

# Tanzu Application Platform v1.4

VMware Tanzu Application Platform 1.4

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

<https://docs.vmware.com/>

**VMware by Broadcom**

3401 Hillview Ave.  
Palo Alto, CA 94304  
[www.vmware.com](http://www.vmware.com)

Copyright © 2024 Broadcom. All Rights Reserved. The term “Broadcom” refers to Broadcom Inc. and/or its subsidiaries. For more information, go to <https://www.broadcom.com>. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies. [Copyright and trademark information](#).

# Contents

<b>Tanzu Application Platform v1.4</b>	<b>108</b>
Tanzu Application Platform overview	108
Simplified workflows	108
Notice of telemetry collection for Tanzu Application Platform	110
<b>Tanzu Application Platform release notes</b>	<b>111</b>
v1.4.13	111
v1.4.13 Security fixes	111
v1.4.13 Known issues	111
v1.4.12	111
v1.4.12 Security fixes	111
v1.4.12 Known issues	115
v1.4.11	115
v1.4.11 Security fixes	116
v1.4.11 Known issues	116
v1.4.10	116
v1.4.10 Security fixes	116
v1.4.10 Known issues	117
v1.4.9	117
v1.4.9 Security fixes	117
v1.4.9 Resolved issues	119
v1.4.9 Resolved issues: Tanzu CLI and plug-ins	119
v1.4.9 Known issues	119
v1.4.9 Known issues: Tanzu Build Service	119
v1.4.8	119
v1.4.8 Security fixes	119
v1.4.8 Resolved issues	120
v1.4.8 Resolved issues: Tanzu Build Service	121
v1.4.8 Known issues	121
v1.4.8 Known issues: Tanzu Build Service	121
v1.4.7	121
v1.4.7 Security fixes	121
v1.4.7 Known issues	121
v1.4.6	122
v1.4.6 Security fixes	122
v1.4.6 Resolved issues	122
v1.4.6 Resolved issues: Tanzu Developer Tools for IntelliJ	122

v1.4.6 Resolved issues: Tanzu Developer Tools for Visual Studio	122
v1.4.6 Resolved issues: Tanzu Developer Tools for VS Code	122
v1.4.6 Known issues	122
v1.4.5	122
v1.4.5 Security fixes	123
v1.4.4	123
v1.4.4 Security fixes	123
v1.4.4 Resolved issues	124
v1.4.4 Resolved issues: Grype Scanner	124
v1.4.4 Resolved issues: Source Controller	124
v1.4.4 Known issues	125
v1.4.4 Known issues: API Auto Registration	125
v1.4.4 Known issues: Grype Scanner	125
v1.4.2	125
v1.4.2 Security fixes	125
v1.4.2 Resolved issues	126
v1.4.2 Resolved issues: Tanzu Build Service	126
v1.4.2 Known issues	126
v1.4.2 Known issues: Grype scanner	126
v1.4.1	127
v1.4.1 Security fixes	127
v1.4.1 Security fixes: Tanzu Application Platform GUI	127
v1.4.1 Resolved issues	127
v1.4.1 Resolved issues: Source Controller	127
v1.4.1 Resolved issues: Tanzu Application Platform GUI	127
v1.4.1 Resolved issues: Tanzu Application Platform plug-ins	127
API Validation and Scoring Toolkit plug-in	127
Application Accelerator plug-in	127
Application Live View plug-in	127
Out of the Box Supply Chain Templates plug-in	128
Security Analysis plug-in	128
Supply Chain Choreographer plug-in	128
Supply Chain Security Tools plug-in	128
v1.4.1 Known issues	128
v1.4.1 Known issues: Grype scanner	128
v1.4.1 Known issues: Security Analysis GUI	129
v1.4.0	129
v1.4.0 Tanzu Application Platform new features	129
v1.4.0 New features by component and area	129
v1.4.0 Features: API Validation and Scoring Toolkit	129
v1.4.0 Features: Application Single Sign-On (AppSSO)	129
v1.4.0 Features: Application Accelerator	130



