target version of the Tanzu Application Platform package repository by running:
 - If Cluster Essentials 1.2 and above installed:

```
tanzu package repository add tanzu-tap-repository \
```

```
--url ${INSTALL_REGISTRY_HOSTNAME}/TARGET-REPOSITORY/tap-packages:$TAP_VERSION \
--namespace tap-install
```

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

- If Cluster Essentials 1.1 and 1.0 installed:

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

Where `$TAP_VERSION` is the target revision of Tanzu Application Platform you are updating to. For example, 1.3.0

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 upgrade of Tanzu Application Platform

Upgrade instructions for Profile-based installation

Important: Before performing the upgrade, ensure `descriptor_name` is either unset or set to `full`, or `lite` in the `tap-values.yaml`.

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 tap-values.yaml -n tap-install
```

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

Upgrade instructions for component-specific installation

For information about upgrading Tanzu Application Platform GUI, see [upgrading Tanzu Application Platform GUI](#).

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                        PACKAG
E-VERSION  STATUS
accelerator                accelerator.apps.tanzu.vmware.com    1.0.2
    Reconcile succeeded
api-portal                 api-portal.tanzu.vmware.com         1.0.9
    Reconcile succeeded
appliveview               run.appliveview.tanzu.vmware.com    1.0.2
    Reconcile succeeded
appliveview-conventions   build.appliveview.tanzu.vmware.com  1.0.2
    Reconcile succeeded
buildservice              buildservice.tanzu.vmware.com       1.4.3
    Reconcile succeeded
cartographer              cartographer.tanzu.vmware.com       0.2.2
    Reconcile succeeded
cert-manager              cert-manager.tanzu.vmware.com       1.5.3+
tap.1    Reconcile succeeded
cnrs                      cnrs.tanzu.vmware.com               1.1.1
    Reconcile succeeded
contour                   contour.tanzu.vmware.com             1.18.2
+tap.1    Reconcile succeeded
conventions-controller    controller.conventions.apps.tanzu.  0.5.1
    Reconcile succeeded
developer-conventions     developer-conventions.tanzu.vmware.  0.5.0
    Reconcile succeeded
fluxcd-source-controller  fluxcd.source.controller.tanzu.vmw  0.16.3
    Reconcile succeeded
grype                     grype.scanning.apps.tanzu.vmware.c  1.0.1
    Reconcile succeeded
image-policy-webhook      image-policy-webhook.signing.apps.  1.0.2
    Reconcile succeeded
learningcenter            learningcenter.tanzu.vmware.com      0.1.1
    Reconcile succeeded
learningcenter-workshops  workshops.learningcenter.tanzu.vmw  0.1.1
    Reconcile succeeded
metadata-store            metadata-store.apps.tanzu.vmware.co  1.0.2
    Reconcile succeeded
ootb-delivery-basic       ootb-delivery-basic.tanzu.vmware.co  0.6.1
    Reconcile succeeded
ootb-supply-chain-basic   ootb-supply-chain-basic.tanzu.vmw  0.6.1
    Reconcile succeeded
ootb-templates            ootb-templates.tanzu.vmware.com     0.6.1
    Reconcile succeeded
scanning                  scanning.apps.tanzu.vmware.com       1.0.1
    Reconcile succeeded
service-bindings          service-bindings.labs.vmware.com     0.6.1
    Reconcile succeeded
services-toolkit          services-toolkit.tanzu.vmware.com    0.5.1
    Reconcile succeeded
source-controller         controller.source.apps.tanzu.vmware  0.2.1
    Reconcile succeeded
spring-boot-conventions   spring-boot-conventions.tanzu.vmw  0.3.0
    Reconcile succeeded
tap                       tap.tanzu.vmware.com                 1.0.2
    Reconcile succeeded
```

tap-gui	tap-gui.tanzu.vmware.com	1.0.2
Reconcile succeeded		
tap-telemetry	tap-telemetry.tanzu.vmware.com	0.1.4
Reconcile succeeded		
tekton-pipelines	tekton.tanzu.vmware.com	0.30.1
Reconcile succeeded		

Migrate Tanzu Application Platform profiles

This document describes how to migrate from one Tanzu Application Platform profile to another.

You can install Tanzu Application Platform by following the instructions in [Installing Tanzu Application Platform](#).

Prerequisites

To migrate from one Tanzu Application Platform profile to another ensure you do the following:

- Install Tanzu Application Platform. See [Install your Tanzu Application Platform profile](#)
- Install or update the Tanzu CLI and plug-ins. See [Install or update the Tanzu CLI and plug-ins](#). If you install Tanzu Application Platform GUI verify the considerations. See [Tanzu Application Platform GUI Considerations](#)
- Verify all packages are reconciled by running `tanzu package installed list -A`
- Note which packages are included in the original profile and the profile migrated to. If the profile you are migrating to includes additional packages, those will be added and additional cluster resources may be needed. If the profile migrated to doesn't include a component, the package is removed. If you do not want to add or remove a package, consider adding an additional cluster instead of migrating.

Add new package repository

Follow these steps to add the new package repository:

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

```
tanzu package repository update tanzu-tap-repository \
  --url registry.tanzu.vmware.com/tanzu-application-platform/tap-packages:TAP
-VERSION \
  --namespace tap-install
```

Where `TAP-VERSION` is your Tanzu Application Platform version. For example, `1.1.0`.

2. Add the new package repository by running:

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

Edit the tap-values.yaml configuration file that was used during installation

During the Tanzu Application Platform installation, a configuration file is built that contains the

necessary configuration values related to the Tanzu Application Platform cluster. See [Installing the Tanzu Application Platform package and profiles](#). To change the profile, edit the `profile` key in this file, typically saved as `tap-values.yaml` by running:

```
profile: PROFILE-NAME
```

Where `PROFILE-NAME` is the desired target profile. Review the documentation on profiles to determine the best fit.

Perform migration of Tanzu Application Platform profile

This step will execute the actual migration that will result in the addition or removal of components based on the selected profile. Consideration should be given to backup any data that may be lost as part of this operation.

To complete the Tanzu Application Platform profile migration, do the following:

```
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 your Tanzu Application Platform version. For example, `1.1.0`.

Getting started with the Tanzu Application Platform

Purpose

Welcome to the Tanzu Application Platform. This document guides you through getting started on the platform. Specifically, you are going to learn how to:

- Develop and promote an application
- Create an application accelerator
- Add testing and security scanning to an application
- Administer, set up, and manage supply chains

Before getting started, you must complete the prerequisites in the next section.

Getting started prerequisites

Verify you have successfully:

- **Installed the Tanzu Application Platform**
See [Installing Tanzu Application Platform](#).
- **Installed the 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](#).

Note: 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 installed packages](#).
- **Installed Tanzu Application Platform GUI**
See [Install Tanzu Application Platform GUI](#).

- **Installed the VS Code Tanzu Extension**
See [Install the Visual Studio Code Tanzu Extension](#) for instructions.

When you have completed the prerequisites, you are ready to get started.

Section 1: Develop your first application on the Tanzu Application Platform

In this section, you are going to:

- Learn about application accelerators
- Deploy your application
- Add your application to Tanzu Application Platform GUI Software Catalog
- Set up your integrated development environment (IDE)
- Iterate on your application
- Live update your application
- Debug your application
- Monitor your running application

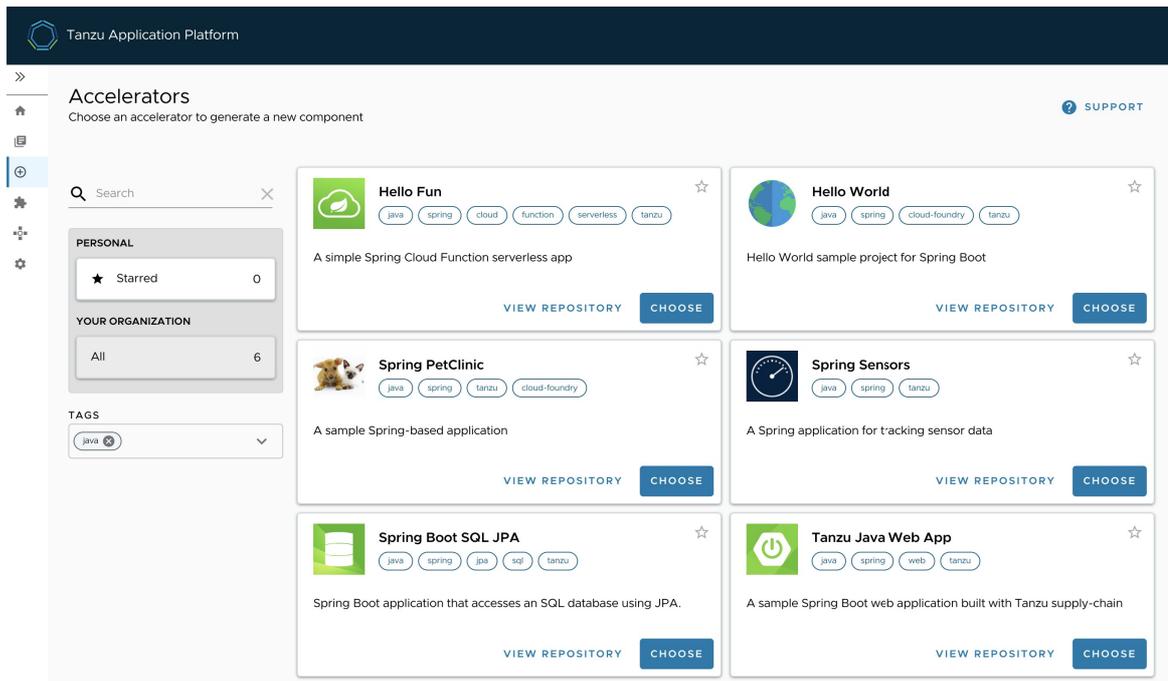
About application accelerators

Application accelerators are templates that not only codify best practices, but also provide important configuration and structures ready and available for use. Developers can create applications and get started with feature development immediately. Admins can create custom application accelerators that reflect desired architectures and configurations, enabling developer use according to the best practices defined. The Application Accelerator plug-in of Tanzu Application Platform GUI assists both application developers and admins with creating and generating application accelerators. To create your own application accelerator, see [Create your accelerator](#).

Deploy your application

To deploy your application, you must download an accelerator, upload it on your Git repository of choice, and run a CLI command. VMware recommends using the accelerator called `Tanzu-Java-Web-App`.

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. For information about connecting to Tanzu Application Platform GUI, see [Accessing Tanzu Application Platform GUI](#).



2. Locate the Tanzu Java Web App accelerator, which is a Spring Boot web app, 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`. Click **NEXT STEP**, verify the provided information, and click **CREATE**.
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. Ensure you have [set up developer namespaces to use installed packages](#).
7. 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 in step 5 and `YOUR-DEVELOPER-NAMESPACE` is the namespace configured in step 6.

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 in step 6.

For more information, see [Tanzu Apps Workload Create](#).

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

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

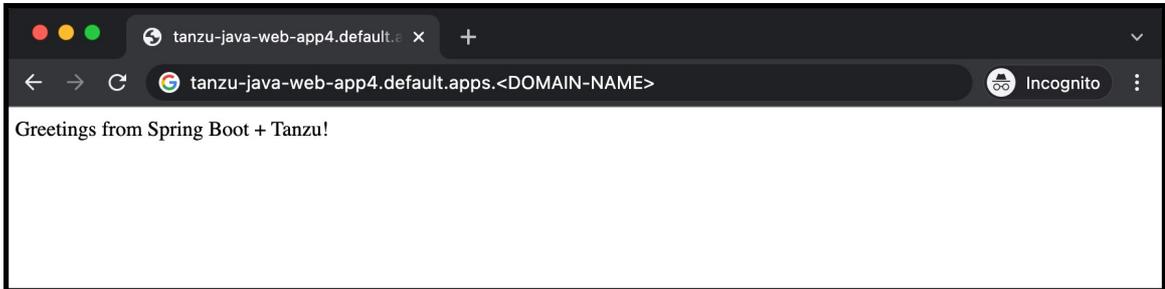
```
tanzu apps workload tail tanzu-java-web-app --since 10m --timestamp --namespace
YOUR-DEVELOPER-NAMESPACE
```

Where `YOUR-DEVELOPER-NAMESPACE` is the namespace configured in step 6.

- 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 command below, and then press **ctrl-click** on the Workload Knative Services URL at the bottom of the command output.

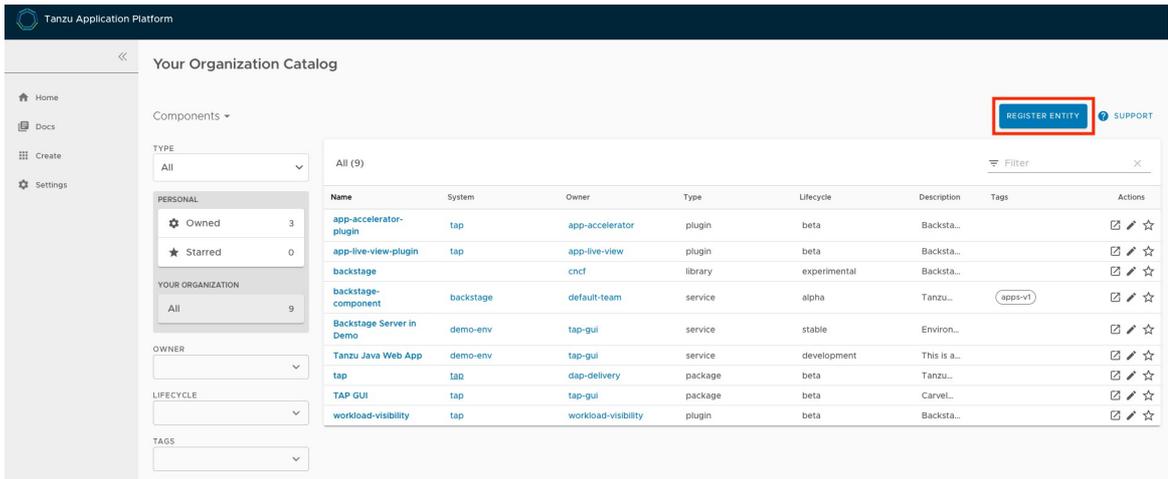
```
tanzu apps workload get tanzu-java-web-app --namespace YOUR-DEVELOPER-NAMESPACE
```

Where `YOUR-DEVELOPER-NAMESPACE` is the namespace configured in step 6.



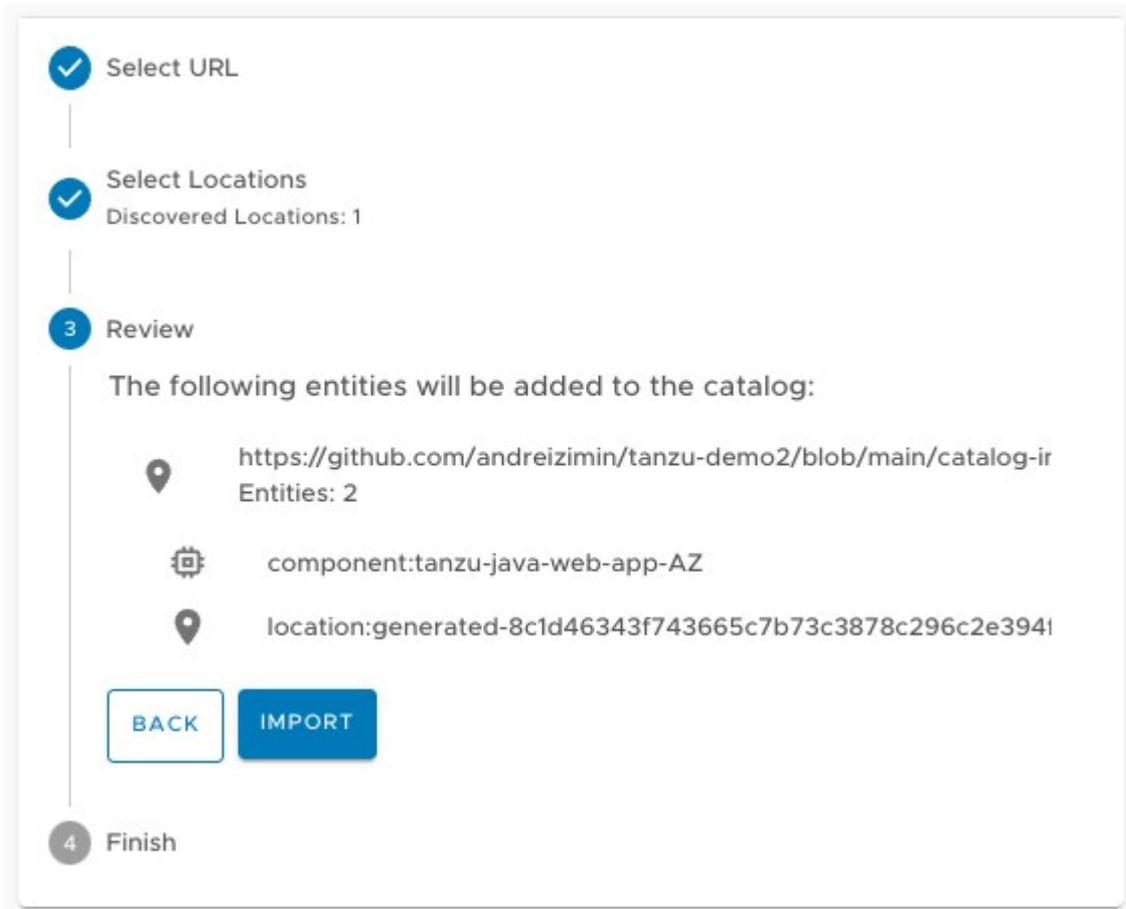
Add your application to Tanzu Application Platform GUI Software Catalog

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



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



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.

Iterate on your application

Now that you have a skeleton workload working, you are ready to iterate on your application and test code changes on the cluster. Tanzu Developer Tools for Visual Studio Code, VMware Tanzu's official IDE extension for VSCode, helps you develop and receive fast feedback on your workloads running on the Tanzu Application Platform.

The VSCode 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 extension, see [Install Tanzu Dev Tools for VSCode](#).

Note: Use Tilt v0.23.2 or a later version for the sample application.

1. Open the Tanzu Java Web App as a project within your VSCode IDE.
2. To ensure your extension assists you with iterating on the correct project, configure its settings using the following instructions.
 - ◆ In Visual Studio Code, navigate to `Preferences > Settings > Extensions > Tanzu`.
 - ◆ In the **Local Path** field, provide the path to the directory containing the Tanzu Java Web App. The current directory is the default.
 - ◆ 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 demonstrate how code changes are going to behave on a production cluster early in the development process.

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. Enter to the IDE and 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.

Follow 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. You are going to see the “Deploy and Connect” Task complete and the debug menu actions are available to you, indicating that the debugger has attached.
3. Navigate to `http://localhost:8080` in your browser. This hits the breakpoint within VSCode. Play to the end of the debug session using VSCode 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.

Follow the following steps to diagnose Spring Boot-based applications using Application Live View:

1. Confirm that the Application Live View components installed successfully. For instructions, see [Verify the Application Live View component](#).
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 [Product Features](#).

Section 2: Create your application accelerator

In this section, you are going to create an application accelerator by using Tanzu Application Platform GUI and CLI.

Create an application accelerator

You can use any Git repository to create an accelerator. You need the repository URL to create an accelerator.

The Git repository must be `public` and contain a `README.md` file. These options are available to configure when you create repositories on GitHub.

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

```

Note: You can use any icon with a reachable URL.

4. Add the new `accelerator.yaml` file, commit this change and push 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. Run the following command to publish the new application accelerator:

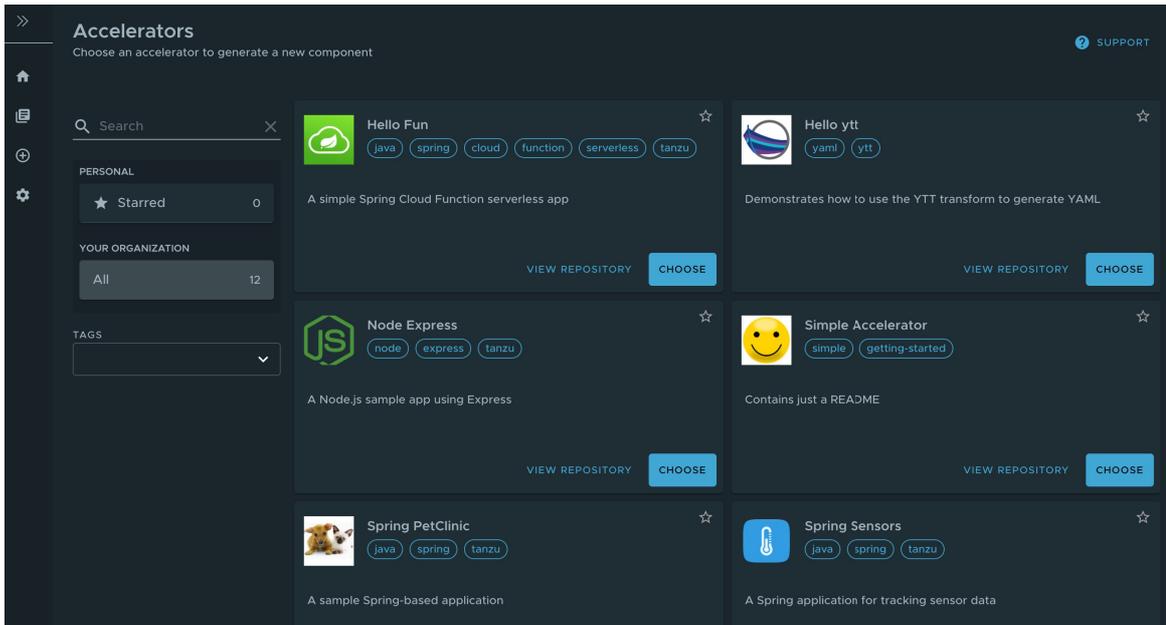
```

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.



Note: It might take a few seconds for Tanzu Application Platform GUI to refresh the catalog and add an entry for 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. You can run the following command to force an immediate reconciliation:

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

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
```

Section 3: Add Testing and Security Scanning to Your Application

In this section, you are going to:

- Learn about supply chains
- Discover available out of the box (OOTB) supply chains
 - ◆ OOTB Basic (default)
 - ◆ OOTB Testing
 - ◆ OOTB Testing+Scanning
- Install OOTB Testing (optional)
- Install OOTB Testing+Scanning (optional)

Introducing a Supply Chain

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 chain to deliver applications through automation where supply chain allows 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, which may be automated and repeatable between teams for applications at scale.

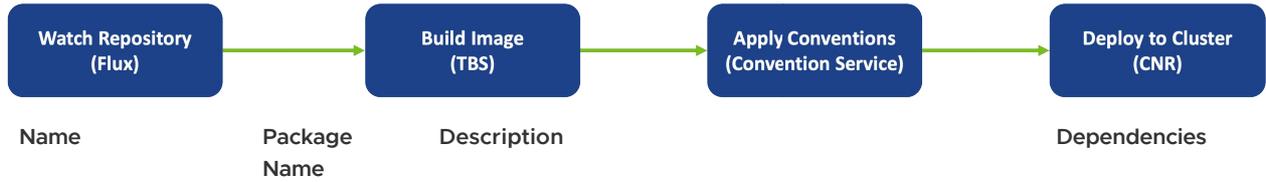
Where tools typically are not able to integrate with one another and additional scripting or webhooks are necessary, there would be a unified automation tool to codify all the interactions between each of the tools. Supply chains used to codify the organization's path to production are configurable, 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 OOTB supply chains to work with the Tanzu Application Platform components, and they include:

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.



<p>Out of the Box Basic (Default - Installed during Installing Part 2)</p>	<p><code>ootb-supply-chain-basic.tanzu.vmware.com</code></p>	<p>This supply chain monitors a repository that is identified in the developer's <code>workload.yaml</code> file. When any new commits are made to the application, the supply chain:</p> <ul style="list-style-type: none"> • Creates a new image. • Applies any predefined conventions. • Deploys the application to the cluster. 	<ul style="list-style-type: none"> • Flux/Source Controller • Tanzu Build Service • Convention Service • Tekton • Cloud Native Runtimes • If using Service References: <ul style="list-style-type: none"> ◊ Server vices Build indings ◊ Server vices Tolk it
---	--	--	--

2: OOTB Testing

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



Name	Package Name	Description	Dependencies
Out of the Box Testing	<code>ootb-supply-chain-testing.tanzu.vmware.com</code>	<p>The 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.</p> <ul style="list-style-type: none"> 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

The OOTB Testing+Scanning supply chain includes integrations for secure scanning tools.



Name	Package Name	Description	Dependencies
Out of the Box Testing and Scanning	<code>ootb-supply-chain-testing-scanning.tanzu.vmware.com</code>	<p>The Out of the Box Testing and Scanning contains all of the same elements as the Out of the Box Testing supply chains but it also includes integrations out of the box with the secure scanning components of Tanzu Application Platform.</p> <ul style="list-style-type: none"> 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. 	<p>All of the Source to URL dependencies, and:</p> <ul style="list-style-type: none"> The secure scanning components included with Tanzu Application Platform

Install OOTB Testing

This section introduces how to install the OOTB Testing supply chain and provides 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.

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

To apply this install method, follow the following steps:

- You can activate the Out of the Box Supply Chain with Testing by updating our profile to use `testing` rather than `basic` as the selected supply chain for workloads in this cluster. Update `tap-values.yaml` (the file used to customize the profile in `Tanzu package install tap --values-file=...`) 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>"

```

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, `1.1.0`.

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 needs to be tested. The operator can also add the Tekton supply chain to a cluster before the developer get 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 itself contained in the `steps` pull the code from the repository indicated in the developers `workload` and run the tests within

the repository. The steps of the Tekton pipeline are configurable and allow the developer to add any additional items that is needed to test their code. Because this step is one of many in the supply chain (and the next step is an image build in this case), the developer is free to focus on testing their code. Any additional steps that the developer adds to the Tekton pipeline is independent for the image being built and any subsequent steps of the supply chain being executed.

The `params` are templated by the Supply Chain Choreographer. Additionally, Tekton pipelines require a Tekton `pipelineRun` in order to execute on the cluster. The Supply Chain Choreographer handles creating the `pipelineRun` dynamically each time that step of the supply 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 update 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
```

```
Create workload:
 1 + |---
 2 + |apiVersion: carto.run/v1alpha1
 3 + |kind: Workload
 4 + |metadata:
 5 + |  labels:
 6 + |    apps.tanzu.vmware.com/has-tests: "true"
 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

? Do you want to create this workload? Yes
Created workload "tanzu-java-web-app"
```

2. After accepting the workload creation, monitor the creation of new resources by the workload by running:

```
kubectl get workload,gitrepository,pipelinerun,images.kpack,podintents,apps,services.serving
```

You will see output similar to the following example that shows the objects that were created by the Supply Chain Choreographer:

NAME	AGE
workload.carto.run/tanzu-java-web-app	109s

NAME	READY	STATUS	URL	
gitrepository.source.toolkit.fluxcd.io/tanzu-java-web-app sample-accelerators/tanzu-java-web-app c8866b7805fb2425130edb69a9853bdfd	True	Fetches revision: main/872ff44	https://github.com/	
		109s		
NAME	COMPLETIONTIME	SUCCEEDED	REASON	START
pipelinerun.tekton.dev/tanzu-java-web-app-4ftlb	77s	True	Succeeded	104s
NAME	LATESTIMAGE	READY	REASON	
image.kpack.io/tanzu-java-web-app	10.188.0.3:5000/foo/tanzu-java-web-app@sha256:1d5bc4d3d1ffeb8629fbb721fcd1c4d28b896546e005f1efd98fbc4e79b7552c	True		
NAME	AGE	READY	REASON	
podintent.conventions.apps.tanzu.vmware.com/tanzu-java-web-app	7s	True		
NAME	AGE	DESCRIPTION	SINCE-DEPLOY	
app.kappctrl.k14s.io/tanzu-java-web-app	2s	Reconcile succeeded	1s	
NAME	REASON	URL	LATESTREADY	READY
service.serving.knative.dev/tanzu-java-web-app	IngressNotConfigured	http://tanzu-java-web-app.developer.example.com	tanzu-java-web-app-00001	Unknown

Install OOTB Testing+Scanning

Follow these steps to install the OOTB Testing+Scanning supply chain:

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.

Important: The grype must be installed for scanning.

1. Supply Chain Security Tools - Scan is installed as part of the profiles. Verify that both Scan Link 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 also must 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 the following code must be in the required namespace. You can either add the namespace flag to the kubectl command or add the namespace field to the template itself. Run:

```
kubectl apply -f - -o yaml << EOF
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: scan-policy
spec:
  regoFile: |
    package policies

    default isCompliant = false

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "UnknownSeverity"
    violatingSeverities := ["Critical","High","UnknownSeverity"]
    ignoreCVEs := []

    contains(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
      fails := contains(violatingSeverities, match.Ratings.Rating[_].Severity)
      not fails
    }

    isSafe(match) {
      ignore := contains(ignoreCVEs, match.Id)
      ignore
    }

    isCompliant = isSafe(input.currentVulnerability)
EOF
```

2. (optional) To persist and query the vulnerability results post-scan, ensure that [Supply Chain Security Tools - Store](#) is installed using the following command. The Tanzu Application Platform profiles install the package by default.

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

If the package is not installed, follow [the installation instructions](#).

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, `1.1.0`.

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 create 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
```

Example output:

```
Create workload:
 1 + |---
 2 + |apiVersion: carto.run/v1alpha1
 3 + |kind: Workload
 4 + |metadata:
 5 + |  labels:
 6 + |    apps.tanzu.vmware.com/has-tests: "true"
 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

? Do you want to create this workload? Yes
Created workload "tanzu-java-web-app"
```

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 the Supply Chain Choreographer created:

NAME	AGE
workload.carto.run/tanzu-java-web-app	109s

NAME	READY	STATUS	URL
gitrepository.source.toolkit.fluxcd.io/tanzu-java-web-app			https://github.com/sample-accelerators/tanzu-java-web-app
sample-accelerators/tanzu-java-web-app	True	Fetches revision: main/872ff44c8866b7805fb2425130edb69a9853bfd	
		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/tanzu-java-web-app	90s

NAME	SUCCEDED	REASON	START
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/fo/tanzu-java-web-app@sha256:1d5bc4d3d1ffeb8629fbb721fcd1c4d28b896546e005f1efd98fbc4e79b7552c	True

NAME	PHASE	SCAN
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/fo/tanzu-java-web-app@sha256:1d5bc4d3d1ffeb8629fbb721fcd1c4d28b896546e005f1efd98fbc4e79b7552c		14s

NAME	READY	REASON
AGE		
podintention.conventions.apps.tanzu.vmware.com/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	READY
LATESTCREATED	LATESTREADY	
REASON		
service.serving.knative.dev/tanzu-java-web-app	http://tanzu-java-web-app.developer.example.com	Unkn
	tanzu-java-web-app-00001	wn
	IngressNotConfigured	

If the source or image scan has a “Failed” phase, then the scan has failed compliance and the supply chain stops.

Query for vulnerabilities

Scan reports are automatically saved to the [Supply Chain Security Tools - Store](#), and can be queried

for vulnerabilities and dependencies. For example, open-source software (OSS) or third party packages.

1. Query the tanzu-java-web-app image dependencies and vulnerabilities with the following commands:

```
insight image get --digest DIGEST
insight image vulnerabilities --digest DIGEST
```

`DIGEST` is the component version, or image digest printed in the `KUBECTL GET` command.

Important: The `Insight CLI` is separate from the Tanzu CLI.

See [Tanzu Insight plug-in overview](#) additional information and examples.

Congratulations! You have successfully deployed your application on the Tanzu Application Platform.

Through the next two sections to learn about recommended supply chain security best practices and access to a powerful Services Journey experience on the Tanzu Application Platform by enabling several advanced use cases.

Section 4: Configure image signing and verification in your supply chain

In this section, you are about to:

- 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 your image builds

1. Configure Tanzu Build Service to sign your container image builds by using cosign. See [Managing Image Resources and Builds](#) for instructions.
2. Create a `values.yaml` file, and install the sign supply chain security tools and image policy web-hook. See [Install Supply Chain Security Tools - Sign](#) for instructions.
3. Configure a `ClusterImagePolicy` resource to verify image signatures when deploying resources. The resource must be named `image-policy`.

For example:

```
---
apiVersion: signing.apps.tanzu.vmware.com/v1beta1
kind: ClusterImagePolicy
metadata:
  name: image-policy
spec:
  verification:
    exclude:
      resources
      namespaces:
```

```

- kube-system
- test-namespace
keys:
- name: first-key
  publicKey: |
    -----BEGIN PUBLIC KEY-----
    <content ...>
    -----END PUBLIC KEY-----
images:
- namePattern: registry.example.org/myproject/*
  keys:
  - name: first-key

```

Note: System namespaces specific to your cloud provider might need to be excluded from the policy.

To prevent the Image Policy Webhook from blocking components of Tanzu Application Platform, VMware recommends configuring exclusions for Tanzu Application Platform system namespaces listed in [Create a ClusterImagePolicy resource](#).

When you apply the [ClusterImagePolicy](#) resource, your cluster requires valid signatures for all images that match the `namePattern`: you define in the configuration. For more information about configuring an image signature policy, see [Configuring Supply Chain Security Tools - Sign](#).

Next steps

- [Overview for Supply Chain Security Tools - Sign](#)
- [Configuring Supply Chain Security Tools - Sign](#)
- [Supply Chain Security Tools - Sign known issues](#)

Scan and Store: Introducing vulnerability scanning and metadata storage to your Supply Chain

Overview

This feature set allows an application operator to introduce source code and image vulnerability scanning, and scan-time rules, to their Tanzu Application Platform Supply Chain. The scan-time rules prevent critical vulnerabilities from flowing to the supply chain unresolved.

[Supply Chain Security Tools - Store](#) takes the vulnerability scanning results and stores them. Users can query for information about CVEs, images, packages, and their relationships by using the `tanzu insight` CLI plug-in, or directly from the API.

Features

- 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 using Open Policy Agent
- Produce vulnerability scan results and post them to the Supply Chain Security Tools - 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?

To try the scan and store features as individual one-off scans, see [Scan samples](#).

To try the scan and store features in a supply chain, see [Section 3: Add testing and security scanning to your application](#).

Next steps

- [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](#)

Section 5: Consuming services on Tanzu Application Platform

In this section you will learn about working with backing services such as RabbitMQ, PostgreSQL and MySQL as part of Tanzu Application Platform.

Particular focus will be given to binding application workloads to service instances, which is the most common use case for 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 any Kubernetes resource which exposes its capability through a well-defined interface. For example, you could consider Kubernetes resources that have `MySQL` as the API Kind to be MySQL service instances. These resources expose their capability over the MySQL protocol. Other examples include resources that have `PostgreSQL` or `RabbitmqCluster` as the API Kind.

Service bindings

Service binding refers to a mechanism in which connectivity information such as service instance credentials are automatically communicated to application workloads. Tanzu Application Platform uses a standard named [Service Binding for Kubernetes](#) to implement this mechanism. To fully understand the services aspect of Tanzu Application Platform, you must learn about this standard.

Resource claims

Resource claims are inspired in part by [Persistent Volume Claims](#) in Kubernetes. Resource Claims provide a mechanism for users to “claim” service instance resources 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. [RabbitMQ Cluster Operator for Kubernetes](#)
2. [VMware SQL with Postgres for Kubernetes](#)
3. [VMware SQL with MySQL for Kubernetes](#)

Whether a service is compatible with Tanzu Application Platform is 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 there is at least one API resource type adhering to the [Provisioned Service](#) duck type defined by the [Service Binding for Kubernetes](#) standard. 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 CRDs and corresponding controllers to reconcile the resources of the CRDs, as is the case with the three Kubernetes Operators listed above.

User roles and responsibilities

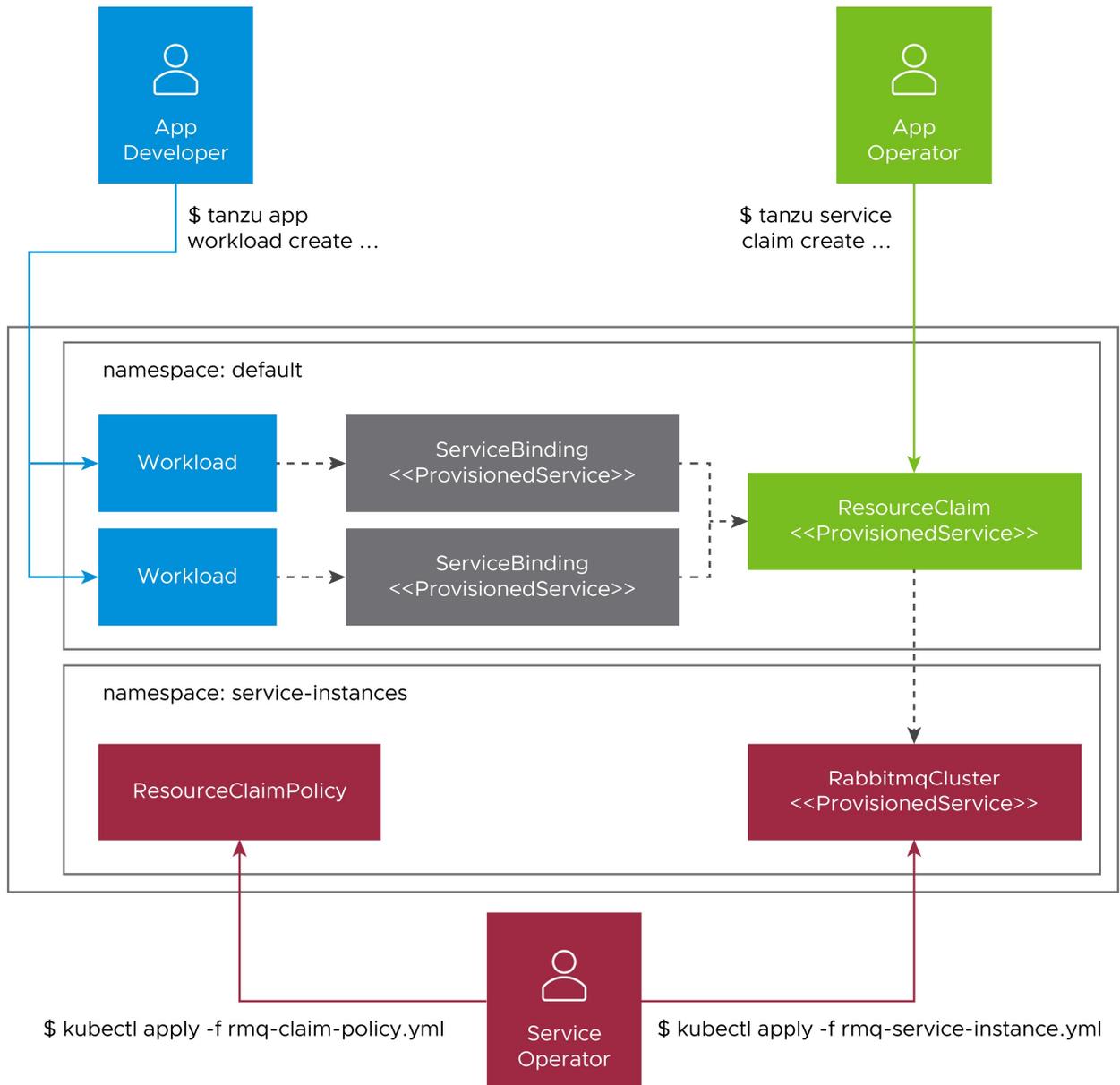
It is important to understand the user roles for services on Tanzu Application Platform along with the responsibilities assumed of each. The following table describes each user role.

User role	Exists as a default role in Tanzu Application Platform?	Responsibilities
Service operator	No (might be introduced in a future release)	<ul style="list-style-type: none"> • 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
-----------------------	---	--

Walkthrough

This section guides you through deploying two application workloads and learning how to configure 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. The following diagram depicts a summary of what this section covers.



Bear the following observations in mind as you work through this section.

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 lifecycle 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.
- 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. Have setup the [default](#) namespace to use installed packages and use it as your developer namespace. For more information, see [Set up developer namespaces to use installed packages](#)).
4. Ensure your Tanzu Application Platform cluster can pull source code from GitHub.
5. Ensure your Tanzu Application Platform cluster can pull the images required by the [RabbitMQ Cluster Kubernetes Operator](#).

Set up a service

This section covers the following:

- Installing the [RabbitMQ Cluster Kubernetes Operator](#)
- Creating the RBAC rules to grant Tanzu Application Platform permission to interact with the newly-installed APIs provided by the RabbitMQ Cluster 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**.

Note: Although this walkthrough uses the RabbitMQ Cluster Kubernetes Operator as an example, the set up steps remain mostly the same for any compatible Operator.

To set up a service:

1. Use [kapp](#) to install the RabbitMQ Cluster Kubernetes Operator by running:

```
kapp -y deploy --app rmq-operator --file https://github.com/rabbitmq/cluster-operator/releases/download/v1.9.0/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:
    resourceclaims.services.apps.tanzu.vmware.com/controller: "true"
rules:
- apiGroups: ["rabbitmq.com"]
  resources: ["rabbitmqclusters"]
  verbs: ["get", "list", "watch", "update"]
```

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

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

3. In a file named `rabbitmqcluster-app-operator-reader.yaml`, define RBAC rules that permit the users of the cluster to interact with the new APIs. For example, to permit application operators to get, list, and watch for `RabbitmqCluster` service instances, apply the following RBAC `ClusterRole`, labeled so that the rules are aggregated to the `app-operator` role:

```
# rabbitmqcluster-app-operator-reader.yaml
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: rabbitmqcluster-app-operator-reader
  labels:
    apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access: "true"
rules:
- apiGroups: ["rabbitmq.com"]
  resources: ["rabbitmqclusters"]
  verbs: ["get", "list", "watch"]
```

4. Apply `rabbitmqcluster-app-operator-reader.yaml` by running:

```
kubectl apply -f rabbitmqcluster-app-operator-reader.yaml
```

3. Make the new API discoverable.

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

```
# rabbitmqcluster-clusterresource.yaml
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ClusterResource
metadata:
  name: rabbitmq
```

```
spec:
  shortDescription: It's a RabbitMQ cluster!
  longDescription: A consistent and easy way to deploy RabbitMQ clusters
to Kubernetes and run them, including "day two" (continuous) operations.
  resourceRef:
    group: rabbitmq.com
    kind: RabbitmqCluster
```

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

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

After applying this resource, it will be listed in the output of the `tanzu service types 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 if desired.

2. Find the list of services that are available on your cluster by running:

```
tanzu service types list
```

Expected output:

```
Warning: This is an ALPHA command and may change without notice.

NAME          DESCRIPTION          APIVERSION          KIND
rabbitmq      It's a RabbitMQ cluster!  rabbitmq.com/v1beta1  RabbitmqCluster
```



Note

: If you see `No service types found.`, ensure you have completed the steps in [Set up a service](#) earlier in this walkthrough.

3. Create a `RabbitmqCluster` service instance.

1. Create a file named `rmq-1-service-instance.yaml` using the `APIVERSION` and `KIND` from the output of the `tanzu service types list` command:

```
# 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
```

4. 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 that are 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 instance list` to view details about service instances.
- 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 them having to create and manage the services themselves. Resource claims provide a mechanism for application operators to say what they want, without having to worry about anything that goes into providing what they want. 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 will use the `tanzu service claim create` command to create 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 information needed to make a resource claim by running:

```
tanzu service instance list -A
```

Expected output:

```
Warning: This is an ALPHA command and may change without notice.

NAMESPACE          NAME    KIND              SERVICE TYPE  AGE
service-instances  rmq-1  RabbitmqCluster  rabbitmq      24h
```

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
```

In the next section you will see how to inspect the claim and to then use it to bind to application workloads.

Bind an application workload to the service instance

This section covers the following:

- 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

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

As a final step, you must create application workloads and to bind them to the service instance using the claim.

In Tanzu Application Platform Service bindings are created when application workloads that specify `.spec.serviceClaims` are created. In this section, you will see how to create such workloads 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:

```
Warning: This is an ALPHA command and may change without notice.

NAME    READY  REASON
rmq-1   True
```

2. Retrieve detailed information about the claim by running:

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

Expected output:

```
Warning: This is an ALPHA command and may change without notice.

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 our `RabbitmqCluster` service instance, is displayed.

Advanced use cases and further reading

There are a couple more advanced service use cases that not covered in the procedures in this topic, such as Direct Secret References and Dedicated Service Clusters.

Advanced Use Case	Short Description
Direct Secret References	Binding to services running external to the cluster, for example, and in-house oracle database. Binding to services that are not conformant with the binding specification.
Dedicated Service Clusters	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](#)

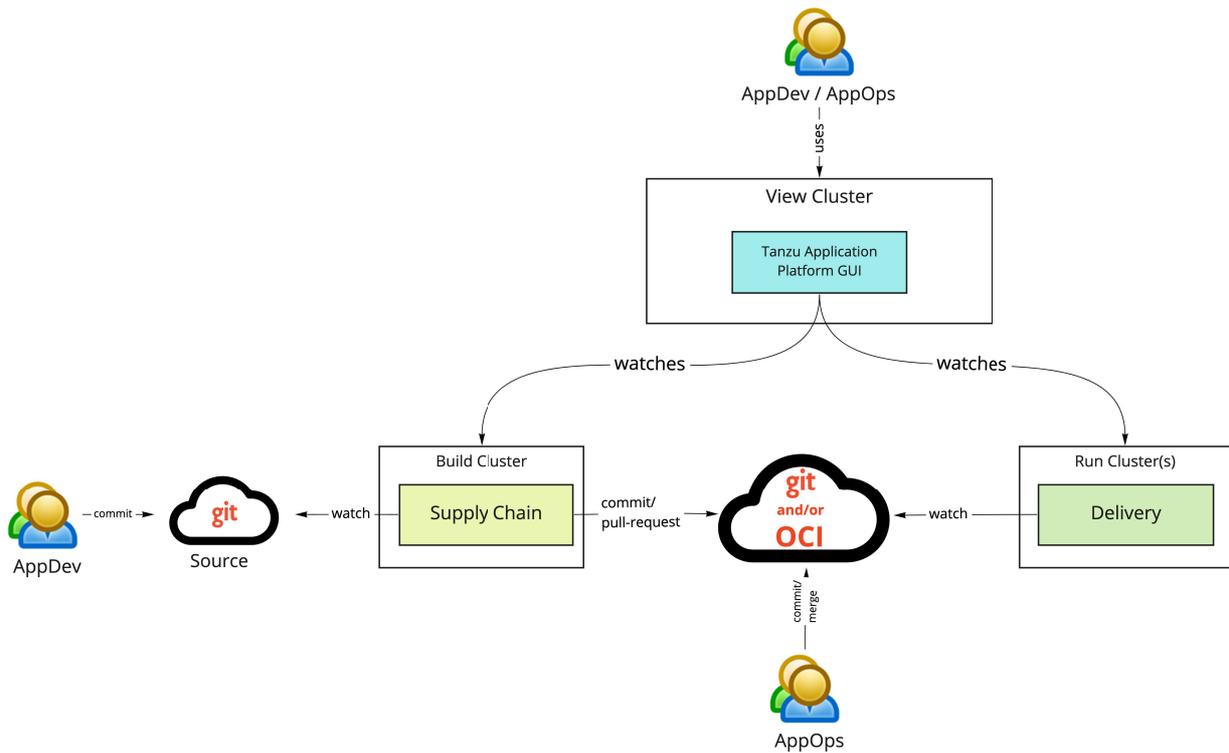
Overview of multicluster Tanzu Application Platform

You can install Tanzu Application Platform 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 three 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

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: `http://tap-gui.INGRESS-DOMAIN`, where `INGRESS-DOMAIN` is the DNS domain you set to point to the shared Contour installation in the `tanzu-system-ingress` namespace with the service `envoy`.
3. Verify that you followed [Multicluster setup](#) for the Supply Chain Security Tools - Store.

Install Build clusters

To install the Build profile cluster, follow the steps described in [Installing the Tanzu Application Platform package and profiles](#) by using a reduced values file as shown in [Build profile](#).

Install Run clusters

To install the Run profile cluster:

1. Follow the steps described in [Install the Tanzu Application Platform package and profiles](#) by using a reduced values file as shown in [Run profile](#).
2. To use Application Live View, set the `INGRESS-DOMAIN` for `appliveview_connector` to match the value you set on the View profile for the `appliveview` in the values file.

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

In this topic, you will validate the implementation of a multicluster topology by taking a sample workload and passing it through the supply chains on the Build and Run clusters. You can take various approaches to configuring the supply chain in this topology, but the following procedures validate the most basic capabilities.

By following the steps in this topic, you will build an application on the Build profile clusters and run

the application on the Run profile clusters. You will be able to view the workload and associated objects from Tanzu Application Platform GUI interface on the View profile cluster.

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 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 similar 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      RE
ADY   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
```

- 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                                     TLS SECRET   STATUS   FQDN   STATUS   DESC
RIPTION
tanzu-java-web-app-contour-a98df54e3629c5ae9c82a395501ee1fdtanz  tanzu-java-we
b-app.default.svc.cluster.local   valid      Valid HTTP
proxy
tanzu-java-web-app-contour-e1d997a9ff9e7dfb6c22087e0ce6fd7ftanz  tanzu-java-we
b-app.default.apps.run.multi.kapplegate.com   valid      Valid HTTP
proxy
tanzu-java-web-app-contour-tanzu-java-web-app.default             tanzu-java-we
b-app.default                               valid      Valid HTTP
proxy
tanzu-java-web-app-contour-tanzu-java-web-app.default.svc        tanzu-java-we
b-app.default.svc                       valid      Valid HTTP
proxy
```

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!".

- View the component in Tanzu Application Platform GUI, by following [these steps](#) and using the [catalog file](#) from the sample accelerator in GitHub.

Build profile

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"
  tanzunet_username: "TANZUNET-USERNAME"
  tanzunet_password: "TANZUNET-PASSWORD"
```

```

supply_chain: testing_scanning
ootb_supply_chain_testing_scanning:
  registry:
    server: "SERVER-NAME"
    repository: "REPO-NAME"
  gitops:
    ssh_secret: "SSH-SECRET-KEY"
scanning:
  metadataStore:
    url: "METADATA-STORE-URL-ON-VIEW-CLUSTER"
    caSecret:
      name: store-ca-cert
      importFromNamespace: metadata-store-secrets
    authSecret:
      name: store-auth-token
grype:
  namespace: "MY-DEV-NAMESPACE" # (optional) Defaults to default namespace.
  targetImagePullSecret: "TARGET-REGISTRY-CREDENTIALS-SECRET"

```

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.
- `TANZUNET-USERNAME` and `TANZUNET-PASSWORD` are the email address and password that you use to log in to VMware Tanzu Network. Your VMware Tanzu Network credentials enable you to configure the dependencies updater. This resource accesses and installs the build dependencies (buildpacks and stacks) Tanzu Build Service needs on your cluster. It can also optionally keep these dependencies up to date as new versions are released on VMware Tanzu Network. This credential can also be configured by using a Secret reference. For more information, see [Install Tanzu Build Service](#).
- `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.
- `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.
- `METADATA-STORE-URL-ON-VIEW-CLUSTER` references the URL of the Supply Chain Security Tools (SCST) - Store deployed on the View cluster. For more information, see [SCST - Store's Ingress and multicluster support](#) for additional details.
- `MY-DEV-NAMESPACE` is the namespace where you want to deploy the `ScanTemplates`. This is the namespace where the scanning feature runs.
- `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the Secret that contains the credentials to pull an image from the registry for scanning. If built images are pushed to the same registry as Tanzu Application Platform images, you can reuse the `tap-registry` Secret created in [Add the Tanzu Application Platform package repository](#).