v1.4.0 Features: Application Live View	130
v1.4.0 Features: Apps plug-in for Tanzu CLI	131
v1.4.0 Features: cert-manager	131
v1.4.0 Features: Eventing	131
v1.4.0 Features: External Secrets Operator (alpha)	131
v1.4.0 Features: Services Toolkit	131
v1.4.0 Features: Tanzu Application Platform GUI plug-ins	132
Security Analysis Plug-in	132
Supply Chain Choreographer plug-in	132
v1.4.0 Features: Supply Chain Security Tools - Policy	132
v1.4.0 Features: Supply Chain Security Tools - Scan	132
v1.4.0 Features: Tanzu Developer Tools for IntelliJ	133
v1.4.0 Features: Tanzu Developer Tools for Visual Studio	133
v1.4.0 Features: Tanzu Developer Tools for Visual Studio Code	133
v1.4.0 Breaking changes	133
v1.4.0 Breaking changes: Application Single Sign-On (AppSSO)	133
v1.4.0 Breaking changes: Out of the Box Supply Chain Templates	133
v1.4.0 Breaking changes: Supply Chain Security Tools - Image Policy Webhook	134
v1.4.0 Breaking changes: Supply Chain Security Tools - Policy Controller	134
v1.4.0 Breaking changes: Tanzu Application Platform GUI	134
v1.4.0 Breaking changes: Tanzu Developer Tools for IntelliJ	134
v1.4.0 Breaking changes: Tanzu Developer Tools for Visual Studio Code	134
v1.4.0 Security fixes	134
v1.4.0 Security fixes: API Auto Registration	134
v1.4.0 Security fixes: Contour	134
v1.4.0 Security fixes: Supply Chain Security Tools - Grype	134
v1.4.0 Security fixes: Remediated vulnerabilities	135
Note about CVE-2022-4378	135
v1.4.0 Resolved issues	135
v1.4.0 Resolved issues: API Auto Registration	135
v1.4.0 Resolved issues: Application Single Sign-On (AppSSO)	135
v1.4.0 Resolved issues: Out of the Box Supply Chain Templates	136
v1.4.0 Resolved issues: Tanzu CLI Apps Plug-in	136
v1.4.0 Resolved issues: Tanzu Application Platform GUI plug-ins	136
v1.4.0 Resolved issues: Supply Chain Choreographer plug-in	136
v1.4.0 Known issues	136
v1.4.0 Known issues: API Auto Registration	136
v1.4.0 Known issues: Application Accelerator for Visual Studio Code	136
v1.4.0 Known issues: Cloud Native Runtimes for VMware Tanzu	136
v1.4.0 Known issues: Grype scanner	137
v1.4.0 Known issues: Namespace Provisioner	137
v1.4.0 Known issues: Out of the Box Supply Chain Templates	137

v1.4.0 Known issues: Tanzu Application Platform GUI plug-ins	137
Security Analysis plug-in	137
Supply Chain Choreographer plug-in	137
v1.4.0 Known issues: Tanzu Developer Tools for IntelliJ	138
v1.4.0 Known issues: Tanzu Developer Tools for Visual Studio	138
v1.4.0 Known issues: Tanzu Developer Tools for Visual Studio Code	138
Deprecations	138
Application Live View deprecations	138
Application Single Sign-On (AppSSO) deprecations	138
Services Toolkit deprecations	139
Supply Chain Security Tools - Image Policy Webhook deprecations	139
Supply Chain Security Tools - Scan deprecations	139
Supply Chain Security Tools - Sign deprecations	139
Tanzu Build Service deprecations	139
Tanzu CLI Apps plug-in deprecations	139
Linux Kernel CVEs	139
<b>Components and installation profiles for Tanzu Application Platform</b>	<b>141</b>
Tanzu Application Platform components	141
Installation profiles in Tanzu Application Platform v1.4	145
Packages: A to C	145
Packages: D to R	146
Packages: S to Z	146
Language and framework support in Tanzu Application Platform	147
Installing Tanzu Application Platform	147
<b>Install Tanzu Application Platform</b>	<b>148</b>
<b>Install Tanzu Application Platform</b>	<b>148</b>
<b>Prerequisites for installing Tanzu Application Platform</b>	<b>148</b>
VMware Tanzu Network and container image registry requirements	148
DNS Records	149
Tanzu Application Platform GUI	149
Kubernetes cluster requirements	150
Resource requirements	151
Tools and CLI requirements	152
Next steps	152
<b>Kubernetes version support for Tanzu Application Platform</b>	<b>152</b>
<b>Install Tanzu CLI</b>	<b>152</b>
Accept the End User License Agreements	153
Example of accepting the Tanzu Application Platform EULA	153

Set the Kubernetes cluster context	155
Install or update the Tanzu CLI and plug-ins	156
Install the Tanzu CLI	156
Install Tanzu CLI Plug-ins	158
List the versions of each plug-in group available across Tanzu	159
List the versions of the Tanzu Application Platform specific plug-in group	159
Install the version of the Tanzu Application Platform plug-in group matching your target environment	159
Verify the plug-in group list against the plug-ins that were installed	159
<b>Install Tanzu Application Platform (online)</b>	<b>159</b>
<b>Install Tanzu Application Platform (online)</b>	<b>160</b>
<b>Install Tanzu Application Platform package and profiles</b>	<b>160</b>
Relocate images to a registry	161
Add the Tanzu Application Platform package repository	162
Install your Tanzu Application Platform profile	165
Full profile	165
CEIP policy disclosure	168
(Optional) Additional Build Service configurations	168
(Optional) Configure your profile with full dependencies	168
(Optional) Configure your profile with the Jammy stack only	169
Install your Tanzu Application Platform package	169
Install the full dependencies package	170
Access Tanzu Application Platform GUI	170
Exclude packages from a Tanzu Application Platform profile	171
Next steps	171
<b>View possible configuration settings for your package</b>	<b>171</b>
<b>Install individual packages</b>	<b>173</b>
Install pages for individual Tanzu Application Platform packages	173
Verify the installed packages	174
Next steps	174
<b>Set up developer namespaces to use your installed packages</b>	<b>175</b>
Additional configuration for testing and scanning	175
Legacy namespace setup	175
Next steps	175
<b>Legacy manual developer namespace setup instructions</b>	<b>175</b>
Enable single user access	175
Enable additional users with Kubernetes RBAC	177

Additional configuration for testing and scanning	179
<b>Install Tanzu Developer Tools for your VS Code</b>	<b>179</b>
Prerequisites	179
Install	179
Configure	180
Uninstall	180
Next steps	181
<b>Install Tanzu Application Platform (offline)</b>	<b>181</b>
<b>Install Tanzu Application Platform (offline)</b>	<b>181</b>
<b>Install Tanzu Application Platform in your air-gapped environment</b>	<b>182</b>
Relocate images to a registry	182
Prepare Sigstore Stack for air-gapped policy controller	186
Install your Tanzu Application Platform profile	186
Full Profile	187
Install your Tanzu Application Platform package	191
Next steps	191
<b>Install the Tanzu Build Service dependencies</b>	<b>191</b>
Next steps	192
<b>Configure custom certificate authorities for Tanzu Application Platform GUI</b>	<b>192</b>
Next steps	193
<b>Configure Application Accelerator</b>	<b>193</b>
Using a Git-Ops style configuration for deploying a set of managed accelerators	194
Functional and Organizational Considerations	194
Examples for creating accelerators	194
A minimal example for creating an accelerator	194
An example for creating an accelerator with customized properties	195
Creating a manifest with multiple accelerators and fragments	196
Configure tap-values.yaml with Git credentials secret	197
Using non-public repositories	198
Examples for a private Git repository	198
Example using http credentials	198
Example using http credentials with self-signed certificate	199
Example using SSH credentials	200
Examples for a private source-image repository	201
Example using image-pull credentials	201
Configure ingress timeouts when some accelerators take longer to generate	202

Configure an ingress timeout overlay secret for each HTTPProxy	202
Apply the timeout overlay secrets in tap-values.yaml	203
Configuring skipping TLS verification for access to Source Controller	203
Enabling TLS for Accelerator Server	203
Configuring skipping TLS verification of Engine calls for Accelerator Server	204
Enabling TLS for Accelerator Engine	204
Next steps	205
<b>Use Grype in offline and air-gapped environments</b>	<b>205</b>
To enable Grype in offline air-gapped environments	205
Troubleshooting	205
ERROR failed to fetch latest cli version	205
Solution	206
Database is too old	206
Solution	207
Grype package overlays are not applied to scantemplates created by Namespace Provisioner	208
Solution	208
Debug Grype database in a cluster	209
<b>Set up developer namespaces to use your installed packages</b>	<b>210</b>
Additional configuration for testing and scanning	210
Legacy namespace setup	210
Next steps	210
<b>Install Tanzu Application Platform (AWS)</b>	<b>210</b>
<b>Install Tanzu Application Platform (AWS)</b>	<b>211</b>
<b>Create AWS Resources for Tanzu Application Platform</b>	<b>211</b>
Prerequisites	212
Export environment variables	212
Create an EKS cluster	212
Install EBS CSI driver	213
Create the container repositories	213
Create the workload container repositories	213
Create IAM roles	214
<b>Install Tanzu Application Platform package and profiles on AWS</b>	<b>218</b>
Relocate images to a registry	218
Install your Tanzu Application Platform profile	221
Full profile (AWS)	222
(Optional) Configure your profile with full dependencies	224
Install your Tanzu Application Platform package	224

Install the full dependencies package	225
Access Tanzu Application Platform GUI	225
Exclude packages from a Tanzu Application Platform profile	226
Next steps	226
<b>View possible configuration settings for your package</b>	<b>226</b>
<b>Install individual packages</b>	<b>228</b>
Install pages for individual Tanzu Application Platform packages	228
Verify the installed packages	229
Next steps	229
<b>Set up developer namespaces to use your installed packages</b>	<b>230</b>
Enable single user access	230
Enable additional users access with Kubernetes RBAC	231
Next steps	233
<b>Install Tanzu Developer Tools for your VS Code</b>	<b>233</b>
Prerequisites	233
Install	233
Configure	234
Uninstall	234
Next steps	234
<b>Install Tanzu Application Platform (OpenShift)</b>	<b>234</b>
<b>Install Tanzu Application Platform (OpenShift)</b>	<b>235</b>
<b>Install Tanzu Application Platform on your OpenShift clusters</b>	<b>235</b>
Relocate images to a registry	236
Install your Tanzu Application Platform profile	239
Full profile	240
(Optional) Additional Build Service configurations	242
(Optional) Configure your profile with full dependencies	242
(Optional) Configure your profile with the Jammy stack only	243
Security Context Constraints	243
(Optional) Exclude components that require RedHat OpenShift privileged SCC	243
Install your Tanzu Application Platform package	243
Install the full dependencies package	244
Access Tanzu Application Platform GUI	245
Exclude packages from a Tanzu Application Platform profile	245
<b>View possible configuration settings for your package</b>	<b>245</b>
<b>Install individual packages</b>	<b>247</b>