When you install Tanzu Application Platform, it is bootstrapped with a set of dependencies (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 in the descriptor file on [Tanzu Network](#). The current installed version of the descriptor is `100.0.293`. Sometimes the dependencies get out of date and require updates. You can do this using a [manual process in a CI/CD context](#), or an [automatic update process](#) in the background by Tanzu Application Platform.

Run profile

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.
supply_chain: basic

cnrs:
  domain_name: INGRESS-DOMAIN

contour:
  envoy:
```

```

service:
  type: LoadBalancer #NodePort can be used if your Kubernetes cluster doesn't support LoadBalancing

appliveview_connector:
  backend:
    sslDisabled: TRUE-OR-FALSE-VALUE
    host: appliveview.APP-LIVE-VIEW-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.
- `APP-LIVE-VIEW-INGRESS-DOMAIN` is the subdomain you setup on the View profile cluster. This matches the value key `appliveview.ingressDomain`. Include the default host name `appliveview.` ahead of the domain.

View profile

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.

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

learningcenter:
  ingressDomain: "DOMAIN-NAME"

tap_gui:
  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
  kubernetes:
    serviceLocatorMethod:
      type: 'multiTenant'
    clusterLocatorMethods:
      - type: 'config'
      clusters:
        - url: CLUSTER_URL

```

```

    name: CLUSTER_NAME
    authProvider: serviceAccount
    serviceAccountToken: CLUSTER_TOKEN
    skipTLSVerify: TRUE-OR-FALSE-VALUE

metadata_store:
  app_service_type: LoadBalancer # (optional) Defaults to LoadBalancer. Change to Node
  Port for distributions that don't support LoadBalancer

appliveview:
  ingressEnabled: TRUE-OR-FALSE-VALUE
  ingressDomain: APP-LIVE-VIEW-INGRESS-DOMAIN

```

Where:

- `DOMAIN-NAME` has a value such as `learningcenter.example.com`.
- `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 troubleshooting information to help resolve issues with Tanzu Application Platform:

- [Troubleshoot installing Tanzu Application Platform](#)
- [Troubleshoot using Tanzu Application Platform](#)
- [Troubleshoot Tanzu Application Platform components](#)

Troubleshoot installing Tanzu Application Platform

In this topic, you'll find troubleshooting information to help resolve issues installing Tanzu Application Platform.

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 PACKAGE-NAME grype -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 images: Working with registry.tanzu.vmware.com/app-live-view/application-live-view-install-bundle@sha256:b13b9ba81bcc985d76607cfc04bcbb8829b4cc2820e64a99e0af840681da12aa: GET https://registry.tanzu.vmware.com/v2/app-live-view/application-live-view-install-bundle/manifests/sha256:b13b9ba81bcc985d76607cfc04bcbb8829b4cc2820e64a99e0af840681da12aa: UNAUTHORIZED: unauthorized to access repository: app-live-view/application-live-view-install-bundle, action: pull: unauthorized to access repository: app-live-view/application-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
0.5.1	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 VERSION	DESCRIPTION	PACKAGE NAME
accelerator	1.0.1	Reconcile succeeded	accelerator.apps.tanzu.vmware.com
	109m		
api-portal	1.0.9	Reconcile succeeded	api-portal.tanzu.vmware.com
	119m		
appliveview	1.0.2-build.2	Reconcile succeeded	run.appliveview.tanzu.vmware.com
	109m		
appliveview-conventions	1.0.2-build.2	Reconcile succeeded	build.appliveview.tanzu.vmware.com
	109m		
buildservice	1.5.0	Reconcile succeeded	buildservice.tanzu.vmware.com
	119m		
cartographer	0.2.1	Reconcile succeeded	cartographer.tanzu.vmware.com

cert-manager	117m	cert-manager.tanzu.vmware.com
1.5.3+tap.1		Reconcile succeeded
cnrs	119m	cnrs.tanzu.vmware.com
1.1.0		Reconcile succeeded
contour	109m	contour.tanzu.vmware.com
1.18.2+tap.1		Reconcile succeeded
conventions-controller	117m	controller.conventions.apps.tanzu.vmware.com
0.5.0		Reconcile succeeded
developer-conventions	117m	developer-conventions.tanzu.vmware.com
0.5.0		Reconcile succeeded
fluxcd-source-controller	109m	fluxcd.source.controller.tanzu.vmware.com
0.16.1		Reconcile succeeded
grype	119m	grype.scanning.apps.tanzu.vmware.com
1.0.0		Reconcile failed: Error (see .status.usefulErrorMessage for details)
image-policy-webhook	109m	image-policy-webhook.signing.apps.tanzu.vmware.com
1.0.1		Reconcile succeeded
learningcenter	117m	learningcenter.tanzu.vmware.com
0.1.0		Reconcile succeeded
learningcenter-workshops	109m	workshops.learningcenter.tanzu.vmware.com
0.1.0		Reconcile succeeded
metadata-store	103m	metadata-store.apps.tanzu.vmware.com
1.0.2		Reconcile succeeded
ootb-delivery-basic	117m	ootb-delivery-basic.tanzu.vmware.com
0.6.1		Reconcile succeeded
ootb-supply-chain-basic	103m	ootb-supply-chain-basic.tanzu.vmware.com
0.6.1		Reconcile succeeded
ootb-templates	103m	ootb-templates.tanzu.vmware.com
0.6.1		Reconcile succeeded
scanning	109m	scanning.apps.tanzu.vmware.com
1.0.0		Reconcile succeeded
service-bindings	119m	service-bindings.labs.vmware.com
0.6.0		Reconcile succeeded
services-toolkit	119m	services-toolkit.tanzu.vmware.com
0.5.1		Reconcile succeeded
source-controller	119m	controller.source.apps.tanzu.vmware.com
0.2.0		Reconcile succeeded
spring-boot-conventions	119m	spring-boot-conventions.tanzu.vmware.com
0.3.0		Reconcile succeeded
tap	109m	tap.tanzu.vmware.com

```

1.0.1      Reconcile failed: Error (see .status.usefulErrorMessage for
details)   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.

- 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`.

- 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](#).
- 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*.

Troubleshoot using Tanzu Application Platform

In this topic, you'll find troubleshooting information to help resolve issues using Tanzu Application Platform.

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:latest'
```

```
: Error while preparing a transport to talk with the registry: Unable to create round
tripper: GET https://auth.ipv6.docker.com/token?scope=repository%3Ashaileshp2922%2Fbui
ld-service%2Ftanzu-java-web-app%3Apush%2Cpull&service=registry.docker.io: unexpected s
tatus 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='[{"op"
: "add", "path": "/rules/-", "value": {"apiGroups": [""], "resources": ["secrets"], "ver
bs": ["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 creating: Internal error occurred: failed calling webhook "image-policy-webhook.signing.apps.tanzu.vmware.com": Post "https://image-policy-webhook-service.image-policy-system.svc:443/signing-policy-check?timeout=10s": no endpoints available for service "image-policy-webhook-service"
```

```
Events:
  Type          Reason          Age          From                      Message
  ----          -
  Warning       FailedCreate    10m         replicaset-controller     Error creating: Internal error occurred: failed calling webhook "image-policy-webhook.signing.apps.tanzu.vmware.com": Post
```

```
"https://image-policy-webhook-service.image-policy-system.svc:443/signing-policy-check?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 disable signature verification in your cluster.

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

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

2. Delete `MutatingWebhookConfiguration` by running:

```
kubectl delete MutatingWebhookConfiguration image-policy-mutating-webhook-configuration
```

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-policy-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-store-db user=metadata-store-user database=metadata-store`: server error (FATAL: password authentication failed for user "metadata-store-user" (SQLSTATE 28P01))
```

Explanation

The database password has 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-store-db user=metadata-store-user database=metadata-store`: server error (FATAL: password authentication failed for user "metadata-store-user" (SQLSTATE 28P01))
```

Explanation

The database password has 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":{"storage class.kubernetes.io/is-default-class":"true"}}}'
```

Supply Chain Security Tools - Sign rejects images

Supply Chain Security Tools - Sign rejects images from private registries.

Explanation

The image is deployed to a non-default namespace.

Solution

Make the private registry secret available to the `default` namespace.

Supply Chain Security Tools - Scan unable to decode CycloneDX

Supply Chain Security Tools - Scan has a known issue where it sets the phase of a scan to `Error` with the message `unable to decode cyclonedx`. This is an intermittent issue that cuts the CycloneDX XML stream to the logs such that the scan controller is unable to process the results properly.

Explanation The root cause of the problem is unknown.

Workaround: See the [Troubleshooting Guide](#) for how to exit this error state.

Troubleshoot Tanzu Application Platform components

For component-level troubleshooting, see these topics:

- [Troubleshoot Tanzu Application Platform GUI](#)
- [Troubleshoot Convention Service](#)
- [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](#)
- [Troubleshoot Tanzu Build Service \(FAQ\)](#)

Uninstalling Tanzu Application Platform

This document describes how to uninstall Tanzu Application Platform packages from the Tanzu Application Platform package repository.

The process for uninstalling Tanzu Application Platform is made up of three tasks:

- [Delete the Packages](#)
- [Delete the Tanzu Application Platform Package Repository](#)
- [Remove Tanzu CLI, plug-ins, and associated files](#)

Delete the packages

To delete the installed packages:

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 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-application-platform-package-repository
--namespace tap-install
- Deleting package repository 'tanzu-application-platform-package-repository'..
.
Deleted package repository 'tanzu-application-platform-package-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
```

Component documentation

Tanzu Application Platform is a modular, composable platform consisting of various 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.

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 [Installing the Tanzu Application Platform Package 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 cert-manager, Contour, and FluxCD](#)
- [Install Cloud Native Runtimes](#)
- [Install Convention Service](#)
- [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 - Sign](#)
- [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 0.4
.2                        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	0.6
.0	Reconcile succeeded	
source-controller	controller.source.apps.tanzu.vmware.com	0.2
.0	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	

Set up developer namespaces to use 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
```

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

- 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
```

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

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

Enable additional users 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.

Tanzu CLI

Tanzu CLI plug-ins

- `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 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.

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 document describes how to install the Apps CLI plug-in.



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

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.

Verify that there is an `apps` entry in the `cli/manifest.yaml` file:

```
plugins:
...
- name: apps
  description: Applications on Kubernetes
  versions: []
```

Create a workload

This document describes how to create a workload from example source code with Tanzu Application Platform.

Prerequisites

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

- Install Kubernetes command line tool (kubectl). For information about installing kubectl, see [Install Tools](#) in the Kubernetes documentation.
- 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 installed packages](#).

Get started with an example workload

Here is how you can get started with an example workload.

- To name the workload and specify a source code location to create the workload from, 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` is 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`.

Check build logs

Once the workload is created, you can tail the workload to view the build and runtime logs.

- 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.

Create a workload from local source code

You can create a workload using code from a local folder.

- Inside the folder that contains the source code, run:

```
tanzu apps workload create pet-clinic --local-path . --source-image springio/pe  
tclinic
```

Respond `y` to the prompt about publishing local source code if the image needs to be updated.

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 for the local source code.

Bind a service to a workload

Multiple services can be configured for each workload. The cluster supply chain is in charge of provisioning those services.

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

```
tanzu apps workload update pet-clinic --service-ref "database=services.tanzu.vmware.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 `{name}={apiVersion}:{kind}:{name}`. For more details, refer to [update command](#).

Next steps

You can add environment variables, export definitions, and use flags with these [commands](#). The following procedure includes example environment variables and flags.

1. To add environment variables, run:

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

2. To export the workload definition into git, or to migrate to another environment, run:

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

3. To see flags available for the workload commands, run:

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

Command reference

- [Tanzu apps](#)
 - ✦ [Workload](#)
 - [Workload apply](#)
 - [Workload create](#)
 - [Workload update](#)
 - [Workload get](#)
 - [Workload delete](#)
 - [Workloads list](#)
 - [Workload tail](#)
- [Cluster supply chain](#)
 - ✦ [List cluster supply chain](#)

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
ncontext defined by kubeconfig)
-h, --help          help for apps
--kubeconfig file  kubeconfig file (default is $HOME/.kube/config)
--no-color          disable 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 may 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
ncontext defined by kubeconfig)
--kubeconfig file  kubeconfig file (default is $HOME/.kube/config)
--no-color          disable 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 will be created.

Workload configuration options include:

- source code to build
- runtime resource limits
- environment variables
- services to bind

```
tanzu apps workload apply [name] [flags]
```

Examples

```
tanzu apps workload apply --file workload.yaml
```

Options

```

--annotation "key=value" pair    annotation is represented as a "key=value" pair
, or "key-" to remove. This flag may be specified 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, or "key-" to remove. This flag may be specified multiple times
--debug                            put the workload in debug mode, --debug=false t
o disable
--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, or "key-" to remove. This flag may be specified 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 repo to checkout
--git-commit SHA                 commit SHA within the git repo to checkout
--git-repo url                   git url to remote source code
--git-tag tag                     tag within the git repo 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, or
"key-" to remove. This flag may be specified multiple times
  --limit-cpu cores                 the maximum number CPU cores allowed (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 disable
  --local-path path                path on the local file system to a directory of
source code to build for the workload
  -n, --namespace name            kubernetes namespace (defaulted from kube confi
g)
  --param "key=value" pair         additional parameters represented as a "key=val
ue" pair, or "key-" to remove. This flag may be specified multiple times
  --request-cpu cores              the minimum number of CPU cores required (500m
= .5 cores)
  --request-memory bytes           the minimum amount of memory required, in bytes
(500Mi = 500MiB = 500 * 1024 * 1024)
  --service-ref object reference  object reference for a service to bind to the w
orkload "database=rabbitmq.com/v1beta1:RabbitmqCluster:[my-broker-ns]:my-broker", or "
database-" to delete. This flag may be specified multiple times.
  --source-image image             destination image repository where source code
is staged before being built
  --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                       disable 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
- Runtime resource limits
- Environment variables
- Services to bind

```
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
```

Options

```

--annotation "key=value" pair    annotation is represented as a "key=value" pair
, or "key-" to remove. This flag may be specified 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, or "key-" to remove. This flag may be specified multiple times
--debug                            put the workload in debug mode, --debug=false t
o disable
--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, or "key-" to remove. This flag may be specified 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 repo to checkout
--git-commit SHA                 commit SHA within the git repo to checkout
--git-repo url                   git url to remote source code
--git-tag tag                    tag within the git repo 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, or
"key-" to remove. This flag may be specified multiple times
--limit-cpu cores                the maximum number of CPU cores allowed (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 disable
--local-path path                path on the local file system to a directory of
source code to build for the workload
-n, --namespace name            kubernetes namespace (defaulted from kube confi
g)
--param "key=value" pair        additional parameters represented as a "key=val
ue" pair, or "key-" to remove. This flag may be specified multiple 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-ref object reference  object reference for a service to bind to the w
orkload "database=rabbitmq.com/v1beta1:RabbitmqCluster:[my-broker-ns]:my-broker", or "
database-" to delete. This flag may be specified multiple times.
--source-image image         destination image repository where source code
is staged before being built
--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                disable 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
- runtime resource limits
- environment variables
- services to bind

```
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
```

Options

```

--annotation "key=value" pair    annotation is represented as a "key=value" pair
, or "key-" to remove. This flag may be specified 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, or "key-" to remove. This flag may be specified multiple times
--debug                            put the workload in debug mode, --debug=false t
o disable
--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, or "key-" to remove. This flag may be specified 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 repo to checkout
--git-commit SHA                 commit SHA within the git repo to checkout
--git-repo url                   git url to remote source code
--git-tag tag                    tag within the git repo 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, or
"key-" to remove. This flag may be specified multiple times
--limit-cpu cores                the maximum number of CPU cores allowed (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 disable
--local-path path               path on the local file system to a directory of
source code to build for the workload
-n, --namespace name            kubernetes namespace (defaulted from kube confi
g)
--param "key=value" pair        additional parameters represented as a "key=val
ue" pair, or "key-" to remove. This flag may be specified multiple times
--request-cpu cores             the minimum number of CPU cores required (500m
= .5 cores)
--request-memory bytes          the minimum amount of memory required, in bytes
(500Mi = 500MiB = 500 * 1024 * 1024)
--service-ref object reference  object reference for a service to bind to the w
orkload "database=rabbitmq.com/v1beta1:RabbitmqCluster:[my-broker-ns]:my-broker", or "
database-" to delete. This flag may be specified multiple times.
--source-image image            destination image repository where source code
is staged before being built
--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         disable 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"

```

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         disable 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 workload; 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 (default 1m0s)
-y, --yes            accept all prompts
```

Options inherited from parent commands

```
--context name      name of the kubeconfig context to use (default is current-context defined by kubeconfig)
--kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
--no-color           disable 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
ncontext defined by kubeconfig)
--kubeconfig file  kubeconfig file (default is $HOME/.kube/config)
--no-color          disable 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
ncontext defined by kubeconfig)
--kubeconfig file  kubeconfig file (default is $HOME/.kube/config)
--no-color          disable 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
ncontext defined by kubeconfig)
--kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
--no-color          disable 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 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
ncontext defined by kubeconfig)
--kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
--no-color          disable 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 should 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: java
spec:
  source:
    git:
      url: https://github.com/sample-accelerators/spring-petclinic
      ref:
        tag: tap-1.1
```

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"
```

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 Build profile or Tanzu Application Platform View profile.

1. [Install the Tanzu Insight CLI plug-in](#)
2. [Configure target endpoint and certificate](#)
3. [Configure access tokens](#)

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.
- Currently, the Tanzu Insight plug-in only supports macOS and Linux.

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.

Note: All [required setup](#) must be completed in addition to configuring the CLI.

Set the target and certificate authority certificate

Set the target endpoint and CA certificate by running:

```
tanzu insight config set-target https://metadata-store-app.metadata-store.svc.cluster.local:PORT --ca-cert PATH
```

Where

- `PORT` is the target endpoint port
- `PATH` is the direct path to the CA certificate

For example:

```
$ tanzu insight config set-target https://metadata-store-app.metadata-store.svc.cluster.local:8443 --ca-cert /tmp/ca.crt
```

```
Using config file: /Users/username/.config/tanzu/insight/config.yaml
Setting trustedcacert in config
Setting endpoint in config to: https://metadata-store-app.metadata-store.svc.cluste
```

```
r.local:8443
✓ Success: Set Metadata Store endpoint
```

Check the connection

Check that your configuration is correct and you are able to make a connection.

```
tanzu insight health
```

For example:

```
$ tanzu insight health
Success: Reached Metadata Store!
```

Configure target endpoint and certificate

The connection to the Store requires TLS encryption, the configuration depends on the kind of installation. Use the following instructions to set up the TLS connection according to the type of your setup:

- [Use Ingress](#)
- [Not use Ingress](#)
 - ◊ [Use LoadBalancer](#)
 - ◊ [Use NodePort](#)

Note: [NodePort](#) is commonly used with local clusters such as kind or minikube.

Use [Ingress](#)

When using an [Ingress setup](#), the Store creates a specific TLS Certificate for HTTPS communications under the `metadata-store` namespace.

To get such certificate, run the following command:

```
kubectl get secret ingress-cert -n metadata-store -o json | jq -r '.data."ca.crt"' | base64 -d > insight-ca.crt
```

The endpoint host is set to `metadata-store.<ingress-domain>`, for example, `metadata-store.example.domain.com`). This value matches the value of `ingress_domain`.

If no accessible DNS record exists for such 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 the target by running:

```
tanzu insight config set-target https://$METADATA_STORE_DOMAIN --ca-cert insight-ca.crt
```

Not use Ingress

If you install the Store without using the Ingress alternative, you must use a different Certificate resource for HTTPS communication. In this case, query the `app-tls-cert` to get the CA Certificate:

```
kubectl get secret app-tls-cert -n metadata-store -o json | jq -r '.data."ca.crt"' | base64 -d > insight-ca.crt
```

Use LoadBalancer

To use a `LoadBalancer` configuration, you must find the external IP address of the `metadata-store-app` service by using `kubectl`.



Note

: For all `kubectl` commands, use the `--namespace metadata-store` flag.

```
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
```

Set the target by running:

```
tanzu insight config set-target https://$METADATA_STORE_DOMAIN:$METADATA_STORE_PORT --ca-cert insight-ca.crt
```

Use NodePort

To use `NodePort`, you must obtain the CA certificate by following the instructions in [Not use Ingress](#), then [Configure port forwarding](#) and [Modify your /etc/hosts file](#).

Configure port forwarding

When using `NodePort`, configure port forwarding for the service so the CLI can access Supply Chain Security Tools - Store. Run:

```
kubectl port-forward service/metadata-store-app 8443:8443 -n metadata-store
```

Note: You must run this command in a separate terminal window.

Modify your `/etc/hosts` file

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
```

Set the target by running:

```
tanzu insight config set-target https://$METADATA_STORE_DOMAIN:$METADATA_STORE_PORT --ca-cert insight-ca.crt
```

Configure access tokens

Service accounts are required to generate the access tokens.

The access token is a `Bearer` token used in the http request header `Authorization`. (ex.

`Authorization: Bearer eyJhbGciOiJSUzI1NiIsImtpZCI6IjhmV0...`)

By default, Supply Chain Security Tools - Store comes with `read-write` service account installed. This service account is cluster-wide.

Service accounts

You can create two types of service accounts:

1. Read-only service account - only able to use `GET` API requests
2. Read-write service account - full access to the API requests

Read-only service account

As a part of the Store installation, the `metadata-store-read-only` cluster role is created by default. This cluster role allows the bound user to have `get` access to all resources. To bind to this cluster role, run the following command:

```
kubectl apply -f - -o yaml << EOF
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: metadata-store-ready-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
automountServiceAccountToken: false
EOF

```

If you do not want to bind to a cluster role, create your own read-only role in the `metadata-store` namespace with a service account. The following example command creates a service account named `metadata-store-read-client`:

```

kubectl apply -f - --o yaml << EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: metadata-store-ro
  namespace: metadata-store
rules:
- resources: ["all"]
  verbs: ["get"]
  apiGroups: [ "metadata-store/v1" ]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: metadata-store-ro
  namespace: metadata-store
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: metadata-store-ro
subjects:
- kind: ServiceAccount
  name: metadata-store-read-client
  namespace: metadata-store
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: metadata-store-read-client
  namespace: metadata-store
automountServiceAccountToken: false
EOF

```

Read-write service account

To create a read-write service account, run the following command. The command creates a service account called `metadata-store-read-write-client`:

```
kubectl apply -f - -o yaml << EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: metadata-store-read-write
  namespace: metadata-store
rules:
- resources: ["all"]
  verbs: ["get", "create", "update"]
  apiGroups: [ "metadata-store/v1" ]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: metadata-store-read-write
  namespace: metadata-store
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: metadata-store-read-write
subjects:
- kind: ServiceAccount
  name: metadata-store-read-write-client
  namespace: metadata-store
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: metadata-store-read-write-client
  namespace: metadata-store
automountServiceAccountToken: false
EOF
```

Getting the Access Token

To retrieve the read-only access token, run the following command:

```
kubectl get secret $(kubectl get sa -n metadata-store metadata-store-read-client -o js
on | jq -r '.secrets[0].name') -n metadata-store -o json | jq -r '.data.token' | base6
4 -d
```

To retrieve the read-write access token run the following command:

```
kubectl get secret $(kubectl get sa -n metadata-store metadata-store-read-write-client
-o json | jq -r '.secrets[0].name') -n metadata-store -o json | jq -r '.data.token' |
base64 -d
```

The access token is a “Bearer” token used in the http request header “Authorization.” (ex.

`Authorization: Bearer eyJhbGciOiJSUzI1NiIsImtpZCI6IjhmMV0...`)

Setting the Access Token

When using the CLI, you’ll need to set the `METADATA_STORE_ACCESS_TOKEN` environment variable, or use the `--access-token` flag. It is not recommended to use the `--access-token` flag as the token will appear in your shell history.

The following command will retrieve the access token from Kubernetes and store it in `METADATA_STORE_ACCESS_TOKEN` where `SERVICE-ACCOUNT-NAME` is the name of the service account you plan to use.

```
export METADATA_STORE_ACCESS_TOKEN=$(kubectl get secrets -n metadata-store -o jsonpath="{.items[?(@.metadata.annotations['kubernetes.io/service-account.name']=='SERVICE-ACCOUNT-NAME')].data.token}" | base64 -d)
```

For example:

```
$ export METADATA_STORE_ACCESS_TOKEN=$(kubectl get secrets -n metadata-store -o jsonpath="{.items[?(@.metadata.annotations['kubernetes.io/service-account.name']=='metadata-store-read-write-client')].data.token}" | base64 -d)
```

Query data

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 vulnerabilities.
- [vulnerabilities](#) - Query vulnerabilities by image, package, or source code.

Use `tanzu insight -h` or see [Tanzu Insight Details](#) for more information.

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 sha256:407d7099d6ce7e3632b6d00682a43028d75d3b088600797a833607bd629d1ed5
Registry: docker.io
Image Name: checkr/flagr:1.1.12
Digest:      sha256:407d7099d6ce7e3632b6d00682a43028d75d3b088600797a833607bd629d1ed5
Packages:
 1. alpine-baselayout@3.1.1.2-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?

Run:

```
tanzu insight source get --repo REPO --org ORG
```

Where:

- `REPO` specifies XML or JSON, the two supported file types
- `ORG` is the source code's organization

You may also use `tanzu insight source get --commit COMMIT` where `COMMIT` is the commit sha. `--repo` and `--org` must be used together.

For example, to get a recent scan for <https://github.com/pivotal/kpack.git>:

```
$ tanzu insight source get --repo kpack --org pivotal
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

See [Add Data](#) for more information about manually adding 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 may 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 Gype or another tool in the [CycloneDX Tool Center](#).

To use Gype to scan an image and generate an image report in CycloneDX format:

1. Install [Gype](#).
2. Scan the image and generate a report by running:

```
gype 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 XML and JSON format. 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 peer 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 Environment Variable METADATA_STORE_ACCESS_TOKEN during the API calls, this will override access token flag. Note: using the the access-token flag stores the token on disk, the 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 add

Add an image report from a report file:

```
tanzu insight image add [flags]
```

Examples

```
tanzu insight image add --cyclonedxtype json --path /path/to/file.json
```

Options

```
    --cyclonedxtype string  cyclonedx file type(xml/json)
-h, --help                help for add
```

```
--path string      path to file
```

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 <image-format>] [flags]
```

Examples

```
tanzu insight image packages --digest sha256:a86859ac1946065d93df9ecb5cb7060adeeb0288fad610b1b659907 --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 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> --pkgmgr
Unknown [--format <format>] [flags]
```

Examples

```
tanzu insight package get --name client --version 1.0.0a --pkgmgr Unknown
```

Options

```
-f, --format string    output format which can be in 'json' or 'text'. If not present,
defaults to text. (default "text")
-h, --help            help for get
-n, --name string      name of the package
-p, --pkgmgr 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 present
, 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 present
, 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 present
, 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 add

Add a source report from a report file:

```
tanzu insight source add [flags]
```

Examples

```
tanzu insight source add --cyclonedxtype json --path /path/to/file.json
```

Options

```

--cyclonedxtype string  cyclonedx file type (xml/json)
-h, --help              help for add
--path string           path to file

```

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 <organization-name> [--format <format>] [flags]

```

Examples

```

tanzu insight source get --repo github.com/org/example --commit b33dfee51 --org company

```

Options

```

-c, --commit string  commit hash
-f, --format string  output format which can be in 'json' or 'text'. If not present, 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>] [--format <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 present
, 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 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 present
, 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 present
, 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 present
, 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 present
, defaults to text. (default "text")
-h, --help           help for sources
```

See also

- [Tanzu insight vulnerabilities](#) - Vulnerabilities commands

Overview

Tanzu Application Platform v1.1 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.

Setting up authentication for Tanzu Application Platform

There are multiple ways to set up authentication for your Tanzu Application Platform 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 a single cluster](#) and [Logging in 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.

Installing Pinniped on a single cluster

Pinniped is used to support authentication on Tanzu Application Platform. This topic introduces how to install Pinniped on a single cluster of Tanzu Application Platform. You will deploy two Pinniped components into the cluster.

The **Pinniped Supervisor** is 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.

The **Pinniped Concierge** is 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.

Install Pinniped Supervisor

Follow these steps to install `pinniped-supervisor`:

1. Create the necessary certificate files.
2. Create the Ingress resources.
3. Create the `pinniped-supervisor` configuration.
4. Apply these resources to the cluster.

Create Certificates (letsencrypt/cert-manager)

Create a ClusterIssuer for `letsencrypt` and a TLS certificate resource for Pinniped Supervisor by creating the following resources and save them into `workspace/pinniped-supervisor/certificates.yaml`.

```
---
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-staging
  namespace: cert-manager
spec:
  acme:
    email: your-mail@example.com
    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:
  - pinniped-supervisor.example.com
  issuerRef:
    name: letsencrypt-staging
    kind: ClusterIssuer

```

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
spec:
  virtualhost:
    fqdn: pinniped-supervisor.example.com
    tls:
      secretName: pinniped-supervisor-tls-cert
  routes:
  - services:
    - name: pinniped-supervisor
      port: 8443

```

Create 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. In the following example, you will use auth0. See the [Pinniped documentation](#) to learn how to configure different identity providers, including OKTA, GitLab, OpenLDAP, Dex, Microsoft AD, and more.

To create Pinniped-Supervisor configuration, create the following resources and save them in `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.
  issuer: https://dev-xyz.us.auth0.com/

  # Specify how to form authorization requests to GitLab.
  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: 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: https://pinniped-supervisor.example.com
  tls:
    secretName: pinniped-supervisor-tls-cert

```

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 --into-ns pinniped-supervisor -f pinniped-supervisor -f https://get.pinniped.dev/v0.12.0/install-pinniped-supervisor.yaml
```

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. Bind the Ingress DNS to the IP address by running:

```
*.example.com A 35.222.xxx.yyy
```

Install Pinniped Concierge

To install Pinniped Concierge:

1. Deploy the Pinniped Concierge by running:

```
kapp deploy -y --app pinniped-concierge \
  -f https://get.pinniped.dev/v0.12.0/install-pinniped-concierge-crds.yaml \
  -f https://get.pinniped.dev/v0.12.0/install-pinniped-concierge.yaml
```

2. Get the CA certificate of the supervisor by running:

```
kubectl get secret pinniped-supervisor-tls-cert -n pinniped-supervisor -o 'go-template={{index .data "tls.crt"}}'
```

Note the `tls.crt` contains the entire certificate chain including the CA certificate for letsencrypt generated certificates

3. 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: https://pinniped-supervisor.example.com
  audience: concierge
  tls:
    certificateAuthorityData: # insert the CA certificate data here
```

4. Deploy the resource by running:

```
kapp deploy -y --app pinniped-concierge-jwt --into-ns pinniped-concierge -f pinniped-concierge/jwt_authenticator.yaml
```

Log in to the cluster

See [Login using Pinniped](#).

Integrating Azure Active Directory

This topic describes how to integrate the Azure Active Directory (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-admin-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-admin-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](#) yet.

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

This topic is a high level overview of each default role. For more information about the specific permissions of each role for every component, see [Detailed Role Permissions Breakdown](#).

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 needed to create running workloads. This role is not for a user.

Detailed role permissions breakdown

Native Kubernetes Resources

```
apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"
```

```
- apiGroups: [""]
```

```

resources: ["configmaps","endpoints","events","persistentvolumeclaims","pods","pods/
log","resourcequotas","services"]
verbs: ["get","list","watch"]
- apiGroups: ["apps"]
resources: ["deployments","replicasets","statefulsets"]
verbs: ["get","list","watch"]
- apiGroups: ["batch"]
resources: ["cronjobs","jobs"]
verbs: ["get","list","watch"]
- apiGroups: ["events.k8s.io"]
resources: ["events"]
verbs: ["get","list","watch"]
- apiGroups: ["networking.k8s.io"]
resources: ["ingresses"]
verbs: ["get","list","watch"]

```

`apps.tanzu.vmware.com/aggregate-to-app-operator: "true"`

```

- apiGroups: [""]
resources: ["configmaps","secrets"]
verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]

```

App Accelerator

`apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"`

```

- apiGroups: ["accelerator.apps.tanzu.vmware.com"]
resources: ["accelerators"]
verbs: ["get","list","watch"]

```

`apps.tanzu.vmware.com/aggregate-to-app-operator: "true"`

```

- apiGroups: ["accelerator.apps.tanzu.vmware.com"]
resources: ["accelerators"]
verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]

```

Cartographer

`apps.tanzu.vmware.com/aggregate-to-app-editor: "true"`

```

- apiGroups: ["carto.run"]
resources: ["deliverables","workloads"]
verbs: ["create","patch","update","delete","deletecollection"]

```

`apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"`

```

- apiGroups: ["carto.run"]
resources: ["deliverables","runnables","workloads"]
verbs: ["get","list","watch"]

```

```
apps.tanzu.vmware.com/aggregate-to-app-viewer-cluster-access:
"true"
```

```
- apiGroups: ["carto.run"]
  resources: ["clusterconfigtemplates","clusterconfigtemplates","clusterdeliveries","c
lusterdeploymenttemplates","clusterimagetemplates","clusterruntemplates","clustersourc
emplates","clustersupplychains","clustertemplates"]
  verbs: ["get","list","watch"]
```

```
apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access
```

```
- apiGroups: ["carto.run"]
  resources: ["clusterconfigtemplates","clusterconfigtemplates","clusterdeliveries","c
lusterdeploymenttemplates","clusterimagetemplates","clusterruntemplates","clustersourc
emplates","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.apps.tanzu.vmware.com"]
  resources: ["podintents"]
  verbs: ["get","list","watch"]
```

`apps.tanzu.vmware.com/aggregate-to-app-viewer-cluster-access: "true"`

```
- apiGroups: ["conventions.apps.tanzu.vmware.com"]
  resources: ["clusterpodconventions"]
  verbs: ["get","list","watch"]
```

`apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access`

```
- 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.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"]
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.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"]
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"]
  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"]
  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 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
```

Use a different kubeconfig location

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.

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.authorization
  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[] | select(.subjects[]?.name == \"\${NAME}\") | .roleRef.name" | xargs -n1 kubectl describe clusterroles
```

2. Get a list of user or group for a specific role:

```
tanzu rbac binding get --role ROLE --namespace NAMESPACE
```

Login using Pinniped

As a prerequisite, the administrator needs to provide users access to resources via `rolebindings`. It can be done with the `tanzu rbac` plug-in. See [Bind a user or group to a default role](#).

To login to your cluster by using Pinniped, follow these steps:

1. Generate and distribute `kubeconfig` to users
2. Login with provided `kubeconfig`

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/concierge-kubeconfig
...
"level"=0 "msg"="validated connection to the cluster"
```

Distribute this `kubeconfig` to your users so they can login by using `pinniped`.

Login with provided kubeconfig

As a user of the cluster, you will need the `kubeconfig` provided by your administrator to login. Logging in is a part of requesting information from the cluster. You can execute any resource request with `kubectl` to get into 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 visit in the browser. You can then log in, copy the auth code and paste it back to the terminal. After the login succeeds, you will either see the resources or get a message that you have no permission to access the resources.

Additional resources

This topic includes additional information about Authentication and Authorization for Tanzu Application Platform. Read the [Overview](#) page first 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

This document describes how to install default roles for Tanzu Application Platform without deploying a Tanzu Application Platform 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 [About Tanzu Application Platform components and 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-install
1
```

```
- 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 for VMware Tanzu

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.

To learn more about Application Accelerator, see:

- [Application Accelerator for VMware Tanzu Documentation](#)
- [Application Accelerator in Tanzu Application Platform GUI](#)

Install Application Accelerator

This document describes how to install Application Accelerator 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 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 FluxCD 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	LoadBalancer	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
tls.secretName	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 usage 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  0.5.1   2021-12-02T00:00:00Z
```

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/0.5.1 --values-
schema --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.

Note: 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 VERSION-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 1.0.0 -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-rolebinding'
| 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:     1.0.0
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.

Application Live View for VMware Tanzu

Application Live View is a lightweight insights and troubleshooting tool that helps app developers and app operators to look inside running apps. It is based on the concept of Spring Boot Actuators.

To learn more about Application Live View, see:

- [Application Live View documentation](#)
- [Application Live View in Tanzu Application Platform GUI](#)

Install Application Live View

This topic describes how to install Application Live View from the Tanzu Application Platform 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 Backend package (`backend.appliveview.tanzu.vmware.com`). This installs the Application Live View Backend 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.

Use the instructions on this page if you do not want to use a profile to install packages. For more information about profiles, see [Installing the Tanzu Application Platform Package and Profiles](#).

Prerequisites

Before installing Application Live View, complete all prerequisites to install Tanzu Application Platform. For more information, see [Prerequisites](#).

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 Backend 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 Backend.

Install Application Live View Backend

To install Application Live View Backend:

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 1.1.1                                2022-04-22T00:00:10Z
```

- (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, `1.1.1`.

For example:

```
$ tanzu package available get backend.appliveview.tanzu.vmware.com/1.1.1 --valu
es-schema --namespace tap-install
```

For more information about values schema options, see the properties listed earlier.

- Create `app-live-view-backend-values.yaml` with the following details:

For single cluster environment, use the following values:

```
ingressEnabled: "false"
```

For a multicluster environment, use 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 `appliveview` 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.

- Install the Application Live View Backend package by running:

```
tanzu package install appliveview -p backend.appliveview.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 appliveview -p backend.appliveview.tanzu.vmware.com -v
1.1.1 -n tap-install -f app-live-view-backend-values.yaml
- Installing package 'backend.appliveview.tanzu.vmware.com'
| Getting namespace 'tap-install'
| Getting package metadata for 'backend.appliveview.tanzu.vmware.com'
| Creating service account 'appliveview-tap-install-sa'
| Creating cluster admin role 'appliveview-tap-install-cluster-role'
| Creating cluster role binding 'appliveview-tap-install-cluster-rolebinding'
| Creating package resource
| Package install status: Reconciling

Added installed package 'appliveview' in namespace 'tap-install'
```

The Application Live View Backend component is deployed in `app-live-view` namespace by default.

5. Verify the Application Live View Backend 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:     1.1.1
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 1.1.1       2022-04-22T00:00:10Z
```

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, `1.1.1`.

For example:

```
$ tanzu package available get connector.appliveview.tanzu.vmware.com/1.1.1 --va
lues-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, use the following values:

```
backend:
  sslDisabled: "true"
```

The Application Live View Connector connects to the `cluster-local` back end to register the applications.

For a multicluster environment, use the following values:

```
backend:
  sslDisabled: "false"
  host: appliveview.INGRESS-DOMAIN
```

Where `INGRESS-DOMAIN` is the top level domain the Application Live View Backend exposes by using `tanzu-shared-ingress` for the Connectors in other clusters to reach the back end. Prepend the `appliveview` subdomain to the provided value.

The `sslDisabled` boolean flag is treated as a string in Kubernetes YAML. Therefore it must be specified in `double-quotes` for the configuration to be picked up.

You can edit the values to suit your project needs or leave the default values as is.

Using the HTTP proxy either on 80 or 443 based on SSL config exposes the Backend service running on port 7000. The connector connects to the Backend 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 appliveview-connector -p connector.appliveview.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, `1.1.1`.

For example:

```
$ tanzu package install appliveview-connector -p connector.appliveview.tanzu.vm
ware.com -v 1.1.1 -n tap-install -f app-live-view-connector-values.yaml
| Installing package 'connector.appliveview.tanzu.vmware.com'
| Getting namespace 'tap-install'
| Getting package metadata for 'connector.appliveview.tanzu.vmware.com'
| Creating service account 'appliveview-connector-tap-install-sa'
| Creating cluster admin role 'appliveview-connector-tap-install-cluster-role'
| Creating cluster role binding 'appliveview-connector-tap-install-cluster-role'
```

```
binding'
- Creating package resource
/ Package install status: Reconciling

Added installed package 'appliveview-connector' in namespace 'tap-install'
```

Each cluster installs the connector as a DaemonSet. The connector is configured to connect to the central instance of the Backend. 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 appliveview-connector -n tap-install
```

For example:

```
tanzu package installed get appliveview-connector -n tap-install
                                                                    5s
| Retrieving installation details for appliveview-connector...
NAME:                appliveview-connector
PACKAGE-NAME:        connector.appliveview.tanzu.vmware.com
PACKAGE-VERSION:     1.1.1
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.vmw  1.1.1                                2022-04-22T00:00:00Z
```

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, `1.1.1`.

For example:

```
$ tanzu package install appliveview-conventions -p conventions.appliveview.tanz
```

```

u.vmware.com -v 1.1.1 -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-role'
|
| Creating cluster role binding 'appliveview-conventions-tap-install-cluster-rolebinding'
- Creating package resource
\ Package install status: Reconciling

Added installed package 'appliveview-conventions' in namespace 'tap-install'

```

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:     1.1.1
STATUS:              Reconcile succeeded
CONDITIONS:          [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:

```

Verify that `STATUS` is `Reconcile succeeded`

For more information about Application Live View, see the [Application Live View documentation](#).

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](#).

Convention Service

Overview

Convention Service provides a means for people in operational roles to express their hard-won knowledge and opinions about how applications should run on Kubernetes as a convention. Convention Service 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. Convention Service currently supports defining and applying conventions for Pods.

About applying conventions

The convention server uses criteria defined in the convention to determine whether the configuration of a workload should be changed. 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 also 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 (for example, 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 given convention server. Images built with Cloud Native Buildpacks reliably include rich descriptive metadata. Images built by some other process may not include the same metadata.

Applying conventions without using image metadata

Conventions can also be defined to apply to workloads without targeting build service metadata. Examples of possible uses of this type of convention include appending a logging/metrics sidecar, adding environment variables, or adding cached volumes. Such conventions are a great way for you to ensure infrastructure uniformity across workloads deployed on the cluster while reducing developer toil.

Note: Adding a sidecar alone does not magically make the log/metrics collection work. This requires collector agents to be already deployed and accessible from the Kubernetes cluster, and also configuring required access through role-based access control (RBAC) policy.

Install Convention Service

This document describes how to install convention controller from the Tanzu Application Platform package repository. Convention controller is a primary component of Convention Service.

Note: Use the instructions on this page if you do not want to use a profile to install packages. Both the full and light profiles include convention controller. For more information about profiles, see [Installing the Tanzu Application Platform Package and Profiles](#).

Convention Service allows app operators to enrich Pod Template Specs with operational knowledge based on specific conventions they define. It includes the following components:

- Convention controller: Provides metadata to the convention server. Implements update requests from the convention server.

- Convention server: Receives and evaluates metadata associated with a workload from convention controller. Requests updates to the Pod Template Spec associated with that workload. There can be one or more convention servers for a single convention controller instance.

In the following procedure, you install convention controller. You install convention servers as part of separate installation procedures. For example, you install an `app-live-view` convention server as part of the `app-live-view` installation.

Prerequisites

Before installing convention 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 convention controller:

1. List version information for the package by running:

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

For example:

```
$ tanzu package available list controller.conventions.apps.tanzu.vmware.com --namespace tap-install
- Retrieving package versions for controller.conventions.apps.tanzu.vmware.com.
..
NAME                                VERSION  RELEASED-AT
controller.conventions.apps.tanzu.vmware.com  0.6.3    2022-03-08T00:00:00Z
```

2. (Optional) Gather values schema:

```
tanzu package available get controller.conventions.apps.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 controller.conventions.apps.tanzu.vmware.com/0.6.3 --values-schema --namespace tap-install

KEY          DEFAULT  TYPE      DESCRIPTION
ca_cert_data          string    Optional: PEM Encoded certificate data for image registries with private CA.
```

3. (Optional) Enable Convention 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.

To provide custom cert, create a file named `convention-controller-values.yaml` that includes the PEM-encoded CA cert data.

For example:

```
ca_cert_data: |
  -----BEGIN CERTIFICATE-----
  MIICpTCCAYUCBgkqhkiG9w0BBQ0wMzAbBgkqhkiG9w0BBQwwDgQIYg9x6gkCAggA
  ...
  9TlA7A4FFpQqbhAuAVH6KQ8WMZIrVxJSQ03c9lKVkI62wQ==
  -----END CERTIFICATE-----
```

4. Install the package by running:

```
tanzu package install convention-controller -p controller.conventions.apps.tanzu.vmware.com -v VERSION-NUMBER -f VALUES-FILE -n tap-install
```

Where:

- ◆ `VERSION-NUMBER` is the version of the package listed in the earlier step.
- ◆ `VALUES-FILE` is the path to the file created in the earlier step.

For example:

```
tanzu package install convention-controller -p controller.conventions.apps.tanzu.vmware.com -v 0.6.3 -f VALUES-FILE convention-controller-values.yaml -n tap-install
/ Installing package 'controller.conventions.apps.tanzu.vmware.com'
| Getting namespace 'tap-install'
- Getting package metadata for 'controller.conventions.apps.tanzu.vmware.com'
| Creating service account 'convention-controller-tap-install-sa'
| Creating cluster admin role 'convention-controller-tap-install-cluster-role'
| Creating cluster role binding 'convention-controller-tap-install-cluster-role-binding'
\ Creating package resource
| Package install status: Reconciling
Added installed package 'convention-controller' in namespace 'tap-install'
```

5. Verify the package install by running:

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

For example:

```
tanzu package installed get conventions-controller -n tap-install
Retrieving installation details for conventions-controller...
NAME:                conventions-controller
PACKAGE-NAME:        controller.conventions.apps.tanzu.vmware.com
PACKAGE-VERSION:     0.6.3
STATUS:              Reconcile succeeded
CONDITIONS:          [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`:

```
kubectl get pods -n conventions-system
```

For example:

```
$ kubectl get pods -n conventions-system
NAME                                READY   STATUS    RESTARTS   AGE
conventions-controller-manager-596c65f75-j9dmn  1/1     Running   0           72s
```

Verify that **STATUS** is **Running**.

Creating conventions

This document describes how to create and deploy custom conventions to the Tanzu Application Platform.

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](#).

[Convention Service](#) is a key component of the supply chain compositions the choreographer calls into action. Convention Service 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. Convention Service 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, let's look at two distinct components of Convention Service: the [convention controller](#) and [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 can be controlled conditionally. The conditional criteria governing the application of a convention is customizable and can be based on the evaluation of a custom Kubernetes resource called [PodIntent](#). PodIntent is the vehicle by which Convention Service as a whole delivers its value.

A PodIntent is created, or updated if already existing, when a workload is run through a Tanzu Application Platform supply chain. The custom resource includes both the PodTemplateSpec (see the [Kubernetes documentation](#)) and the OCI image metadata associated with a workload. The conditional criteria for a convention can be 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 will not take action unless triggered to do so by the second component of Convention Service, 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 Convention Service 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 Convention Service components, let's look at how to 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 specs.

Prerequisites

The following prerequisites must be met before a convention can be 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 [from GitHub](#). (These instructions use `ko` to build an image, but 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](#).

- The example `server.go` sets up the `ConventionHandler` to ingest the webhook requests(`PodConventionContext`) from the convention controller. 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 is going to modify. 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.

- 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 configure and create the *http* `webhook` server.
- ◆ `port` is the calculated port of the server to listen for requests. It needs to 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/start web server (`func NewConventionServer(...)`) are defined in the convention controller within the `webhook` package, but a custom one can be used.

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, images []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 PodTemplateSpec
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 key.
    ...
}
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 TSL configuration.
    server := &http.Server{
        Addr: addr,
        TLSConfig: &tls.Config{
            GetCertificate: func(_ *tls.ClientHelloInfo) (*tls.Certificate, error) {
                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 can be used to define the criteria for applying conventions. The following are a few examples.