Install pages for individual Tanzu Application Platform packages	247
Verify the installed packages	248
Next steps	249
<b>Set up developer namespaces to use your installed packages</b>	<b>249</b>
Additional configuration for testing and scanning	249
Legacy namespace setup	249
Next steps	249
<b>Install Tanzu Developer Tools for your VS Code</b>	<b>249</b>
Prerequisites	249
Install	250
Configure	250
Uninstall	251
Next steps	251
<b>Custom Security Context Constraint details for Tanzu Application Platform</b>	<b>251</b>
Application Accelerator on OpenShift	251
Application Live View on OpenShift	252
Application Single Sign-On for OpenShift cluster	253
Contour for OpenShift cluster	254
Developer Conventions for OpenShift cluster	255
Tanzu Build Service for OpenShift cluster	256
<b>Customize your package installation</b>	<b>258</b>
Customize a package that was manually installed	258
Customize a package that was installed by using a profile	258
<b>Upgrade your Tanzu Application Platform</b>	<b>259</b>
Prerequisites	259
Update the new package repository	259
Perform the upgrade of Tanzu Application Platform	260
Upgrade instructions for Profile-based installation	260
Upgrade the full dependencies package	261
Multicluster upgrade order	261
Upgrade instructions for component-specific installation	262
Verify the upgrade	262
<b>Opt out of telemetry collection</b>	<b>263</b>
<b>Overview of security and compliance in Tanzu Application Platform</b>	<b>265</b>
<b>Overview of security and compliance in Tanzu Application Platform</b>	<b>265</b>

Overview of security and compliance in Tanzu Application Platform	265
Secure Ingress certificates in Tanzu Application Platform	265
Replacing the default ingress issuer	266
Deactivating TLS for ingress	266
Overriding TLS for components	266
Use custom CA certificates in Tanzu Application Platform	266
Use External Secrets Operator in Tanzu Application Platform (alpha)	267
Installing the External Secrets Operator	267
Using the External Secrets Operator	267
Connecting to a secret manager	267
Example : Google Secret Manager	267
Create a sychronized secret	269
Using a sychronized secret	269
Assess Tanzu Application Platform against the NIST 800-53 Moderate Assessment	269
Overview of multicluster Tanzu Application Platform	277
Next steps	277
Overview of multicluster Tanzu Application Platform	278
Next steps	278
Install multicluster Tanzu Application Platform profiles	278
Prerequisites	278
Multicluster Installation Order of Operations	279
Install View cluster	279
Install Build clusters	279
Install Run clusters	279
Install Iterate clusters	280
Add Build, Run and Iterate clusters to Tanzu Application Platform GUI	280
Next steps	280
Get started with multicluster Tanzu Application Platform	280
Prerequisites	280
Start the workload on the Build profile cluster	281
Install Tanzu Application Platform Build profile	283
Prerequisites	283
Example values.yaml	283
Install Tanzu Application Platform Run profile	285



Install Tanzu Application Platform View profile	286
Install Tanzu Application Platform Iterate profile	288
Get started with Tanzu Application Platform	291
Prerequisites	291
Next steps	291
Get started with Tanzu Application Platform	291
Prerequisites	292
Next steps	292
Create an accelerator	292
What you will do	293
Set up Visual Studio Code	293
Create a simple project	293
Set up the project directory	293
Prepare the README.md and accelerator.yaml	293
Test the accelerator	295
Upload the project to a Git repository	296
Register the accelerator to the Tanzu Application Platform and verify project generation output	296
Verify project generation output by using Tanzu Application Platform GUI	296
Learn more about Application Accelerator	298
Create an accelerator	299
What you will do	299
Set up Visual Studio Code	299
Create a simple project	299
Set up the project directory	300
Prepare the README.md and accelerator.yaml	300
Test the accelerator	301
Upload the project to a Git repository	302
Register the accelerator to the Tanzu Application Platform and verify project generation output	302
Verify project generation output by using Tanzu Application Platform GUI	303
Learn more about Application Accelerator	304
Add testing and scanning to your application	305
What you will do	305
Overview	305
Install OOTB Supply Chain with Testing	305
Tekton pipeline config example	306
Workload update	307