Matching criteria by labels or annotations

When using labels or annotations to define whether a convention should be applied, the server checks the [PodTemplateSpec](#) of workloads.

- [PodTemplateSpec](#)

```

```yaml
...
template:
 metadata:
 labels:
 awesome-label: awesome-value
 annotations:
 awesome-annotation: awesome-value
...
```

```

- [Handler](#)

```

```go
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 we want to validate. + `awesome-annotation` is the **annotation** that we 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 should be present in the `PodTemplateSpec.spec.containers[*].env`. and we can validate the value.

- PodTemplateSpec

```

```yaml
...
template:
 spec:
 containers:
 - name: awesome-container
 env:
...
...

```

- Handler

```

```go
package convention
...
func conventionHandler(template *corev1.PodTemplateSpec, images []model.ImageConfig) ([]string, error) {
    if len(template.Spec.Containers[0].Env) == 0 {
        template.Spec.Containers[0].Env = append(template.Spec.Containers[0].Env, 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 can be 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 for the webhook to run 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 also created. For this example, the exposed port is the default 443, but if it is changed, the `ClusterPodConvention` needs to be updated with the proper one.

```

...
---
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.apps.tanzu.vmware.com/v1alpha1
kind: ClusterPodConvention
metadata:
  name: awesome-convention
  annotations:
    conventions.apps.tanzu.vmware.com/inject-ca-from: "awesome-convention/aweso
me-webhook-cert"
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 needs to be built and deployed, use the [ko] tool on GitHub (<https://github.com/google/ko>). It compiles your *go* code into a docker image and pushes it to the registry(`KO_DOCKER_REGISTRY`).

```
ko apply -f dist/server.yaml
```

- ✦ If a different tool is used to build the image, the configuration can be 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 check the status of the convention server, check for the running convention Pods:

- ✦ If the server is running, `kubectl get all -n awesome-convention` returns something like:

NAME	READY	STATUS	RESTARTS	A
GE				
pod/awesome-webhook-1234567890-12345h	1/1	Running	0	8

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/awesome-webhook/TCP	ClusterIP	10.56.12.49	<none>	443	28h

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/awesome-webhookh	1/1	1	1	28h

NAME	DESIRED	CURRENT	READY
replicaset.apps/awesome-webhook-123456321323h	0	0	0
replicaset.apps/awesome-webhook-5b79d5cb5928h	0	0	0
replicaset.apps/awesome-webhook-5bf557c9f820h	1	1	1
replicaset.apps/awesome-webhook-77c647c98723h	0	0	0
replicaset.apps/awesome-webhook-79d9c6f74c23h	0	0	0
replicaset.apps/awesome-webhook-7d9d667b8d9h	0	0	0
replicaset.apps/awesome-webhook-8668664d7523h	0	0	0
replicaset.apps/awesome-webhook-9b6957476	0	0	0

24h

- ◆ To verify the conventions are being applied, check the `PodIntent` of a workload that matches the convention criteria:

```
kubectl -o yaml get podintents.conventions.apps.tanzu.vmware.co awesome-app
```

```
apiVersion: conventions.apps.tanzu.vmware.com/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.apps.tanzu.vmware.com/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.apps.tanzu.vmware.com/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

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", "ClusterPodConvention": "
base-convention", "error": "unable to find valid certificaterequests for certific
ate \"convention-template/webhook-certificate\", \"stacktrace\": \"reflect.Value.Ca
ll\\n\\treflect/value.go:339\\ngithub.com/vmware-labs/reconciler-runtime/reconcile
rs.(*SyncReconciler).sync\\n\\tgithub.com/vmware-labs/reconciler-runtime@v0.3.0/r
econcilers/reconcilers.go:287\\ngithub.com/vmware-labs/reconciler-runtime/reconc
ilers.(*SyncReconciler).Reconcile\\n\\tgithub.com/vmware-labs/reconciler-runtime@
v0.3.0/reconcilers/reconcilers.go:276\\ngithub.com/vmware-labs/reconciler-runtim
e/reconcilers.Sequence.Reconcile\\n\\tgithub.com/vmware-labs/reconciler-runtime@v
0.3.0/reconcilers/reconcilers.go:815\\ngithub.com/vmware-labs/reconciler-runtim
e/reconcilers.(*ParentReconciler).reconcile\\n\\tgithub.com/vmware-labs/reconcil
er-runtime@v0.3.0/reconcilers/reconcilers.go:146\\ngithub.com/vmware-labs/reconcil
er-runtime/reconcilers.(*ParentReconciler).Reconcile\\n\\tgithub.com/vmware-labs/
reconciler-runtime@v0.3.0/reconcilers/reconcilers.go:120\\nsigs.k8s.io/controlle
r-runtime/pkg/internal/controller.(*Controller).Reconcile\\n\\tsigs.k8s.io/contr
oller-runtime/pkg/internal/controller/controller.go:114\\nsigs.k8s.io/contr
oller-runtime/pkg/internal/controller.(*Controller).reconcileHandler\\n\\tsigs.
k8s.io/controller-runtime@v0.10.3/pkg/internal/controller/controller.go:311\\nsi
gs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).processNextW
orkItem\\n\\tsigs.k8s.io/controller-runtime@v0.10.3/pkg/internal/controller/contr
oller.go:266\\nsigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controll
er).Start.func2.2\\n\\tsigs.k8s.io/controller-runtime@v0.10.3/pkg/internal/contro
ller/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.apps.tanzu.vmware.com/inject-ca-from` which must be set to the used *Certificate*.

Note: Do not set annotation `conventions.apps.tanzu.vmware.com/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.PodIntent.ApplyConventions", "msg": "failed to apply convention", "Convention": { "Name": "base-convention", "Selectors": null, "Priority": "Normal", "ClientConfig": { "service": { "namespace": "convention-template", "name": "webhook", "port": 443 }, "caBundle": "..."}, "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/reconcilers.(*SyncReconciler).sync\n\tgithub.com/vmware-labs/reconciler-runtime@v0.3.0/reconcilers/reconcilers.go:287\ngithub.com/vmware-labs/reconciler-runtime/reconcilers.(*SyncReconciler).Reconcile\n\tgithub.com/vmware-labs/reconciler-runtime@v0.3.0/reconcilers/reconcilers.go:276\ngithub.com/vmware-labs/reconciler-runtime/reconcilers.Sequence.Reconcile\n\tgithub.com/vmware-labs/reconciler-runtime@v0.3.0/reconcilers/reconcilers.go:815\ngithub.com/vmware-labs/reconciler-runtime/reconcilers.(*ParentReconciler).reconcile\n\tgithub.com/vmware-labs/reconciler-runtime@v0.3.0/reconcilers/reconcilers.go:146\ngithub.com/vmware-labs/reconciler-runtime/reconcilers.(*ParentReconciler).Reconcile\n\tgithub.com/vmware-labs/reconciler-runtime@v0.3.0/reconcilers/reconcilers.go:120\nnsigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).Reconcile\n\tgithub.com/vmware-labs/reconciler-runtime@v0.10.0/pkg/internal/controller/controller.go:114\nnsigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).reconcileHandler\n\tgithub.com/vmware-labs/reconciler-runtime@v0.10.0/pkg/internal/controller/controller.go:311\nnsigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).processNextWorkItem\n\tgithub.com/vmware-labs/reconciler-runtime@v0.10.0/pkg/internal/controller/controller.go:266\nnsigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).Start.func2.2\n\tgithub.com/vmware-labs/reconciler-runtime@v0.10.0/pkg/internal/controller/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 `PodIntents`. It is only

applied to the new `PodIntents`.

- ◆ 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": { "Name": "
base-convention", "Selectors": null, "Priority": "Normal", "ClientConfig": { "service"
: { "namespace": "convention-template", "name": "webhook", "port": 443 }, "caBundle": ".
." }, "error": "Post \"https://webhook.convention-template.svc:443/?timeout=30s\"
: dial tcp 10.56.13.206:443: connect: connection refused", "stacktrace": "reflect
.Value.call\n\treflect/value.go:543\nreflect.Value.Call\n\treflect/value.go:339
\ngithub.com/vmware-labs/reconciler-runtime/reconcilers.(*SyncReconciler).sync\n
\ngithub.com/vmware-labs/reconciler-runtime@v0.3.0/reconcilers/reconcilers.go:
287\ngithub.com/vmware-labs/reconciler-runtime/reconcilers.(*SyncReconciler).Re
concile\n\tgithub.com/vmware-labs/reconciler-runtime@v0.3.0/reconcilers/reconci
lers.go:276\ngithub.com/vmware-labs/reconciler-runtime/reconcilers.Sequence.Rec
oncile\n\tgithub.com/vmware-labs/reconciler-runtime@v0.3.0/reconcilers/reconci
lers.go:815\ngithub.com/vmware-labs/reconciler-runtime/reconcilers.(*ParentRecon
ciler).reconcile\n\tgithub.com/vmware-labs/reconciler-runtime@v0.3.0/reconciler
s/reconcilers.go:146\ngithub.com/vmware-labs/reconciler-runtime/reconcilers.(*P
arentReconciler).Reconcile\n\tgithub.com/vmware-labs/reconciler-runtime@v0.3.0/
reconcilers/reconcilers.go:120\nsigs.k8s.io/controller-runtime/pkg/internal/con
troller.(*Controller).Reconcile\n\tgithub.com/vmware-labs/reconciler-runtime@v0.10.0/pkg/i
nternal/controller/controller.go:114\nsigs.k8s.io/controller-runtime/pkg/intern
al/controller.(*Controller).reconcileHandler\n\tgithub.com/vmware-labs/reconciler-
runtime@v0.10.0/pkg/internal/controller/controller.go:311\nsigs.k8s.io/controller-runti
me/pkg/internal/controller.(*Controller).processNextWorkItem\n\tgithub.com/vmware-
labs/reconciler-runtime@v0.10.0/pkg/internal/controller/controller.go:266\nsigs.k8s.io/
controller-runtime/pkg/internal/controller.(*Controller).Start.func2.2\n\tgithub.
com/vmware-labs/reconciler-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 assigned convention-template/webhook-594d75d69b-4w4s8 to pool
Normal    Pulling     14m          kubelet        Pulling image "awesome-repo/awesome-user/awesome-convention-..."
Normal    Pulled      14m          kubelet        Successfully pulled 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 failed: 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 failed: 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 "awesome-repo/awesome-user/awesome-convention" already present on machine
Warning   BackOff     4m22s (x32 over 11m)    kubelet        Back-off restarting 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.apps.tanzu.vmware.com/inject-ca-from` as a pointer to the deployed `Certificate` resource.

Convention Resources

The convention controller is open to extension. These resources are typically consumed by platform developers and operators rather than by application developers.

Convention Service Resources

There are several `resources` involved in the application of conventions to workloads.

API Structure

References

- [ImageConfig](#)
- [PodConventionContextSpec](#)
- [PodConventionContextStatus](#)
- [PodConventionContext](#)
- [Cluster Pod Convention](#)
- [PodIntent](#)
- [BOM](#)

ImageConfig

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

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

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.apps.tanzu.vmware.com/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

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.

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.apps.tanzu.vmware.com/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":"7fef8ba6a0786143efc6e66b0bbfbfbab02afeef522b4e09833c5b550d7`..
.
      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

`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.

Webhook servers are currently the only way to define conventions.

```
apiVersion: conventions.apps.tanzu.vmware.com/v1alpha1
kind: ClusterPodConvention
metadata:
  name: base-convention
  annotations:
    conventions.apps.tanzu.vmware.com/inject-ca-from: "convention-template/webhook-cert"
spec:
  webhook:
    clientConfig:
      service:
        name: webhook
        namespace: convention-template
```

PodIntent

`PodIntent` applies conventions to a workload. The `.spec.template`'s `PodTemplateSpec` is enriched by the conventions and exposed as the `.status.templates` `PodTemplateSpec`. A log of which sources and conventions applied is captured with the [conventions.apps.tanzu.vmware.com/applied-conventions](#) annotation on the `PodTemplateSpec`.

```
apiVersion: conventions.apps.tanzu.vmware.com/v1alpha1
kind: PodIntent
metadata:
  name: sample
spec:
  template:
    spec:
      containers:
        - name: workload
          image: ubuntu
```

BOM

The [BOM](#) is a type/structure wrapping a Software Bill of Materials (SBOM) describing the software

components and their dependencies.

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, and FluxCD Source Controller

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](#).

FluxCD Source Controller is a Kubernetes operator that helps you acquire artifacts from external sources such as Git, Helm repositories, and S3 buckets. For more information about FluxCD Source Controller, see the [fluxcd/source-controller](#) project on GitHub.

Install cert-manager, Contour, and FluxCD Source Controller

This document describes how to install cert-manager, Contour, and FluxCD Source Controller 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 cert-manager, contour, and FluxCD Source Controller. For more information about profiles, see [Components and installation profiles](#).

Prerequisites

Before installing cert-manager, Contour, and FluxCD 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: 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
  CRPTION
  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/
  docs/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 on
    . 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,
    aws
    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.

Install FluxCD source-controller

To install FluxCD 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.vmw  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.tanz
u.vmware.com -v VERSION-NUMBER -n tap-install
```

Where:

- ◆ **VERSION-NUMBER** is the version of the package listed in step 1.

For example:

```
tanzu package install fluxcd-source-controller -p fluxcd.source.controller.tanz
u.vmware.com -v 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-rol
e'
| Creating cluster role binding 'fluxcd-source-controller-tap-install-cluster-r
olebinding'
| Creating package resource
- Waiting for 'PackageInstall' reconciliation for 'fluxcd-source-controller'
| 'PackageInstall' resource install status: Reconciling

Added installed package 'fluxcd-source-controller'
```

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-118fj  1/1     Running   0           3m38s
```

Verify that **STATUS** is **Running**

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 document describes how to install Cloud Native Runtimes 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 full and light profiles include Cloud Native Runtimes. For more information about profiles, see [Installing the Tanzu Application Platform Package and Profiles](#).

Prerequisites

Before installing Cloud Native Runtimes:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see [Prerequisites](#).

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 where an existing Contour is installed on your cluster (for external services) 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 where an existing Contour is installed on your cluster (for internal services) 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 domain for CoreDNS. Only applicable when "local_dns.enable" is set to "true" 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 local DNS.
pdb.enable                         true     boolean        Optional: Set to true to enable Pod Disruption Budget. If provider local is set to "local", the PDB will be disabled automatically.
provider                           <nil>    string        Optional: Kubernetes cluster provider. To be specified if deploying CNR on a local Kubernetes cluster 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 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.

Spring Boot conventions

This topic describes the Spring Boot convention server.

Overview

The Spring Boot convention server is 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.

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 describes 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 [About Tanzu Application Platform components and profiles](#).

Prerequisites

Before installing Spring Boot conventions:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see [Prerequisites](#).
- Ensure that Convention Service is installed on the cluster. For more information, see [Install Convention Service](#) section.

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 the package install 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**

Conventions

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.apps.tanzu.vmware.com/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 using the Tanzu CLI or `workload.yaml` file.

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.apps.tanzu.vmware.com/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.apps.tanzu.vmware.com/v1alpha1
kind: PodIntent
metadata:
  annotations:
    kubectrl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"conventions.apps.tanzu.vmware.com/v1alpha1","kind":"PodIntent","me
tadata":{"annotations":{},"name":"spring-sample","namespace":"default"},"spec":{"templ
ate":{"spec":{"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.apps.tanzu.vmware.com/applied-conventions: |-
        spring-boot-convention/spring-boot
    labels:
      conventions.apps.tanzu.vmware.com/framework: spring-boot
  spec:
    containers:
  - image: index.docker.io/springio/petclinic@sha256:...
    name: workload
    resources: {}
```

Spring boot graceful shut down convention

If any of the following dependencies are in the metadata within the SBOM file under `dependencies`, the Spring Boot graceful shutdown 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.apps.tanzu.vmware.com/v1alpha1
kind: PodIntent
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"conventions.apps.tanzu.vmware.com/v1alpha1","kind":"PodIntent","m
etadata":{"annotations":{},"name":"spring-sample","namespace":"default"},"spec":{"temp
late":{"spec":{"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.apps.tanzu.vmware.com/applied-conventions: |-
          spring-boot-convention/spring-boot
          spring-boot-convention/spring-boot-graceful-shutdown
      labels:
        conventions.apps.tanzu.vmware.com/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 [PodTemplateSpec](#). If the `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.apps.tanzu.vmware.com/v1alpha1
kind: PodIntent
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"conventions.apps.tanzu.vmware.com/v1alpha1","kind":"PodIntent","m
etadata":{"annotations":{},"name":"spring-sample","namespace":"default"},"spec":{"temp
late":{"spec":{"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.apps.tanzu.vmware.com/applied-conventions: |-
          spring-boot-convention/spring-boot
          spring-boot-convention/spring-boot-web
      labels:
        conventions.apps.tanzu.vmware.com/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 the following actions:

- Sets the management port in the `JAVA_TOOL_OPTIONS` environment variable to `8081`.
- Sets the base path in the `JAVA_TOOL_OPTIONS` environment variable to `/actuator`.
- 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 the 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.apps.tanzu.vmware.com/v1alpha1
kind: PodIntent
metadata:
  annotations:
    kubectrl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"conventions.apps.tanzu.vmware.com/v1alpha1","kind":"PodIntent","m
etadata":{"annotations":{},"name":"spring-sample","namespace":"default"},"spec":{"temp
late":{"spec":{"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.apps.tanzu.vmware.com/applied-conventions: |-
      spring-boot-convention/spring-boot
      spring-boot-convention/spring-boot-web
      spring-boot-convention/spring-boot-actuator
  labels:
    conventions.apps.tanzu.vmware.com/framework: spring-boot
spec:
  containers:
  - env:
    - name: JAVA_TOOL_OPTIONS
      value: Dmanagement.endpoints.web.base-path="/actuator" -Dmanagement.server.p
ort="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 greater than or equal to 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 the value `true`:
 - ◊ `-Dmanagement.health.probes.enabled`
 - ◊ `-Dmanagement.endpoint.health.probes.add-additional-paths`

The Spring Boot Actuator Probes convention does the following actions:

- 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](#)
- 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.apps.tanzu.vmware.com/v1alpha1
kind: PodIntent
metadata:
  annotations:
    kubectrl.kubernetes.io/last-applied-configuration: |

```

```

    {"apiVersion":"conventions.apps.tanzu.vmware.com/v1alpha1","kind":"PodIntent","m
etadata":{"annotations":{},"name":"spring-sample","namespace":"default"},"spec":{"temp
late":{"spec":{"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.apps.tanzu.vmware.com/applied-conventions: |-
          spring-boot-convention/spring-boot
          spring-boot-convention/spring-boot-web
          spring-boot-convention/spring-boot-actuator
      labels:
        conventions.apps.tanzu.vmware.com/framework: spring-boot
    spec:
      containers:
      - env:
        - name: JAVA_TOOL_OPTIONS
          value: -Dmanagement.endpoint.health.probes.add-additional-paths="true" -Dman
agement.endpoints.web.base-path="/actuator" -Dmanagement.health.probes.enabled="true"
-Dmanagement.server.port="8081" -Dserver.port="8080"
          image: index.docker.io/springio/petclinic@sha256:...
          name: workload
          livenessProbe:
            httpGet:
              path: /livez
              port: 8080
              scheme: HTTP
          ports:
          - containerPort: 8080
            protocol: TCP
          readinessProbe:
            httpGet:
              path: /readyz
              port: 8080
              scheme: HTTP
          resources: {}

```

Service intent conventions

The Service intent conventions do not change the behavior of the final deployment, but you can use them as added information to process in the supply chain, such as when an app requires to be bound to a database service. This convention adds an annotation and a label to `PodTemplateSpec` for each detected dependency. It also adds an annotation and a label to `conventions.apps.tanzu.vmware.com/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

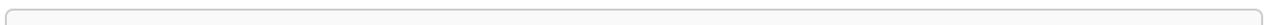
- **Name:** `service-intent-kafka`
- **Label:** `services.conventions.apps.tanzu.vmware.com/kafka`
- **Dependencies:** `kafka-clients`

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:
    kubect1.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"conventions.apps.tanzu.vmware.com/v1alpha1","kind":"PodIntent",
"metadata":{"annotations":{},"name":"spring-sample","namespace":"default"},"spec":{"te
mplate":{"spec":{"containers":[{"image":"springio/petclinic","name":"workload"}]}}}
    creationTimestamp: "...
    generation: 1
    name: spring-sample
    namespace: default
    resourceVersion: "...
    uid: ...
spec:
  serviceAccountName: default
  template:
    metadata: {}
    spec:
      containers:
        - image: springio/petclinic
          name: workload
          resources: {}
status:
  conditions:
    - lastTransitionTime: "... # This status indicates that all worked as expected
      status: "True"
      type: ConventionsApplied
    - lastTransitionTime: "...
      status: "True"
      type: Ready
  observedGeneration: 1
  template:
    metadata:
      annotations:
        boot.spring.io/actuator: http://:8080/actuator
        boot.spring.io/version: 2.3.3.RELEASE
        conventions.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.server
.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 describes 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-conventions/server.go:83","msg":"Successfully applied convention: spring-boot","component":"spring-boot-conventions"}
{"level":"info","timestamp":"2021-11-11T16:00:26.597Z","caller":"spring-boot-conventions/server.go:83","msg":"Successfully applied convention: spring-boot-graceful-shutdown","component":"spring-boot-conventions"}
{"level":"info","timestamp":"2021-11-11T16:00:26.597Z","caller":"spring-boot-conventions/server.go:83","msg":"Successfully applied convention: spring-boot-webhook","component":"spring-boot-conventions"}
{"level":"info","timestamp":"2021-11-11T16:00:26.597Z","caller":"spring-boot-conventions/server.go:83","msg":"Successfully applied convention: spring-boot-actuator","component":"spring-boot-conventions"}
{"level":"info","timestamp":"2021-11-11T16:00:26.597Z","caller":"spring-boot-conventions/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.

Service Bindings for Kubernetes

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.

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 document describes how to install Service Bindings 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 Service Bindings. For more information about profiles, see [About Tanzu Application Platform components and profiles](#).

Prerequisites

Before installing Service Bindings:

- Complete all prerequisites to install Tanzu Application Platform. 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

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/nfig
.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/run
/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 election"
}
{"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.dev
/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb
0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkItem
\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\nk
native.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg@v0.0.0-202103310
65221-952fdd90dbb0/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.dev
/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb
0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkItem
\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\nk
native.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg@v0.0.0-202103310
65221-952fdd90dbb0/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.dev
/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb
0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkItem
\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\nk
native.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg@v0.0.0-202103310
65221-952fdd90dbb0/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.dev
/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb
0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkItem
\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\nk
native.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg@v0.0.0-202103310
65221-952fdd90dbb0/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.dev
/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb
0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkItem
\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\nk
native.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg@v0.0.0-202103310
65221-952fdd90dbb0/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":"182.402µs
","error":"deployments.apps \"spring-petclinic\" not found","stacktrace":"knative.dev/

```

```

pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb0
/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkItem\
n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\nkn
ative.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg@v0.0.0-2021033106
5221-952fdd90dbb0/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.dev
/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb
0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkItem
\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\nk
native.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg@v0.0.0-202103310
65221-952fdd90dbb0/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.dev
/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb
0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkItem
\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\nk
native.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg@v0.0.0-202103310
65221-952fdd90dbb0/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":"179.203µs
","error":"deployments.apps \"spring-petclinic\" not found","stacktrace":"knative.dev/
pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb0
/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkItem\
n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\nkn
ative.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg@v0.0.0-2021033106
5221-952fdd90dbb0/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.dev
/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb
0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkItem
\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\nk
native.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg@v0.0.0-202103310
65221-952fdd90dbb0/controller/controller.go:468"}

```

Resources

ServiceBinding (servicebinding.io/v1alpha3)

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

```

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, etc.) 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 Product 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 [About Tanzu Application Platform components and 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  0.6.0     2022-04-12T00:00:00Z
```

2. Install Services Toolkit by running:

```
tanzu package install services-toolkit -n tap-install -p services-toolkit.tanzu
.vmware.com -v VERSION-NUMBER
```

Where `VERSION-NUMBER` is the Services Toolkit version you want to install. For example, `0.6.0`.

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:     0.6.0
STATUS:              Reconcile succeeded
CONDITIONS:          [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Source Controller

Tanzu Source Controller provides a common interface for artifact acquisition. With it, an `ImageRepository` resource can resolve source from the contents of an image in an image registry. This functionality enables app developers to create and update workloads from local source code or from a code repository.

Tanzu Source Controller extends the functionality of the FluxCD Source Controller Kubernetes operator. For more information about FluxCD Source Controller, see the [fluxcd/source-controller](#) project on GitHub.

Install Source Controller

This document describes how to install Source Controller 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 Source Controller. For more information about profiles, see [About Tanzu Application Platform components and 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, Contour](#).

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
```

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/0.3.3 --values-schema --namespace tap-install
Retrieving package details for controller.source.apps.tanzu.vmware.com/0.3.3...
.
  KEY          DEFAULT  TYPE      DESCRIPTION
  ca_cert_data          string    Optional: PEM Encoded certificate data for image registries with private CA.
```

3. (Optional) Enable 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.

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
  ...
```

```
9T1A7A4FFpQqbhAuAVH6KQ8WMZIrVxJSQ03c91KVkI62wQ==
-----END CERTIFICATE-----
```

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 0.3.3 -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:     0.3.3
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-f68dc7bb6-41rn6  1/1     Running   0           100s
```

Verify that `STATUS` is `Running`.

Troubleshoot 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

The following reference documentation exists.

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.

Developer Conventions for Tanzu Application Platform

Overview

Developer Conventions is a set of [conventions](#) that enable your workloads to support live-update and debug operations. It is used alongside the [Tanzu CLI Apps plug-in](#) and the [Tanzu Developer Tools for Visual Studio Code](#) 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 describes how to install Developer 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 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 [Convention Service](#).

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 --namespace 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`.

Learning Center for Tanzu Application Platform

Overview

Learning Center provides a platform for creating and self-hosting workshops. It allows content creators 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. The workshop environment can be destroyed 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 as it is usually not necessary to provide elevated access to the Kubernetes cluster, and role-based access controls (RBAC) can be used to prevent users from interfering with each other. Quotas can also be set so that users are restricted to how much resources they can use.

When needing to run workshops that deal with cluster operations, for which users need cluster admin access, a separate cluster is created 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 can be done at runtime, are as follows:

- Everything for the workshop must be stored 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.
- Instructions can be annotated 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.
- Text can be annotated 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.
- Additional Kubernetes resources specific to a workshop session can be created in advance of the session. This enables the deployment of applications for each user session.
- Additional Kubernetes resources common to all workshop sessions can be deployed 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

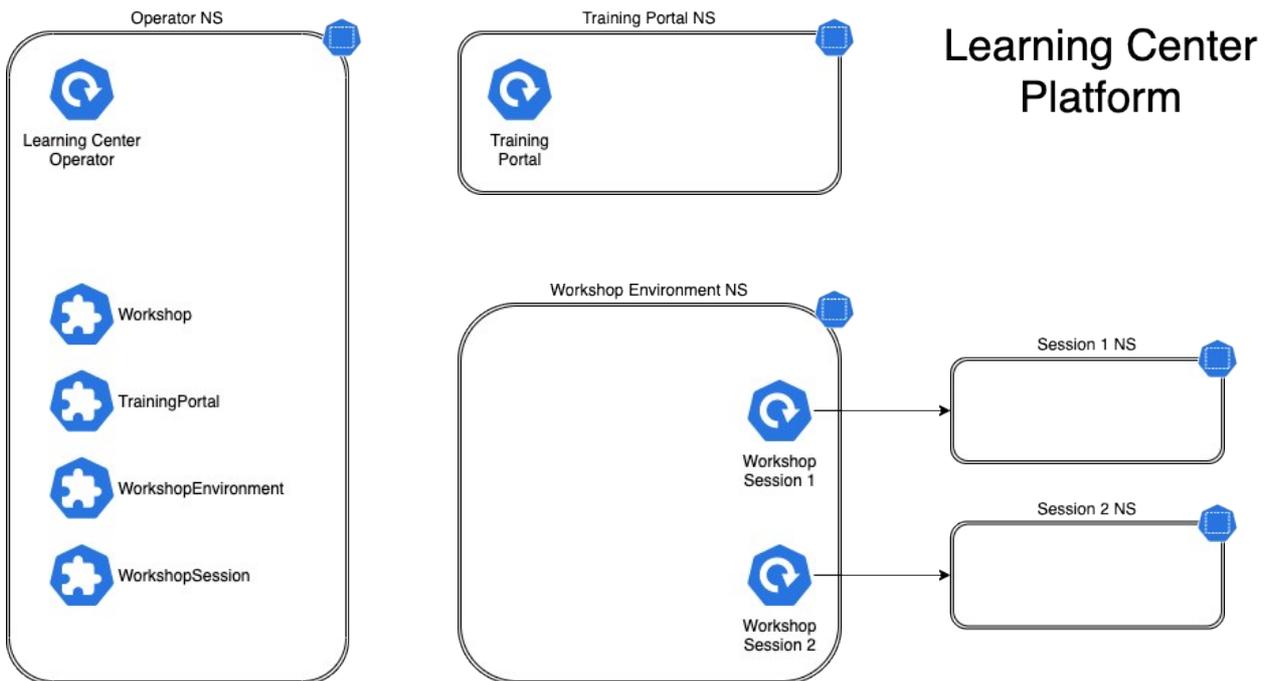
edit files during the workshop.

- 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](#)

Install Learning Center

This document describes how to install Learning Center 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 Learning Center. For more information about profiles, see [About Tanzu Application Platform components and 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 [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 TAP package through the full profile or light profile, it already deploys a contour ingress controller.
- The operator, when deploying instances of the workshop environments, needs to 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 wish 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 need to override which is used.

Install

To install Learning Center:

1. List version information for the package by running:

```
tanzu package available list learningcenter.tanzu.vmware.com --namespace tap-in
stall
```

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. Add the parameter `ingressDomain` to `learning-center-config.yaml`, as in this 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. This access is needed to discover the domain name that can be used as a suffix to hostnames 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 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 will cause a failure. Internal services needing to connect to each other will connect to themselves instead because the address would resolve 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/V2NCTG9uXa9aAiYt7WJ3MA0GCSqG4Ib3DQEBwUAA
    . . .
    dHa6Ly9yMy5vamxlbmNyLm9yZzAiBggrBgEFBQawAoYWaHR0cDoaL3IzLmkubGVu
    -----END CERTIFICATE-----
  privateKey: |
    -----BEGIN PRIVATE KEY-----
    MIIEvQIBADAABgkqhkiG9w0BAQEFAASCBCkwggSjAgEAAoIBAAcX4nyc2xwaVOzf
    . . .
    IY/9SatMcJZivH3Fla7SXL98PawPIOSR7986P7rLFHzNjaQQ0DWTaXBRt+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.

Wildcard certificates can be created 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.vmwar
```

```
e.com --version 0.x.x -f learning-center-config.yaml
```

The command above will create a default namespace in your Kubernetes cluster called `learningcenter`, and the operator, along with any required namespaced resources, is created in it. A set of custom resource definitions and a global cluster role binding are also created.

You can check that the operator deployed successfully by running:

```
kubectl get all -n learningcenter
```

The pod for the operator should be marked as running.

Procedure to install the Self-Guided Tour Training Portal and Workshop

To install the Self-Guided Tour Training Portal and Workshop:

1. Make sure 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 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 "learningcenter" 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 Center
#! 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 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.

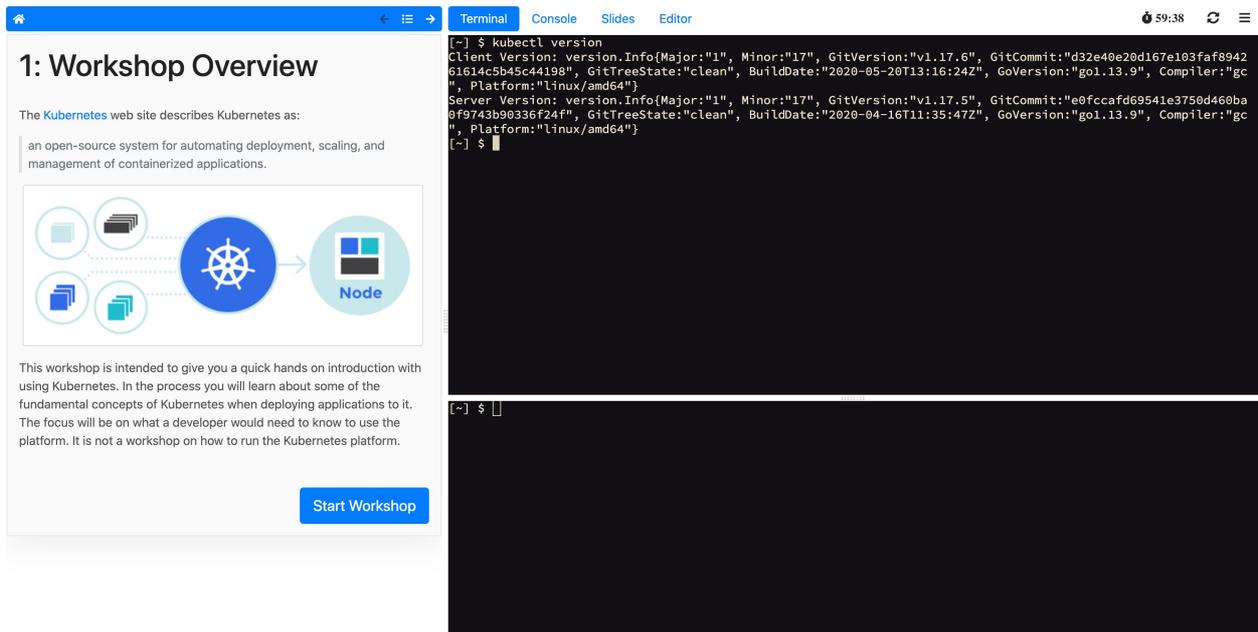
```

```
network:
  blockCIDsRs:
    - 169.254.169.254/32
    - fd00:ec2::254/128
```

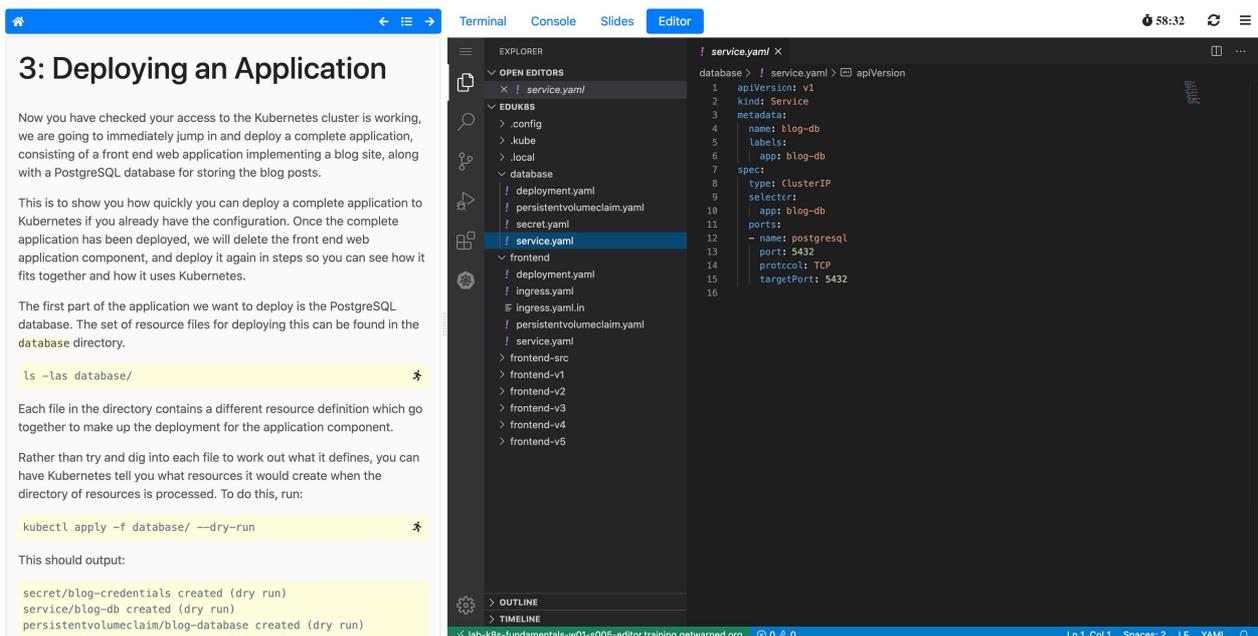
See [Restricting Network Access](#) for more information on blocking CIDRs.

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 requiring commands to be run, one or more terminal shells are provided. More detailed information about Workshops including creating your own: [Creating Learning Center Workshops](#)



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.



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.

2: 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:

```
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 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.

When run, you should see output similar to:

```
Client Version: version.Info{Major:"1", Minor:"17",
GitVersion:"v1.17.0", GitCommit:"70132b0f130acc0bed193d9
ba59dd186f0e634c", GitTreeState:"clean", BuildDate:"2019-12-
07T21:20:10Z", GoVersion:"go1.13.4", Compiler:"
```

The slide also features a visual equation: a blue Kubernetes logo (a ship's wheel) followed by an equals sign, followed by a broken egg emoji with a sad face.

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.

4: Creating the Resources

By doing a dry run deployment, you have seen the resources that will be created. To actually deploy the database component, now run:

```
kubectl apply -f database/
```

As with the dry run, `kubectl apply` will list the resources, except this time the resources will be created.

```
secret/blog-credentials created
service/blog-db created
persistentvolumeclaim/blog-database created
deployment.apps/blog-db created
```

The key resource in this list is deployment. It specifies the name of the container image to be deployed for an application, how many instances should be started, and the strategy for how the deployment should be managed.

To monitor progress of the deployment, and know when it has completed, you can run the command:

```
kubectl rollout status deployment/blog-db
```

The argument is the full name of the resource, including the type of resource and the name for this instance. In this case the instance was called `blog-db`.

With the database deployed, now deploy the front end web application by running:

```
kubectl apply -f frontend/
```

The screenshot shows the Kubernetes dashboard interface. The left sidebar lists various resources like Cluster Roles, Namespaces, Nodes, etc. The main area shows 'Workload Status' with three green circles representing Deployments, Pods, and Replica Sets. Below that is a table of Deployments:

Name	Labels	Pods	Created	Images
blog	app: blog	2 / 2	10 minutes ago	quay.io/educ8s-labs/app-k8s-fundamentals-frontend:latest
blog-db	app: blog-db	1 / 1	11 minutes ago	centos/postgresql-96-centos7:latest

Alternatively, you can enable Octant as the web console.

1: Workshop Overview

Octant is an open source developer-centric web interface for Kubernetes that lets you inspect a Kubernetes cluster and its applications.

It provides an alternative to the de-facto Kubernetes dashboard that is typically available with a Kubernetes cluster. Whereas the Kubernetes dashboard would be hosted in the cluster, Octant is deployed to your own local desktop machine. Octant works with your local Kubernetes client configuration, meaning you can use it with whatever Kubernetes cluster you are working with. You can easily switch contexts, allowing you to operate against a cluster with a different identity, or even change clusters.

Is this workshop you will get a quick introduction to Octant, how to navigate its interface to access details of your Kubernetes cluster and the

The screenshot shows the Octant interface with the following sections:

- Overview**: Summary of the cluster.
- Deployments**:

Name	Labels	Status	Age	Containers	Selector
blog	app.blog	2/2	33s	blog	app.blog
blog-db	app.blog-db	1/1	1m	postgresql	app.blog-db
- Pods**:

Name	Labels	Ready	Phase	Restarts	Node	Age
blog-79b5449985-7gnpr	app.blog	1/1	Running	0	minikube	33s
blog-79b5449985-8en55	app.blog	1/1	Running	0	minikube	33s
blog-db-59b8fcd66-jbww	app.blog-db	1/1	Running	0	minikube	1m
- ReplicaSets**: Summary of replica sets.

Getting started with Learning Center

To view information about Learning Center, see [Learning Center for Tanzu Application Platform](#). Before deploying workshops, 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 information about installing Learning Center, see [Install Learning Center](#).

Other useful information about getting started with Learning Center:

- [Learning Center operator](#)
- [Deleting an operator](#)
- [Workshops](#)
- [TrainingPortal](#)

Learning Center operator

Before deploying workshops, 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

The following is additional information about installing and setting up the Learning Center operator.

The Learning Center operator can be deployed 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 long-lived websocket connections can be used. 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 should have been defined so that persistent volume claims do not need to specify a storage class. For some Kubernetes distributions, including from IBM, it is necessary to 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.

Note: For the custom domain you are using, DNS must have been configured 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` domain name because such domain names require using 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` domain name cannot work.

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`.

```
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-----
    MIIIEvQIBADAaBgkqhkiG9waBAQEFAASCBCkcgwSjAgEAAoIBAAcX4nyc2xwaVOzf
    . . .
    IY/9SatMcJZivH3Fla7SXL98PawPIOSR7986P7rLFHzNjaQQ0DWTaXBRT+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=workshops
.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 won't 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=workshops.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=contour
```

This applies only to the ingress created for the training portal and workshop sessions. It does not apply to the 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 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`.

Deleting Learning Center

Follow these steps to delete Learning Center:

1. Delete all current workshop environments by running:

```
kubectl delete workshops,trainingportals,workshoprequests,workshopsessions,workshopenvironments --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,workshopenvironments --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 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](#). The following helps you get started with workshops. 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-environment.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
lab-k8s-fundamentals	lab-k8s-fundamentals	lab-k8s-fundamentals	{YOUR-REGISTRY-URL}/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-request.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 OgZvfHM7m	http://lab-k8s-fundamentals-cvh51.test	learningcenter	buQ

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.

TrainingPortal

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-portal.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
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*

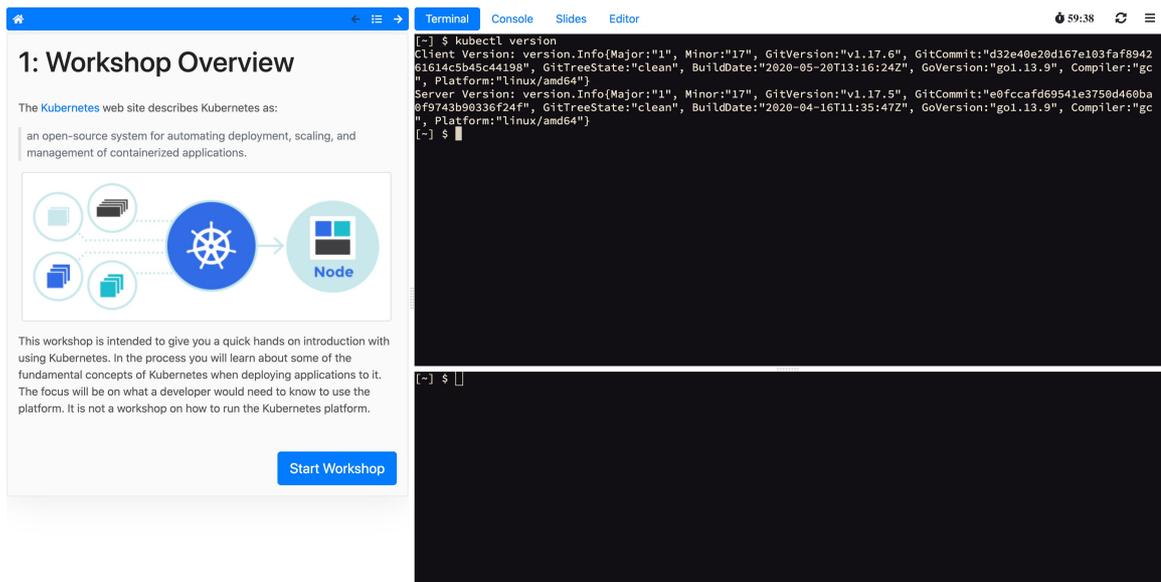
Enter the same password as before, for verification.

[Register](#)

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.

- 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.



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
```

Learning Center local install guides

The following guides tell you how to install Learning Center on your local environment:

- [Installing on Kind](#)
- [Installing on Minikube](#)

Installing on 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/latest/download/release.yml
```

To install secret-gen controller, run:

```
kapp deploy -a sg -f https://github.com/vmware-tanzu/carvel-secretgen-controller/releases/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, `1.1.0`.

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:

```
#! The namespace in which to deploy Learning Center. For now this must be "learningcenter" 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
#! Primary image registry where Learning Center container images are stored. It
#! is only necessary to define the host and credentials when that image
#! registry requires authentication to access images. This principally
#! exists to allow relocation of images through Carvel image bundles.
imageRegistry:
host: null
username: null
password: null
#! Container image versions for various components of Learning Center. The Lear
ning Center
#! operator needs to be modified to read names of images for the registry
#! and docker-in-docker from config map to enable disconnected install.
#! https://github.com/educ8s/educ8s-operator/issues/112
#! 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
#! Override operator image. Only used during development of Learning Center.
operatorImage: null

```

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/educ8s-operator -n educ8s 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.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 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 use 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-registry.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-registry.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-registry.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-registry.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-registry.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](#).

Installing on 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/latest/download/release.yml
```

To install secret-gen controller, run:

```
kapp deploy -a sg -f https://github.com/vmware-tanzu/carvel-secretgen-controller/releases/latest/download/release.yml
```

Note: Type “y” and enter to continue when prompted during installation of both kapp and secretgen 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:VERSION-NUMBER \
  --namespace tap-install
```

Where `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.1.0`.

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 the Learning Center package

Create a file called `learningcenter-value.yaml` in your current directory with the following data:

```
#! The namespace in which to deploy Learning Center.
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. 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. Storage group might need to be
#! set where a cluster has pod security policies enabled, usually setting it
#! to one. Storage user and storage group can be set in exceptional cases
#! where storage class used maps to NFS storage and storage server requires that
#! 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 can be obtained from the custom resource
#! for the training portal.
portalCredentials:
  systemAdmin:
    username: learningcenter
    password: null
  clientAccess:
    username: robot@learningcenter
    password: null
#! Primary image registry where Learning Center container images are stored. You
#! need only define the host and credentials when that image
#! registry requires authentication to access images. This principally
#! exists to allow relocation of images through Carvel image bundles.
imageRegistry:
  host: null
  username: null
  password: null
#! Container image versions for various components of Learning Center. The Learning Ce
n
ter
#! Operator must be modified to read names of images for the registry
#! and docker-in-docker from config map to enable disconnected install.
#! https://github.com/educ8s/educ8s-operator/issues/112
#! Prepull images to nodes in cluster. This is an empty list if no images
#! are prepulled. Normally you only 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
#! Override operator image. Only used during development of Learning Center.
operatorImage: null

```

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=192.168.64.1.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.

Creating Learning Center workshops

This section includes information on creating Learning Center workshops.

- [Workshop configuration](#)
- [Workshop images](#)
- [Workshop content](#)
- [Building an image](#)
- [Workshop instructions](#)
- [Workshop runtime](#)
- [Workshop slides](#)

Workshop configuration

There are two main parts to the configuration for a 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-GITHUB-URL-FOR-LAB-MARKDOWN-SAMPLE
  content:
    image: quay.io/educ8s/lab-markdown-sample:main
  duration: 15m
  session:
    namespaces:
      budget: small
    applications:
      console:
        enabled: true
      editor:
        enabled: true
```

Where:

- `YOUR-GITHUB-URL-FOR-LAB-MARKDOWN-SAMPLE` is the Git repository URL for `lab-markdown-sample`. For example, <https://github.com/educ8s/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-GITHUB-URL-FOR-LAB-MARKDOWN-SAMPLE
  content:
    files: YOUR-GITHUB-URL-FOR-LAB-MARKDOWN-SAMPLE
  duration: 15m
  session:
    namespaces:
```

```

budget: small
applications:
  console:
    enabled: true
  editor:
    enabled: true

```

Where:

- `YOUR-GITHUB-URL-FOR-LAB-MARKDOWN-SAMPLE` is the Git repository URL for `lab-markdown-sample`. For example, <https://github.com/educ8s/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](#).

Workshop images

The workshop environment for the Learning Center is packaged as a container 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 need to 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.

Workshop 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. There are several best practices for speeding up the iterative loop of editing and testing a workshop when developing workshop content.

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>/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: ghcr.io/eduk8s-labs/lab-sample-workshop:main
    files: <YOUR-GIT-REPO>/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: ghcr.io/educ8s-labs/custom-environment:main
  files: github.com/educ8s-labs/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.

Building an image

This topic explains how to 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:a8870aa60b45495d298df5b65c69b3d7972608da4367bd6e69d6e392ac969dd4

COPY --chown=1001:0 . /home/educ8s/

RUN mv /home/educ8s/workshop /opt/workshop

RUN fix-permissions /home/educ8s
```

The default `Dockerfile` action is to:

- Copy all files from a registry to the `/home/educ8s` 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/educ8s/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:a8870aa60b45495d298df5b65c69b3d7972608da4367bd6e69d6e392ac969dd4
```

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 ${IMAGE_REPOSITORY}/pkgs-java-tools as java-tools
FROM registry.tanzu.vmware.com/tanzu-application-platform/tap-packages@sha256:a8870aa60b45495d298df5b65c69b3d7972608da4367bd6e69d6e392ac969dd4
COPY --from=java-tools --chown=1001:0 /opt/jdk11 /opt/java
COPY --from=java-tools --chown=1001:0 /opt/gradle /opt/gradle
COPY --from=java-tools --chown=1001:0 /opt/maven /opt/maven
COPY --from=java-tools --chown=1001:0 /opt/code-server/extensions/. /opt/code-server/extensions/
COPY --from=java-tools --chown=1001:0 /home/educ8s/. /home/educ8s/
COPY --from=java-tools --chown=1001:0 /opt/educ8s/. /opt/educ8s/
ENV PATH=/opt/java/bin:/opt/gradle/bin:/opt/maven/bin:$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/educ8s`. 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/educ8s`, thinking it is the `root` home directory, because the `HOME` environment variable is by default set to `/home/educ8s`. 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/educ8s/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/educ8s`, 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.

Workshop instructions

Individual module files making up the workshop instructions can use either [Markdown](#) or [AsciiDoc](#) markup formats. The extension used on the file should be `.md` or `.adoc`, corresponding to which formatting markup style you use.

Annotation of executable commands

In conjunction with the standard Markdown and AsciiDoc, additional annotations can be applied 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."