Install OOTB Supply Chain with Testing and Scanning	308
Prerequisites	308
Workload update	310
Query for vulnerabilities	311
Next steps	311
<b>Configure image signing and verification in your supply chain</b>	<b>312</b>
What you will do	312
Configure your supply chain to sign and verify your image builds	312
Next steps	313
<b>Set up services for consumption by developers</b>	<b>313</b>
What you will do	314
Overview	314
Prerequisites	315
Set up a service	315
Create a service instance	317
Claim a service instance	318
Further use cases and reading	319
Next steps	319
<b>Deploy an app on Tanzu Application Platform</b>	<b>320</b>
What you will do	320
Generate a new project using an application accelerator	320
Deploy your application through Tanzu Application Platform GUI	322
Add your application to Tanzu Application Platform GUI software catalog	323
Next steps	324
<b>Deploy an app on Tanzu Application Platform</b>	<b>324</b>
What you will do	325
Generate a new project using an application accelerator	325
Deploy your application through Tanzu Application Platform GUI	327
Add your application to Tanzu Application Platform GUI software catalog	328
Next steps	329
<b>Iterate on your new app</b>	<b>329</b>
What you will do	329
Prepare your IDE to iterate on your application	330
Live update your application	330
Debug your application	331
Monitor your running application	331
Next steps	332
<b>Consume services on Tanzu Application Platform</b>	<b>332</b>

What you will do	332
Overview	332
Prerequisites	333
Bind an application workload to the service instance	334
Further use cases and reading	335
Next steps	336
<b>Deploy an air-gapped workload on Tanzu Application Platform</b>	<b>336</b>
What you will do	336
Create a workload from Git	336
Create a basic supply chain workload	337
Create a testing supply chain workload	338
Create a testing scanning supply chain workload	338
<b>Learn about Tanzu Application Platform</b>	<b>339</b>
<b>Application accelerators on Tanzu Application Platform</b>	<b>339</b>
What are application accelerators	339
Working with accelerators	339
Next steps	340
<b>Supply chains on Tanzu Application Platform</b>	<b>340</b>
What are supply chains	340
A path to production	340
Available supply chains	340
1: OOTB Basic (default)	340
2: OOTB Testing	341
3: OOTB Testing+Scanning	341
Next steps	342
<b>Vulnerability scanning, storing, and viewing for your supply chain</b>	<b>342</b>
Features	342
Components	343
Next steps	343
Troubleshooting	343
<b>About consuming services on Tanzu Application Platform</b>	<b>343</b>
Key concepts	343
Service instances	343
Service bindings	344
Resource claims	344
Services you can use with Tanzu Application Platform	344
User roles and responsibilities	345
Next steps	345

<b>Set up Tanzu Service Mesh</b>	<b>346</b>
Prerequisites	346
Activate your Tanzu Service Mesh subscription	346
Onboard clusters	346
Set up Tanzu Application Platform	347
End-to-end workload build and deployment scenario	348
Apply a workload resource to a build cluster	348
Configure egress for Tanzu Build Service	349
Create a global namespace	349
Run cluster deployment	349
Deployment use case: Hungryman	350
Create an initial set of configuration files from the accelerator	350
Apply the workload resources to your build cluster	351
Install service claim resources on the cluster	351
Run cluster deployment	352
Create a global namespace	353
Deployment use case: ACME Fitness Store	354
Deploy AppSSO	354
Apply the workload resources to your build cluster	355
Create the Istio ingress resources	355
Deploy Redis	356
Run cluster deployment	356
Deploy Spring Cloud Gateway	357
Install the Spring Cloud Gateway package	357
Create a global namespace	357
<b>Set up Tanzu Service Mesh</b>	<b>358</b>
Prerequisites	358
Activate your Tanzu Service Mesh subscription	359
Onboard clusters	359
Set up Tanzu Application Platform	360
End-to-end workload build and deployment scenario	360
Apply a workload resource to a build cluster	360
Configure egress for Tanzu Build Service	362
Create a global namespace	362
Run cluster deployment	362
Deployment use case: Hungryman	362
Create an initial set of configuration files from the accelerator	363
Apply the workload resources to your build cluster	363
Install service claim resources on the cluster	363
Run cluster deployment	364
Create a global namespace	365

Deployment use case: ACME Fitness Store	366
Deploy AppSSO	366
Apply the workload resources to your build cluster	367
Create the Istio ingress resources	368
Deploy Redis	368
Run cluster deployment	368
Deploy Spring Cloud Gateway	369
Install the Spring Cloud Gateway package	369
Create a global namespace	370
<b>Overview of workloads</b>	<b>371</b>
Workload features	371
Available workload types	371
<b>Overview of workloads</b>	<b>372</b>
Workload features	372
Available workload types	372
<b>Use web workloads</b>	<b>373</b>
Overview	373
Use the web workload type	374
Calling web workloads within a cluster	374
Example of service to service communication for web and server workloads	374
<b>Use server workloads</b>	<b>375</b>
Overview	375
Use the server workload type	375
server-specific workload parameters	376
Expose server workloads outside the cluster	376
Manual configuration for HTTP workloads	377
Define a workload type that exposes server workloads outside the cluster	377
<b>Use worker workloads</b>	<b>381</b>
Overview	381
Use the worker workload type	382
<b>Parameter reference</b>	<b>382</b>
Workload Parameter Reference	382
List of Supply Chain Resources for Workload Object	382
source-provider	383
GitRepository	383
ImageRepository	384
MavenArtifact	384
source-tester	385