```

## 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 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 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 can be 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_cert`: 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.

Workshop runtime

Your workshop content can script the steps a user must run for a workshop. In some cases, you must parameterize that 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/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, a number of predefined environment variables are also available that can be referenced directly.

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/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/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/educ8s` 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 lose 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 targeting 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
```

Presenter slides

If a 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`.

Using 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.

Using 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.

Learning Center runtime environment

This section includes information about the Custom Resource Definitions (CRDs) that are part of the Learning Center:

- [Custom resource overview](#)
- [Workshop resource](#)
- [WorkshopEnvironment resource](#)
- [WorkshopRequest resource](#)
- [WorkshopSession resource](#)
- [TrainingPortal resource](#)
- [SystemProfile resource](#)

Custom resources

You can deploy workshop images directly to a container runtime. 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: github.com/educ8s/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.

Workshop resource

The `Workshop` custom resource defines a 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: github.com/educ8s/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-GITHUB-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-GITHUB-URL-FOR-LAB-MARKDOWN-SAMPLE

```

Where:

- The `url` field is the Git repository URL for `lab-markdown-sample`. For example, <https://github.com/educ8s/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, `https://github.com/educ8s/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: github.com/educ8s/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=main` 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: quay.io/eduk8s/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: github.com/educ8s-tests/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`, `: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/educ8s-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: github.com/educ8s/lab-markdown-sample
  session:
    env:
      - name: REPOSITORY-URL
        value: YOUR-GITHUB-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, <https://github.com/educ8s/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: quay.io/educ8s/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: quay.io/educ8s/lab-markdown-sample:main
  session:
    resources:
      storage: 5Gi
```

The persistent volume is mounted on top of the `/home/educ8s` 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: quay.io/educ8s/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 |
| 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 |

| Budget | Minimum | Maximum | Request | Limit |
|-----------|---------|---------|---------|-------|
| 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: quay.io/educ8s/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/educ8s-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/educ8s-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/educ8s-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/educ8s-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/educ8s-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/educ8s-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/educ8s-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/educ8s-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/educ8s-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/educ8s-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/educ8s-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/educ8s-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/educ8s-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`, `pkc-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/educ8s-tests/lab-policy-testing
  session:
    namespaces:
      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: quay.io/eduk8s-tests/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: quay.io/eduk8s-tests/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: quay.io/eduk8s-tests/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: quay.io/eduk8s-tests/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: quay.io/eduk8s-tests/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: quay.io/eduk8s-tests/lab-application-testing:main
  session:
    applications:
      workshop:
        url: $(ingress_protocol)://$(workshop_namespace)-instructions.$(ingress_domain)
)
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.

Disabling 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: quay.io/educ8s-tests/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: quay.io/educ8s-tests/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: quay.io/educ8s-tests/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: quay.io/educ8s-tests/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: quay.io/educ8s-tests/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: quay.io/educ8s-tests/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: quay.io/educ8s-tests/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: quay.io/educ8s-tests/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: quay.io/educ8s-tests/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: quay.io/eduk8s-tests/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: quay.io/eduk8s-tests/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: quay.io/eduk8s-tests/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: quay.io/educ8s-tests/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: quay.io/educ8s-tests/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: quay.io/eduk8s-tests/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: quay.io/educ8s-tests/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: quay.io/eduk8s-tests/lab-application-testing:main
  session:
    dashboards:
      - name: Example
        url: terminal:example
```

WorkshopEnvironment resource

The `WorkshopEnvironment` custom resource defines a 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-GITHUB-URL-FOR-LAB-MARKDOWN-SAMPLE

```

Where `YOUR-GITHUB-URL-FOR-LAB-MARKDOWN-SAMPLE` is the Git repository URL for `lab-markdown-sample`. For example, <https://github.com/educ8s/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 to 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.

WorkshopRequest resource

The [WorkshopRequest](#) custom resource defines a 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 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

TrainingPortal resource

The `TrainingPortal` custom resource triggers the deployment of a set of 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-GITHUB-URL-FOR-LAB-MARKDOWN-SAMPLE

```

Where `YOUR-GITHUB-URL-FOR-LAB-MARKDOWN-SAMPLE` is the Git repository URL for `lab-markdown-sample`. For example, <https://github.com/educ8s/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: 0kGmMlYw46BZT2vCntyrRuFflgQq5ohi
  Username: learningcenter
Robot:
  Password: QrnY67ME9yGasNhq20TbgWA4RzipUvo5
  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 disable 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 disabled, 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.

SystemProfile resource

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-profile -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.blockCIDs`. 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 have 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:
    blockCIDs:
      - 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 disable 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 at 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-environment: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:
      "quay.io/educ8s-labs/lab-k8s-fundamentals:main": "registry.test/lab-k8s-fundamentals: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

```

WorkshopSession resource

The `WorkshopSession` custom resource defines a 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. At the same time, you must 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 you want.

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 a 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 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-GITHUB-URL-FOR-LAB-MARKDOWN-SAMPLE

```

Where `YOUR-GITHUB-URL-FOR-LAB-MARKDOWN-SAMPLE` is the Git repository URL for `lab-markdown-sample`. For example, <https://github.com/educ8s/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.

Learning Center Portal Rest API

This section includes information about the Portal Rest API that you can leverage to gain information and manage your Learning Center instance.

- [Anonymous access](#)
- [Workshop catalog](#)
- [Session management](#)
- [Client authentication](#)

Anonymous access

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.

Workshop catalog

A single training portal can host one or more workshops. The REST API endpoints for the workshops catalog provide a means to list the available workshops and get information about them.

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&state=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.

Session management

The REST API endpoints for session management allow you to request that a workshop session be allocated.

Disabling portal user registration

When you use the REST API to trigger creation of workshop sessions, VMware recommends that you disable 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.test
/
```

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
D8Bhf0egVmsEKYlaOcTywrrpQJGi&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=https://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&state=STOPPING

```

Include the `state` query string parameter more than once to see workshop environments in both `RUNNING` and `STOPPING` states.

Client authentication

The training portal web interface is a quick way of providing access to a set of workshops when running a supervised training workshop. For integrating access to workshops into an existing website or for creating a custom web interface for accessing workshops hosted across one or more training portals, you can use the portal REST API.

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:    0kGmMlYw46BZT2vCntyrRuFflgQq5ohi
        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": "tg31ied56f0o4axuhuZLHj5JpUYCEL",
  "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-id>&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 section includes a list of known issues with troubleshooting and recovery steps for Learning Center.

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.

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 TAP profile and you get this error:

```
Internal error occurred: failed calling webhook "image-policy-webhook.signing.run.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

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.

Cannot update parameters

The training portals do not work or do not show updated parameters.

Run one of the following 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

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.

Supply Chain Choreographer for Tanzu

This topic introduces 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 paved road to production. Orchestrating supply chain resources - 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 like 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 document describes how to install Supply Chain Choreographer 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 full and light profiles include Supply Chain Choreographer. For more information about profiles, see [Installing the Tanzu Application Platform Package and Profiles](#).

Supply Chain Choreographer provides the custom resource definitions the supply chain uses. Each pre-approved supply chain creates a clear road to production and orchestrates supply chain resources. You can test, build, scan, and deploy. Developers can focus on delivering value to users. Application operators can rest assured that all code in production has passed through an approved workflow.

For example, Supply Chain Choreographer passes the results of fetching source code to the component that builds a container image of it, and then passes the container image to a component

that deploys the image.

Prerequisites

Before installing Supply Chain Choreographer:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see [Prerequisites](#).

Install

To install Supply Chain Choreographer:

1. Install v0.3.0 of the `cartographer.tanzu.vmware.com` package, naming the installation `cartographer`.

```
tanzu package install cartographer \
  --namespace tap-install \
  --package-name cartographer.tanzu.vmware.com \
  --version 0.3.0
```

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

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

This document describes how to install Out of the Box Delivery Basic 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 full and light profiles include Out of the Box Delivery Basic. For more information about profiles, see [Installing the Tanzu Application Platform Package and 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-rolebinding'
| 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

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 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
```

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 `imagePullSecrets` 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 v1.1 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 v1.1, 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

This document describes how to install Out of the Box Supply Chain Basic 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 full and light profiles include Out of the Box Supply Chain Basic. For more information about profiles, see [Installing the Tanzu Application Platform Package and 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 registry server where the application images from the workload should be pushed (required). |
| registry.server | Name of the registry server where application images should be pushed to (required). |
| git_implementation | Determines which git client library to use. Valid options are go-git or libgit2. |
| gitops.username | Default user name to be used for the commits produced by the supply chain. |
| gitops.branch | Default branch to use for pushing Kubernetes configuration files produced by the supply chain. |
| gitops.commit_message | Default git commit message to write when publishing Kubernetes configuration files produced by the supply chain to git. |
| gitops.email | Default user email to be used for the commits produced by the supply chain. |
| gitops.repository_prefix | Default prefix to be used for forming Git SSH URLs for pushing Kubernetes configuration produced by the supply chain. |
| gitops.ssh_secret | Name of the default Secret containing SSH credentials to lookup in the developer namespace for the supply chain to fetch source |

| | |
|-------------------------------|--|
| | code from and push configuration to. |
| cluster_builder
to | Name of the Tanzu Build Service (TBS) ClusterBuilder
use by default on image objects managed by the supply
chain. |
| service_account
e Workload | Name of the service account in the namespace where th
is submitted to utilize for providing registry creden
tials to |
| ploying the | Tanzu Build Service (TBS) Image objects as well as de
ploying the
application. |

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:
  repository_prefix: git@github.com:vmware-tanzu/
  branch: main
  user_name: supplychain
  user_email: supplychain
  commit_message: supplychain@cluster.local
  ssh_secret: git-ssh

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-role'
|
| 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

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 make 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/workload-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.labels["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

This document describes how to install Out of the Box Supply Chain with Testing 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 full and light profiles include Out of the Box Supply Chain with Testing. For more information about profiles, see [Installing the Tanzu Application Platform Package and 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 --namespace tap-install
```

Example output:

```
Deleting installed package 'ootb-supply-chain-testing-scanning' in namespace 'tap-install'.
Are you sure? [y/N]: y

| Uninstalling package 'ootb-supply-chain-testing-scanning' from namespace 'tap-install'
\ Getting package install for 'ootb-supply-chain-testing-scanning'
- Deleting package install 'ootb-supply-chain-testing-scanning' from namespace 'tap-install'
| Deleting admin role 'ootb-supply-chain-testing-scanning-tap-install-cluster-role'
| Deleting role binding 'ootb-supply-chain-testing-scanning-tap-install-cluster-rolebinding'
| Deleting secret 'ootb-supply-chain-testing-scanning-tap-install-values'
| Deleting service account 'ootb-supply-chain-testing-scanning-tap-install-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
here | Name of the repository in the image registry server where the application images from the workload should be pushed (required). |
| registry.server
should | Name of the registry server where application images should be pushed to (required). |
| git_implementation | Determines which git client library to use. Valid options are go-git or libgit2. |
| gitops.username
by the | Default user name to be used for the commits produced by the supply chain. |
| gitops.branch
ation files | Default branch to use for pushing Kubernetes configuration files produced by the supply chain. |
| gitops.commit_message
ubernetes | Default git commit message to write when publishing Kubernetes configuration files produced by the supply chain to git. |
| gitops.email
d by the | Default user email to be used for the commits produced by the supply chain. |
| gitops.repository_prefix
r pushing | Default prefix to be used for forming Git SSH URLs for Kubernetes configuration produced by the supply chain. |
| gitops.ssh_secret
to lookup
tch source | Name of the default Secret containing SSH credentials in the developer namespace for the supply chain to fetch code from and push configuration to. |
| cluster_builder
to | Name of the Tanzu Build Service (TBS) ClusterBuilder to use by default on image objects managed by the supply chain. |
| service_account
e Workload
tials to
ploying the | Name of the service account in the namespace where the Workload is submitted to utilize for providing registry credentials to Tanzu Build Service (TBS) Image objects as well as deploying 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:
  repository_prefix: git@github.com:vmware-tanzu/
  branch: main
  user_name: supplychain
  user_email: supplychain
  commit_message: supplychain@cluster.local
  ssh_secret: git-ssh

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

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 not only pass the source code through testing and vulnerability scanning, but also the container image produced.

This package includes all the capabilities of the Out of the Box Supply Chain With Testing, but adds

source and image scanning using Gype.

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 Gype
 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 make use this supply chain, ensure:

- 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.

To verify 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-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

As mentioned in the prerequisites section, 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 ones, 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.

Note that the names of the objects **must** match the ones in the example.

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 one below (`scan-policy`):

```
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
```

```

kind: ScanPolicy
metadata:
  name: scan-policy
spec:
  regoFile: |
    package policies

    default isCompliant = false

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "UnknownSeverity"
    violatingSeverities := ["Critical","High","UnknownSeverity"]
    ignoreCVEs := []

    contains(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
      fails := contains(violatingSeverities, match.Ratings.Rating[_].Severity)
      not fails
    }

    isSafe(match) {
      ignore := contains(ignoreCVEs, match.Id)
      ignore
    }

    isCompliant = isSafe(input.currentVulnerability)

```

See [Writing Policy Templates](#) for more details.

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](#).

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.labels["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 (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

```

Install Out of the Box Supply Chain with Testing and Scanning

This document describes how to install Out of the Box Supply Chain with Testing and Scanning 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. The full profile includes Out of the Box Supply Chain with Testing and Scanning. For more information about profiles, see [Installing the Tanzu Application Platform Package and 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-role
'
| 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.com
/0.7.0 \
--values-schema \
-n tap-install
```

For example:

| KEY | DESCRIPTION |
|---------------------|--|
| registry.repository | Name of the repository in the image registry server w
here

the application images from the workload should be pu
shed (required). |
| registry.server | Name of the registry server where application images
should

be pushed to (required). |
| git_implementation | Determines which git client library to use.
Valid options are go-git or libgit2. |
| gitops.username | Default user name to be used for the commits produced
by the

supply chain. |

| | |
|---|---|
| <code>gitops.branch</code>
Default branch to use for pushing Kubernetes configuration files | Default branch to use for pushing Kubernetes configuration files produced by the supply chain. |
| <code>gitops.commit_message</code>
Default git commit message to write when publishing Kubernetes configuration files produced by the supply chain to git. | Default git commit message to write when publishing Kubernetes configuration files produced by the supply chain to git. |
| <code>gitops.email</code>
Default user email to be used for the commits produced by the supply chain. | Default user email to be used for the commits produced by the supply chain. |
| <code>gitops.repository_prefix</code>
Default prefix to be used for forming Git SSH URLs for pushing Kubernetes configuration produced by the supply chain. | Default prefix to be used for forming Git SSH URLs for pushing Kubernetes configuration produced by the supply chain. |
| <code>gitops.ssh_secret</code>
Name of the default Secret containing SSH credentials to lookup for the supply chain to push configuration to. | Name of the default Secret containing SSH credentials for the supply chain to push configuration to. |
| <code>cluster_builder</code>
Name of the Tanzu Build Service (TBS) ClusterBuilder to use by default on image objects managed by the supply chain. | Name of the Tanzu Build Service (TBS) ClusterBuilder to use by default on image objects managed by the supply chain. |
| <code>service_account</code>
Name of the service account in the namespace where the Workload is submitted to utilize for providing registry credentials to Tanzu Build Service (TBS) Image objects as well as deploying the application. | Name of the service account in the namespace where the Workload is submitted to utilize for providing registry credentials to Tanzu Build Service (TBS) Image objects as well as deploying the application. |
| <code>cluster_builder</code>
Name of the Tanzu Build Service (TBS) ClusterBuilder to use by default on image objects managed by the supply chain. | Name of the Tanzu Build Service (TBS) ClusterBuilder to use by default on image objects managed by the supply chain. |

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:
  repository_prefix: git@github.com:vmware-tanzu/
  branch: main
  user_name: supplychain
  user_email: supplychain
  commit_message: supplychain@cluster.local
  ssh_secret: git-ssh

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.vmware.com'
| Creating service account 'ootb-supply-chain-testing-scanning-tap-install-sa'
| Creating cluster admin role 'ootb-supply-chain-testing-scanning-tap-install-cluster-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-scanning'
\ 'PackageInstall' resource install status: Reconciling

Added installed package 'ootb-supply-chain-testing-scanning' in namespace 'tap-install'
```

Out of the Box Templates

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: Use the instructions on this page if you do not want to use a profile to install packages. Both the full and light profiles include Out of the Box Templates. For more information about profiles, see

Installing the Tanzu Application Platform Package and 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-rolebinding'
```

```
'
| 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'
```

Building from source

Regardless of the Out of the Box Supply Chain Package you've installed, you can provide source code for the workload either from a directory in your local computer's file system or from a Git repository.

```
Supply Chain

-- fetch source          * either from Git or local directory
-- test
-- build
  -- scan
    -- apply-conventions
      -- push config
```

This document provides details about both approaches.

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 is referenced by the `GitRepository` object created for that workload. See [How It Works](#) to learn more about the underlying process of 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-auth / 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 as explained in the earlier section. 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 auth

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
```

To learn more about how to generate the keys and set it up 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 looks like this:

```
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/default/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 can be 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 can be 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; that is, 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 image registry, the cluster must also have the proper credentials, so it can pull that container image, unpack it, run tests, build the application, and so on.

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`. Then, 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.

Using a prebuilt image

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-app'
[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: sha256:7140722ea396af69fb3d0ad12e9b4419bc3e67d9c5d8a2f6a1421decc4828ace 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.

Git authentication

To either fetch or push source code from or to a repository that requires credentials, you must provide those through a Kubernetes secret object referenced by the intended Kubernetes object created for performing the action.

The following sections provide details about how to appropriately set up Kubernetes secrets for carrying those credentials forward to the proper resources.



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.

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 OPENSSSH PRIVATE KEY-----
    AAAA
    ....
    ....
    -----END OPENSSSH PRIVATE KEY-----
  known_hosts: |
    <known hosts entrys for git provider>
  identity: |
    -----BEGIN OPENSSSH PRIVATE KEY-----
    AAAA
    ....
    ....
    -----END OPENSSSH PRIVATE KEY-----
  identity.pub: ssh-ed25519 AAAABBBCCCCDDDDdeeeeFFFF user@example.com

```

- 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

```

GitOps vs. RegistryOps

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 certain parameters are set in the supply chain:

- `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 is 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.

Authentication

Regardless of how the supply chains are configured, if pushing to Git is configured by repository prefix or repository name, you must provide credentials for the remote provider (for example, GitHub) 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 / 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`.

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 via 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 need to be set up, because the secret containing the credentials for the container image registry were already configured during the setup of the workload namespace.

Authoring supply chains

The Out of the Box Supply Chain, Delivery Basic, and Templates packages provide a set of Kubernetes objects aiming at covering a reference path to production prescribed by VMware. Because VMware recognizes that each organization has their own needs, all of these objects are customizable, whether 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 1.1, 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 excluded_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

In case either the shape of a supply chain or the templates that it points to must be changed, VMware recommends 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:f2ad401bb3e850940...
```

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:f2ad401bb3e850940... \
  -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 through `tap-values.yaml` to be interpolated through 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-

```

Note: 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 through `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: Here 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/Images` to hardcode an environment variable.

1. Exclude the `kpack-template` from the set of templates that `ootb-templates` installs by updating `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:a5e177f38d7287f2ca7ee2afd67ff178645d8f1b1e47af4f192a5ddd6404825e
```

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:a5e177f38d7287f2ca7ee2afd67ff178645d8f1b1e47af4f192a5ddd6404825e \
  -o ootb-templates
```

```
Pulling bundle 'registry.tanzu.vmware.com/tanzu-...
  Extracting layer 'sha256:a5e177f38d7287f2ca7ee2afd67ff178645d8f1b1e47af4f192a5ddd6404825e'...

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 what is provided by 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/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/supply chains are persisted until you revert the `PackageInstalls` to not being paused. To persist the changes, follow the steps outlined in the earlier sections.

Supply Chain Security Tools - 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 (CRDs) for Image and Source Scan.
- CRD for a scanner plug-in. Example is available by using Anchore's Syft and Grype.
- CRD for policy enforcement.
- Enhanced scanning coverage by analyzing the Cloud Native Buildpack SBOMs that Tanzu Build Service images provide.

A Note on Vulnerability Scanners

Although vulnerability scanning is an important practice in DevSecOps and the benefits of it are widely recognized and accepted, 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 can not 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 document describes how to install Supply Chain Security Tools - Scan 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. The full profile includes Supply Chain Security Tools - Scan. For more information about profiles, see [Installing the Tanzu Application Platform Package and 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. It can be present on the same cluster or a different one. You can install Supply Chain Security Tools - Scan by using the CA Secret name for Supply Chain Security Tools - Store present in the same cluster, with Token Secret name for Supply Chain Security Tools - Store in different cluster, or without Supply Chain Security Tools - Store. After you complete installing Supply Chain Security Tools - Store, you must update the Supply Chain Security Tools - Scan values file.

For usage instructions, see [Using the Supply Chain Security Tools - Store](#).

- Install the Tanzu Insight CLI plug-in to query the Supply Chain Security Tools - Store for CVE results. See [Install the Tanzu Insight CLI plug-in](#).

Scanner support

| Out-Of-The-Box Scanner | Version |
|-------------------------------|---------|
| Anchore Grype | v0.33.1 |

Let us know if there's a scanner you'd like us to support.

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-ins
tall
```

For example:

```
$ tanzu package available list scanning.apps.tanzu.vmware.com --namespace tap-i
ninstall
```

```

/ Retrieving package versions for scanning.apps.tanzu.vmware.com...
NAME                                VERSION      RELEASED-AT
scanning.apps.tanzu.vmware.com      1.1.0

```

- (Optional) Make changes to the default installation settings by running:

```

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**.

- Gather the values schema.
- Install the package with default configuration by running:

```

tanzu package install scan-controller \
  --package-name scanning.apps.tanzu.vmware.com \
  --version VERSION \
  --namespace tap-install

```

Where **VERSION** is your package version number. For example, **1.1.0**.

- (Optional) Configure Supply Chain Security Tools - Store in a different cluster

```

---
metadataStore:
  url: META-DATA-STORE-URL
  authSecret:
    name: AUTH-SECRET-NAME

```

Where:

- ◆ **META-DATA-STORE-URL** is the URL pointing to the Supply Chain Security Tools - Store ingress in the cluster that has your Supply Chain Security Tools - Store deployment. For example, <https://metadata-store.example.com:8443>.
- ◆ **AUTH-SECRET-NAME** is the name of the secret that has the auth token to post to the Supply Chain Security Tools - Store.

To install Supply Chain Security Tools - Scan (Grype scanner):

- List version information for the package by running:

```

tanzu package available list grype.scanning.apps.tanzu.vmware.com --namespace tap-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

```

- (Optional) Make changes to the default installation settings by running:

```

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. (Optional) You can define the `--values-file` flag to customize the default configuration. Create a `grype-values.yaml` file by using the following configuration:

```
---
namespace: DEV-NAMESPACE # defaults to default
targetImagePullSecret: TARGET-REGISTRY-CREDENTIALS-SECRET
targetSourceSshSecret: TARGET-REPOSITORY-CREDENTIALS-SECRET
syft:
  failOnSchemaErrors: FAIL-ON-SCHEMA-ERRORS # defaults to true
```

Where:

- ◆ `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 Grype 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.
- ◆ `TARGET-REPOSITORY-CREDENTIALS-SECRET` is the name of the secret that contains the credentials to pull source code from a private repository for scanning. This field is not optional if the source code is located in a public repository.
- ◆ `FAIL-ON-SCHEMA-ERRORS` is a boolean (either `true` or `false`). When `true` the image scan will exit with an error if the provided Syft schema version embedded in the image is incompatible with the schema version supported in the [Grype version](#).

4. VMware recommends using the default values for this package. To change the default

values, see the Scan controller instructions for more information.

5. 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'
```

Spec reference

With the Scan Controller and Grype Scanner installed (see [Install Supply Chain Security Tools - Scan from Installing Individual Packages](#)), 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
```

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 |
|---|---|
| <code>public-source-scan-template</code> | Clones and scans source code from a public repository. |
| <code>private-source-scan-template</code> | Connects with SSH credentials to clone and scan source code from a private repository. |
| <code>public-image-scan-template</code> | Pulls and scans images from a public registry. |
| <code>private-image-scan-template</code> | Connects with the registry credentials to pull and scan images from a private registry. |
| <code>blob-source-scan-template</code> | To be used in a Supply Chain. Gets a <code>.tar.gz</code> available file with <code>wget</code> , 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

This section provides samples on multiple use cases 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

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
spec:
  regoFile: |
    package policies

    default isCompliant = false

    # Accepted Values: "UnknownSeverity", "Critical", "High", "Medium", "Low", "Negligible"
    violatingSeverities := ["Critical"]
    ignoreCVEs := []

    contains(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
      fails := contains(violatingSeverities, match.Ratings.Rating[_].Severity)
      not fails
    }

    isSafe(match) {
      ignore := contains(ignoreCVEs, match.Id)
      ignore
    }

    isCompliant = isSafe(input.currentVulnerability)
  ---
```

```

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

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
spec:
  regoFile: |
    package policies

    default isCompliant = false

    # Accepted Values: "UnknownSeverity", "Critical", "High", "Medium", "Low",
    "Negligible"
    violatingSeverities := ["Critical"]
    ignoreCVEs := []

    contains(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
      fails := contains(violatingSeverities, match.Ratings.Rating[_].Severity)
      not fails
    }
  }
```

```

    isSafe(match) {
      ignore := contains(ignoreCVEs, match.Id)
      ignore
    }

    isCompliant = isSafe(input.currentVulnerability)
  }
}

---
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,scanpolicies -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-NAMESPACE

```

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"]
  ...
```

- 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
```

- Re-describe the SourceScan CR by running:

```
kubectl describe sourcescan sample-public-source-scan-with-compliance-check -n
DEV-NAMESPACE
```

- 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

- Clean up by running:

```
kubectl delete -f sample-public-source-scan-with-compliance-check.yaml -n DEV-N
AMESPACE
```

Sample private image scan

This example performs a scan against an image located in a private registry.

Define the resources

Set up target image pull secret

- 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](#).
- 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 DEV-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

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 OPENSSSH PRIVATE KEY-----
  ....
  ....
  -----END OPENSSSH 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 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAA
IbmlzdHAyNTYAAABBBEmKSEnjQEezOmxkZMy7opKgwFB9nkt5YRrYMjNuG5N87uRgg
6CLrbo5wAdT/y6v0mKV0U2w0WZ2YB/++Tpockg=
github.com ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIOMqqnkVzrm0SdG6U0o
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

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 DEV-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.

Observe Supply Chain Security Tools - Scan

This section outlines observability and troubleshooting methods and issues for using the Supply Chain Security Tools - Scan components.

Watching in-flight jobs

The scan will run inside the job, which creates a Pod. Both the job and Pod will be cleaned up automatically after completion. You can set a watch on the job and Pod before applying a new scan to observe the job deployment.

```
watch kubectl get sourcescans,imagescans,pods,taskruns,scantemplates,scanpolicies -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

Troubleshooting Supply Chain Security Tools - Scan

If you run into any problems or face non-expected behavior, you can always address the logs to get more feedback.

```
kubectl -n scan-link-system logs -f deployment/scan-link-controller-manager -c manager
```

Missing target image pull secret

Scanning an image from a private registry requires an image pull secret to exist in the Scan CR's namespace and be referenced as `grype.targetImagePullSecret` in `tap-values.yaml`. See [Installing the Tanzu Application Platform Package and Profiles](#) for more information.

If a private image scan is triggered and the secret is not configured, the scan job will fail with the error as follows:

```
Job.batch "scan-${app}-${id}" is invalid: [spec.template.spec.volumes[2].secret.secretName: Required value, spec.template.spec.containers[0].volumeMounts[2].name: Not found : "registry-cred"]
```

Disable Supply Chain Security Tools - Store

Supply Chain Security Tools - Store is a prerequisite for installing Supply Chain Security Tools - Scan. If you choose to install without the Supply Chain Security Tools - Store, you need to edit the configurations to disable 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 supported 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 [installation steps](#)). 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 “Unable to decode cyclonedx”

Supply Chain Security Tools - Scan intermittently sets the phase of a scan to `Error` with the message `unable to decode cyclonedx`. To resolve this issue:

- If you’re applying the scan manually, you can delete the failed scan job and re-apply with `kubectl apply -f PATH-TO-IMAGE-SCAN-OR-SOURCE-SCAN -n DEV-NAMESPACE` to retrigger the scan.
- If this happens while running an out of the box Tanzu Application Platform Supply Chain, run `kubectl get imagescans -n WORKLOAD-NAMESPACE` OR `kubectl get sourcescans -n WORKLOAD-NAMESPACE` to get the scan name. Delete the failed scan by running `kubectl delete IMAGE-SCAN-OR-SOURCE-SCAN SCAN-NAME -n WORKLOAD-NAMESPACE`. The Choreographer controller then recreates the scan.

Blob Source Scan is reporting wrong source URL

A Source Scan for a blob artifact might result in reporting the `status.artifact` and `status.compliantArtifact` for the wrong URL for the resource. This passes the remote SSH URL instead of the cluster local fluxcd URL. 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
```

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
status:
  artifact:
    blob:
      ...
      url: ssh://git@github.com:sample/repo.git
  compliantArtifact:
    blob:
      ...
```

```
url: ssh://git@github.com:sample/repo.git
```

Workaround: The following workarounds fix this issue:

1. This problem is resolved in SCST - Scan [v1.2.0](#). Upgrade your SCST - Scan and Grype Scanner deployment to [v1.2.0](#) or later.
2. Configure your SourceScan or Workload to connect to the repository by using HTTPS instead of using SSH.
3. Edit the FluxCD GitRepository resource to not include the [.git](#) directory.

Additional resources

- [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](#)

Configure code repositories and image artifacts to be scanned

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 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 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 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 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
```

Enforce compliance policy using Open Policy Agent

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.

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
spec:
  # A multiline string defining a valid Rego file for policy validation
  regoFile: |
    # Define the package policies
    package policies

    # Give default value to isCompliant to be returned
    # if no change to `true` is applied
    default isCompliant = false

    # Not fail on any CVE with this severities in it
    ignoreSeverities := ["Low"]

    contains(array, elem) = true {
      array[_] = elem
    } else = false { true }

    # Define the rule structure for evaluating CVEs
    isCompliant {
      # Check if the severity level in any of the ratings associated
```

```

# with the current CVEs is present in the ignoreSeverities
# array.
ignore := contains(ignoreSeverities, input.currentVulnerability.Ratings.R
ating[_].Severity)
# If the severity level is in the array, isCompliant will be true
# since `ignore` is. isCompliant will have the default value if `ignore`
is false.
ignore
}

```

2. Deploy the scan policy to the cluster by running:

```

kubectl apply -f <path_to_scan_policy>/<scan_policy_filename>.yaml -n <desired_
namespace>

```

Create a ScanTemplate

The `ScanTemplate` custom resource (CR) defines the Pod with the scanner image that you use for vulnerability scanning. There's a default scanner image you can use out-of-the-box.

Structure

```

apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanTemplate
spec:
  # Required. This field must specify a valid pod.spec.
  # This has the instructions for the scan to be successfully executed.
  # See Pod Requirements section below for more details
  template:

```

Pod requirements

You can define any valid [Kubernetes Pod](#) into the `ScanTemplate` CR if you follow these requirements:

1. **Scanner Container**

The Pod scan must define a container named `scanner` to hold the scanning result.

- ◆ **stdout Logs**

The scan result must be printed in the `stdout` of the `scanner` container having a valid [CycloneDX](#) XML format.

2. **XML Extra Fields**

Component Name

For the scan controller to keep track of your report, provide the name of the scanned artifact in the `bom>metadata>component>name` field of the XML generated as an output. Use the `url` for a source repository. Use the `image` name for an image scan. **Component Digest**

For the Scan Controller to keep track of your report, provide your artifact's digest or most unique identifier of your artifact into the `bom>metadata>component>version` field of the XML generated as an output.

Scanner Name

Provide the name of the scanner you are using in the `bom>metadata>tools>tool>name` field of

the XML generated as an output.

Scanner Vendor

Provide the name of the vendor from the scanner that you are using in the `bom>metadata>tools>tool>vendor` field of the XML generated as an output.

Scanner Version

Provide the version of the scanner you are using in `bom>metadata>tools>tool>version` field of the XML generated as an output.

If the `scanner` Pod is not defined or the logs retrieved from the `stdout` do not have a valid format, the scanning condition fails.

Best practices

1. SourceScan

◊ Init Container

If you're doing a `SourceScan`, we recommend defining the cloning of the repository in an init container named `repo`. Any output in `stdout` in this init container is prompted if an error occurs, so you can have more context about what failed inside the job.

View scan status conditions

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 - Sign

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:

- `PodS`,
- `ReplicaSetS`
- `DeploymentS`
- `JobS`
- `StatefulSetS`
- `DaemonSetS`
- `CronJobS`.

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

Supply Chain Security Tools - Sign is released as an individual Tanzu Application Platform component and is not included in either the full or light profile.

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.vmwara
e.com...
NAME                                VERSION    RELEASED-A
T
image-policy-webhook.signing.apps.tanzu.vmware.com  1.1.2     2022-04-20
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.com/1.1.2 --values-schema --namespace tap-install
| Retrieving package details for image-policy-webhook.signing.apps.tanzu.vmware.com/1.1.2...
  KEY                                DEFAULT                                TYPE                                DESCRIPTION
  allow_unmatched_images             false                                boolean                            Feature flag for enabling admission of images that do not match any patterns in the image policy configuration.
                                                                              Set to true to allow images that do not match any patterns into the cluster with a warning.
  custom_ca_secrets                  <nil>                                array                               List of custom CA secrets that should be included in the application container for registry communication.
                                                                              An array 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 where the referred secret resides.
  custom_cas                          <nil>                                array                               List of custom CA contents that should be included in the application container for registry communication.
                                                                              An array of items containing a ca_content field with the PEM-encoded contents of a certificate authority.
  deployment_namespace              image-policy-system                  string                             Deployment namespace specifies the namespace where this component should be deployed to.
                                                                              If not specified, "image-policy-system" is assumed.
  limits_cpu                          200m                                string                             The CPU limit defines a hard ceiling on how much CPU time that the Image Policy Webhook controller manager container can use.
                                                                              https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/#meaning-of-cpu
  limits_memory                       256Mi                                string                             The memory limit defines a hard ceiling on how much memory that the Image Policy Webhook controller manager container can use.
                                                                              https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/#meaning-of-memory
  quota.pod_number                    5                                    string                             The maximum number of Image Policy Webhook Pods allowed to be created with the priority class system-cluster-critical. This value must be enclosed in quotes (""). If this value is not specified then a default 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 value 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 manager. During CPU contention, CPU request is used as a weighting where higher CPU requests are allocated more CPU time.
https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/#meaning-of-cpu

requests_memory 50Mi string The memory request defines the minimum memory amount for the Image Policy Webhook controller manager.
https://kubernetes.io/docs/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 1.1.2 \
  --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-rolebinding'
| 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

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
    keys:
      - name: cosign-key
        publicKey: |
          -----BEGIN PUBLIC KEY-----
          MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEhyQCx0E9wQWSFI9ULGwy3BuRk1nt
          IqozONbbdbqz11hlRjy9c7SG+hdCF19jE9uE/dwtuwU2MqU9T/cNOYkWww==
          -----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 `ServiceAccounts` 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 | <code>registry.example.org/myproject/my-image:mytag</code> | <code>registry.example.org/myproject/my-image:mytag</code> |
| Any Tag | <code>registry.example.org/myproject/my-image</code> | <code>registry.example.org/myproject/my-image:mytag</code>
<code>registry.example.org/myproject/my-image:other-tag</code> |
| Any Tag | <code>registry.example.org/myproject/my-image:*</code> | <code>registry.example.org/myproject/my-image:mytag</code>
<code>registry.example.org/myproject/my-image:other-tag</code> |
| Any Image and Tag | <code>registry.example.org/myproject/*</code> | <code>registry.example.org/myproject/my-image:mytag</code>
<code>registry.example.org/myproject/anotherimage:another-tag</code> |
| Any Project | <code>registry.example.org/*/my-image:mytag</code> | <code>registry.example.org/myproject/my-image:mytag</code>
<code>registry.example.org/anotherproject/my-image:mytag</code> |
| Any Project and Tag | <code>registry.example.org/*/my-image</code> | <code>registry.example.org/myproject/my-image:mytag</code>
<code>registry.example.org/myproject/my-image:another-tag</code> |
| Registry | <code>registry.example.org/*</code> | <code>registry.example.org/myproject/my-image:mytag</code>
<code>registry.example.org/anotherproject/anotherimage:another-tag</code> |
| Any Subdomain | <code>*.example.org/*</code> | <code>my-registry.example.org/myproject/my-image:mytag</code>
<code>registry.example.org/anotherproject/anotherimage:another-tag</code> |
| Anything | <code>*</code> | <code>my-registry.example.org/myproject/my-image:mytag</code>
<code>registry.example.org/anotherproject/anotherimage:another-tag</code>
<code>registry.io/project/image:tag</code> |



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 signed.): 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 |

| Key | Description |
|------------|---|
| 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 occurred.
Only present with error level |

The possible log messages the webhook emits and their explanations are summarized in the following table:

| Log Message | Explanation |
|---|---|
| <code>clusterimagepolicies.signing.apps.tanzu.vmware.com "image-policy" not found. Image policy enforcement was not applied.</code> | The Image Policy was not created in the cluster and the webhook did not check any container images for signatures. |
| <code><Namespace> is excluded. The ImagePolicy will not be applied.</code> | <ul style="list-style-type: none"> An image policy is present in the cluster. The namespace is present in the <code>verification.exclude.resources.namespaces</code> property of the policy. Any container images trying to get created in this namespace will not be checked for signatures. |
| <code>Could not verify against any image policies for container image: <ContainerImage>.</code> | <ul style="list-style-type: none"> An image policy is present in the cluster. The <code>AllowUnMatchedImages</code> flag is set to <code>false</code> 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. |

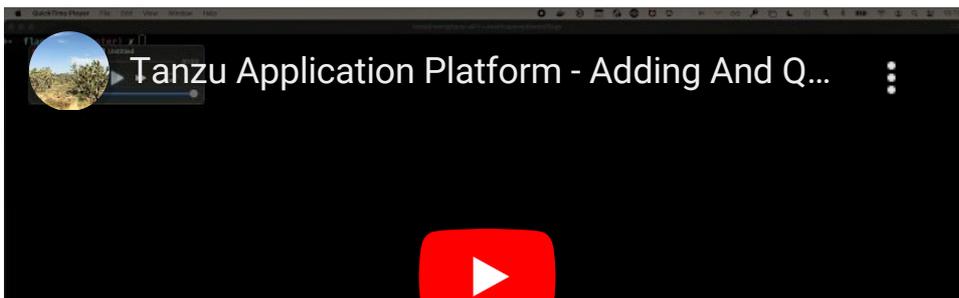
| Log Message | Explanation |
|---|---|
| <pre><ContainerImage> did not match any image policies. Container will be created as AllowUnmatchedImages flag is true.</pre> | <ul style="list-style-type: none"> • An image policy is present in the cluster. • The <code>AllowUnmatchedImages</code> flag is set to <code>true</code>. • 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. |
| <pre>failed to find signature for image.</pre> | <ul style="list-style-type: none"> • 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. |
| <pre>The image: <ContainerImage> is not signed.</pre> | <ul style="list-style-type: none"> • 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. |
| <pre>failed to decode resource</pre> | <ul style="list-style-type: none"> • The resource type is not supported. • Currently supported v1 versions of: <ul style="list-style-type: none"> ✦ Pod ✦ Deployment ✦ StatefulSet ✦ DaemonSet ✦ ReplicaSet ✦ Job ✦ CronJob (and v1beta1) |

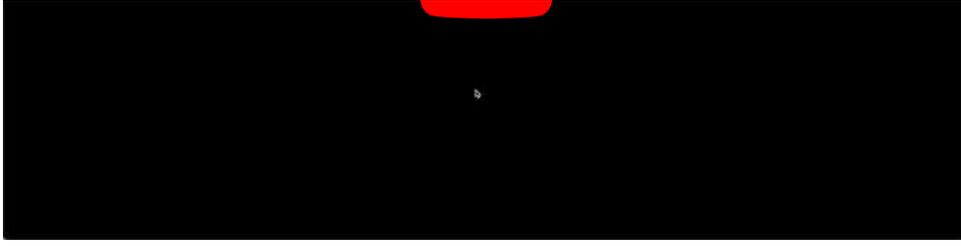
| Log Message | Explanation |
|--|---|
| <pre>failed to verify</pre> | <ul style="list-style-type: none"> • An image policy is present in the cluster. • The namespace you are installing your resource in is not excluded. • Image of the container being installed matches a pattern. • The webhook can not verify the signature. |
| <pre>matching pattern: <Pattern> against image <ContainerImage> matching registry patterns: [{<Image NamePattern, Keys, SecretRef>}]</pre> | <ul style="list-style-type: none"> • Provide the pattern that matches the container image. • Provide the corresponding <code>Image</code> configuration from the <code>ClusterImagePolicy</code> that matches the container image. |
| <pre>service account not found</pre> | <ul style="list-style-type: none"> • 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 <code>imagePullSecrets</code> for container images and their co-located <code>cosign</code> signatures. |
| <pre>unmatched image policy: <ContainerImage></pre> | <p>Container image does not match any policy image patterns.</p> |

Supply Chain Security Tools for Tanzu – Store

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 four-minute demo of scanning an image for CVEs and querying the database for CVEs and dependencies.





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 [Ingress and multicluster support](#) for information about how to set up Supply Chain Security Tools Scan and Store to work together in a multicluster setup.

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.

Install Supply Chain Security Tools - Store independent from Tanzu Application Platform profiles

This document describes how to install Supply Chain Security Tools - Store from the Tanzu Application Platform package repository.

Note: VMware recommends installing Supply Chain Security Tools - Store by using Tanzu Application Platform Profiles. See [Installing the Tanzu Application Platform Package and Profiles](#). Use the following instructions if you do not want to use a profile to install the Supply Chain Security Tools - Store package.

Prerequisites

Before installing Supply Chain Security Tools - Store:

- 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, Contour](#).
- See [Deployment Details and Configuration](#) to review what resources will be deployed. For more information, see the [overview](#).

Install

To install Supply Chain Security Tools - Store:

1. The deployment assumes the user has set up the Kubernetes cluster to provision persistent volumes on demand. Make sure a default storage class is available in your cluster. Check whether default storage class is set in your cluster by running:

```
kubectl get storageClass
```

For example:

```
$ kubectl get storageClass
NAME                                PROVISIONER                RECLAIMPOLICY   VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANSION  AGE
standard (default)      rancher.io/local-path     Delete          WaitForFirstConsumer
er  false                7s
```

2. List version information for the package by running:

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

For example:

```
$ tanzu package available list metadata-store.apps.tanzu.vmware.com --namespace tap-install
- Retrieving package versions for metadata-store.apps.tanzu.vmware.com...
NAME                                VERSION  RELEASED-AT
metadata-store.apps.tanzu.vmware.com  1.0.2
```

3. (Optional) List out all the available deployment configuration options:

```
tanzu package available get metadata-store.apps.tanzu.vmware.com/VERSION --values-schema -n tap-install
```

Where **VERSION** is the your package version number. For example, **1.0.2**.

For example:

```
$ tanzu package available get metadata-store.apps.tanzu.vmware.com/1.0.2 --values-schema -n tap-install
| Retrieving package details for metadata-store.apps.tanzu.vmware.com/1.0.2...
KEY                                DEFAULT                TYPE    DESCRIPTION
app_service_type                   LoadBalancer          string  The type of service to use for the metadata app service. This can be set to 'NodePort' or 'LoadBalancer'.
auth_proxy_host                    0.0.0.0               string  The binding ip address of the kube-rbac-proxy sidecar
db_host                             metadata-store-db     string  The address to the postgres database host that the metadata-store app uses to connect. The default is set to metadata-store-db which is the postgres service name. Changing this does not change the postgres service name
db_replicas                         1                     integer The number of replicas for the metadata-store-db
db_sslmode                         verify-full           string  Determines the security connection between API server and Postgres database. This can be set to 'verify-ca' or 'verify-full'
```

| | | | |
|---|---------------------|---------|--|
| <code>pg_limit_memory</code> | 4Gi | string | Memory limit for postgres container in metadata-store-db deployment |
| <code>app_req_cpu</code> | 100m | string | CPU request for metadata-store-app container |
| <code>app_limit_memory</code> | 512Mi | string | Memory limit for metadata-store-app container |
| <code>app_req_memory</code> | 128Mi | string | Memory request for metadata-store-app container |
| <code>auth_proxy_port</code> | 8443 | integer | The external port address of the kube-rbac-proxy sidecar |
| <code>db_name</code> | metadata-store | string | The name of the database to use. |
| <code>db_port</code> | 5432 | string | The database port to use. This is the port to use when connecting to the database pod. |
| <code>api_port</code> | 9443 | integer | The internal port for the metadata app api endpoint. This will be used by the kube-rbac-proxy sidecar. |
| <code>app_limit_cpu</code> | 250m | string | CPU limit for metadata-store-app container |
| <code>app_replicas</code> | 1 | integer | The number of replicas for the metadata-store-app |
| <code>db_user</code> | metadata-store-user | string | The database user to create and use for updating and querying. The metadata postgres section create this user. The metadata api server uses this username to connect to the database. |
| <code>pg_req_memory</code> | 1Gi | string | Memory request for postgres container in metadata-store-db deployment |
| <code>priority_class_name</code> | | string | If specified, this value is the name of the desired PriorityClass for the metadata-store-db deployment |
| <code>use_cert_manager</code> | true | string | Cert manager is required to be installed to use this flag. When true, this creates certificates object to be signed by cert manager for the API server and Postgres database. If false, the certificate object have to be provided by the user. |
| <code>api_host</code> | localhost | string | The internal hostname for the metadata api endpoint. This will be used by the kube-rbac-proxy sidecar. |
| <code>db_password</code> | <auto-generated> | string | The database user password. If not specified, the password will be auto-generated. |
| <code>storage_class_name</code> | | string | The storage class name of the persistent volume used by Postgres database for storing data. The default value will use the default class name defined on the cluster. |
| <code>database_request_storage</code> | 10Gi | string | The storage requested of the persistent volume used by Postgres database for storing data. |
| <code>add_default_rw_service_account</code> | true | string | Adds a read-write service account which can be used to obtain access token to use metadata-store CLI |
| <code>log_level</code> | default | string | Sets the log level. This can be set to "minimum", "less", "default", "more", "debug" or "trace". "minimum" currently does not output logs. "less" outputs log configuration options only. "default" and "more" outputs API endpoint access information. "debug" and "trace" outputs extended API endpoint access information (such as body payload) and other debug information. |

- (Optional) Modify one of the deployment configurations by creating a configuration YAML with the custom configuration values you want. For example, if your environment does not support `LoadBalancer`, and you want to use `NodePort`, then create a `metadata-store-values.yaml` and configure the `app_service_type` property.

```
---
app_service_type: "NodePort"
```

See [Deployment details and configuration](#) for more information about configuration options.

See [Ingress and multicluster support](#) for more information about ingress and custom domain name support.

5. Install the package by running:

```
tanzu package install metadata-store \
  --package-name metadata-store.apps.tanzu.vmware.com \
  --version VERSION \
  --namespace tap-install \
  --values-file metadata-store-values.yaml
```

Where:

- ◆ `--values-file` is an optional flag. Only use it to customize the deployment configuration.
- ◆ `VERSION` is the package version number. For example, `1.0.2`.

For example:

```
$ tanzu package install metadata-store \
  --package-name metadata-store.apps.tanzu.vmware.com \
  --version 1.0.2 \
  --namespace tap-install \
  --values-file metadata-store-values.yaml

- Installing package 'metadata-store.apps.tanzu.vmware.com'
/ Getting namespace 'tap-install'
- Getting package metadata for 'metadata-store.apps.tanzu.vmware.com'
/ Creating service account 'metadata-store-tap-install-sa'
/ Creating cluster admin role 'metadata-store-tap-install-cluster-role'
/ Creating cluster role binding 'metadata-store-tap-install-cluster-rolebinding'
'
/ Creating secret 'metadata-store-tap-install-values'
| Creating package resource
- Package install status: Reconciling

Added installed package 'metadata-store' in namespace 'tap-install'
```

Configure target endpoint and certificate

The connection to the Store requires TLS encryption, the configuration depends on the kind of installation. Use the following instructions to set up the TLS connection according to the type of your setup:

- [Use Ingress](#)
- [Not use Ingress](#)
 - ◆ [Use LoadBalancer](#)
 - ◆ [Use NodePort](#)

Note: `NodePort` is commonly used with local clusters such as `kind` or `minikube`.