source-scanner	386
image-provider	386
Kpack Image	386
Runnable (TaskRuns for Dockerfile-based builds)	387
Pre-built image (ImageRepository)	388
image-scanner	388
config-provider	389
app-config	390
service-bindings	390
api-descriptors	391
config-writer (git or registry)	391
deliverable	392
Deliverable Parameters Reference	392
List of Cluster Delivery Resources for Deliverable Object	392
source-provider	393
GitRepository	393
ImageRepository	393
app deployer	394
App	394
<b>Use functions (Beta)</b>	<b>394</b>
Overview	395
Supported languages and frameworks	395
Prerequisites	395
Create a function project from an accelerator	396
Create a function project using the Tanzu CLI	397
Deploy your function	397
<b>Use functions (Beta)</b>	<b>398</b>
Overview	399
Supported languages and frameworks	399
Prerequisites	399
Create a function project from an accelerator	400
Create a function project using the Tanzu CLI	401
Deploy your function	401
<b>Troubleshoot Tanzu Application Platform</b>	<b>403</b>
<b>Troubleshoot Tanzu Application Platform</b>	<b>403</b>
<b>Troubleshoot installing Tanzu Application Platform</b>	<b>403</b>
Developer cannot be verified when installing Tanzu CLI on macOS	403
Access .status.usefulErrorMessage details	404

“Unauthorized to access” error	404
“Serviceaccounts already exists” error	405
After package installation, one or more packages fails to reconcile	405
Failure to accept an End User License Agreement error	409
Ingress is broken on Kind cluster	409
<b>Troubleshoot using Tanzu Application Platform</b>	<b>409</b>
Use events to find possible causes	409
Missing build logs after creating a workload	409
Explanation	410
Solution	410
Workload creation stops responding with “Builder default is not ready” message	410
Explanation	410
Solution	410
“Workload already exists” error after updating the workload	410
Explanation	411
Solution	411
Workload creation fails due to authentication failure in Docker Registry	411
Explanation	411
Solution	411
Telemetry component logs show errors fetching the “reg-creds” secret	412
Explanation	412
Solution	412
Debug convention might not apply	412
Explanation	412
Solution	412
Execute bit not set for App Accelerator build scripts	412
Explanation	412
Solution	412
“No live information for pod with ID” error	413
Explanation	413
Solution	413
“image-policy-webhook-service not found” error	413
Explanation	413
Solution	413
“Increase your cluster resources” error	413
Explanation	413
Solution	413
MutatingWebhookConfiguration prevents pod admission	414
Explanation	414
Solution	414
Priority class of webhook’s pods preempts less privileged pods	415

Explanation	415
Solution	415
CrashLoopBackOff from password authentication fails	415
Explanation	416
Solution	416
Password authentication fails	416
Explanation	416
Solution	416
metadata-store-db pod fails to start	417
Explanation	417
Solution	417
Missing persistent volume	417
Explanation	417
Solution	417
Failure to connect Tanzu CLI to AWS EKS clusters	418
Explanation	418
Solution	418
Invalid repository paths are propagated	418
Explanation	418
Solution	418
x509: certificate signed by unknown authority	418
Explanation	418
Solution	419
Option 1: Configure the Shared Ingress Issuer's Certificate Authority as a trusted Certificate Authority	419
Option 2: Deactivate the shared ingress issuer	419
Troubleshoot Tanzu Application Platform components	420
Uninstall Tanzu Application Platform	421
Delete the packages	421
Delete the Tanzu Application Platform package repository	422
Remove Tanzu CLI, plug-ins, and associated files	422
Remove Cluster Essentials	422
Component documentation for Tanzu Application Platform	423
Component documentation for Tanzu Application Platform	423
Overview of Tanzu CLI	423
Tanzu CLI	423
Tanzu CLI Architecture	423
Tanzu CLI Installation	423
Tanzu CLI Command Groups	424



Install New Plug-ins	424
Install Local Plug-ins	424
<b>Overview of Tanzu CLI</b>	<b>425</b>
Tanzu CLI	425
Tanzu CLI Architecture	425
Tanzu CLI Installation	425
Tanzu CLI Command Groups	426
Install New Plug-ins	426
Install Local Plug-ins	426
<b>Overview of Tanzu CLI plug-ins</b>	<b>427</b>
<b>Tanzu Apps CLI overview</b>	<b>427</b>
About workloads	427
Tutorials	428
How-to-guides	428
<b>Tanzu Apps CLI overview</b>	<b>428</b>
About workloads	428
Tutorials	428
How-to-guides	428
<b>Install Tanzu Apps CLI plug-in</b>	<b>428</b>
Prerequisites	428
Install Tanzu Apps CLI plug-in	428
Uninstall Apps CLI plug-in	429
Change clusters	429
Override the default kubeconfig	429
Autocompletion	429
Bash	430
Zsh	430
Create workloads	430
Debug and troubleshoot workloads	430
<b>Create a workload</b>	<b>430</b>
Prerequisites	430
Get started with an example workload	431
Create a workload from GitHub repository	431
Create a workload from local source code	431
Exclude Files	431
Create workload from an existing image	432
Create a workload from Maven repository artifact	432
Working with YAML files	432