Use Ingress

When using an [Ingress setup](#), the Store creates a specific TLS Certificate for HTTPS communications under the `metadata-store` namespace.

To get such certificate, run the following command:

```
kubectl get secret ingress-cert -n metadata-store -o json | jq -r '.data."ca.crt"' | base64 -d > insight-ca.crt
```

The endpoint host is set to `metadata-store.<ingress-domain>`, for example, `metadata-store.example.domain.com`). This value matches the value of `ingress_domain`.

If no accessible DNS record exists for such 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 the target by running:

```
tanzu insight config set-target https://$METADATA_STORE_DOMAIN --ca-cert insight-ca.crt
```

Not use Ingress

If you install the Store without using the Ingress alternative, you must use a different Certificate resource for HTTPS communication. In this case, query the `app-tls-cert` to get the CA Certificate:

```
kubectl get secret app-tls-cert -n metadata-store -o json | jq -r '.data."ca.crt"' | base64 -d > insight-ca.crt
```

Use LoadBalancer

To use a `LoadBalancer` configuration, you must find the external IP address of the `metadata-store-app` service by using `kubectl`.



Note

: For all `kubectl` commands, use the `--namespace metadata-store` flag.

```
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-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 "$METADATA_STORE_IP $METADATA_STORE_DOMAIN" | sudo tee -a /etc/hosts > /dev/null
```

Set the target by running:

```
tanzu insight config set-target https://$METADATA_STORE_DOMAIN:$METADATA_STORE_PORT --
ca-cert insight-ca.crt
```

Use NodePort

To use [NodePort](#), you must obtain the CA certificate by following the instructions in [Not use Ingress](#), then [Configure port forwarding](#) and [Modify your /etc/hosts file](#).

Configure port forwarding

When using [NodePort](#), configure port forwarding for the service so the CLI can access Supply Chain Security Tools - Store. Run:

```
kubectl port-forward service/metadata-store-app 8443:8443 -n metadata-store
```

Note: You must run this command in a separate terminal window.

Modify your /etc/hosts file

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
```

Set the target by running:

```
tanzu insight config set-target https://$METADATA_STORE_DOMAIN:$METADATA_STORE_PORT --
ca-cert insight-ca.crt
```

Configure access tokens

Service accounts are required to generate the access tokens.

The access token is a [Bearer](#) token used in the http request header [Authorization](#). (ex.

Authorization: Bearer eyJhbGciOiJSUzI1NiIsImtpZCI6IjhmV0...)

By default, Supply Chain Security Tools - Store comes with `read-write` service account installed. This service account is cluster-wide.

Service accounts

You can create two types of service accounts:

1. Read-only service account - only able to use `GET` API requests
2. Read-write service account - full access to the API requests

Read-only service account

As a part of the Store installation, the `metadata-store-read-only` cluster role is created by default. This cluster role allows the bound user to have `get` access to all resources. To bind to this cluster role, run the following command:

```
kubectl apply -f - --o yaml << EOF
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: metadata-store-read-only
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: metadata-store-read-only
subjects:
- kind: ServiceAccount
  name: metadata-store-read-client
  namespace: metadata-store
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: metadata-store-read-client
  namespace: metadata-store
automountServiceAccountToken: false
EOF
```

If you do not want to bind to a cluster role, create your own read-only role in the `metadata-store` namespace with a service account. The following example command creates a service account named `metadata-store-read-client`:

```
kubectl apply -f - --o yaml << EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: metadata-store-ro
  namespace: metadata-store
rules:
- resources: ["all"]
  verbs: ["get"]
  apiGroups: [ "metadata-store/v1" ]
```

```

---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: metadata-store-ro
  namespace: metadata-store
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: metadata-store-ro
subjects:
- kind: ServiceAccount
  name: metadata-store-read-client
  namespace: metadata-store
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: metadata-store-read-client
  namespace: metadata-store
automountServiceAccountToken: false
EOF

```

Read-write service account

To create a read-write service account, run the following command. The command creates a service account called `metadata-store-read-write-client`:

```

kubectl apply -f - -o yaml << EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: metadata-store-read-write
  namespace: metadata-store
rules:
- resources: ["all"]
  verbs: ["get", "create", "update"]
  apiGroups: [ "metadata-store/v1" ]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: metadata-store-read-write
  namespace: metadata-store
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: metadata-store-read-write
subjects:
- kind: ServiceAccount
  name: metadata-store-read-write-client
  namespace: metadata-store
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: metadata-store-read-write-client
  namespace: metadata-store

```

```
automountServiceAccountToken: false
EOF
```

Getting the Access Token

To retrieve the read-only access token, run the following command:

```
kubectl get secret $(kubectl get sa -n metadata-store metadata-store-read-client -o json | jq -r '.secrets[0].name') -n metadata-store -o json | jq -r '.data.token' | base64 -d
```

To retrieve the read-write access token run the following command:

```
kubectl get secret $(kubectl get sa -n metadata-store metadata-store-read-write-client -o json | jq -r '.secrets[0].name') -n metadata-store -o json | jq -r '.data.token' | base64 -d
```

The access token is a “Bearer” token used in the http request header “Authorization.” (ex.

```
Authorization: Bearer eyJhbGciOiJSUzI1NiIsImtpZCI6IjhmV0...
```

Setting the Access Token

When using the CLI, you’ll need to set the `METADATA_STORE_ACCESS_TOKEN` environment variable, or use the `--access-token` flag. It is not recommended to use the `--access-token` flag as the token will appear in your shell history.

The following command will retrieve the access token from Kubernetes and store it in `METADATA_STORE_ACCESS_TOKEN` where `SERVICE-ACCOUNT-NAME` is the name of the service account you plan to use.

```
export METADATA_STORE_ACCESS_TOKEN=$(kubectl get secrets -n metadata-store -o jsonpath="{.items[?(@.metadata.annotations['kubernetes.io/service-account.name']=='SERVICE-ACCOUNT-NAME')].data.token}" | base64 -d)
```

For example:

```
$ export METADATA_STORE_ACCESS_TOKEN=$(kubectl get secrets -n metadata-store -o jsonpath="{.items[?(@.metadata.annotations['kubernetes.io/service-account.name']=='metadata-store-read-write-client')].data.token}" | base64 -d)
```

Security details

Application security

TLS encryption

Supply Chain Security Tools - Store requires TLS connection. If certificates are not provided, the application will 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

Supply Chain Security Tools - Store uses [kube-rbac-proxy](#) as the only entry point to its API.

Authentication and Authorization must be completed successfully via the [kube-rbac-proxy](#) before its API is accessible.

Authentication

The [kube-rbac-proxy](#) uses [Token Review](#) to verify if the token is valid. [Token Review](#) is a Kubernetes API to ensure 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 an access token, please refer to the [Create Service Account Access Token Docs](#).

Authorization

The [kube-rbac-proxy](#) uses [Subject Access Review](#) to ensure users access certain operations. [Subject Access Review](#) is a Kubernetes API that uses [Kubernetes RBAC](#) to determine if the user can perform specific actions. Please refer to the [Create Service Account Access Token doc](#).

There are only two supported roles: [Read Only](#) cluster role and [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 `add_default_rw_service_account` property to `"false"` in the `metadata-store-values.yaml` file [during deployment](#).

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.

Note: There is no support for roles with access to only specific types of resources (i.e., images, packages, vulnerabilities, etc.)

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 Postgres DB pod:

```
allowPrivilegeEscalation: false
runAsUser: 999
fsGroup: 999
```

Note: 65532 is the uid for the “nobody” user. 999 is the uid for the “postgres” 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 Gripe 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

- [API details](#)
- [API walkthrough](#)
- [Deployment details and configuration](#)
- [Install independent from Tanzu Application Platform profiles](#)
- [AWS RDS Postgres configuration](#)
- [Database backup recommendations](#)
- [Log configuration and usage](#)
- [Troubleshooting upgrading](#)
- [Failover, redundancy, and backups](#)
- [Ingress and multicluster support](#)

API details

See [API walkthrough](#) for a walkthrough and 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 |

| Code | Status | Description | Has headers | 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 | <code>string</code> | | | | |
| id | query | int64 (formatted integer) | <code>int64</code> | | | | |

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 | <code>string</code> | | ✓ | | |

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 | <code>path</code> | string | <code>string</code> | | ✓ | | |

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. (*GetSources*)**

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 | <code>string</code> | | ✓ | | |

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 | <code>string</code> | | ✓ | | |

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 | <code>string</code> | | | in: body | <code>something went wrong</code> |

Image

Properties

| Name | Type | Go type | Required | Default | Description | Example |
|----------|---------------------------------------|-----------------------------|----------|---------|-------------|---|
| Digest | string | <code>string</code> | ✓ | | | <code>9n38274ods897fmay487gsdyfga678wr82</code> |
| ID | uint64 (formatted integer) | <code>uint64</code> | | | | |
| Name | string | <code>string</code> | ✓ | | | <code>myorg/application</code> |
| Packages | [] Package (#package) | []* Package | | | | |
| Registry | string | <code>string</code> | ✓ | | | <code>docker.io</code> |
| Sources | [] Source (#source) | []* Source | | | | |

MethodType

Properties

| Name | Type | Go type | Required | Default | Description | Example |
|-----------|------------------------------|------------------------------|----------|---------|-------------|---------|
| CreatedAt | date-time (formatted string) | <code>strfmt.DateTime</code> | | | | |
| DeletedAt | DeletedAt | <code>DeletedAt</code> | | | | |
| ID | uint64 (formatted integer) | <code>uint64</code> | | | | |
| Name | string | <code>string</code> | | | | |

| Name | Type | Go type | Required | Default | Description | Example |
|-----------|-------------------------------------|---------------------------------|----------|---------|-------------|---------|
| 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 | | | | |

| Name | Type | Go type | Required | Default | Description | Example |
|-----------------|---|-----------------------------------|----------|---------|-------------|---------|
| 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 |

| Name | Type | Go type | Required | Default | Description | Example |
|-------------|-----------------------------------|--------------------------|----------|---------|-------------|---------|
| Description | string | <code>string</code> | | | | |
| ID | uint64 (formatted integer) | <code>uint64</code> | | | | |
| Packages | <code>[]Package</code> (#package) | <code>[]*Package</code> | | | | |
| Ratings | <code>[]Rating</code> (#rating) | <code>[]*Rating</code> | | | | |
| References | <code>StringArray</code> | <code>StringArray</code> | | | | |
| URL | string | <code>string</code> | | | | |

API walkthrough

This topic includes an example API call. For information about using the Supply Chain Security Tools - 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 the following:

```
kubectl port-forward service/metadata-store-app 8443:8443 -n metadata-store
```

2. Retrieve the `metadata-store-read-write-client` access token. Ensure the Service Account is [created](#). Run:

```
export METADATA_STORE_ACCESS_TOKEN=$(kubectl get secrets -n metadata-store -o jsonpath="{.items[?(@.metadata.annotations['kubernetes.io/service-account.name']=='metadata-store-read-write-client')].data.token}" | base64 -d)
```

3. Retrieve the CA Certificate and store it locally. Run the following:

```
kubectl get secret app-tls-cert -n metadata-store -o json | jq -r '.data."ca.crt"' | base64 -d > /tmp/ca.crt
```

4. Run the Curl POST Command:

```
curl https://metadata-store-app:8443/api/imageReport \
  --resolve metadata-store-app:8443:127.0.0.1 \
  --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@45asd61asasssdfsdffddssghjkdfsdffasdasdsdasdassdfghjdd
asfddfsadfadfgfshdasdfsdffsdasdsdfsdffadssdfdasdfaasdsdfsdffsdasgsasddffdgf
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

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 and persistent volume claim.
- External IP address (based on a deployment configuration set to use [LoadBalancer](#)).
- A Kubernetes secret to allow pulling Supply Chain Security Tools - Store images from a registry.
- A namespace called `metadata-store`.
- A service account with read-write privileges named `metadata-store-read-write-client`. 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

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 still a secure deployment.

Using AWS RDS postgres database

Users can also configure the deployment to use their own RDS database instead of the default. See [AWS RDS Postgres Configuration](#).

Custom database password

By default, a database password is generated automatically upon deployment. To configure a custom password, use the `db_password` property in the `metadata-store-values.yaml` during deployment.

```
db_password: "PASSWORD-0123"
```

If you're deploying with Tanzu Application Platform profiles, in `tap-values.yaml`, put:

```
metadata_store:
```

```
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](#).

App service type

If your environment does not support `LoadBalancer`, and you want to use `NodePort`, configure the `app_service_type` property in your `metadata-store-values.yaml`:

```
app_service_type: "LoadBalancer"
```

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 [Configure access tokens](#).

The store creates a read-only cluster role, which can be bound to a service account through `ClusterRoleBinding`. To create service accounts to bind to this cluster role, see [Configure access tokens](#).

Exporting certificates

Supply Chain Security Tools - Store creates `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.

Ingress support

Supply Chain Security Tools - 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 Supply Chain Security Tools - Store's API. For example:

```
ingress_enabled: "true"
ingress_domain: "example.com"
```

An `HTTPProxy` object is then installed with `metadata-store.example.com` as the fully qualified domain name. See [Ingress and multicluster support](#).

Install Supply Chain Security Tools - Store independent from Tanzu Application Platform profiles

This document describes how to install Supply Chain Security Tools - Store from the Tanzu Application Platform package repository.

Note: VMware recommends installing Supply Chain Security Tools - Store by using Tanzu Application Platform Profiles. See [Installing the Tanzu Application Platform Package and Profiles](#). Use the following instructions if you do not want to use a profile to install the Supply Chain Security Tools - Store package.

Prerequisites

Before installing Supply Chain Security Tools - Store:

- 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, Contour](#).
- See [Deployment Details and Configuration](#) to review what resources will be deployed. For more information, see the [overview](#).

Install

To install Supply Chain Security Tools - Store:

1. The deployment assumes the user has set up the Kubernetes cluster to provision persistent volumes on demand. Make sure a default storage class is available in your cluster. Check whether default storage class is set in your cluster by running:

```
kubectl get storageClass
```

For example:

```
$ kubectl get storageClass
NAME                                PROVISIONER          RECLAIMPOLICY   VOLUMEBINDINGMODE
  ALLOWVOLUMEEXPANSION  AGE
standard (default)  rancher.io/local-path  Delete          WaitForFirstConsumer
er  false                7s
```

2. List version information for the package by running:

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

For example:

```
$ tanzu package available list metadata-store.apps.tanzu.vmware.com --namespace tap-install
- Retrieving package versions for metadata-store.apps.tanzu.vmware.com...
NAME                                VERSION  RELEASED-AT
metadata-store.apps.tanzu.vmware.com  1.0.2
```

3. (Optional) List out all the available deployment configuration options:

```
tanzu package available get metadata-store.apps.tanzu.vmware.com/VERSION --valu
```

```
es-schema -n tap-install
```

Where **VERSION** is the your package version number. For example, **1.0.2**.

For example:

```
$ tanzu package available get metadata-store.apps.tanzu.vmware.com/1.0.2 --valu
es-schema -n tap-install
| Retrieving package details for metadata-store.apps.tanzu.vmware.com/1.0.2...
  KEY                                DEFAULT                                TYPE    DESCRIPTION
  app_service_type                    LoadBalancer                            string  The type of s
ervice to use for the metadata app service. This can be set to 'NodePort' or 'L
oadBalancer'.
  auth_proxy_host                      0.0.0.0                                  string  The binding ip
address of the kube-rbac-proxy sidecar
  db_host                              metadata-store-db                        string  The address t
o the postgres database host that the metdata-store app uses to connect. The de
fault is set to metadata-store-db which is the postgres service name. Changing
this does not change the postgres service name
  db_replicas                          1                                        integer The number of
replicas for the metadata-store-db
  db_sslmode                           verify-full                              string  Determines th
e security connection between API server and Postgres database. This can be set
to 'verify-ca' or 'verify-full'
  pg_limit_memory                      4Gi                                      string  Memory limit
for postgres container in metadata-store-db deployment
  app_req_cpu                           100m                                     string  CPU request f
or metadata-store-app container
  app_limit_memory                     512Mi                                    string  Memory limit
for metadata-store-app container
  app_req_memory                       128Mi                                    string  Memory reques
t for metadata-store-app container
  auth_proxy_port                      8443                                     integer  The external
port address of the of the kube-rbac-proxy sidecar
  db_name                              metadata-store                            string  The name of t
he database to use.
  db_port                              5432                                     string  The database
port to use. This is the port to use when connecting to the database pod.
  api_port                             9443                                     integer  The internal
port for the metadata app api endpoint. This will be used by the kube-rbac-prox
y sidecar.
  app_limit_cpu                        250m                                     string  CPU limit for
metadata-store-app container
  app_replicas                         1                                        integer  The number of
replicas for the metadata-store-app
  db_user                              metadata-store-user                      string  The database
user to create and use for updating and querying. The metadata postgres section
create this user. The metadata api server uses this username to connect to the
database.
  pg_req_memory                       1Gi                                      string  Memory reques
t for postgres container in metadata-store-db deployment
  priority_class_name                  string                                   string  If specified,
this value is the name of the desired PriorityClass for the metadata-store-db
deployment
  use_cert_manager                     true                                     string  Cert manager
is required to be installed to use this flag. When true, this creates certifica
tes object to be signed by cert manager for the API server and Postgres databas
e. If false, the certificate object have to be provided by the user.
  api_host                             localhost                                string  The internal
```

```

hostname for the metadata api endpoint. This will be used by the kube-rbac-proxy
sidecar.
  db_password                <auto-generated>    string    The database
user password. If not specified, the password will be auto-generated.
  storage_class_name        string    The storage c
lass name of the persistent volume used by Postgres database for storing data.
The default value will use the default class name defined on the cluster.
  database_request_storage  10Gi    string    The storage r
equested of the persistent volume used by Postgres database for storing data.
  add_default_rw_service_account  true    string    Adds a read-w
rite service account which can be used to obtain access token to use metadata-s
tore CLI
  log_level                 default  string    Sets the log
level. This can be set to "minimum", "less", "default", "more", "debug" or "tra
ce". "minimum" currently does not output logs. "less" outputs log configuration
options only. "default" and "more" outputs API endpoint access information. "d
ebug" and "trace" outputs extended API endpoint access information(such as body
payload) and other debug information.

```

- (Optional) Modify one of the deployment configurations by creating a configuration YAML with the custom configuration values you want. For example, if your environment does not support `LoadBalancer`, and you want to use `NodePort`, then create a `metadata-store-values.yaml` and configure the `app_service_type` property.

```

---
app_service_type: "NodePort"

```

See [Deployment details and configuration](#) for more information about configuration options.

See [Ingress and multicluster support](#) for more information about ingress and custom domain name support.

- Install the package by running:

```

tanzu package install metadata-store \
  --package-name metadata-store.apps.tanzu.vmware.com \
  --version VERSION \
  --namespace tap-install \
  --values-file metadata-store-values.yaml

```

Where:

- ◆ `--values-file` is an optional flag. Only use it to customize the deployment configuration.
- ◆ `VERSION` is the package version number. For example, `1.0.2`.

For example:

```

$ tanzu package install metadata-store \
  --package-name metadata-store.apps.tanzu.vmware.com \
  --version 1.0.2 \
  --namespace tap-install \
  --values-file metadata-store-values.yaml

- Installing package 'metadata-store.apps.tanzu.vmware.com'
/ Getting namespace 'tap-install'
- Getting package metadata for 'metadata-store.apps.tanzu.vmware.com'

```

```

/ Creating service account 'metadata-store-tap-install-sa'
/ Creating cluster admin role 'metadata-store-tap-install-cluster-role'
/ Creating cluster role binding 'metadata-store-tap-install-cluster-rolebinding
'
/ Creating secret 'metadata-store-tap-install-values'
| Creating package resource
- Package install status: Reconciling

Added installed package 'metadata-store' in namespace 'tap-install'

```

AWS RDS Postgres configuration

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.

Database backup recommendations

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

This topic covers configuring the Supply Chain Security Tools - Store to output detailed log information and interpret them. re-boot

Log levels

There are six log levels that the Supply Chain Security Tools - Store supports.

| Level | Description |
|---------|---|
| Trace | Output extended debugging logs |
| Debug | Output standard debugging log |
| 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 log level, all logs of that level and lower are outputted to the console. For example, setting the log 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 log 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 log level is specified when the Store is installed, the log level is set to `default`.

Error Logs

Error logs are always outputted regardless of the log 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:
...
```

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` log 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` log 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` log level.
4. When the endpoint returns a response, it outputs a `Request response` line. This line is shown at the `default` log 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` log level.

Format

When the Store handles a request, it outputs some API endpoint access information in the following format:

```
I1122 20:30:21.869528      1 images.go:26] MetadataStore "msg"="Processing request" "
endpoint"="/api/images?digest=sha256%3A20521f76ff3d27f436e03dc666cc97a511bbe71e8e8495f
851d0f4bf57b0bab6" "hostname"="metadata-store-app-564f8995c8-r8d6n" "method"="GET"
```

The log is broken down into three sections: The header, name, and key/value pairs.

Log header

`I1122 20:30:21.869528 1 images.go:26]` is the logging header. The [Logging header formats](#) section in GitHub explains each part in more detail.

Name

The string that follows the header is a name that helps identify what produced the log entry. For Stores, the name always starts with `MetadataStore`.

For log entries that display the raw SQL queries, the name is `MetadataStore/gorm`.

Key-value pairs

Key-value pairs compose the rest of the log output. 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 | Log Level | Description |
|----------|---------|-----------|--|
| msg | string | default | A short description of the logged event |
| endpoint | string | default | The API endpoint the Metadata Store attempts to handle the request. This also includes any query and path parameters passed in. |
| hostname | string | default | 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. |
| function | string | debug | The function name that handles the request |
| 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." |

| Key | Type | Log Level | Description |
|-------|--------|-----------|--|
| error | string | all | The error message which is only available in error log entries |

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:

```
I1122 20:30:21.869791      1 images.go:34] MetadataStore "msg"="Request parameters" "
endpoint"="/api/images?digest=sha256%3A20521f76ff3d27f436e03dc666cc97a511bbe71e8e8495f
851d0f4bf57b0bab6" "hostname"="metadata-store-app-564f8995c8-r8d6n" "method"="GET" "di
gest"="sha256:20521f76ff3d27f436e03dc666cc97a511bbe71e8e8495f851d0f4bf57b0bab6" "id"=0
```

These key/value pairs show up in all subsequent log lines of the same call. For example:

```
I1122 20:30:21.878749      1 images.go:56] MetadataStore "msg"="Request response" "di
gest"="sha256:20521f76ff3d27f436e03dc666cc97a511bbe71e8e8495f851d0f4bf57b0bab6" "endpo
int"="/api/images?digest=sha256%3A20521f76ff3d27f436e03dc666cc97a511bbe71e8e8495f851d0
f4bf57b0bab6" "hostname"="metadata-store-app-564f8995c8-r8d6n" "id"=0 "method"="GET" "
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 log level to `debug`, the Store logs the body payload data for both the request and response of an API call.

The `debug` log 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 log 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:

```
I0111 20:14:30.816833      1 connection.go:40] MetadataStore/gorm "msg"="Sql Call" "hostname"="metadata-store-app-56799fc4f9-phlv7" "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

This topic contains troubleshooting and known issues for Supply Chain Security Tools - Store.

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. This is caused by the persistent volume used by postgres retaining old data, even though the retention policy is set to `DELETE`.

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 steps below 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-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"}}}'
```

Multicluster Support: Error sending results to SCST - Store running in a different cluster

Symptom

The [Store Ingress and multicluster support](#) document instructs you on how to create `SecretExports` to share secrets for communicating with the Store. During installation, Supply Chain Security Tools - Scan (Scan) creates the `SecretImport` for ingesting the TLS CA certificate secret, but misses the `SecretImport` for the RBAC Auth token.

Solution

Follow the AWS documentation to install the [Amazon EBS CSI Driver](#) before installing Store or before upgrading to Kubernetes v1.23.

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
```

Troubleshooting upgrading

This topic describes upgrading issues and resolutions.

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 v1.1 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.pgdg18.04+1) on x86_64-pc-linux-gnu, compiled by gcc (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0, 64-bit
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 signal 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

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](#).

Ingress and multicluster support

Supply Chain Security Tools - Store has ingress support by using Contour's HTTPProxy resources. To enable ingress support, a Contour installation must be available in the cluster.

Supply Chain Security Tools - Store's configuration includes two options to configure the proxy: `ingress_enabled` and `ingress_domain`.

For example:

```
ingress_enabled: "true"
ingress_domain: "example.com"
```

Supply Chain Security Tools - Store installation creates an HTTPProxy entry with host routing by using the qualified name `metadata-store.<ingress_domain>` (`metadata-store.example.com`). The create route supports HTTPS communication through a self-signed certificate with the same subject Alternative Name.

Contour and DNS setup are not part of Supply Chain Security Tools - Store installation. Access to Supply Chain Security Tools - Store through Contour depends on the correct configuration of these two components.

Make the proper DNS record available to clients to resolve `metadata-store.<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 insecure (`-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.

Multicluster setup

To support multicluster setup of Supply Chain Security Tools - Store, some communication secrets must be shared across the cluster.

Set up the cluster containing Supply Chain Security Tools - Store first and enable Supply Chain Security Tools - Store ingress for ease of installation. When configuring a second Tanzu Application Platform cluster, components such as Supply Chain Security Tools - Scan need access to the Store's API. This requires access to the TLS CA certificate for HTTPS support and the Authorization access token.

TLS CA certificate

To get Supply Chain Security Tools - Store's TLS CA certificate, run:

```

# On the Supply Chain Security Tools - Store's cluster
$ CA_CERT=$(kubectl get secret -n metadata-store ingress-cert -o json | jq -r ".data.\
"ca.crt\"")
$ cat <<EOF > store_ca.yaml
---
apiVersion: v1
kind: Secret
type: kubernetes.io/tls
metadata:
  name: store-ca-cert
  namespace: metadata-store-secrets
data:
  ca.crt: $CA_CERT
  tls.crt: ""
  tls.key: ""
EOF

# On the second Cluster

# Create secrets namespace
$ kubectl create ns metadata-store-secrets

# Create the CA Certificate secret
$ kubectl apply -f store_ca.yaml

```

RBAC Auth token

To get the Supply Chain Security Tools - Store's Auth token, run:

```
$ AUTH_TOKEN=$(kubectl get secrets -n metadata-store -o jsonpath="{.items[?(@.metadata.annotations['kubernetes\.io/service-account\.name']=='metadata-store-read-write-client')].data.token}" | base64 -d)
```

Create the corresponding secret on the second cluster. Run:

```
$ kubectl create secret generic store-auth-token --from-literal=auth_token=$AUTH_TOKEN -n metadata-store-secrets
```

This secret is created in the metadata-store-secrets namespace to be imported by the Supply Chain Security Tools - Scan.

Supply Chain Security Tools - Scan installation

To allow Supply Chain Security Tools - Scan to access the created secrets, `SecretExport` resources must be created.

Note: Corresponding `SecretImport` resources that receive the exported secrets are installed with the Supply Chain Security Tools - Scan package.

Here is an example for supporting Supply Chain Security Tools - Scan installation on the default namespace `scan-link-system`:

```
$ cat <<EOF > store_secrets_export.yaml
---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretExport
metadata:
  name: store-ca-cert
  namespace: metadata-store-secrets
spec:
  toNamespace: scan-link-system
---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretExport
metadata:
  name: store-auth-token
  namespace: metadata-store-secrets
spec:
  toNamespace: scan-link-system
EOF

# Export secrets to the Supply Chain Security Tools - Scan namespace
$ kubectl apply -f store_secrets_export.yaml
```

Install Supply Chain Security Tools - Scan with the following configuration:

```
---
scanning:
  metadataStore:
    url: https://metadata-store.example.com
    caSecret:
      name: store-ca-cert
```

```
importFromNamespace: metadata-store-secrets
authSecret:
  name: store-auth-token
  importFromNamespace: metadata-store-secrets
```

Overview of VMware Tanzu Developer Tools for Visual Studio Code

Tanzu Developer Tools for Visual Studio Code (VS Code) is the official VMware Tanzu IDE extension for VS Code. It helps you develop with the Tanzu Application Platform.

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.

Installing Tanzu Developer Tools for Visual Studio Code

This topic explains 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](#)

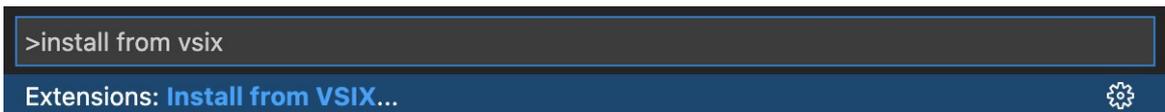
If you are an app developer, someone else in your organization might have already set up the Tanzu Application Platform environment.

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

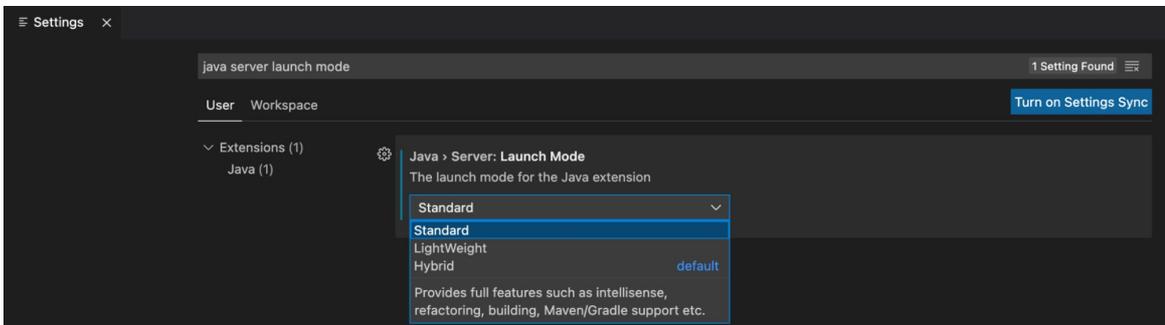
Install

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`.



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**.



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 information, 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.
 - ◆ **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 Visual Studio 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](#).

Ensure you have completed the [Installation](#) before continuing on to the following sections.

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.

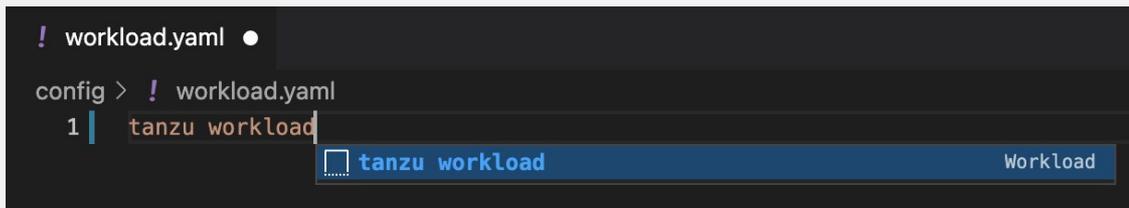
Before beginning to write your `workload.yaml` file, ensure that you know:

- 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.



4. Fill in the template by pressing the Tab key.

Manual

To create your `workload.yaml` file manually, follow this example:

```
apiVersion: carto.run/v1alpha1
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.
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.

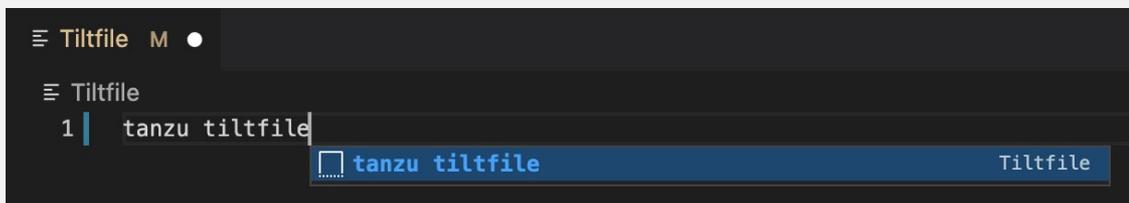
Before beginning to write your Tiltfile file, ensure that you know:

- 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.



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 " + NAMESPACE + " --yes" ,
    deps=['pom.xml', './target/classes'],
    container_selector='workload',
    live_update=[
        sync('./target/classes', '/workspace/BOOT-INF/classes')
    ]
)

k8s_resource('APP-NAME', port_forwards=["8080:8080"],
    extra_pod_selectors=[{'carto.run/workload-name': 'APP-NAME', 'app.kubernetes.io/component': 'run'}])
allow_k8s_contexts('CONTEXT-NAME')

```

Where:

- `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](#).

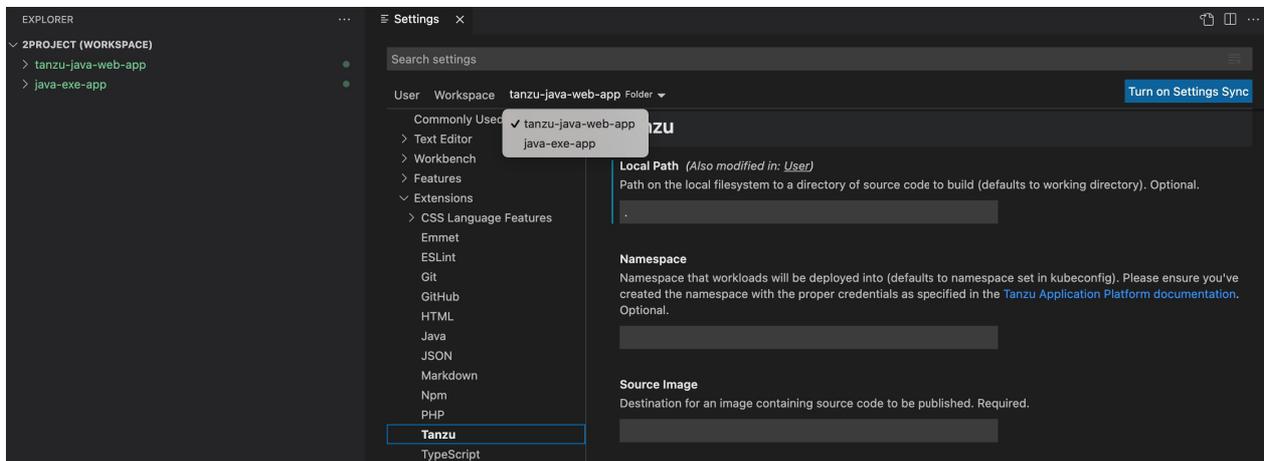
Using Tanzu Developer Tools for Visual Studio Code

Ensure the project that you want to use the extension with has the required files specified in [Getting started with Tanzu Developer Tools for Visual Studio 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 Tanzu Dev Tools Extension settings on a per-project basis by using the dropdown selector in the [Settings](#) page.



Debugging on the cluster

The extension enables you to debug your application on your Kubernetes cluster that has Tanzu Application Platform.

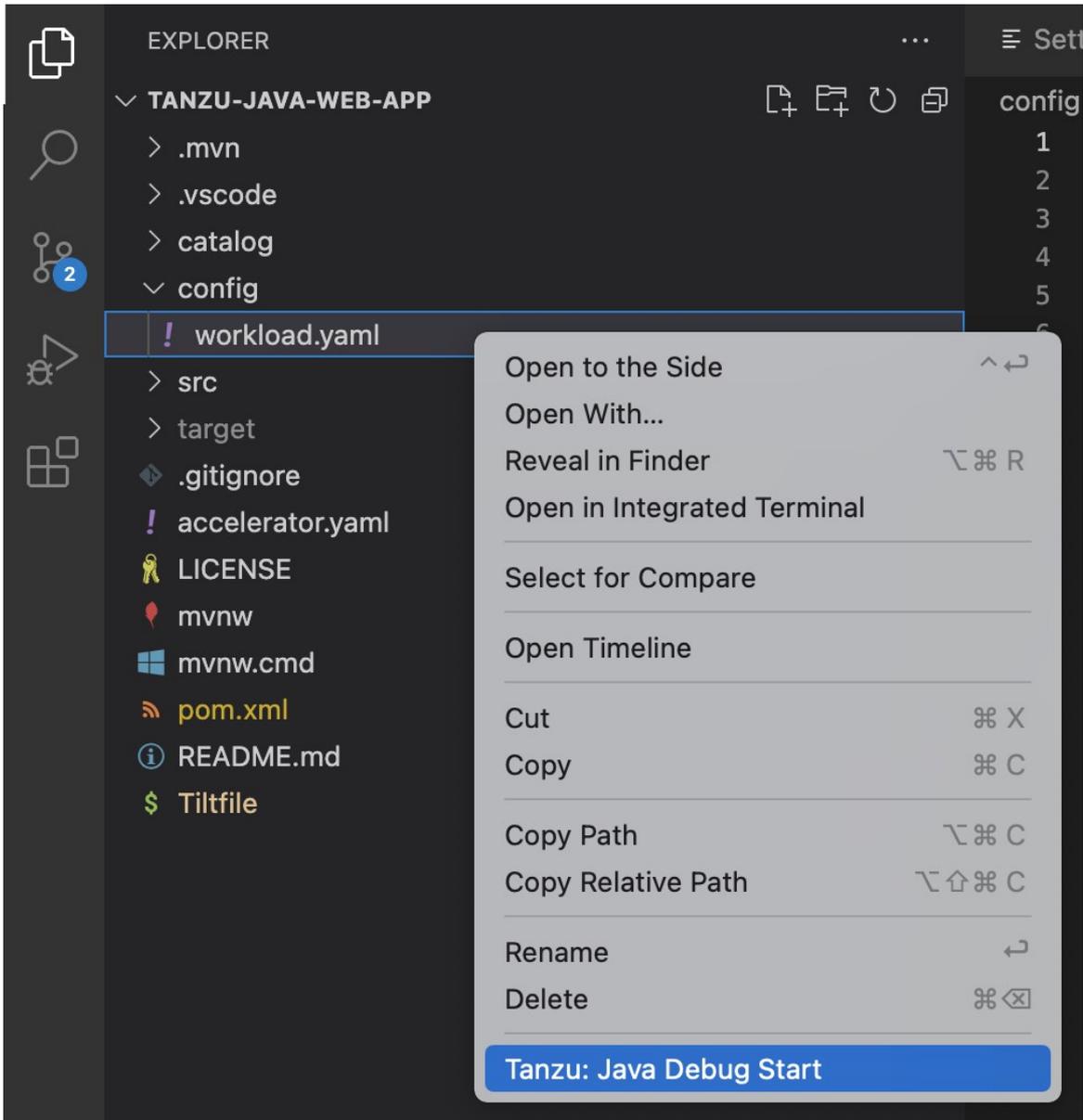
Debugging requires a `workload.yaml` file in your project. For information about creating a `workload.yaml` file, see [Set up Tanzu Developer Tools](#).

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

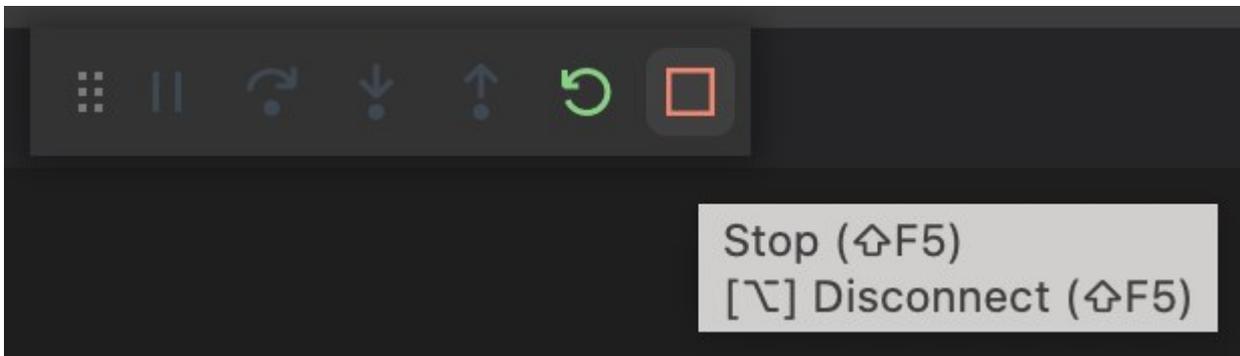
To start debugging on the cluster:

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.

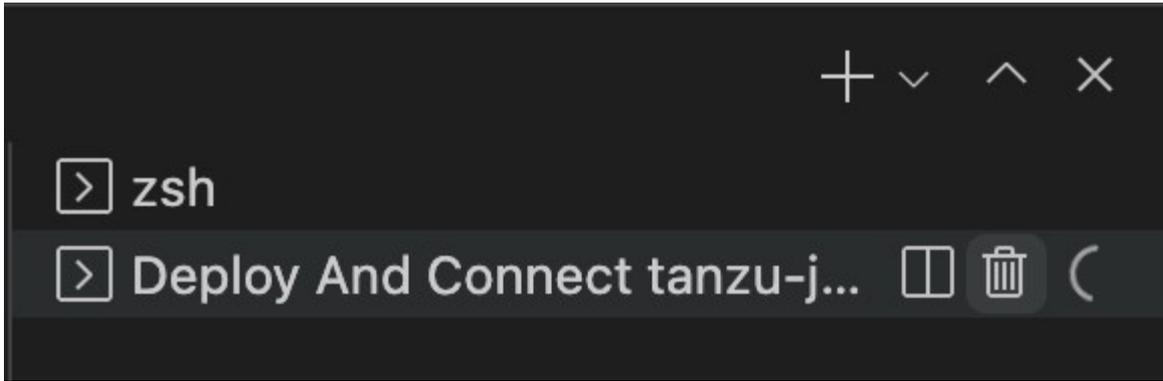


Stop Debugging on the cluster

To stop debugging on the cluster, you can click the stop button in the Debug overlay.



Alternatively, you can press $\text{⌘}+\text{J}$ (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.

```
! [The VS Code interface showing the Explorer tab with the Tiltfile file right-click menu open and the Tanzu: Live Update Start option highlighted.] (../images/vscode-startliveupdate1.png)
```

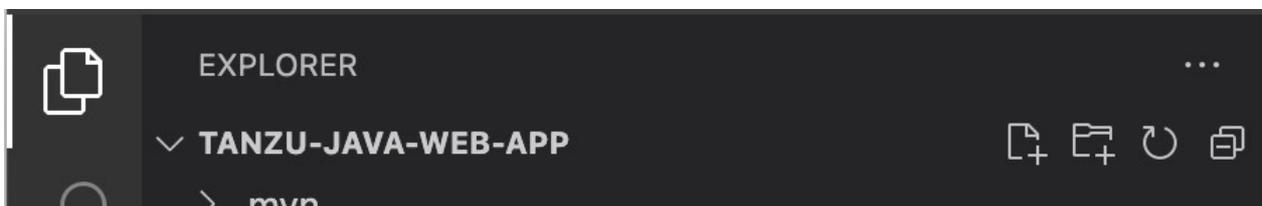
Alternatively, you can press `⇧⌘P` to open the Command Palette and run the **Tanzu: Live Update Start** command.

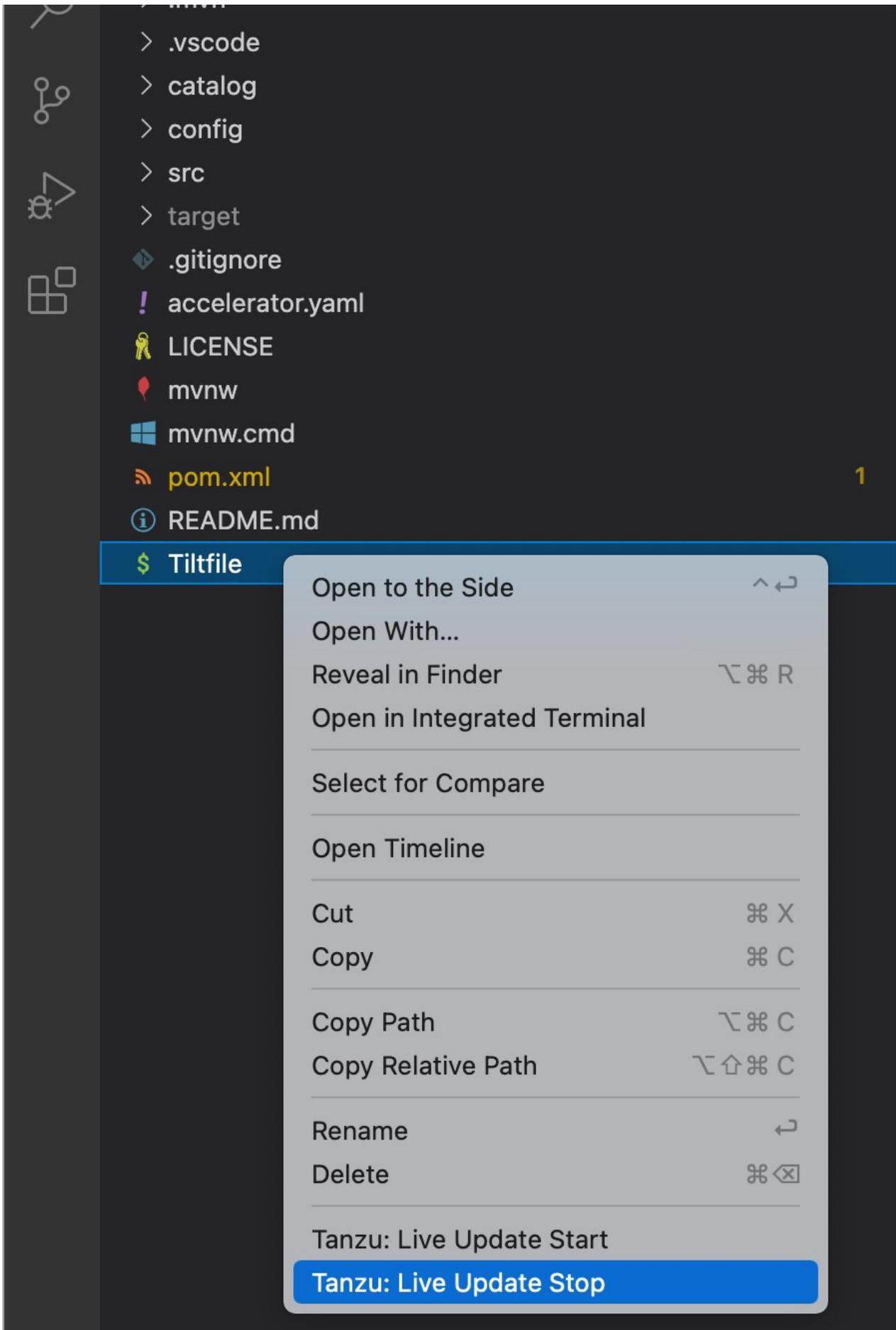
```
! [Command palette open showing text Tanzu: Live Update Start.] (../images/vscode-startliveupdate2.png)
```

Stop Live Update

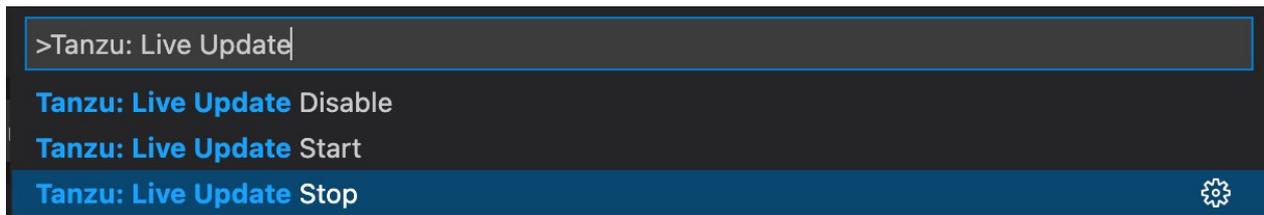
When Live Update stops, your application continues to run on the cluster, but the changes you made and saved in your editor are not present in your running application unless you redeploy your application to the cluster.

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`.

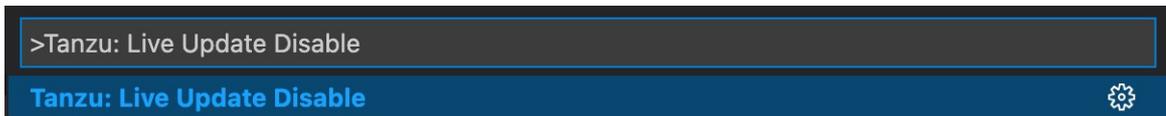


Deactivate Live Update

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`.



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

The Live Update status bar entry can be hidden by right-clicking on it and selecting **Hide 'Tanzu Developer Tools (Extension)'**.

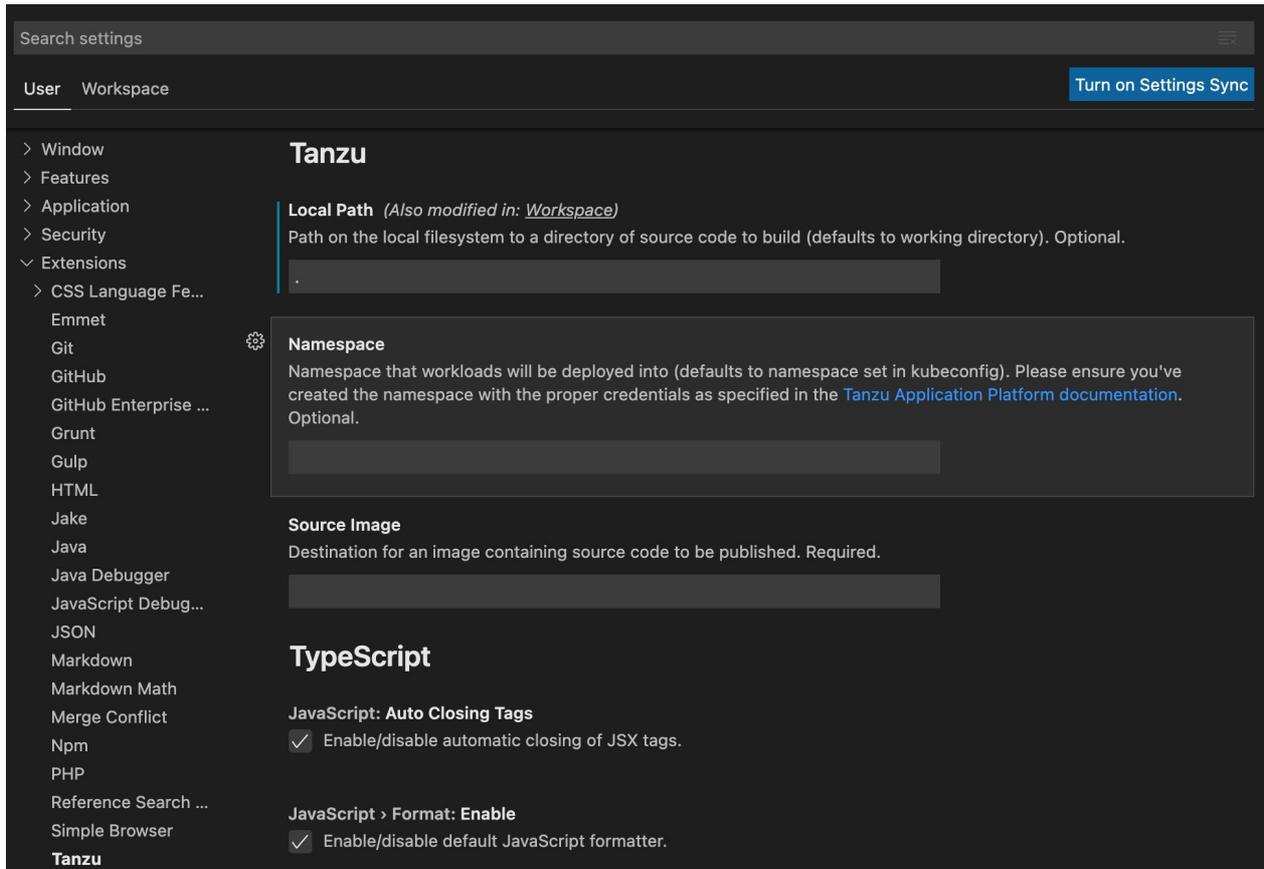


Switch Namespace

To switch the namespace where you created the workload:

1. Navigate to settings (**Code > Preferences > Settings**).
2. Expand the **Extensions** section of the Settings and select **Tanzu**.

- In the Namespace option, add the namespace you want to deploy to. This defaults to the default namespace.



Pinniped compatibility

This topic covers 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.

Tanzu API portal

API portal for VMware Tanzu enables API consumers to find APIs they can use in their own applications. Consumers can view detailed API documentation and try out an API to see if it can meet 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 in a single instance.

For more information about Tanzu API portal, see [API portal for VMware Tanzu](#).

Install Tanzu API portal

This document describes how to install Tanzu API portal from the Tanzu Application Platform package repository.



Note

Follow the steps in this topic if you do not want to use a profile to install API portal. For more information about profiles, see [About Tanzu Application Platform components and 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.com -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'
```

Tanzu Application Platform GUI

See the following topics for information about Tanzu Application Platform GUI.

- [Overview of Tanzu Application Platform GUI](#)
- [Installing Tanzu Application Platform GUI](#)
- [Accessing Tanzu Application Platform GUI](#)
- [Catalog operations](#)
- [Viewing resources on multiple clusters](#)
- [Authentication](#)
- [Adding integrations](#)
- [Database configuration](#)
- [TechDocs](#)
- [Tanzu Application Platform GUI plug-ins](#)
- [Upgrading Tanzu Application Platform GUI](#)
- [Troubleshoot Tanzu Application Platform GUI](#)

Overview of Tanzu Application Platform GUI

Tanzu Application Platform 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.

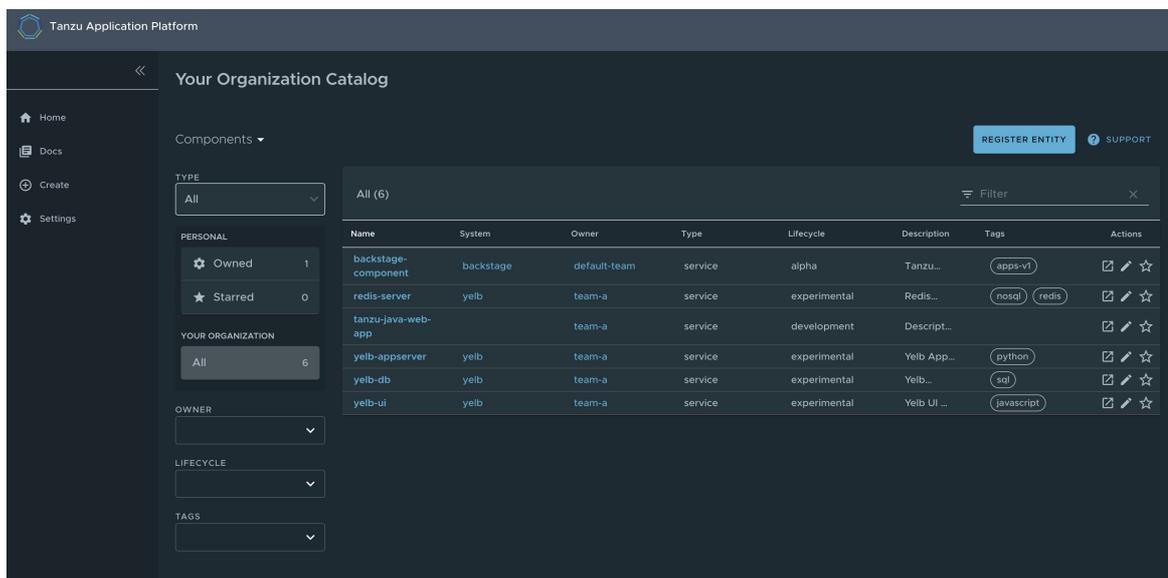
Tanzu Application Platform GUI is built from the [Cloud Native Computing Foundation's](#) project [Backstage](#).

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.



- **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 describes how to install Tanzu Application Platform 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 [About Tanzu Application Platform components and 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-schema 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 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 desired version. For example, `1.0.1`.

For example:

```

$ tanzu package install tap-gui --package-name tap-gui.tanzu.vmware.com --version 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`.

- To access Tanzu Application Platform GUI, use the service you exposed in the `service_type` field in the values file.

Accessing Tanzu Application Platform GUI

Use one of the following methods to access Tanzu Application Platform GUI:

- Access with the LoadBalancer method (default)
- Access with the shared Ingress method

Access with the LoadBalancer method (default)

- 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
```

- Obtain the external IP address of your LoadBalancer by running:

```
kubectl get svc -n tap-gui
```

- 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 GUI uses the shared `tanzu-system-ingress` instance of Contour that is installed as part of the Profile installation.

- 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/TCP
  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.

- Specify parameters in `tap-values.yaml` related to Ingress. For example:

```
tap_gui:
  service_type: ClusterIP
  ingressEnabled: "true"
  ingressDomain: 'example.com' # This makes the host name tap-gui.example.com
```

- Update your other host names in the `tap_gui` section of your `tap-values.yaml` with the new host name. For example:

```
tap_gui:
  service_type: ClusterIP
  ingressEnabled: "true"
  ingressDomain: 'example.com' # This makes the host name tap-gui.example.com
# 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`.

- Update your package installation with your changed `tap-values.yaml` file by running:

```
tanzu package installed update tap --package-name tap.tanzu.vmware.com --version
VERSION-NUMBER \
--values-file tap-values.yaml -n tap-install
```

Where `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.1.0`.

- 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](#).

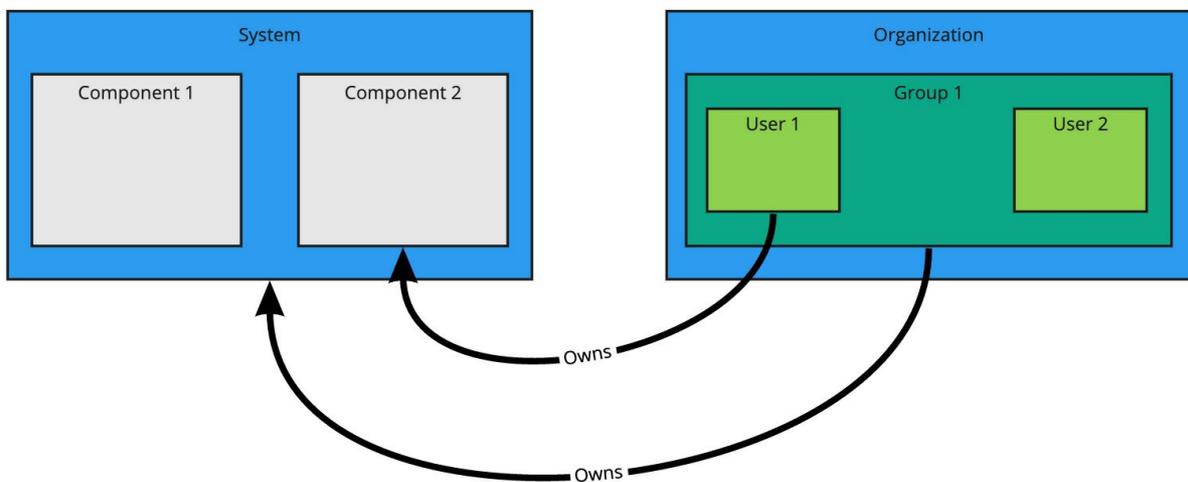
Adding catalog entities

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?background=%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 Kubernetes 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:

```

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:

```
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 to retrieve Kubernetes object details from multiple clusters and then surface those details in the Runtime Resources Visibility plug-in.

Set up a Service Account to view resources on a cluster

To view resources on a cluster, you must create a service account on the cluster that can `get`, `watch`, and `list` resources on that cluster. You first create a `ClusterRole` with these rules and a `ServiceAccount` in its own `Namespace`, and then bind the `ClusterRole` to the `ServiceAccount`.

To do so:

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', '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: ['serverlesservices']
  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
  verbs: ['get', 'watch', 'list']
- apiGroups: ['conventions.apps.tanzu.vmware.com']
  resources:
  - podintents
  verbs: ['get', 'watch', 'list']
- apiGroups: ['kpack.io']
  resources:
  - images
  - builds
  verbs: ['get', 'watch', 'list']
- apiGroups: ['scanning.apps.tanzu.vmware.com']
  resources:
  - sourcescans
  - imagescans
  - scanpolicies
  verbs: ['get', 'watch', 'list']
- apiGroups: ['tekton.dev']
  resources:

```

```

- taskruns
- pipelineruns
verbs: ['get', 'watch', 'list']
- apiGroups: ['kappctrl.k14s.io']
resources:
- apps
verbs: ['get', 'watch', 'list']

```

This YAML content creates [Namespace](#), [ServiceAccount](#), [ClusterRole](#), and [ClusterRoleBinding](#).

2. Create [Namespace](#), [ServiceAccount](#), [ClusterRole](#), and [ClusterRoleBinding](#) by running:

```
kubectl create -f tap-gui-viewer-service-account-rbac.yaml
```

This ensures the `kubeconfig` context is set to the cluster with resources to be viewed in Tanzu Application Platform GUI.

3. Discover the `CLUSTER_URL` and `CLUSTER_TOKEN` values by running:

```

CLUSTER_URL=$(kubectl config view --minify -o jsonpath='{.clusters[0].cluster.server}')

CLUSTER_TOKEN=$(kubectl -n tap-gui get secret $(kubectl -n tap-gui get sa tap-gui-viewer -o=json \
| jq -r '.secrets[0].name') -o=json \
| jq -r '.data["token"]' \
| base64 --decode)

echo CLUSTER_URL: $CLUSTER_URL
echo CLUSTER_TOKEN: $CLUSTER_TOKEN

```

4. Record the `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 cluster 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` file that Tanzu Application Platform GUI uses. This section must have an entry for each cluster that has resources to view.

To do so:

1. Copy this YAML content into `tap-gui-values.yaml`:

```

app_config:
  kubernetes:
    serviceLocatorMethod:
      type: 'multiTenant'
    clusterLocatorMethods:
      - type: 'config'
      clusters:
        - url: CLUSTER-URL

```

```
name: CLUSTER-NAME
authProvider: serviceAccount
serviceAccountToken: "CLUSTER-TOKEN"
skipTLSVerify: 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 cluster that hosts Tanzu Application Platform GUI, add an entry to `clusters` for it as well.

2. Update the `tap-gui` package by running this command:

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

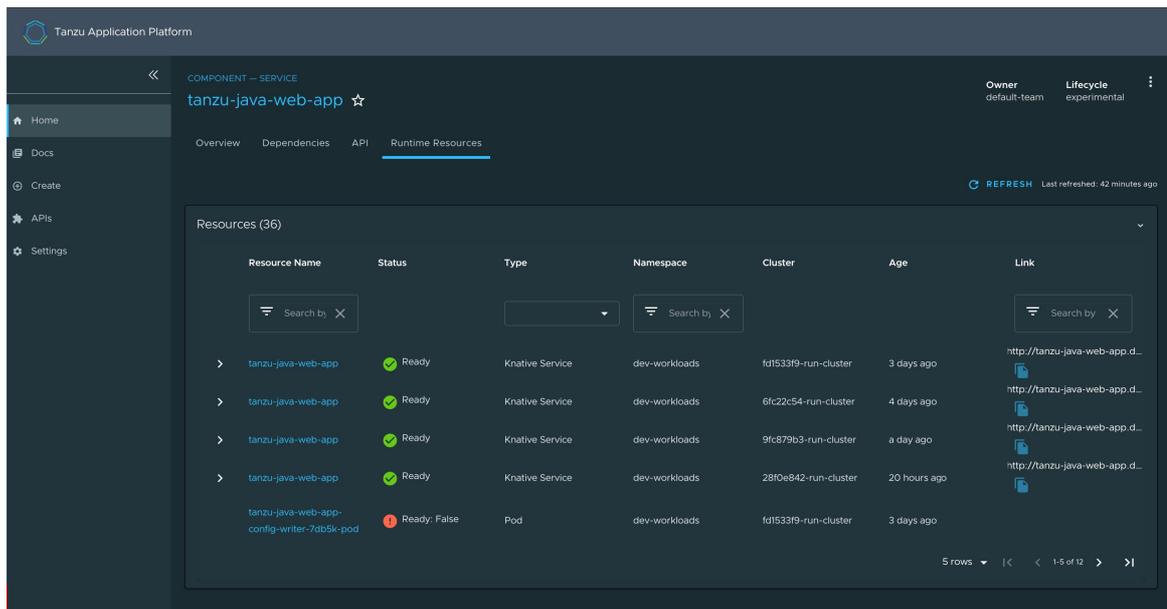
3. Wait a moment for the `tap-gui` package to update and then verify that `STATUS` is `Reconcile succeeded` by running:

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

View resources on 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.



Setting up a Tanzu Application Platform GUI authentication provider

Tanzu Application Platform GUI extends the current Backstage's 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:

```
tap_gui:
  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
#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 # default='RS256'
    scope: AUTH-OIDC-SCOPE # default='openid profile email'
    prompt: auto # default=none (allowed values: auto, none, consent, login)

```

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 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) 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.

Support menu customization

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:

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/index.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:

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.

Adding Tanzu Application Platform GUI integrations

You can integrate Tanzu Application Platform 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 `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 VERSION-NUMBER --values-file tap-values.yaml -n tap-install
```

Where `VERSION-NUMBER` is the Tanzu Application Platform version. For example, `1.1.0`.

For example:

```
$ tanzu package installed update tap --package-name tap.tanzu.vmware.com --version 1.0.0 --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 catalog allows for 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 --version VERSION-NUMBER --values-file tap-values.yaml -n tap-install
```

Where `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.1.0`.

For example:

```
$ tanzu package installed update tap --package-name tap.tanzu.vmware.com --version 1.1.0 --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'
```

TechDocs

This guide explains how to generate and publish TechDocs for catalogs. For more information, 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 `npm` 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
```

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.

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.

Tanzu Application Platform GUI plug-ins

Overview

Tanzu Application Platform GUI has many pre-integrated plug-ins. You do not need to configure the plug-ins. To use the plug-in, you must install the Tanzu Application Platform component.

Tanzu Application Platform includes the following GUI plug-ins:

- [Runtime Resources Visibility](#)
- [Application Live View](#)

- [Application Accelerator](#)
- [API Documentation](#)
- [Supply Chain Choreographer](#)

Runtime resources visibility

This topic describes runtime resources visibility.

Introduction

Runtime Resources Visibility plug-in part of Tanzu Application Platform GUI allows users to visualize their Kubernetes resources associated with their Workloads.

Prerequisite

In order to access the Runtime Resources Visibility plug-in, you must first have successfully installed [Tanzu Application Platform](#), which includes Tanzu Application Platform GUI.

Visualize Workloads on Tanzu Application Platform GUI

In order to view your applications on Tanzu Application Platform GUI, use the following steps:

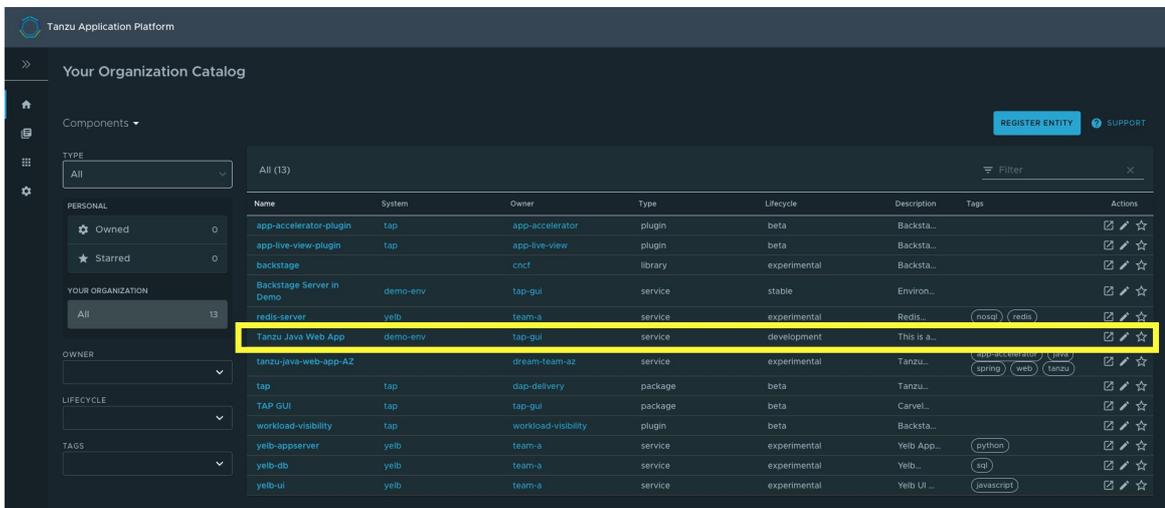
1. [Develop your application on the Tanzu Application Platform via Application Accelerators](#)
2. [Add your application to Tanzu Application Platform GUI Software Catalog](#)

Navigate to the Runtime Resources Visibility screen

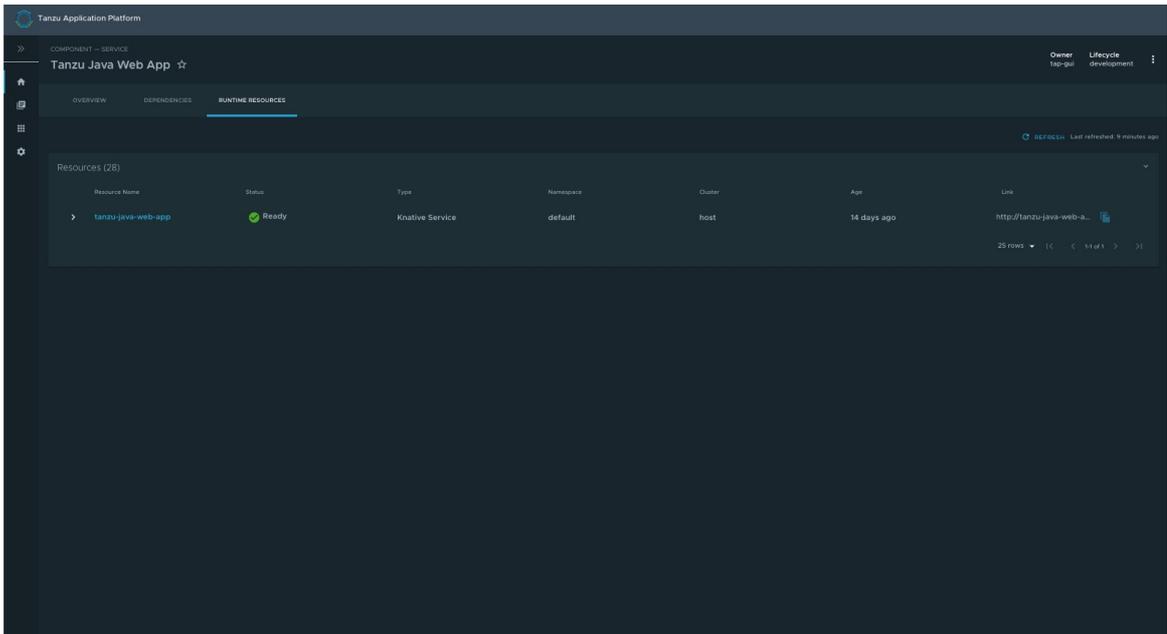
You can view the list of running resources and the details of their status, type, namespace, cluster, and public URL if applicable for the resource type.

To view the list of your running resources:

1. Select your component from the Catalog index page.



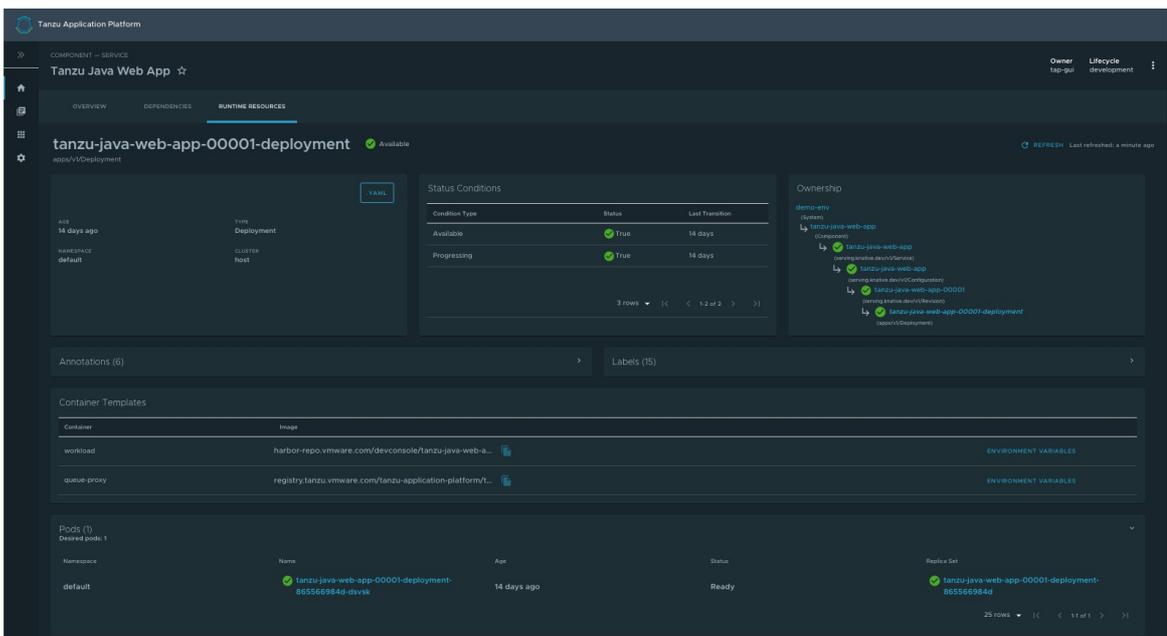
2. Select the **Runtime Resources** tab.



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



View details for a specific resource

The Resources index table shows Knative Services, Deployments, pods, ReplicaSets and Kubernetes Services that match the label indicated in the component's definition.

You can see a hierarchical structure showing the owner-dependent relationship between the objects. Resources without an owner are listed in the table as independent elements.

For information about owners and dependents, see the [Kubernetes documentation](#).

See the following example of an expanded index table showing one of the owner resources and its dependents.

| Resource Name | Status | Type | Namespace | Cluster | Age | Link |
|---|---------------|-----------------|-----------|---------|-------------|---|
| tanzu-java-web-app | Ready | Knative Service | default | host | 14 days ago | http://tanzu-java-web-a... |
| tanzu-java-web-app-00001-deployment | Available | Deployment | default | host | 14 days ago | |
| tanzu-java-web-app-00001-deployment-955569944 | ReplicasReady | ReplicaSet | default | host | 14 days ago | |
| tanzu-java-web-app-00001-deployment-955569944-pod | Ready | Kubernetes Pod | default | host | 14 days ago | |
| tanzu-java-web-app-00002-deployment | Available | Deployment | default | host | 13 days ago | |
| tanzu-java-web-app-00003-deployment | Available | Deployment | default | host | 10 days ago | |
| tanzu-java-web-app-00004-deployment | Available | Deployment | default | host | 10 days ago | |
| tanzu-java-web-app-00005-deployment | Available | Deployment | default | host | 9 days ago | |
| tanzu-java-web-app-00006-deployment | Available | Deployment | default | host | 9 days ago | |
| tanzu-java-web-app-00007-deployment | Available | Deployment | default | host | 8 days ago | |
| tanzu-java-web-app-00008-deployment | Available | Deployment | default | host | 8 days ago | |
| tanzu-java-web-app-00009-deployment | Available | Deployment | default | host | 3 days ago | |

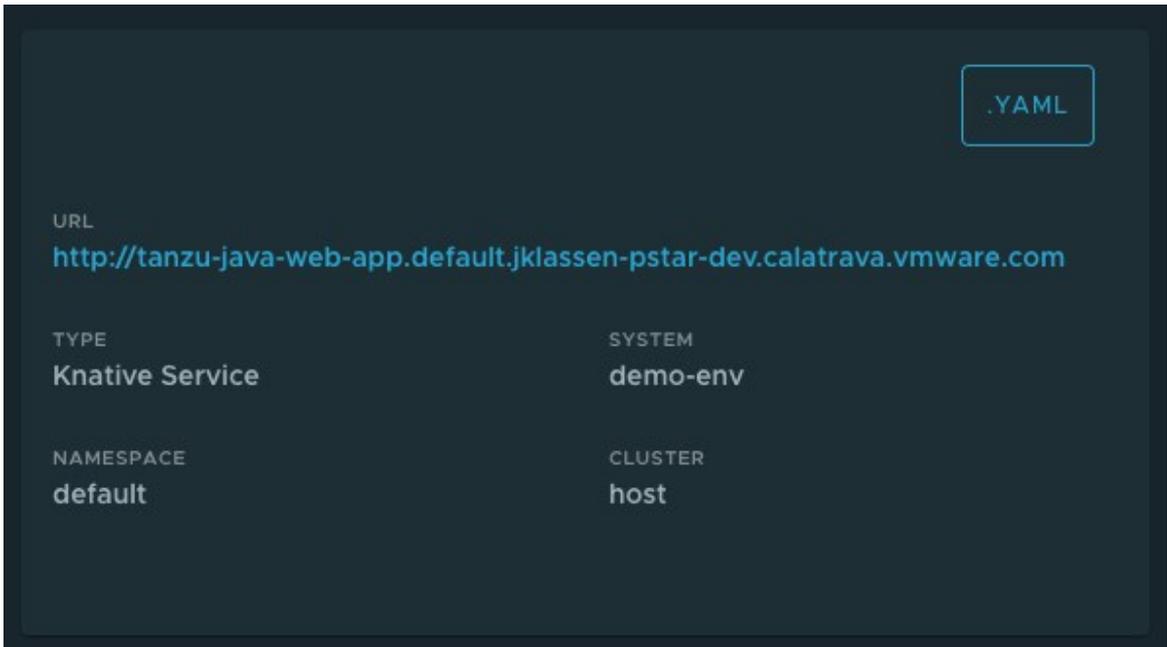
Detail pages

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:

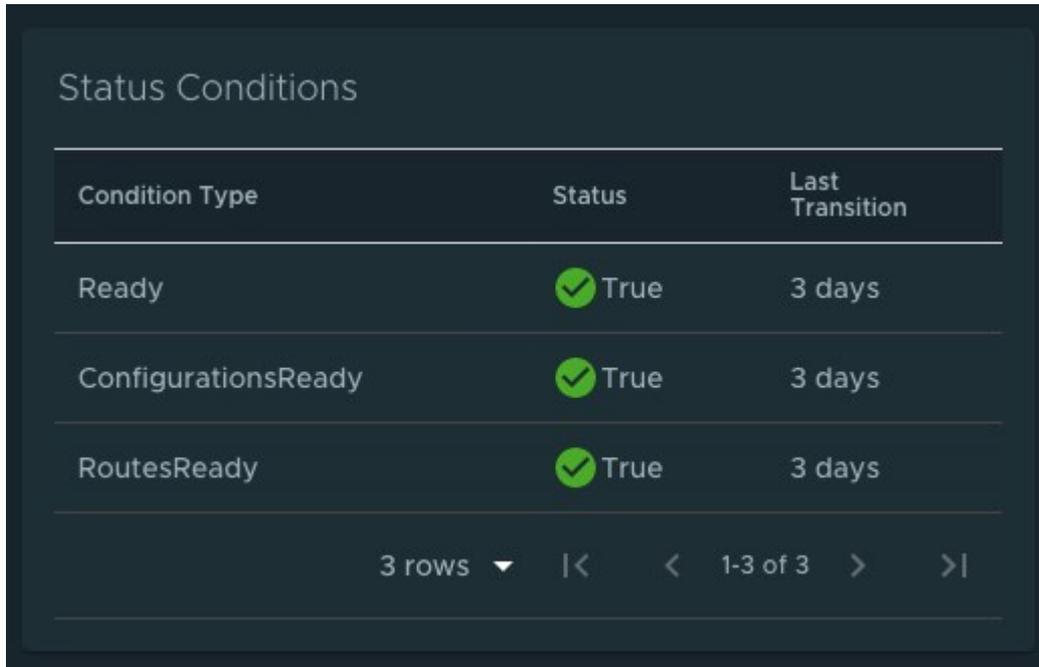
- .YAML button
- URL (URL is available for Knative services and Kubernetes services)
- Type
- System
- Namespace
- Cluster



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, see [Concepts - Object Spec and Status](#) in the Kubernetes documentation.



Ownership card

Depending on the resource that you are viewing, the ownership section presents all the resources specified in the `metadata.ownerReferences`. You can use this section to navigate between resources.

See [Owners and Dependents](#) in the Kubernetes documentation.



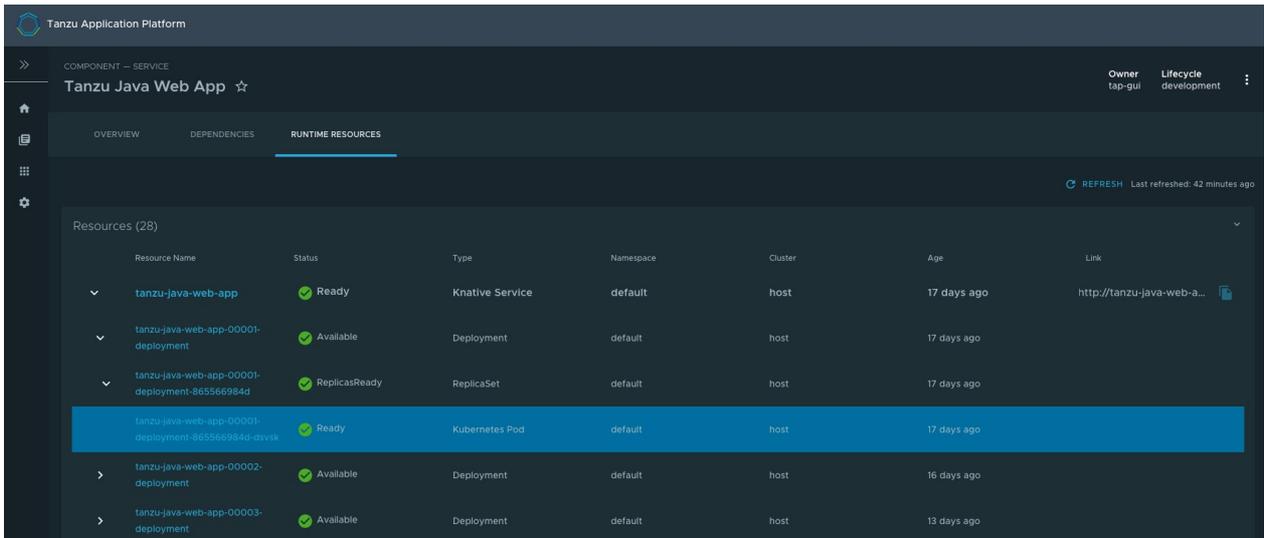
Annotations and Labels

The Annotations and Labels card show information about `metadata.annotations` and `metadata.labels`.

| Annotations (5) | Labels (5) |
|----------------------------------|-----------------------------|
| kapp.k14s.io/identity | app.kubernetes.io/component |
| kapp.k14s.io/original | app.kubernetes.io/part-of |
| kapp.k14s.io/original-diff-md5 | carto.run/workload-name |
| serving.knative.dev/creator | kapp.k14s.io/app |
| serving.knative.dev/lastModifier | kapp.k14s.io/association |

Navigating to Pod Details Page

You can navigate directly to the Pod Details page from the Resources index table.



Alternatively, you can see the pod table in each resource details page as shown in the following screenshot.



Navigating to Application Live View

To view additional information about your running applications, see the [Application Live View](#) section in the Pod Details page.

The screenshot displays the 'Tanzu Application Platform' interface for a 'Tanzu Java Web App'. The main view is 'Application Live View', which provides detailed information about the application's runtime resources. The interface is divided into several sections:

- Overview:** Shows the application name 'tanzu-java-web-app-00002-deployment-745c7bb8d5-z2qm7' with a 'Ready' status and a 'v1Pod' label.
- Runtime Resources:** A table showing the status of various conditions:

| Condition Type | Status | Last Transition |
|-----------------|--------|-----------------|
| Ready | True | 17 days |
| ContainersReady | True | 17 days |
| Initialized | True | 17 days |
- Ownership:** A tree view showing the hierarchy of resources, including 'demo-env', 'tanzu-java-web-app', and 'tanzu-java-web-app-00002'.
- Annotations (8) and Labels (18):** Links to view additional metadata.
- Live View:** A detailed view of the application's configuration and health:

| Information Category | Details |
|------------------------|---|
| Application Name | demo |
| Instance ID | a4648f0e-28d2-4a4c-89e1-e47cf2b2b07 |
| Location | http://192.168.46.238:8080/ |
| Actuator Location | http://backstage-akassen-pstar-dev.cabrava.vmware.com/api/proxy/app-live-view/instance/a4648f0e-28d2-4a4c-89e1-e47cf2b2b07/actuator |
| Health Endpoint | http://192.168.46.238:8080/actuator/health |
| Direct Actuator Access | http://192.168.46.238:8080/actuator |
| Framework | Spring Boot |
| Version | 2.5.4 |
| New Patch Version | 2.5.7 |
| New Minor Version | 2.6.1 |
| New Major Version | |
| Build Version | 0.01-SNAPSHOT |
- Containers:** A table showing the status of containers:

| Container | Status | Restarts | Image |
|-------------|--------|----------|--|
| queue-proxy | ✓ | 0 | sha256:ec56f53d50a4be4071c0445051c2c5ce3b... |
| workload | ✓ | 0 | sha256:8a543cfff2a1bceba2b82a4565cbd9fec79a... |

Application Live View in Tanzu Application Platform GUI

This topic describes Application Live View in Tanzu Application Platform GUI.

Overview

The Application Live View features of the 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 Spring Boot 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. Application Live View 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:

- Select the relevant component under the **Organization Catalog** in Tanzu Application Platform GUI
- Select the desired service under **Runtime Resources** tab
- Select the desired pod from the **Pods** section under **Runtime Resources** tab
- The user can see all the details, do some lightweight troubleshooting and interact with the application in certain boundaries under the **Live View** section

Application Live View pages

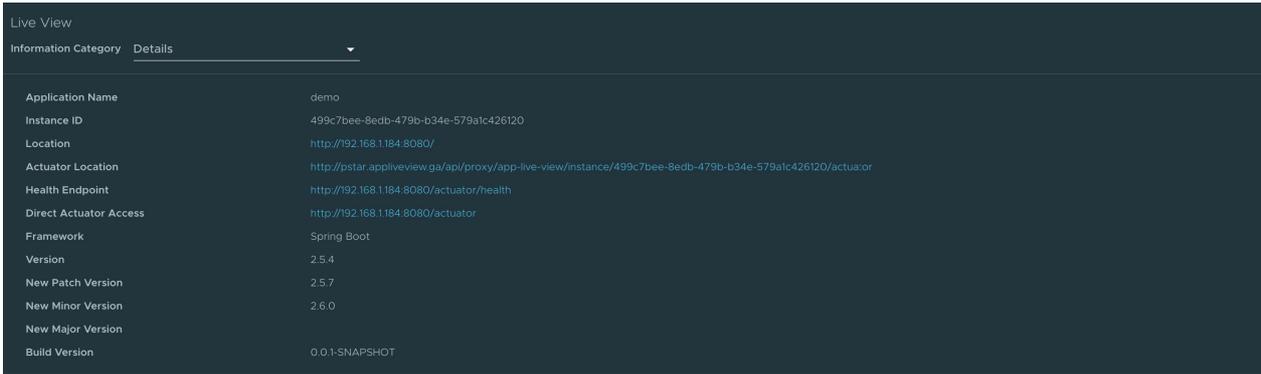
The following sections describe Application Live View pages.

Details page

This is the default page loaded in the **Live View** section. This page gives a tabular overview containing the following information:

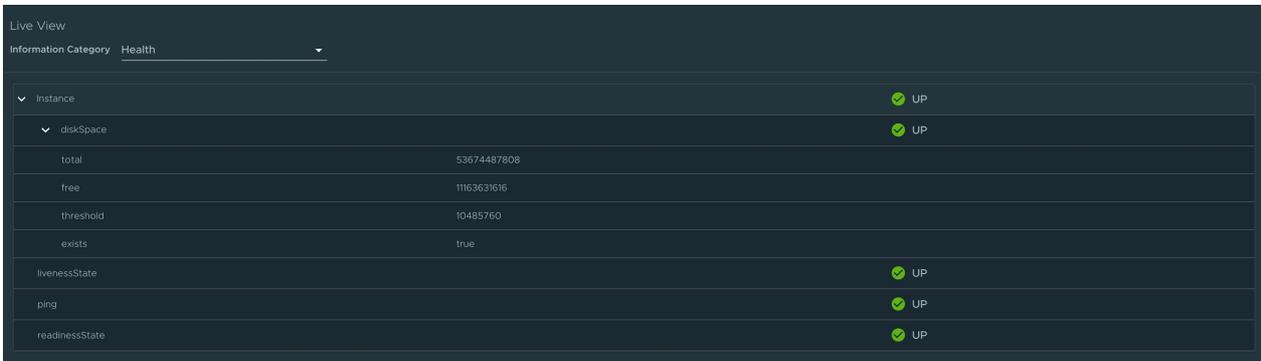
- application name
- instance ID
- location
- actuator location
- health endpoint
- direct actuator access
- framework
- version
- new patch version
- new major version
- build version

The user can navigate between **Information Categories** by selecting from the drop-down menu on the top right corner of the page.



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.



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.

Live View
Information Category Environment

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

REFRESH SCOPE Search by Property or Value...

Applied Overrides

Property Name Value ADD CLEAR

server.ports

| | |
|-------------------|------|
| local.server.port | 8080 |
|-------------------|------|

servletContextInitParams

No Properties Set

systemProperties

| | | | |
|---|----------------------|----------|--------|
| management.endpoints.web.exposure.include | * | | |
| awt.toolkit | sun.awt.X11.XToolkit | Override | Cancel |
| 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 | | |

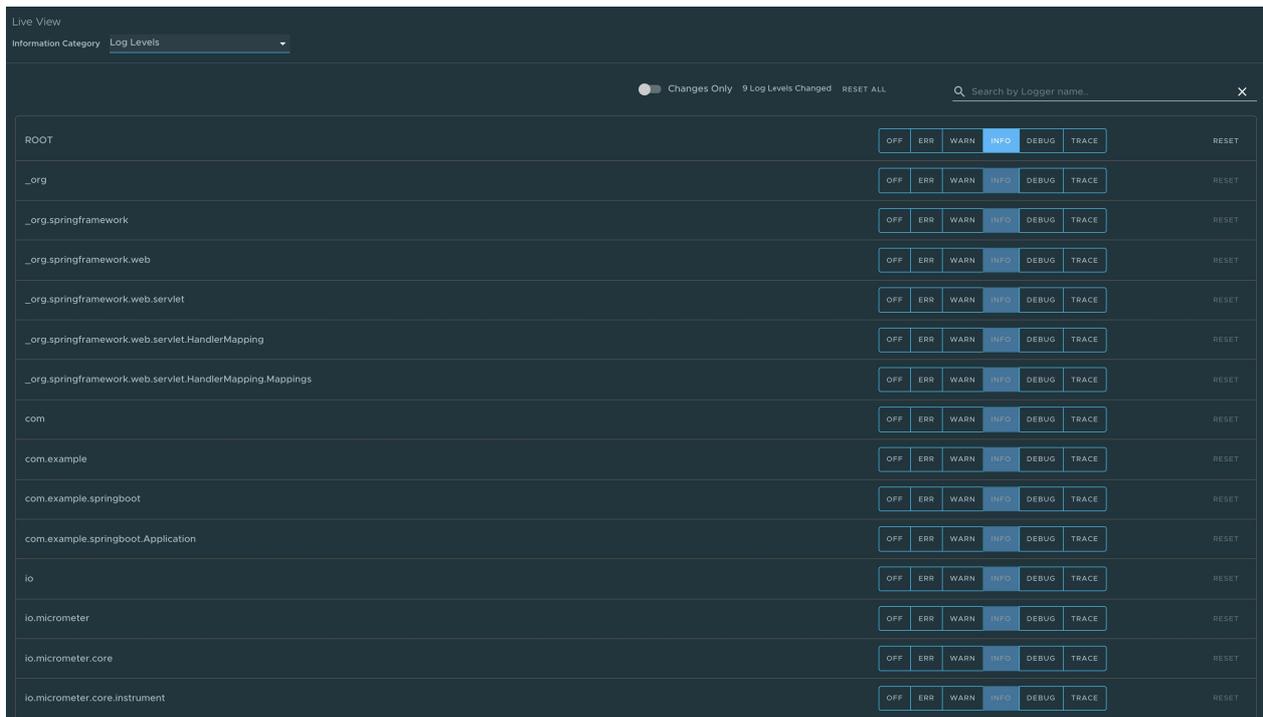
Log Levels page

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.



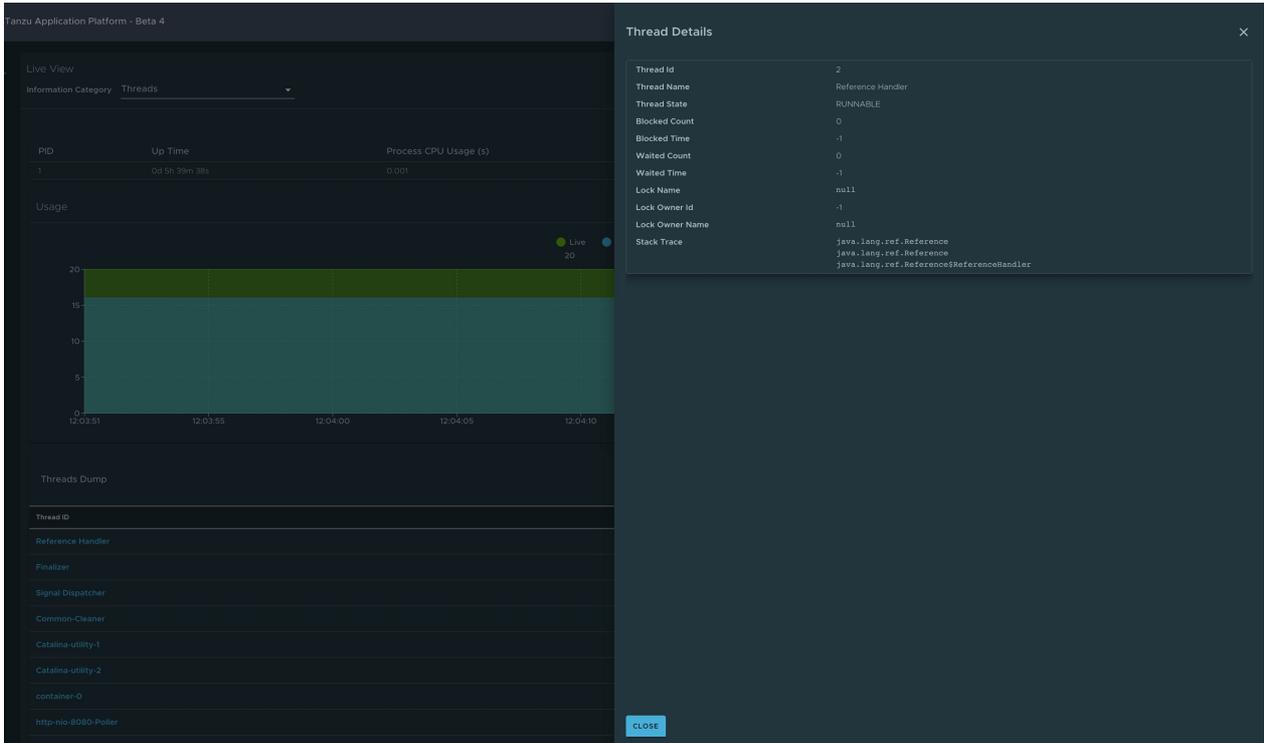
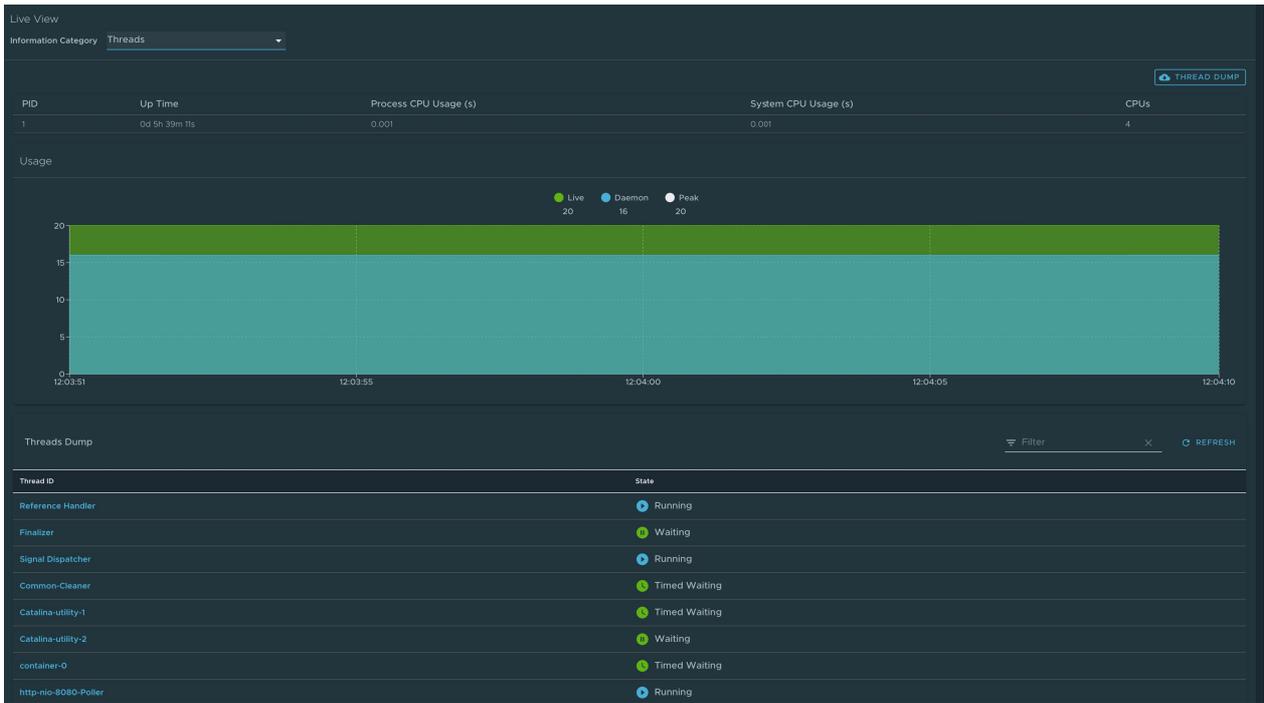
Threads page

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.