Bind a service to a workload	433
Next steps	434
<b>Workload Examples</b>	<b>434</b>
Custom registry credentials	434
–live-update and –debug	435
Spring Boot application example	435
–export	436
–output	437
–sub-path	440
.tanzuignore file	441
Example of a .tanzuignore file	441
–dry-run	441
–update-strategy	442
<b>Debug workloads</b>	<b>444</b>
Verify build logs	444
Check build logs	444
Get the workload status and details	444
Common workload errors	445
Local Path Development Error Cases	445
WorkloadLabelsMissing/SupplyChainNotFound	445
MissingValueAtPath	445
TemplateRejectedByAPIServer	446
Review supply chain steps	446
Additional Troubleshooting References	447
<b>Tanzu Apps CLI commands</b>	<b>447</b>
<b>Tanzu Apps CLI commands</b>	<b>447</b>
<b>tanzu apps cluster-supply-chain</b>	<b>447</b>
Tanzu apps cluster supply chain list	448
Default view	448
Tanzu apps cluster supply chain get	448
Default view	448
<b>tanzu apps workload apply</b>	<b>449</b>
Default view	449
Workload Apply flags	450
--annotation	450
--app / -a	450
--build-env	451
--debug	452

--dry-run	452
--env / -e	453
--file, -f	454
--git-repo	454
--git-branch	454
--git-tag	455
--git-commit	455
--image / -i	455
--label / -l	456
--limit-cpu	457
--limit-memory	457
--live-update	458
--local-path	459
--maven-artifact	459
--maven-group	460
--maven-type	460
--maven-version	460
--source-image, -s	460
--namespace, -n	460
--param / -p	461
--param-yaml	462
--registry-ca-cert	462
--registry-password	463
--registry-token	463
--registry-username	463
--request-cpu	463
--request-memory	464
--service-account	464
--service-ref	465
--sub-path	466
--tail	467
--tail-timestamp	467
--type / -t	468
--update-strategy	469
--wait	470
--wait-timeout	470
--yes, -y	471
<b>tanzu apps workload delete</b>	<b>471</b>
Default view	471
Workload Delete flags	471
--all	472

--file, -f	472
--namespace, -n	472
wait	472
--wait-timeout	472
--yes, -f	473
<b>tanzu apps workload get</b>	<b>473</b>
Default view	473
--export	474
--output/-o	475
--namespace/-n	477
<b>tanzu apps workload list</b>	<b>478</b>
Default view	478
>Workload List flags	478
--all-namespaces, -A	478
--app	479
--namespace, -n	479
--output, -o	479
<b>tanzu apps workload tail</b>	<b>481</b>
Default view	481
>Workload Tail flags	481
--component	482
--namespace, -n	482
--since	483
--timestamp, -t	484
<b>Tanzu Accelerator CLI overview</b>	<b>484</b>
Server API connections for operators and developers	485
Installation	485
Command reference	485
<b>Tanzu Accelerator CLI overview</b>	<b>485</b>
Server API connections for operators and developers	485
Installation	486
Command reference	486
<b>Install Tanzu Accelerator CLI</b>	<b>486</b>
Prerequisites	486
Install	487
<b>Command reference</b>	<b>487</b>
<b>Command reference</b>	<b>488</b>

<b>tanzu accelerator</b>	<b>488</b>
Options	488
SEE ALSO	488
<b>tanzu accelerator</b>	<b>489</b>
Options	489
SEE ALSO	489
<b>tanzu accelerator apply</b>	<b>489</b>
tanzu accelerator apply	489
Synopsis	489
Examples	489
Options	489
Options inherited from parent commands	490
SEE ALSO	490
<b>tanzu accelerator create</b>	<b>490</b>
Synopsis	490
Examples	490
Options	490
Options inherited from parent commands	490
SEE ALSO	491
<b>tanzu accelerator delete</b>	<b>491</b>
Synopsis	491
Examples	491
Options	491
Options inherited from parent commands	491
SEE ALSO	491
<b>tanzu accelerator fragment</b>	<b>491</b>
Synopsis	491
Examples	491
Options	492
Options inherited from parent commands	492
SEE ALSO	492
<b>tanzu accelerator fragment create</b>	<b>492</b>
Synopsis	492
Example	492
Options	492
Options inherited from parent commands	493
SEE ALSO	493