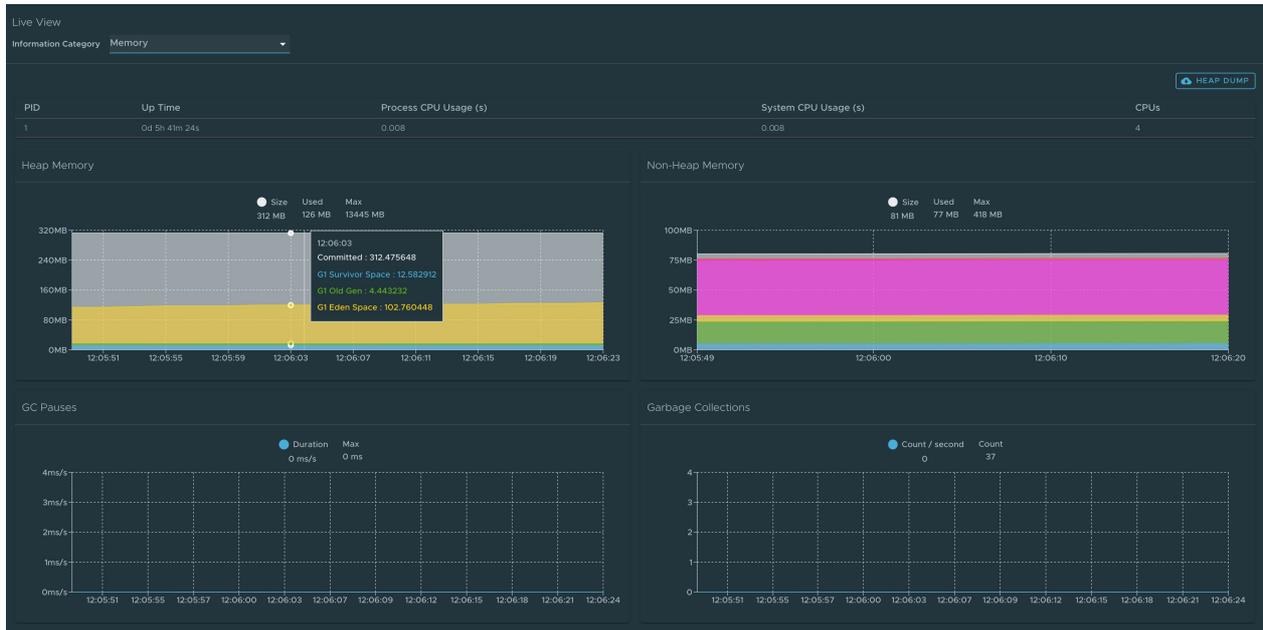


Memory page

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.



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.

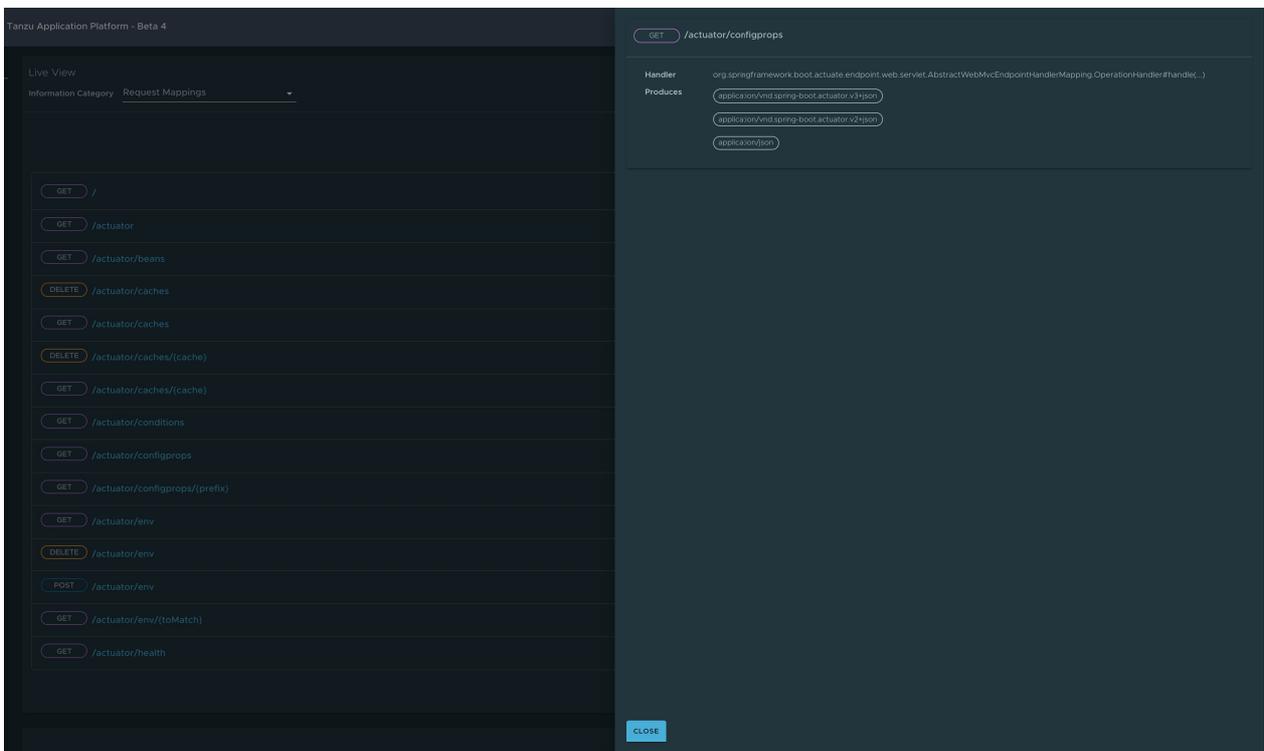
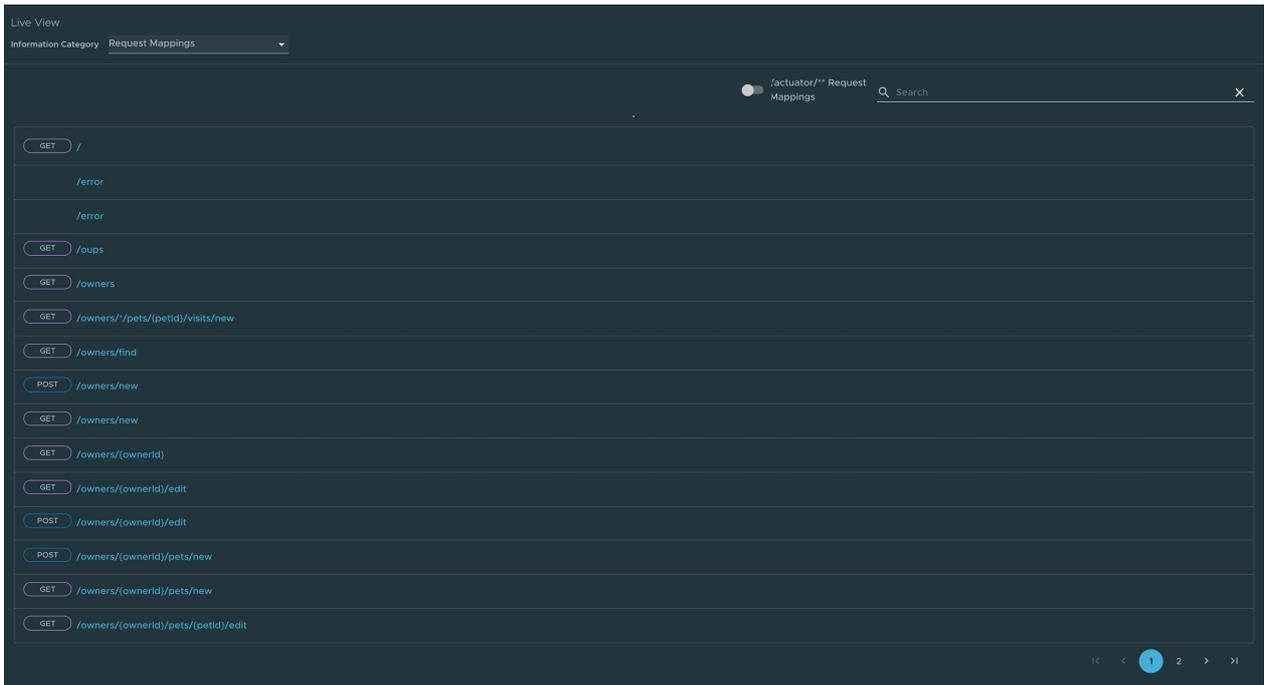
Request Mappings page

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.



HTTP Requests page

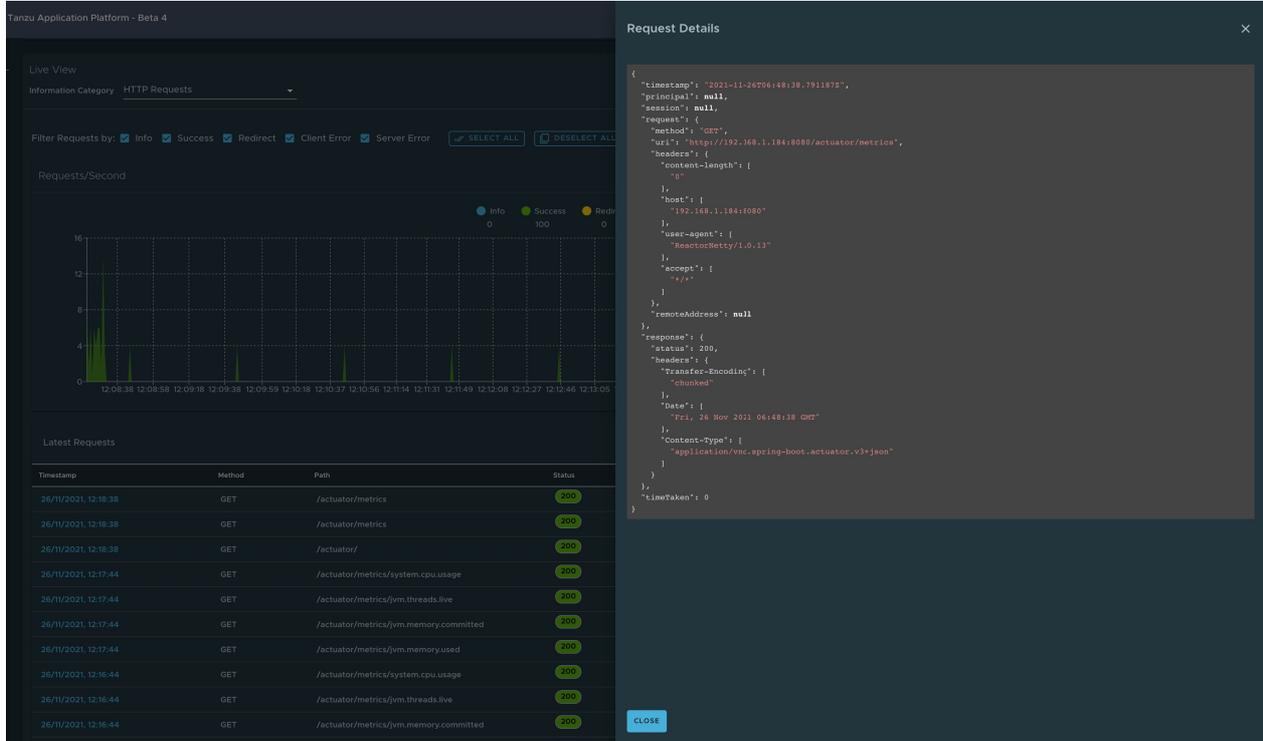
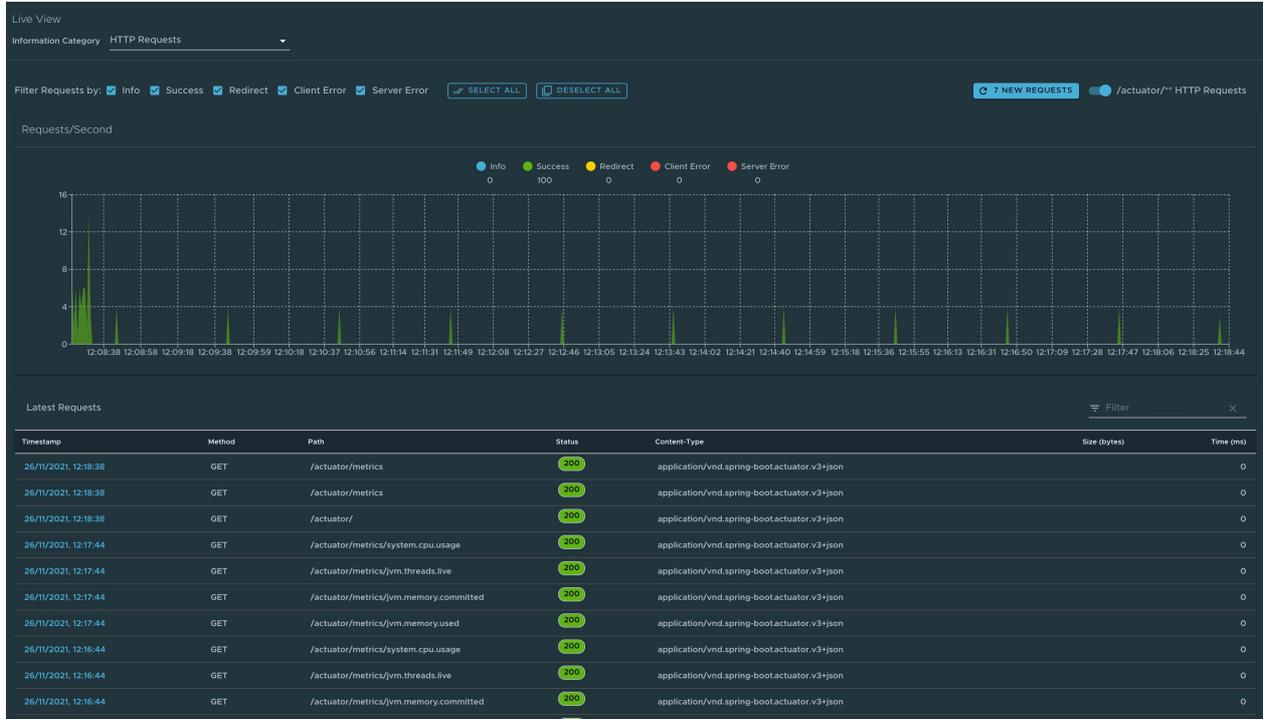
To navigate to the HTTP Requests page, the user should select the **HTTP Requests** option from the **Information Category** drop-down menu. The HTTP Requests page provides information about HTTP request-response exchanges to the application.

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.

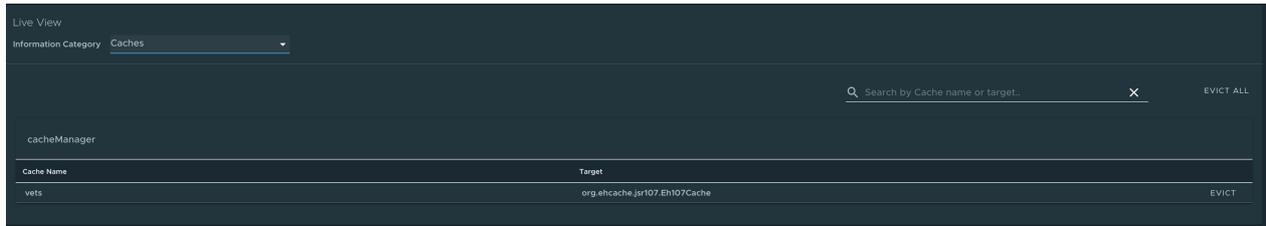


Caches page

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.

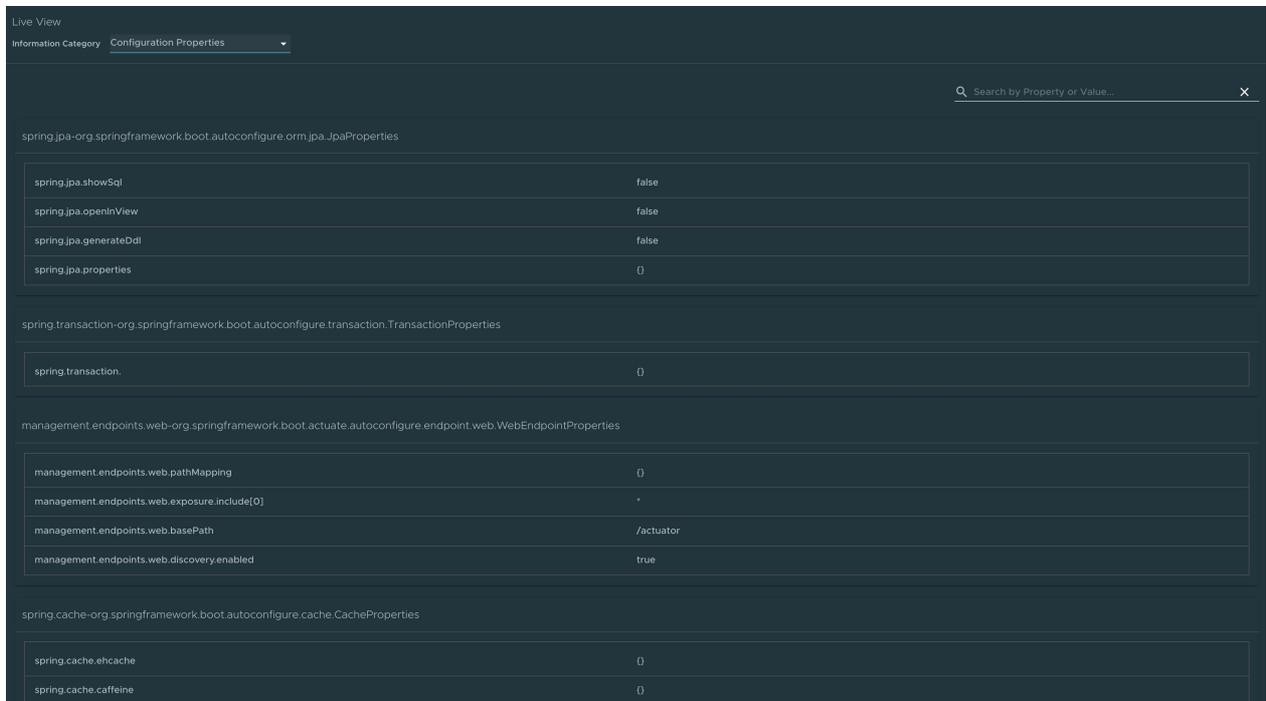
The search feature in the Caches Page enables the user to search for a specific cache/cache manager. The user can clear individual caches by clicking **Evict**. The user can clear all the caches completely by clicking **Evict All**. If there are no cache managers for the application, the message **No cache managers available for the application** is displayed.



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.

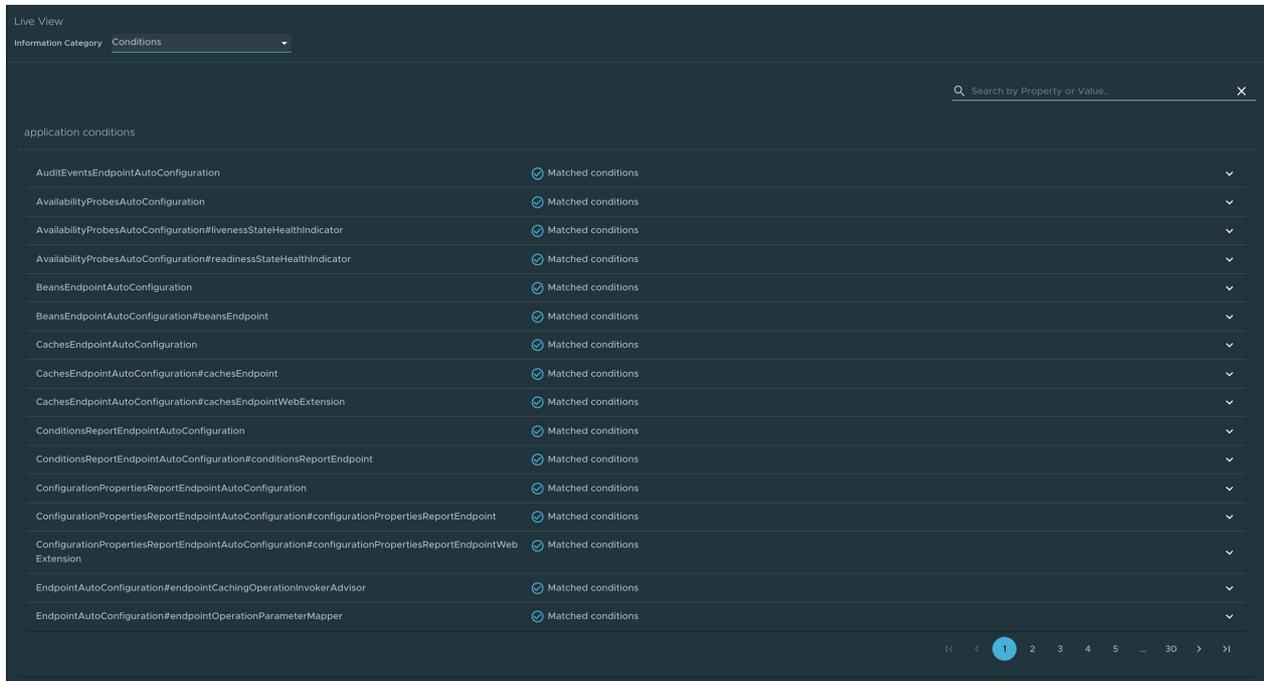


Conditions page

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.

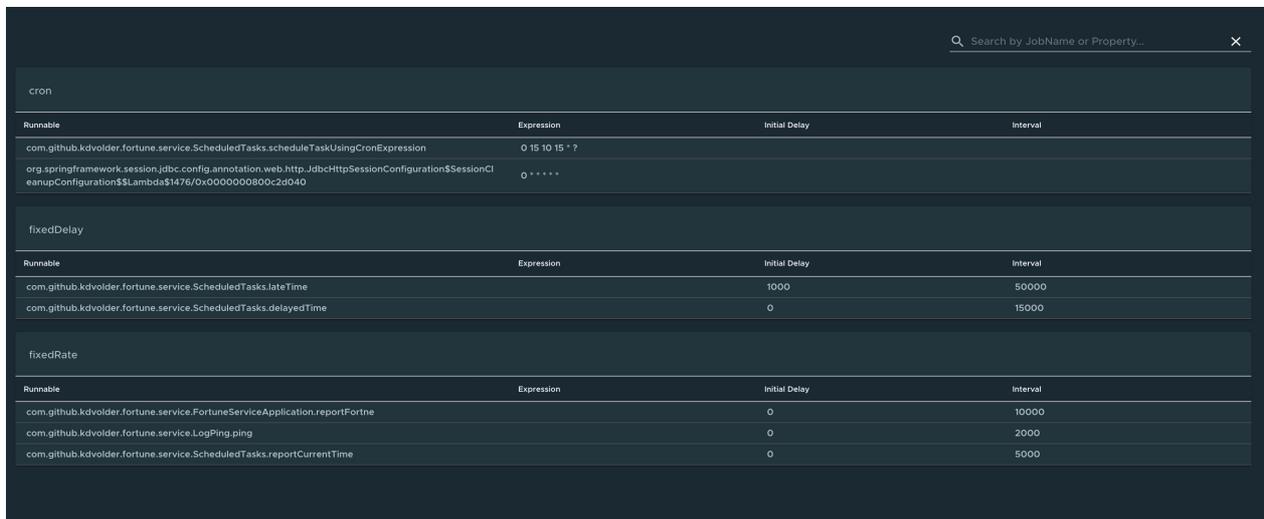


Scheduled Tasks page

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.



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.



The screenshot shows the 'Beans' page in a dark-themed interface. At the top left, there is a 'Live View' indicator and an 'Information Category' dropdown menu set to 'Beans'. A search bar on the right contains the text 'Search by Property or Value...'. Below the search bar, the text 'application beans' is displayed. A table lists various beans, each with a name and a type (all are 'singleton'). The table has a vertical scrollbar on the right. At the bottom right, there is a pagination control showing '1' selected out of 24 items.

| Bean Name | Type |
|---|-----------|
| 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 |

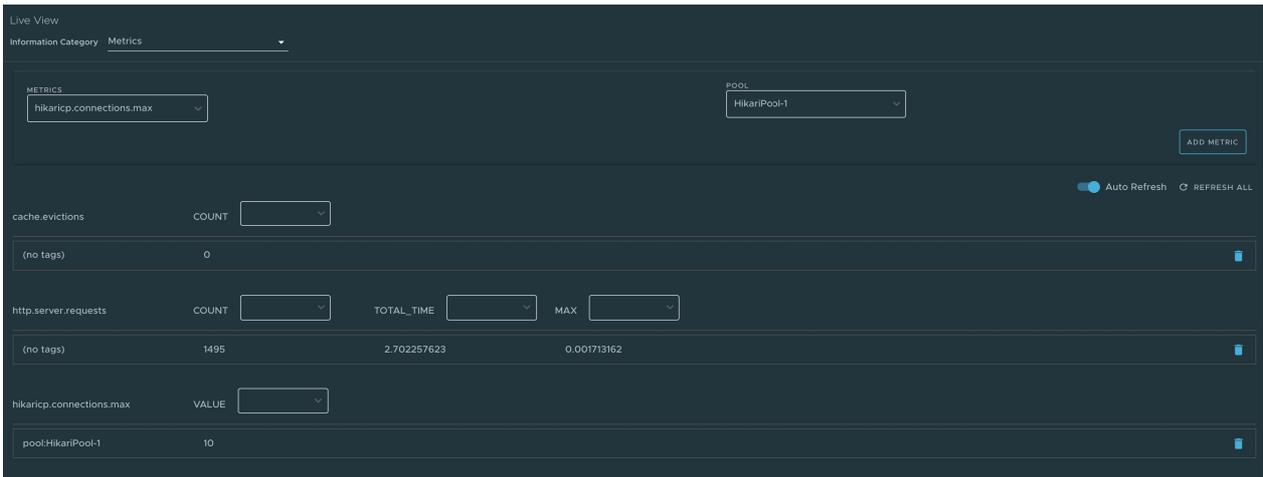
Metrics page

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.

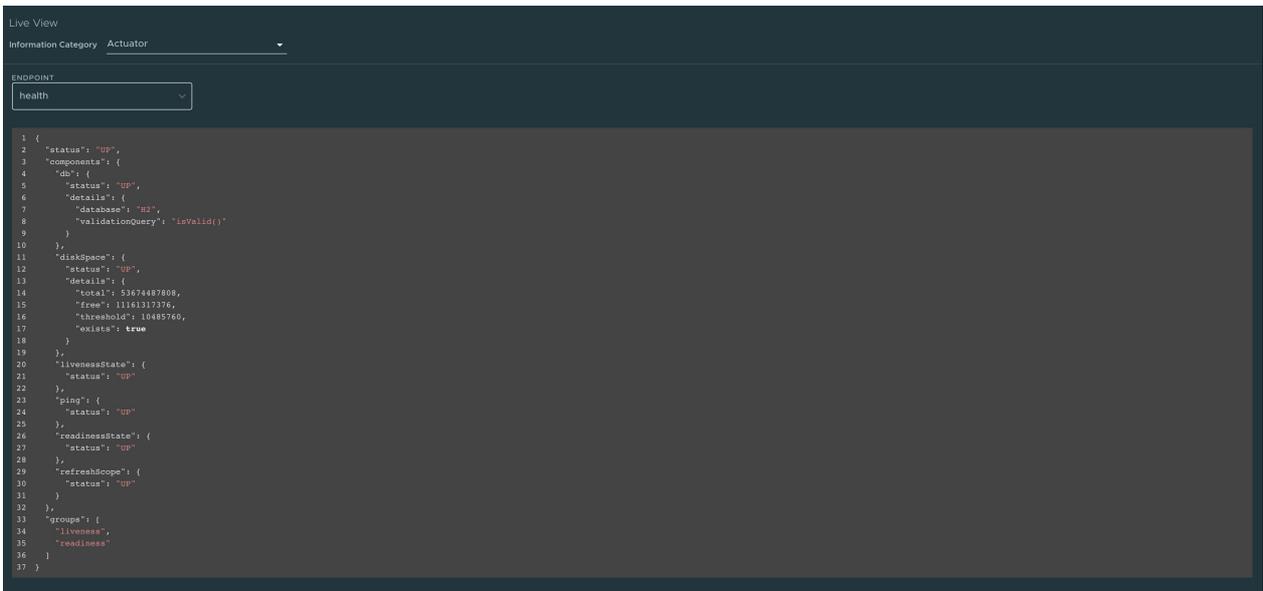
After the metric is chosen, the user can view the associated tags. The user can choose the value of each of the tags based on filtering criteria. Clicking **Add Metric** adds the metric to the page which is refreshed every 5 seconds by default.

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.



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.



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.

Install Application Live View

This topic describes how to install Application Live View from the Tanzu Application Platform 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 Backend package (`backend.appliveview.tanzu.vmware.com`). This installs the Application Live View Backend

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.

Use the instructions on this page if you do not want to use a profile to install packages. For more information about profiles, see [Installing the Tanzu Application Platform Package and Profiles](#).

Prerequisites

Before installing Application Live View, complete all prerequisites to install Tanzu Application Platform. For more information, see [Prerequisites](#).

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 Backend 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 Backend.

Install Application Live View Backend

To install Application Live View Backend:

1. List version information for the package by running:

```
tanzu package available list backend.appliveview.tanzu.vmware.com --namespace tap-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 1.1.1     2022-04-22T00:00:10Z
```

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, `1.1.1`.

For example:

```
$ tanzu package available get backend.appliveview.tanzu.vmware.com/1.1.1 --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 single cluster environment, use the following values:

```
ingressEnabled: "false"
```

For a multicluster environment, use 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 `appliveview` 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 Backend package by running:

```
tanzu package install appliveview -p backend.appliveview.tanzu.vmware.com -v VERSION-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 appliveview -p backend.appliveview.tanzu.vmware.com -v 1.1.1 -n tap-install -f app-live-view-backend-values.yaml
- Installing package 'backend.appliveview.tanzu.vmware.com'
| Getting namespace 'tap-install'
| Getting package metadata for 'backend.appliveview.tanzu.vmware.com'
| Creating service account 'appliveview-tap-install-sa'
| Creating cluster admin role 'appliveview-tap-install-cluster-role'
| Creating cluster role binding 'appliveview-tap-install-cluster-rolebinding'
```

```
| Creating package resource
| Package install status: Reconciling

Added installed package 'appliveview' in namespace 'tap-install'
```

The Application Live View Backend component is deployed in `app-live-view` namespace by default.

5. Verify the Application Live View Backend 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:     1.1.1
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 1.1.1       2022-04-22T00:00:10Z
```

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, `1.1.1`.

For example:

```
$ tanzu package available get connector.appliveview.tanzu.vmware.com/1.1.1 --va
lues-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, use the following values:

```
backend:
  sslDisabled: "true"
```

The Application Live View Connector connects to the `cluster-local` back end to register the applications.

For a multicluster environment, use the following values:

```
backend:
  sslDisabled: "false"
  host: appliveview.INGRESS-DOMAIN
```

Where `INGRESS-DOMAIN` is the top level domain the Application Live View Backend exposes by using `tanzu-shared-ingress` for the Connectors in other clusters to reach the back end. Prepend the `appliveview` subdomain to the provided value.

The `sslDisabled` boolean flag is treated as a string in Kubernetes YAML. Therefore it must be specified in `double-quotes` for the configuration to be picked up.

You can edit the values to suit your project needs or leave the default values as is.

Using the HTTP proxy either on 80 or 443 based on SSL config exposes the Backend service running on port 7000. The connector connects to the Backend 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 appliveview-connector -p connector.appliveview.tanzu.vmware.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, `1.1.1`.

For example:

```
$ tanzu package install appliveview-connector -p connector.appliveview.tanzu.vmware.com -v 1.1.1 -n tap-install -f app-live-view-connector-values.yaml
| Installing package 'connector.appliveview.tanzu.vmware.com'
| Getting namespace 'tap-install'
| Getting package metadata for 'connector.appliveview.tanzu.vmware.com'
| Creating service account 'appliveview-connector-tap-install-sa'
| Creating cluster admin role 'appliveview-connector-tap-install-cluster-role'
| Creating cluster role binding 'appliveview-connector-tap-install-cluster-role-binding'
- Creating package resource
/ Package install status: Reconciling

Added installed package 'appliveview-connector' in namespace 'tap-install'
```

Each cluster installs the connector as a DaemonSet. The connector is configured to connect to the central instance of the Backend. 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 appliveview-connector -n tap-install
```

For example:

```
tanzu package installed get appliveview-connector -n tap-install
                                                                    5s
| Retrieving installation details for appliveview-connector...
NAME:                               appliveview-connector
PACKAGE-NAME:                       connector.appliveview.tanzu.vmware.com
PACKAGE-VERSION:                    1.1.1
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.vmw  1.1.1           2022-04-22T00:00:00Z
```

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, **1.1.1**.

For example:

```
$ tanzu package install appliveview-conventions -p conventions.appliveview.tanz
u.vmware.com -v 1.1.1 -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-role
'
| 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:                     1.1.1
STATUS:                              Reconcile succeeded
CONDITIONS:                          [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`

For more information about Application Live View, see the [Application Live View documentation](#).

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](#).

Application Accelerator in Tanzu Application Platform GUI

This topic describes how to use Application Accelerator in Tanzu Application Platform GUI.

Overview

Application Accelerator for VMware Tanzu helps you bootstrap developing and deploying your applications in a discoverable and repeatable way.

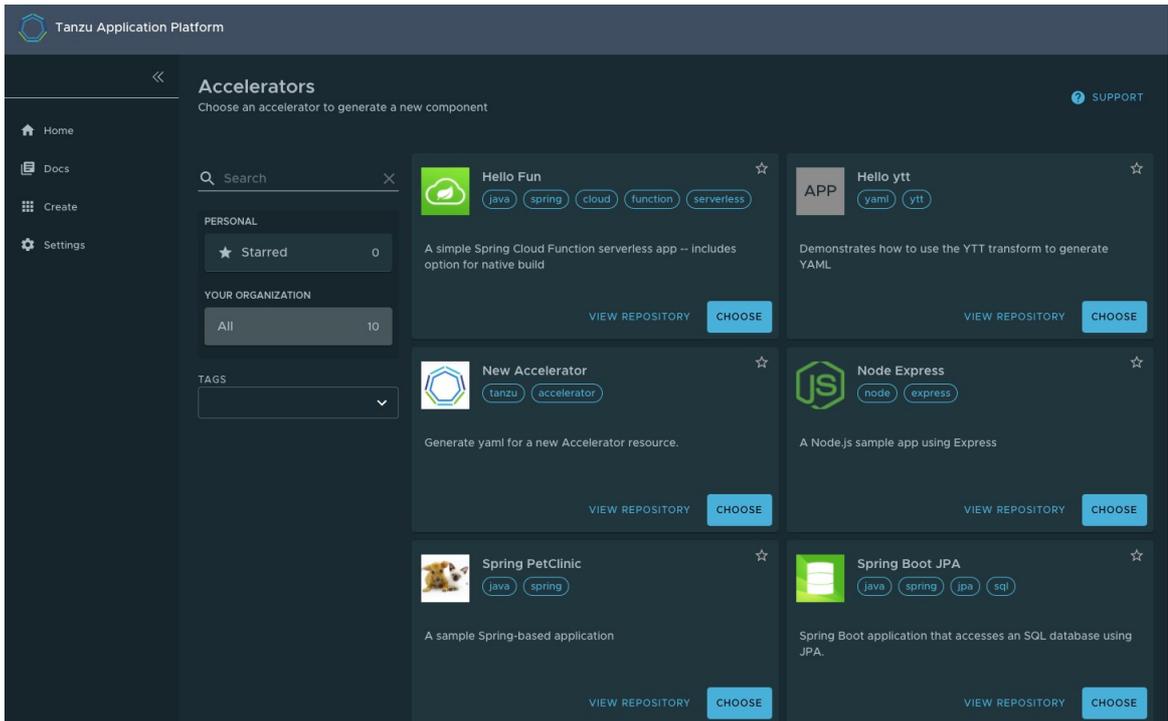
Enterprise architects author and publish accelerator projects that provide developers and operators with ready-made, enterprise-conforming code and configurations. You can then use Application Accelerator to create new projects based on those accelerator projects.

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.



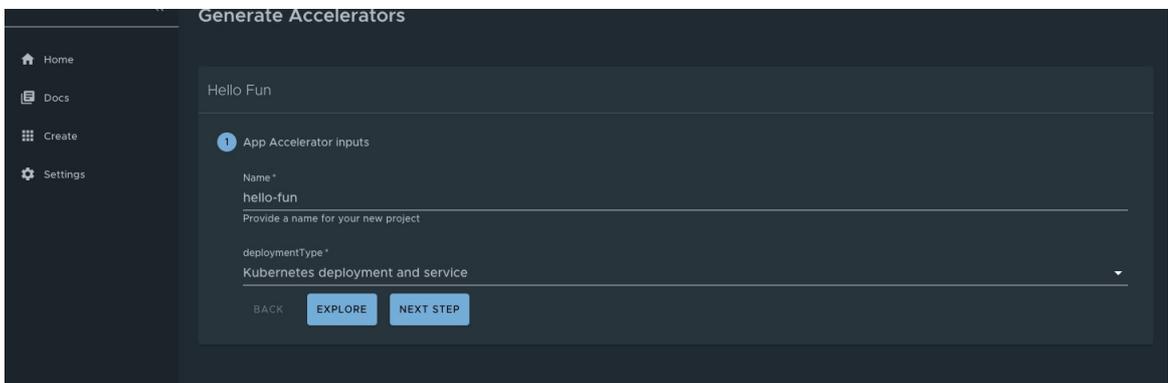
Here you can view accelerators already registered with the system. Developers can add new accelerators by registering them with Kubernetes.

2. Every accelerator has a title and short description. Click **VIEW REPOSITORY** to view an accelerator definition. This opens the accelerator’s Git repository in a new browser tab.
3. Search and filter based on text and tags associated with the accelerators to find the accelerator representing the project you want to create.
4. Click **CHOOSE** for the accelerator you want. This opens the **Generate Accelerators** page.

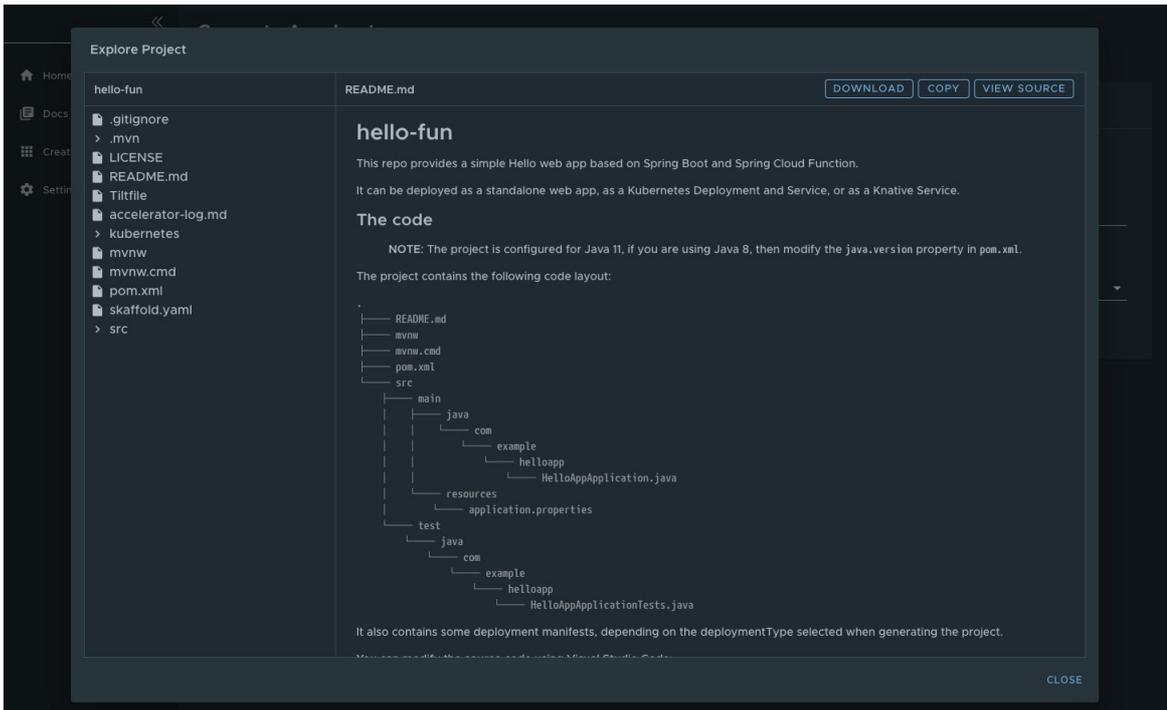
Configure project generation

To configure how projects are generated:

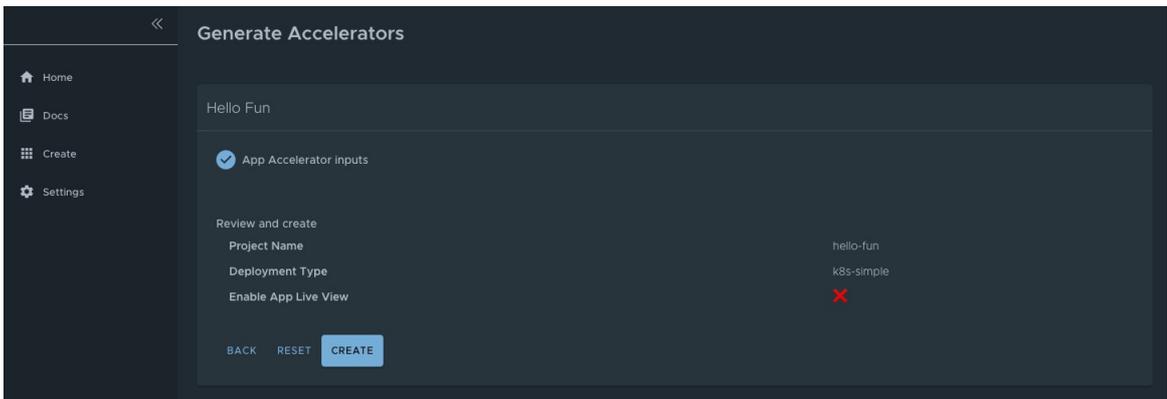
1. On the **Generate Accelerators** page, add any configuration values needed to generate the project. The application architect defined these values in `accelerator.yaml` in the accelerator definition. Filling some text boxes can cause other text boxes to appear. Fill them all in.



2. Click **EXPLORE** to open the **Explore Project** page and view the project before it is generated.



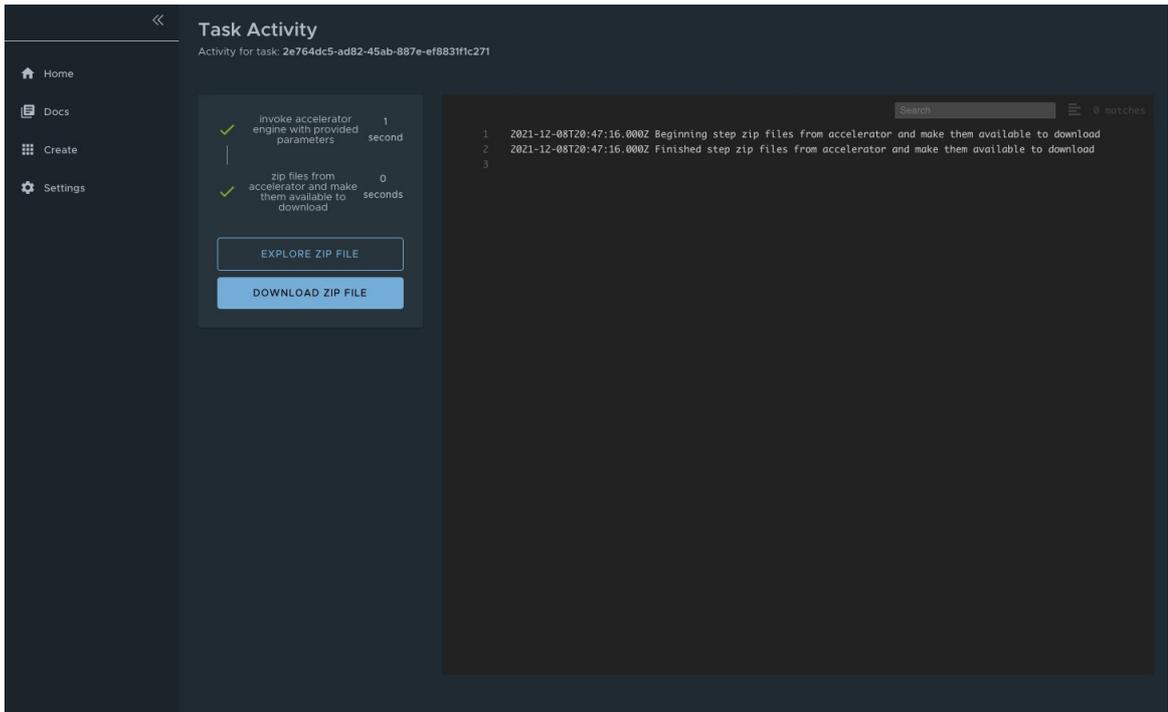
3. After configuring your project, click **NEXT STEP** to see the project summary page.
4. Review the values you specified for the configurable options.
5. Click **BACK** to make more changes, if necessary. Otherwise, proceed to [create the project](#).



Create the project

To create the project:

1. Click **Create** to start generating your project. See the progress on the **Task Activity** page. A detailed log is displayed on the right.

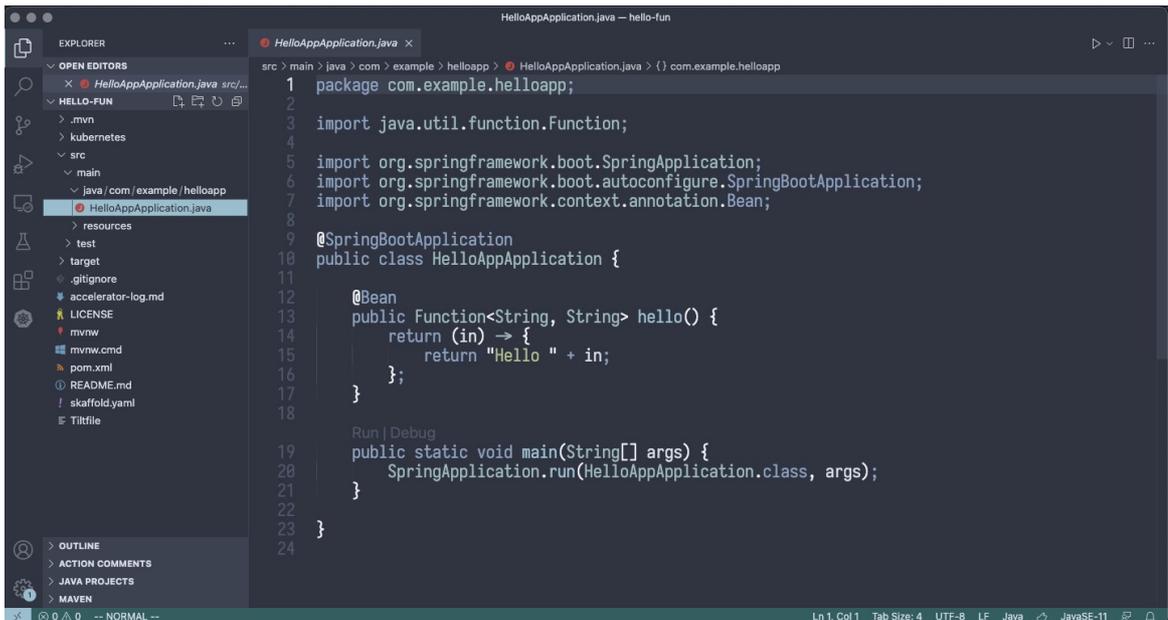


2. After the project is generated, click **EXPLORE ZIP FILE** to open the **Explore Project** page to verify configuration.
3. Click **DOWNLOAD ZIP FILE** to download the project in a ZIP file.

Develop your code

To develop your code:

1. Expand the ZIP file.
2. Open the project in your integrated development environment (IDE).



Next steps

To learn more about Application Accelerator for VMware Tanzu, see the [Application Accelerator documentation](#).

Install Application Accelerator

This document describes how to install Application Accelerator 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 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 FluxCD 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 | LoadBalancer | 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 |

| Property | Default | Description |
|--|----------------------|---|
| 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 |
| tls.secretName | 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_invocations_on_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 usage 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 tap-install
- Retrieving package versions for accelerator.apps.tanzu.vmware.com...
NAME                                VERSION  RELEASED-AT
accelerator.apps.tanzu.vmware.com  0.5.1   2021-12-02T00:00:00Z
```

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/0.5.1 --values-
schema --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.

Note: 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
1.0.0 -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:   1.0.0
STATUS:            Reconcile succeeded
CONDITIONS:       [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:

```

Verify that `STATUS` is `Reconcile succeeded`.

- 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 section provides a general overview of the API documentation plug-in of the Tanzu Application Platform GUI. For more information, see [Getting started with 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

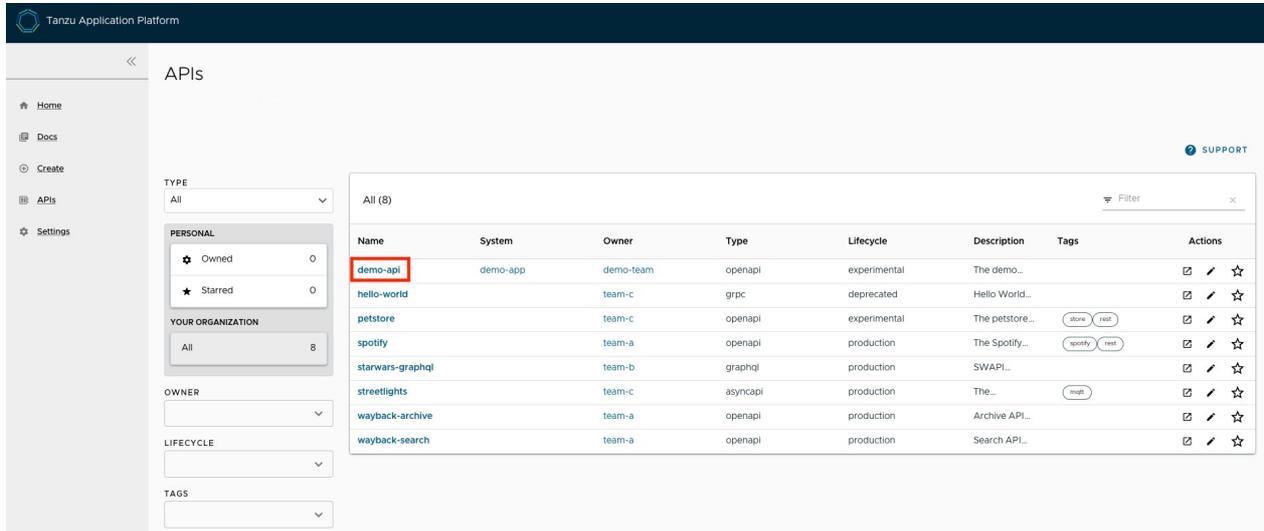
```

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

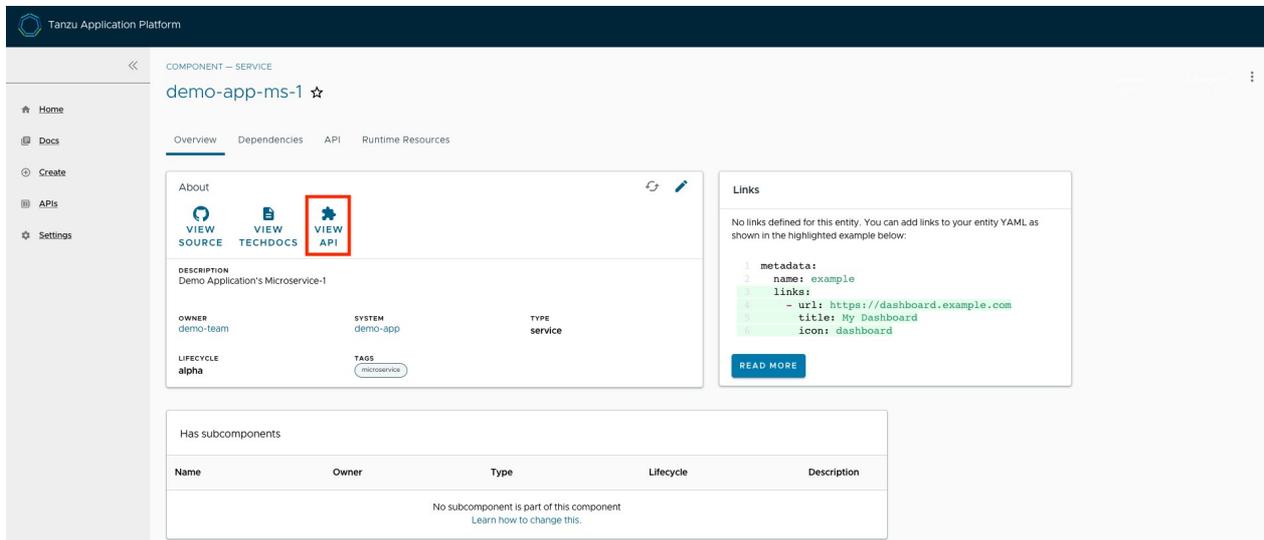
The API documentation plug-in is part of Tanzu Application Platform GUI.

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**.

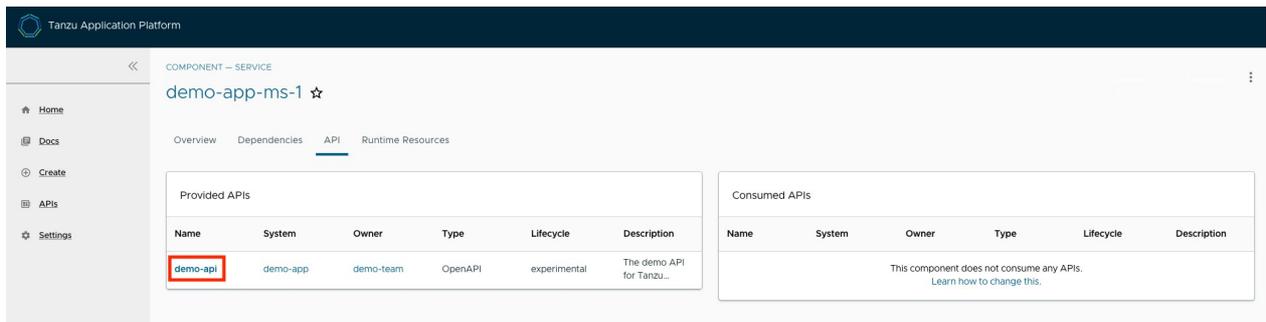


On that page, you can view all the APIs already registered in the catalog regardless of whether they are associated with components or systems.

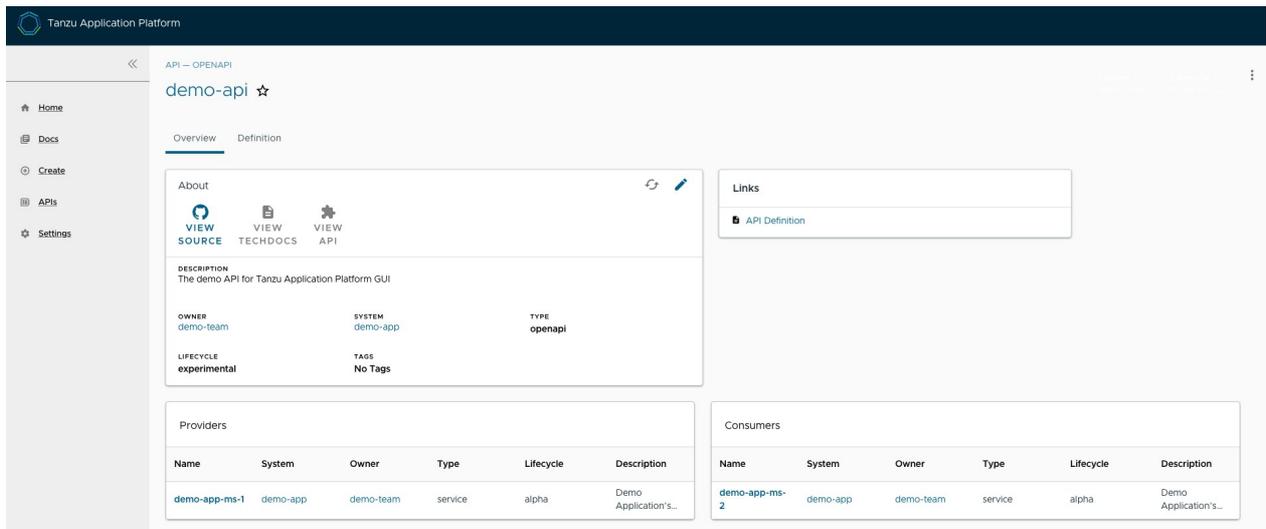
The second way to use the API documentation plug-in is by using components and systems of the software catalog, listed on the home page of Tanzu Application Platform GUI. If there is an API entity associated with the selected component or system, the **VIEW API** icon is active.



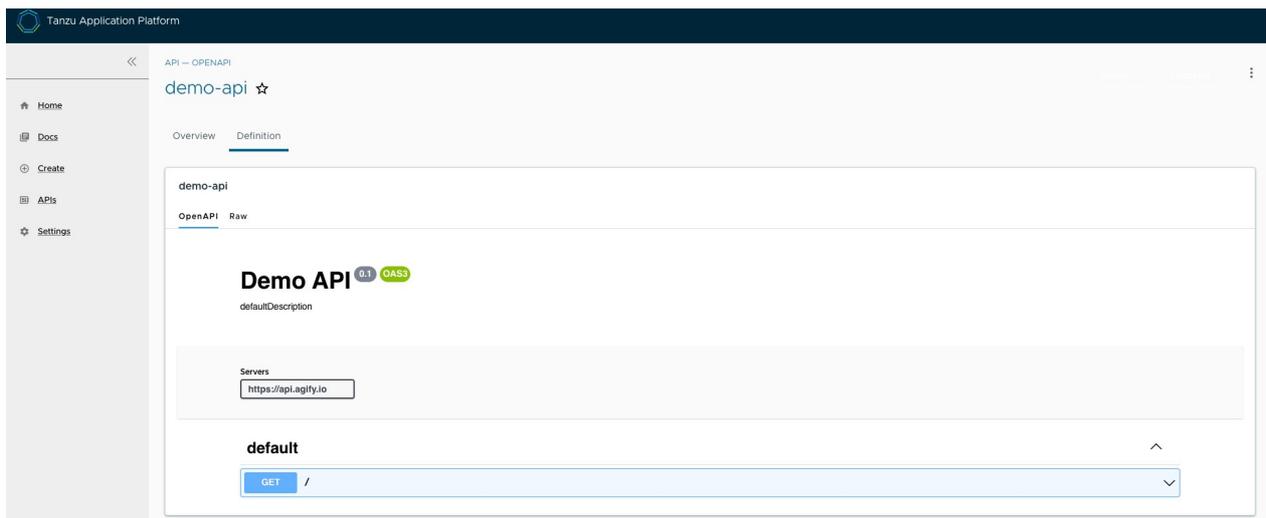
The **VIEW API** tab displays which APIs are being consumed by a component and which APIs are being provided by the component.



Clicking on the API itself takes you to the catalog entry for the API, which the Kind type listed in the upper-left corner denotes. Every API entity has a title and short description, including a reference to the team that owns the definition of that API and the software catalog objects that are connected to it.



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:

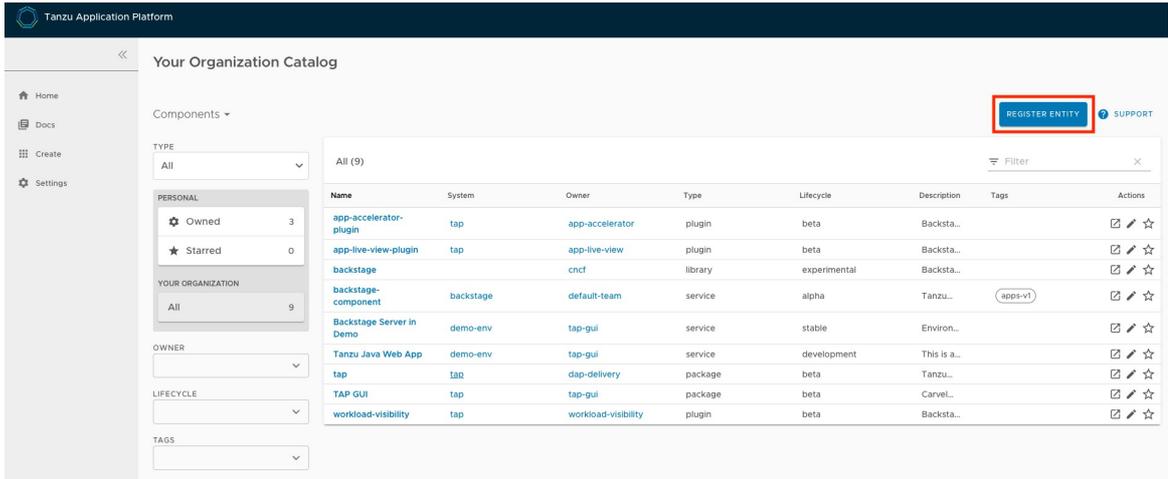
- OpenAPI 2 & 3
- AsyncAPI
- GraphQL

- 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**.



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.

```

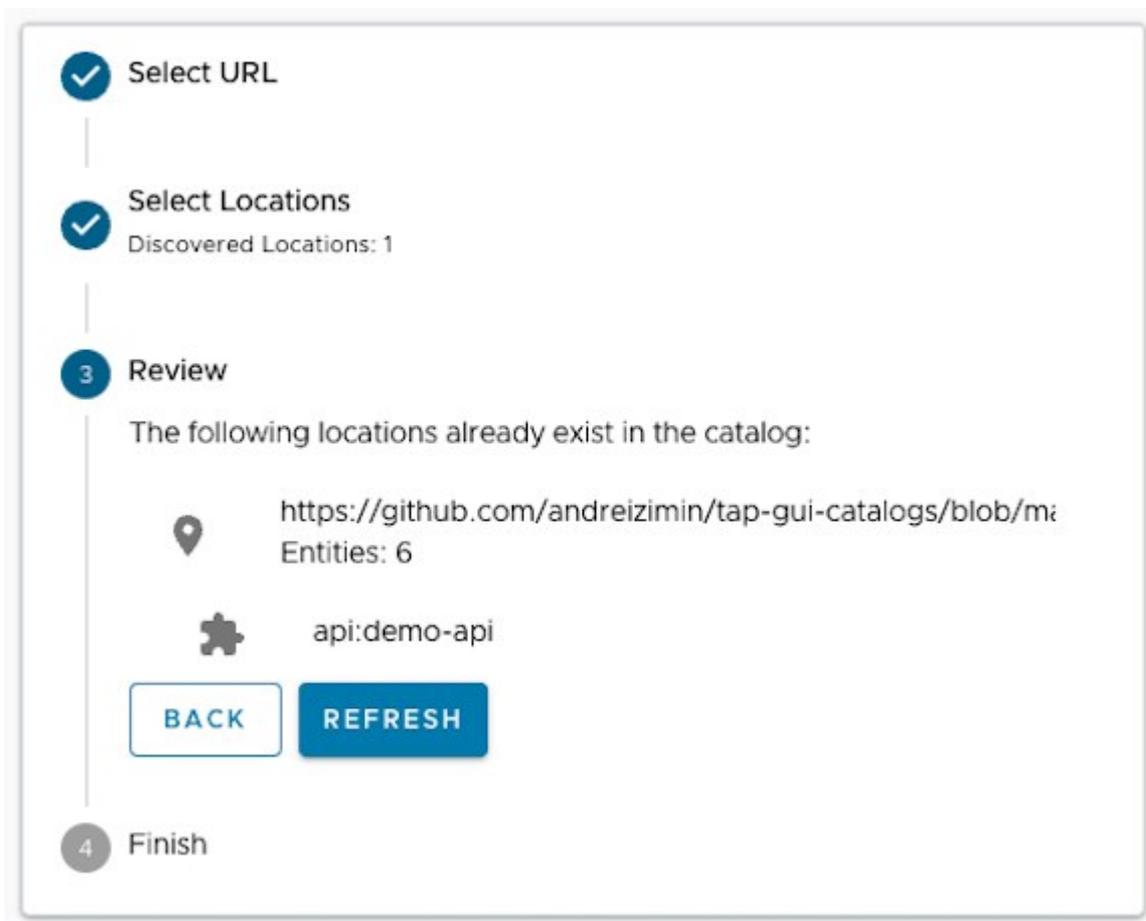
apiVersion: backstage.io/v1alpha1
kind: API
metadata:
  name: demo-api
  description: The demo API for Tanzu Application Platform GUI
  links:
    - url: https://api.agify.io
      title: API Definition
      icon: docs
spec:
  type: openapi
  lifecycle: experimental
  owner: demo-team
  system: demo-app # 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.



5. Click **IMPORT**.
6. Click **APIs** on the left-hand side navigation panel to view entries on the **API** page.

Getting started with API documentation plug-in

This topic describes how to get started with the API documentation plug-in.

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

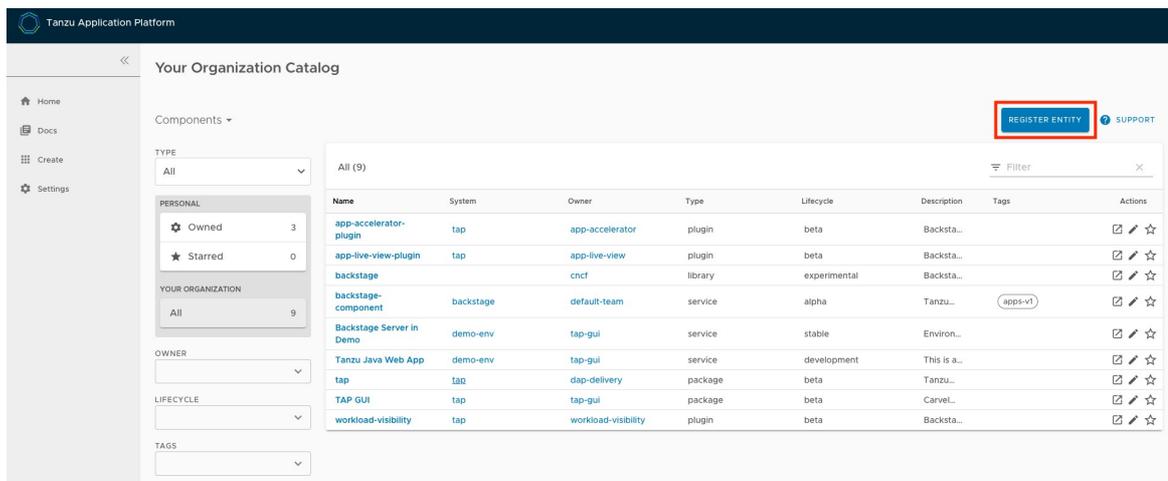
The list of API entities is visible on the left-hand side navigation panel of Tanzu Application Platform GUI. It is also visible on the overview page of specific components on the home page. APIs are a definition of the interface between components.

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**.



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

```

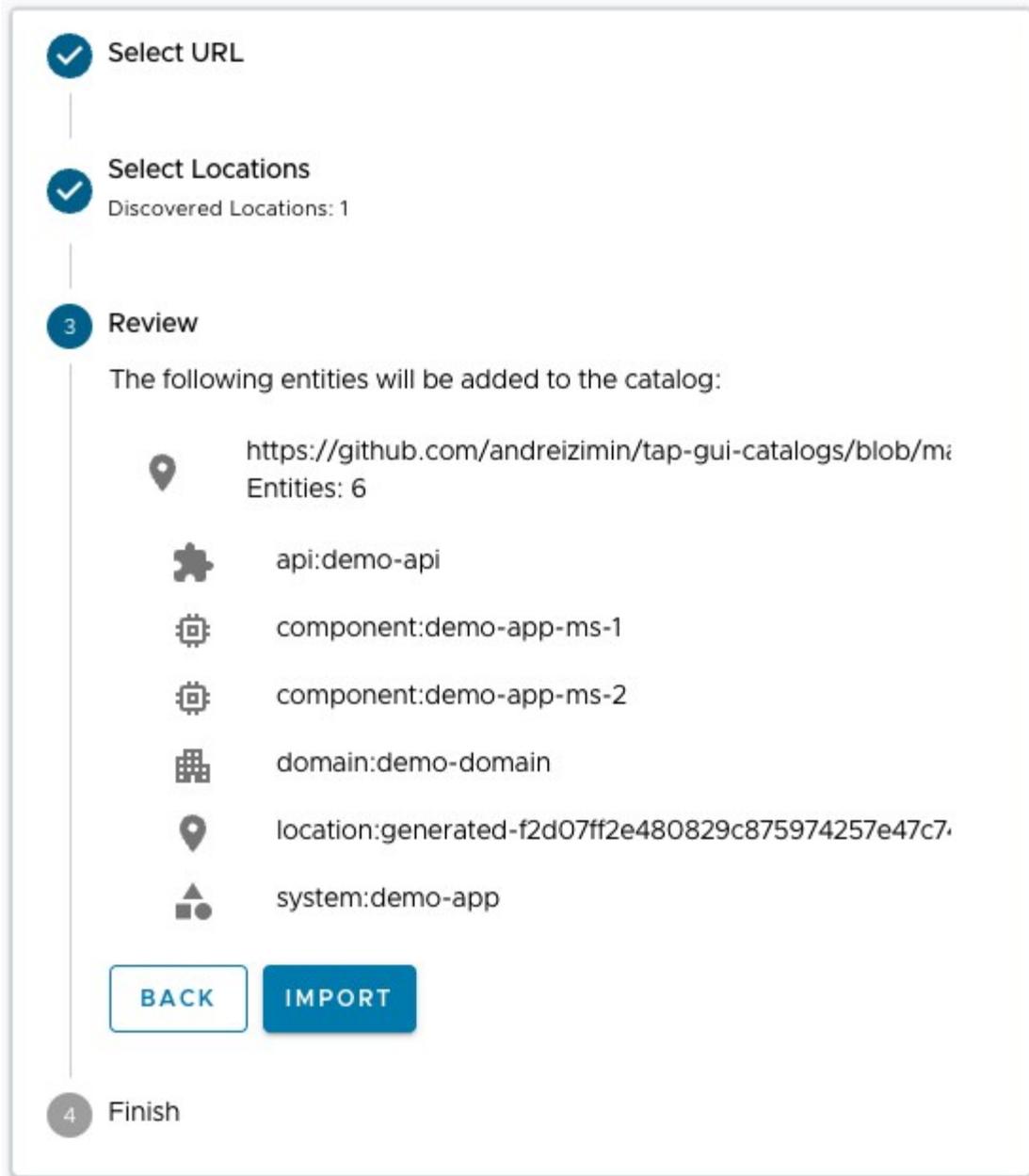
```

---

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**.



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.

The screenshot shows the 'demo-api' overview page. The 'About' section includes 'VIEW SOURCE', 'VIEW TECHDOCS', and 'VIEW API' buttons. A red box highlights the edit icon (pencil) next to the 'VIEW API' button. The 'DESCRIPTION' field contains 'The demo API for Tanzu Application Platform GUI'. The 'OWNER' is 'demo-team', 'SYSTEM' is 'demo-app', and 'TYPE' is 'openapi'. The 'LIFECYCLE' is 'experimental' and 'TAGS' are 'No Tags'. There are 'Providers' and 'Consumers' tables below.

| Name | System | Owner | Type | Lifecycle | Description |
|---------------|----------|-----------|---------|-----------|-----------------------|
| demo-app-ms-1 | demo-app | demo-team | service | alpha | Demo Application's... |

| 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 describes Supply Chain Choreographer in Tanzu Application Platform 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

You must have the Full profile or View profile installed on your cluster, which includes Tanzu Application Platform GUI, or have installed the Tanzu Application Platform GUI package.

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.

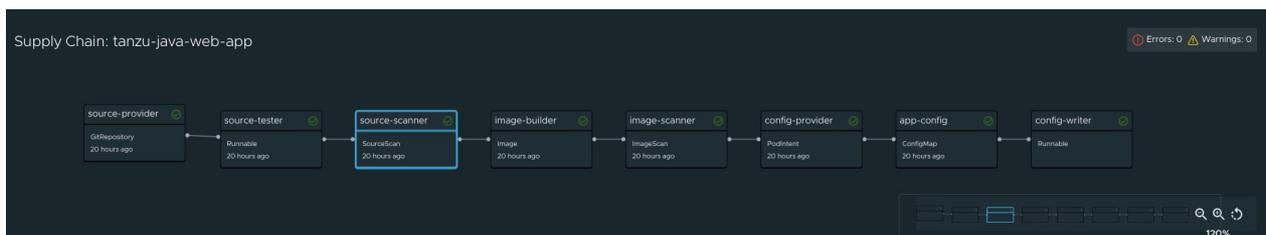
Use the left sidebar navigation to access your workload and visualize it in the supply chain that is installed on your cluster.

The example workload described in this topic is named `tanzu-java-web-app`.

| 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 |

5 rows | < < 1-2 of 2 > >

Click `tanzu-java-web-app` in the **WORKLOADS** table to navigate to the visualization of the supply chain.



There are two sections within this view:

- The box-and-line diagram at the top shows all the configured 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

re-provider (postitory, urs ago) → source-tester (Runnable, 20 hours ago) → source-scanner (SourceScan, 20 hours ago) → **image-builder (Image, 20 hours ago)** → image-scanner (ImageScan, 20 hours ago) → config-provider (PodIntert, 20 hours ago) → app-config (ConfigMap, 20 hours ago) → config-write (Runnable)

106%

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 | 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

Upgrade Tanzu Application Platform GUI

This topic describes how to upgrade Tanzu Application Platform 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 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 -p tap-gui.tanzu.vmware.com -v VERSION --va
lues-file TAP-GUI-VALUES.yaml -n tap-install
```

Where:

- ◆ **VERSION** is the desired target version of Tanzu Application Platform GUI.
- ◆ **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 describes troubleshooting information for problems with installing Tanzu Application Platform 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-h11q9            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-h11q9`.

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
```

Tanzu Build Service

VMware 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](#).

Tanzu Build Service Dependencies

Tanzu Build Service requires dependencies in the form of [Buildpacks](#) and [Stacks](#) to build OCI images.

Configuration

On non-air-gapped clusters, dependencies are imported as a part of installation of a Tanzu Application Platform profile or the Tanzu Build Service component.

When creating the values file during installation, include the `tanzunet_username`, `tanzunet_password`, and `descriptor_name` key-value pairs, in addition to any other `buildservice` key-value pairs, as in this example:

```
... kp_default_repository: REPOSITORY kp_default_repository_username: REGISTRY-USERNAME
kp_default_repository_password: REGISTRY-PASSWORD tanzunet_username: TANZUNET-USERNAME
tanzunet_password: TANZUNET-PASSWORD descriptor_name: DESCRIPTOR-NAME ...
```

Where:

- `TANZUNET-USERNAME` and `TANZUNET-PASSWORD` are the email address and password that you use to log in to VMware Tanzu Network.
- `DESCRIPTOR-NAME` is the name of the descriptor to import automatically. 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.

Descriptors

Tanzu Build Service descriptors are curated sets of dependencies, including stacks and buildpacks, that are continuously released on [VMware Tanzu Network](#) to resolve all workload Critical and High CVEs. Descriptors are imported into Tanzu Build Service to update the entire cluster.

There are two types of descriptor, `lite` and `full`. The different descriptors can apply to different use cases and workload types. You can configure which descriptor is imported after installation when installing Tanzu Application Platform or the Tanzu Build Service component.

For more information, see [Descriptors](#).

Automatic Dependency Updates

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.

When creating the values file during installation, include the key-value pair

`enable_automatic_dependency_updates: true`, in addition to any other `buildservice` keys, as in this example:

```
...
kp_default_repository: REPOSITORY
kp_default_repository_username: REGISTRY-USERNAME
kp_default_repository_password: REGISTRY-PASSWORD
tanzunet_username: TANZUNET-USERNAME
tanzunet_password: TANZUNET-PASSWORD
descriptor_name: DESCRIPTOR-NAME
enable_automatic_dependency_updates: true
...
```

Manual Control of Dependency Updates

Sometimes you might not want Tanzu Build Service to automatically update dependencies in the background.

In this case, you can manually manage and update your dependencies individually or automate the updating configuration yourself in a CI/CD context.

The Tanzu Build Service package in Tanzu Application Platform behaves identically to the standalone Tanzu Build Service product described in the [Tanzu Build Service documentation](#).

For updating dependencies manually, see [Updating Build Service Dependencies](#).

Install Tanzu Build Service

This document describes how to install Tanzu Build Service from the Tanzu Application Platform package repository by using the Tanzu CLI.

Note: Use the instructions on this page if you do not want to use a profile to install packages. Both the full and light profiles include Tanzu Build Service. For more information about profiles, see [Installing the Tanzu Application Platform Package and Profiles](#).

Note: The following procedure might not include some configurations required for your specific environment. For more advanced details on installing Tanzu Build Service, see [Installing Tanzu Build Service](#).

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 descriptor.

- Your Docker registry must be accessible with username and password credentials.

Install Tanzu Build Service by using the Tanzu CLI

To install Tanzu Build Service by using the Tanzu CLI:

1. List version information for the package by running:

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

For example:

```
$ tanzu package available list buildservice.tanzu.vmware.com --namespace tap-in
stall
- Retrieving package versions for buildservice.tanzu.vmware.com...
NAME                                VERSION  RELEASED-AT
buildservice.tanzu.vmware.com      1.5.0    2022-12-17T00:00:00Z
```

2. (Optional) To make changes to the default installation settings, run:

```
tanzu package available get buildservice.tanzu.vmware.com/VERSION-NUMBER --valu
es-schema --namespace tap-install
```

Where `VERSION-NUMBER` is the version of the package listed in step 1 above.

For example:

```
$ tanzu package available get buildservice.tanzu.vmware.com/1.5.0 --values-sche
ma --namespace tap-install
```

3. Gather the values schema by running:

```
tanzu package available get buildservice.tanzu.vmware.com/1.5.0 --values-schema
--namespace tap-install
```

4. Create a `tbs-values.yaml` file.

```
---
kp_default_repository: "KP-DEFAULT-REPO"
kp_default_repository_username: "KP-DEFAULT-REPO-USERNAME"
kp_default_repository_password: "KP-DEFAULT-REPO-PASSWORD"
tanzunet_username: "TANZUNET-USERNAME"
tanzunet_password: "TANZUNET-PASSWORD"
enable_automatic_dependency_updates: TRUE-OR-FALSE-VALUE # Optional, set a
s true or false. Not a string.
```

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 name of the user who can write to `KP-DEFAULT-REPO`. You can write 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 write to this location with this credential. This credential can also be configured by using a Secret reference. For more information, see [Installation using Secret References for registry credentials](#) for details.
 - For Google Cloud Registry, use the contents of the service account json file.
- ◆ `TANZUNET-USERNAME` and `TANZUNET-PASSWORD` are the email address and password that you use to log in to VMware Tanzu Network. Your VMware Tanzu Network credentials enable you to configure the dependencies updater. This resource accesses and installs the build dependencies (buildpacks and stacks) Tanzu Build Service needs on your cluster. It can also optionally keep these dependencies up to date as new versions are released on VMware Tanzu Network. This credential can also be configured by using a Secret reference. See [Installation using Secret References for registry credentials](#) for details.
- ◆ `DESCRIPTOR-NAME` is the name of the descriptor to import. For more information about which descriptor to choose for your workload and use case, see [Descriptors](#). Available options are:
 - `lite` is the default if unset. It has a smaller footprint, which enables faster installations.
 - `full` is optimized to speed up builds and includes dependencies for all supported workload types.

Note: By using the `tbs-values.yaml` configuration, `enable_automatic_dependency_updates: true` causes the dependency updater to update Tanzu Build Service dependencies (buildpacks and stacks) when they are released on VMware Tanzu Network. You can set `enable_automatic_dependency_updates` as `false` to pause the automatic update of Build Service dependencies. When automatic updates are paused, the pinned version of the descriptor for Tanzu Application Platform v1.1.0 is `100.0.293`. If left undefined, this value is `false`. For information about updating dependencies manually, see [Manual Control of Dependency Updates](#).

5. Install the package by running:

```
tanzu package install tbs -p buildservice.tanzu.vmware.com -v 1.5.0 -n tap-inst
all -f tbs-values.yaml --poll-timeout 30m
```

For example:

```

$ tanzu package install tbs -p buildservice.tanzu.vmware.com -v 1.5.0 -n tap-in
stall -f tbs-values.yaml --poll-timeout 30m
| 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'

```

Note: Installing the `buildservice.tanzu.vmware.com` package with Tanzu Network credentials automatically relocates buildpack dependencies to your cluster. This install process can take some time and the `--poll-timeout` flag increases the timeout duration. Using the `lite` descriptor speeds this up significantly. If the command times out, periodically run the installation verification step provided in the following optional step. Image relocation continues in the background.

6. (Optional) Verify the clusterbuilders that the Tanzu Build Service installation created by running:

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

Install Tanzu Build Service using the Tanzu CLI air-gapped

Tanzu Build Service can be installed to a Kubernetes Cluster and registry that are air-gapped from external traffic.

These steps assume that you have installed the TAP packages in your air-gapped environment.

To install the Tanzu Build Service package air-gapped:

1. Gather the values schema by running:

```
tanzu package available get buildservice.tanzu.vmware.com/1.5.0 --values-schema
--namespace tap-install
```

2. Create a `tbs-values.yaml` file. The required fields for an air-gapped installation are:

```

---
kp_default_repository: REPOSITORY
kp_default_repository_username: REGISTRY-USERNAME
kp_default_repository_password: REGISTRY-PASSWORD
ca_cert_data: CA-CERT-CONTENTS

```

Where:

- ◆ `REPOSITORY` is the fully qualified path to the Tanzu Build Service repository. This path must be writable. For example:
 - Harbor: `harbor.io/my-project/build-service`

- **Artifactory:** `artifactory.com/my-project/build-service`
 - ◆ `REGISTRY-USERNAME` and `REGISTRY-PASSWORD` are the user name and password for the internal registry.
 - ◆ `CA-CERT-CONTENTS` are the contents of the PEM-encoded CA certificate for the internal registry
3. Install the package by running:

```
tanzu package install tbs -p buildservice.tanzu.vmware.com -v 1.5.0 -n tap-install -f tbs-values.yaml
```

For example:

```
$ tanzu package install tbs -p buildservice.tanzu.vmware.com -v 1.5.0 -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'
```

4. Keep Tanzu Build Service dependencies up-to-date.

When installing Tanzu Build Service to an air-gapped environment, dependencies cannot be automatically pulled in from the external internet. So dependencies must be imported and kept up to date manually. To import dependencies to an air-gapped Tanzu Build Service, follow the official [Tanzu Build Service docs](#).

Installation using Secret references for registry credentials

Tanzu Build Service requires credentials for the `kp_default_repository` and the Tanzu Network registry.

You can apply them in the `values.yaml` configuration directly in-line by using `_username` and `_password` fields such as `kp_default_repository_username/kp_default_repository_password` and `tanzunet_username/tanzunet_password`.

If you do not want credentials saved in ConfigMaps in plaintext, you can use Secret references in the `values.yaml` configuration to use existing Secrets.

To use Secret references you must create Secrets of type `kubernetes.io/dockerconfigjson` containing credentials for `kp_default_repository` and the VMware Tanzu Network registry.

Use the following alternative configuration for `values.yaml`:

```
---
kp_default_repository: "KP-DEFAULT-REPO"
kp_default_repository_secret:
  name: "KP-DEFAULT-REPO-SECRET-NAME"
```

```
namespace: "KP-DEFAULT-REPO-SECRET-NAMESPACE"
tanzunet_secret:
  name: "TANZUNET-SECRET-NAME"
  namespace: "TANZUNET-SECRET-NAMESPACE"
enable_automatic_dependency_updates: TRUE-OR-FALSE-VALUE
```

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-SECRET-NAME` is the name of the `kubernetes.io/dockerconfigjson` Secret containing credentials for `KP-DEFAULT-REPO`. You can write to this location with this credential.
- `KP-DEFAULT-REPO-SECRET-NAMESPACE` is the namespace of the `kubernetes.io/dockerconfigjson` Secret containing credentials for `KP-DEFAULT-REPO`. You can write to this location with this credential.
- `TANZUNET-SECRET-NAME` is the name of the `kubernetes.io/dockerconfigjson` Secret containing credentials for VMware Tanzu Network registry.
- `TANZUNET-SECRET-NAMESPACE` is the namespace of the `kubernetes.io/dockerconfigjson` Secret containing credentials for the VMware Tanzu Network registry.
- `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.

Descriptors

This topic describes the descriptors that are available so you can choose which option to configure depending on your use case.

About descriptors

Tanzu Build Service descriptors are curated sets of dependencies, including stacks and buildpacks, that are continuously released on VMware Tanzu Network to resolve all workload Critical and High CVEs. Descriptors are imported into Tanzu Build Service to update the entire cluster.

There are two types of descriptor, `lite` and `full`, available on the [Tanzu Network Build Service Dependencies](#) page. The different descriptors can apply to different use cases and workload types. For the differences between the descriptors, see [Descriptor comparison](#).

You configure which descriptor is imported when installing Tanzu Build Service.

Lite descriptor

The Tanzu Build Service `lite` descriptor is the default descriptor selected if none is configured.

It contains a smaller footprint to speed up installation time. However, it does not support all workload types. For example, the `lite` descriptor does not contain the PHP buildpack.

The `lite` descriptor only contains the `base` stack. The `default` stack is installed, but is identical to the `base` stack. For more information, see [Stacks](#).

Full descriptor

The Tanzu Build Service `full` descriptor contains more dependencies, which allows for more workload types.

The dependencies are pre-packaged so builds don't have to download them from the Internet. This can speed up build times.

The `full` descriptor contains the following stacks, which support different use cases:

- `base`
- `default` (identical to `base`)
- `full`
- `tiny`

For more information, see [Stacks](#). Due to the larger footprint of `full`, installations might take longer.

Descriptor comparison

Both `lite` and `full` descriptors are suitable for production environments.

| | <code>lite</code> | <code>full</code> |
|------------------------------|-------------------|-------------------|
| Faster installation time | Yes | No |
| Dependencies pre-packaged | 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 |

| | lite | full |
|---------------------------|------|------|
| Supports static workloads | Yes | Yes |
| Supports binary workloads | Yes | Yes |

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 describes 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 [About Tanzu Application Platform components and 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.com -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-rolebinding'
/ 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 PipelineRuns in your cluster, skip this step because namespace configuration is covered in [Set up developer namespaces to use 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>
```

Note: The service account has access to the `pull-secret` image pull secret.

For more details about Tekton Pipelines, see the [Tekton documentation](#) and the [GitHub repository](#).

For information about getting started with Tekton, see the Tekton [tutorial](#) in GitHub and the [getting started guide](#) in the Tekton documentation.

Note: Windows workloads are deactivated and cause an error if any Tasks try to use Windows scripts.

Workload types

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

The [web](#) workload type allows you to deploy web applications on Tanzu Application Platform. Using an application workload specification, you can turn source code into a scalable, stateless application that runs in a container with an automatically-assigned URL.

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.run/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.containers):
    - #@ 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.yml: #@ yaml.encode(services())
---
apiVersion: carto.run/v1alpha1
kind: ClusterConfigTemplate
metadata:
  name: apply-bindings
spec:
  configPath: .data
  ytt: |
    #@ load("@ytt:data", "data")
    #@ load("@ytt:yml", "yml")
    #@ 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 hasattr(data.values.workload.metadata.annotations, claims_extension_key):
    #@     return None
    #@   end
    #@
    #@   extension = json.decode(data.values.workload.metadata.annotations[claims_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", "deletecollection"]
- apiGroups: ["apps"]
  resources: ["deployments"]
  verbs: ["get", "list", "watch", "create", "patch", "update", "delete", "deletecollection"]
```

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: git-ssh
  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
  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
  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
  templateRef:
    kind: ClusterTemplate
    name: config-writer-template
selector:
  apps.tanzu.vmware.com/workload-type: tcp

```

Where:

- ◆ `REGISTRY-SERVER` is the registry server from your `tap-values.yaml` file.
- ◆ `REGISTRY-REPO` is the registry repository from 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.values.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", "deletecollection"]
- apiGroups: ["apps"]
  resources: ["deployments"]
  verbs: ["get", "list", "watch", "create", "patch", "update", "delete", "deletecollection"]

```

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: git-ssh
      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
      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
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
templateRef:
  kind: ClusterTemplate
  name: config-writer-template
selector:
  apps.tanzu.vmware.com/workload-type: tcp

```

Where:

- ◆ `REGISTRY-SERVER` is the registry server from your `tap-values.yaml` file.
- ◆ `REGISTRY-REPO` is the registry repository from 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-sensor` 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-sensor` to use the `queue` supply chain by changing the workload type by running:

```
tanzu apps workload update spring-sensors-sensor --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 intent.
- The workload no longer autoscales based on request traffic. For the `spring-sensors-sensor` workload, this means that it does not scale down to zero instances when there is no request traffic.

Functions (Beta)

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.

In this section:

- [Using functions](#)
- [Iterate on your function](#)

Using functions (Beta)

This topic describes how to create and deploy an HTTP function from an application accelerator starter template.

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 descriptor you used in the `buildservice` section of your `tap-values.yaml` file:

- For the **full descriptor**, 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 descriptor**, 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
```

- 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.

- 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

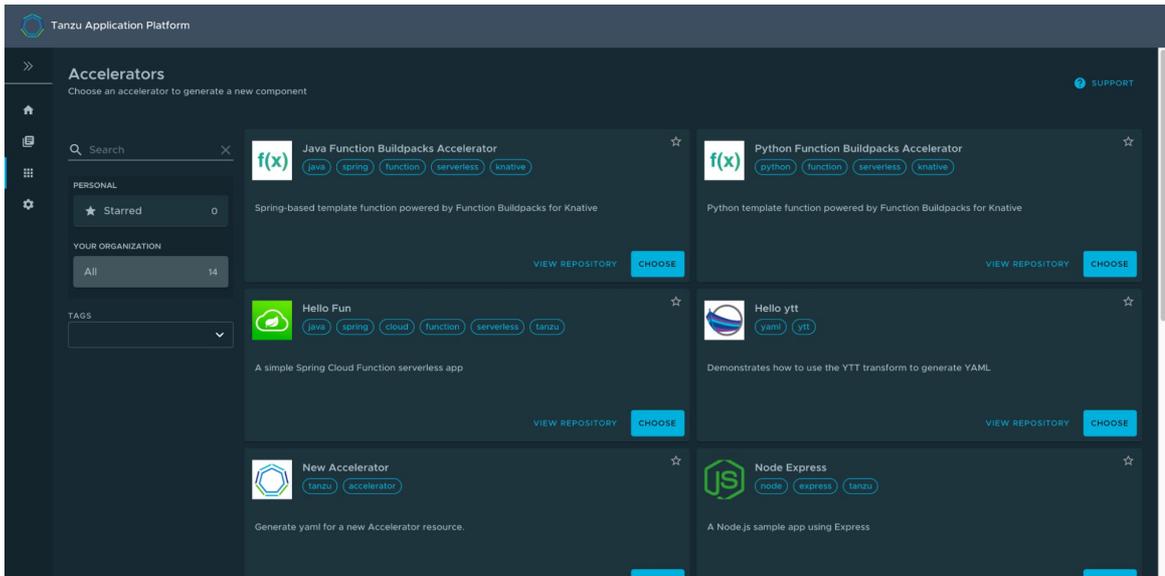
Application Accelerator is a component of Tanzu Application Platform. An accelerator contains your enterprise-conformant code and configurations that developers can use to create new projects that automatically follow the standards defined in your accelerators.

The accelerator ZIP file contains a file called `k8s-resource.yaml`. This file contains the resource manifest for the function accelerator.

- Download the ZIP file for the appropriate accelerator:
 - ◆ [Python HTTP Function](#) on GitHub.
 - ◆ [Java HTTP Function](#) on GitHub.
- Expand the accelerator ZIP file in your target cluster with Tanzu Application Platform GUI installed.
- 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
```

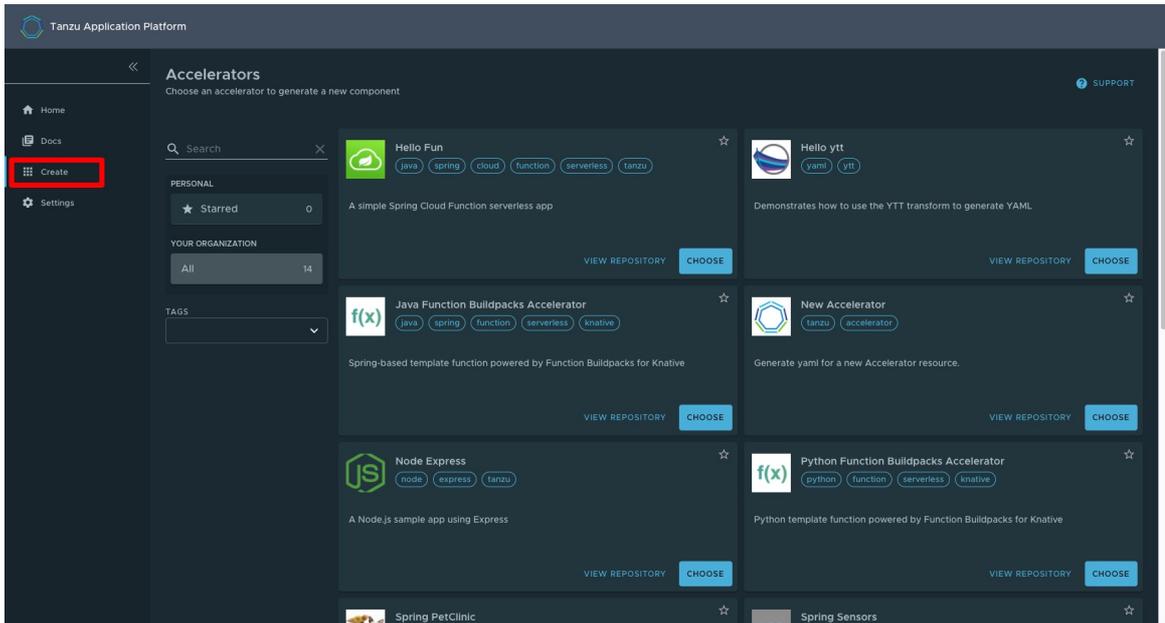
- Refresh Tanzu Application Platform GUI to reveal function accelerator(s).
-



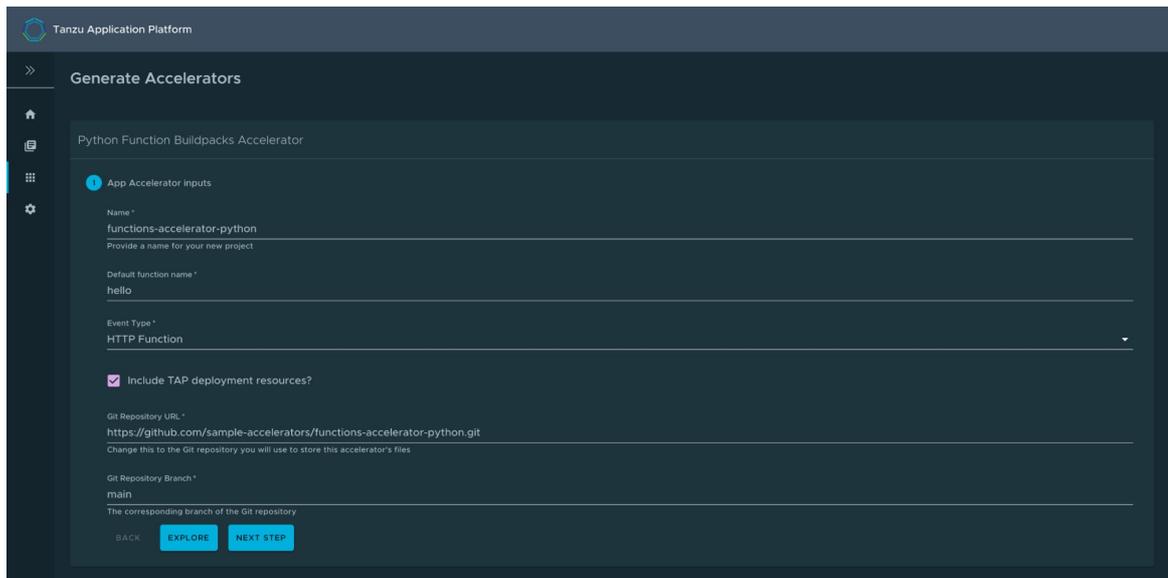
It might take time for Tanzu Application Platform GUI to refresh the catalog to see your added function accelerators.

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.



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**.



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
```

Iterating on your function

This topic provides instructions about how to iterate on your function using the VMware Tanzu

Developer Tools extension for Visual Studio Code. This extension enables live updates of your application while running on the cluster, and allows you to debug your application directly on the cluster.

Prerequisites

Before you can iterate on your function, you must have:

- [Tanzu Developer Tools for Visual Studio Code](#)
- [Tilt](#) v0.27.2 or later.

Note: The Tanzu Developer Tools extension currently only supports Java Functions.

Configure the Tanzu Developer Tools extension

Before iterating on your application, you must configure the Tanzu Developer Tools extension as follows:

1. Open your function as a project within your VSCode IDE.
2. To ensure your extension assists you with iterating on the correct project, configure its settings as follows:
 1. In Visual Studio Code, navigate to Preferences > Settings > Extensions > Tanzu.
 2. In the Local Path field, provide the path to the directory containing your function project. 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,


```
index.docker.io/myteam/java-function.
```

You are now ready to iterate on your application.

Live update your application

Deploy your function application to view it updating live on the cluster. This demonstrates how code changes will behave on a production cluster early in the development process.

To live update your application:

1. Open the Command Palette by pressing `⇧⌘P`.
2. From the Command Palette, 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.
3. Depending on the type of cluster you use, you might see an error message 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.

If you see this error, add the line `allow_k8s_contexts('CLUSTER-NAME')` to your Tiltfile, where `CLUSTER-NAME` is the name of your cluster.

4. When the Live Update status in the status bar is visible and says `Live Update Started`, navigate to `http://localhost:8080` in your browser and view your running application.
5. Enter the IDE and make a change to the source code.
6. The container is updated when the logs stop streaming. Navigate to your browser and refresh the page.
7. View the changes to your workload running on the cluster.
8. If necessary, continue making changes to the source code.
9. When you have finished making changes, stop and deactivate the live update. To do so, open the command palette by pressing `⇧⌘P`, type `Tanzu`, and select **Tanzu: Live Update Stop**.

Debug your application

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

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. You will see the Deploy and Connect task complete and the debug menu actions available to you, indicating that the debugger has attached.

3. Navigate to `http://localhost:8080` in your browser. This hits the breakpoint within VSCode. Play to the end of the debug session using VSCode debugging controls.