<b>tanzu accelerator fragment delete</b>	<b>493</b>
tanzu accelerator fragment delete	493
Synopsis	493
Examples	493
Options	493
Options inherited from parent commands	493
SEE ALSO	493
<b>tanzu accelerator fragment get</b>	<b>493</b>
Synopsis	493
Examples	494
Options	494
Options inherited from parent commands	494
SEE ALSO	494
<b>tanzu accelerator fragment list</b>	<b>494</b>
Synopsis	494
Examples	494
Options	494
Options inherited from parent commands	494
SEE ALSO	495
<b>tanzu accelerator fragment update</b>	<b>495</b>
Synopsis	495
Examples	495
Options	495
Options inherited from parent commands	495
SEE ALSO	495
<b>tanzu accelerator generate</b>	<b>495</b>
tanzu accelerator generate	495
Synopsis	496
Examples	496
Options	496
Options inherited from parent commands	496
SEE ALSO	496
<b>Tanzu accelerator generate-from-local</b>	<b>496</b>
Synopsis	496
Examples	497
Options	497
Options inherited from parent commands	497
<b>tanzu accelerator get</b>	<b>497</b>

Synopsis	498
Examples	498
Options	498
Options inherited from parent commands	498
SEE ALSO	498
<b>tanzu accelerator list</b>	<b>498</b>
Synopsis	498
Examples	498
Options	499
Options inherited from parent commands	499
SEE ALSO	499
<b>tanzu accelerator push</b>	<b>499</b>
tanzu accelerator push	499
Synopsis	499
Examples	499
Options	499
Options inherited from parent commands	499
SEE ALSO	499
<b>tanzu accelerator update</b>	<b>500</b>
Synopsis	500
Examples	500
Options	500
Options inherited from parent commands	500
SEE ALSO	500
<b>Overview of Tanzu Insight plug-in</b>	<b>500</b>
<b>Overview of Tanzu Insight plug-in</b>	<b>501</b>
<b>Install your Tanzu Insight CLI plug-in</b>	<b>501</b>
<b>Configure your Tanzu Insight CLI plug-in</b>	<b>501</b>
Set the target and certificate authority (CA) certificate	502
<b>Single Cluster setup</b>	<b>502</b>
<b>Set Target</b>	<b>502</b>
Set the access token	502
Verify the connection	503
<b>Query vulnerabilities, images, and packages</b>	<b>503</b>
Supported use cases	503

Query using the Tanzu Insight CLI plug-in	503
Example #1: What packages and CVEs does a specific image contain?	503
Example #2: What packages and CVEs does my source code contain?	504
Determining source code org, repo, and commit SHA	504
Source code query with repo and org	505
Source code query with commit SHA	505
Example #3: What dependencies are affected by a specific CVE?	505
Add data	506
<b>Add data to your Supply Chain Security Tools - Store</b>	<b>506</b>
Supported formats and file types	506
Generate a CycloneDX file	506
Add data with the Tanzu Insight plug-in	507
Example #1: Add an image report	507
Example #2: Add a source report	507
<b>Tanzu insight CLI plug-in command reference</b>	<b>508</b>
Synopsis	508
Options	508
See also	508
<b>tanzu insight config set-target</b>	<b>508</b>
tanzu insight config set-target	508
Synopsis	508
Examples	509
Options	509
See also	509
<b>tanzu insight config</b>	<b>509</b>
Options	509
See also	509
<b>tanzu insight health</b>	<b>509</b>
tanzu insight health	509
Synopsis	509
Examples	509
Options	509
See also	510
<b>tanzu insight image</b>	<b>510</b>
Options	510
See also	510
<b>tanzu insight image add</b>	<b>510</b>



Examples	510
Options	510
See also	510
<b>tanzu insight image get</b>	<b>510</b>
Synopsis	511
Examples	511
Options	511
See Also	511
<b>tanzu insight image packages</b>	<b>511</b>
Synopsis	511
Examples	511
Options	511
See also	511
<b>tanzu insight image vulnerabilities</b>	<b>511</b>
Examples	512
Options	512
See also	512
<b>tanzu insight package</b>	<b>512</b>
Options	512
See also	512
<b>tanzu insight package get</b>	<b>512</b>
Synopsis	512
Examples	513
Options	513
See also	513
<b>tanzu insight package images</b>	<b>513</b>
Synopsis	513
Examples	513
Options	513
See also	513
<b>tanzu insight package sources</b>	<b>513</b>
Synopsis	513
Examples	514
Options	514
See also	514
<b>tanzu insight package vulnerabilities</b>	<b>514</b>
Synopsis	514

Examples	514
Options	514
See also	514
<b>tanzu insight source</b>	<b>514</b>
Options	514
See also	515
<b>tanzu insight source add</b>	<b>515</b>
Examples	515
Options	515
See also	515
<b>tanzu insight source get</b>	<b>515</b>
Synopsis	515
Examples	515
Options	516
See also	516
<b>tanzu insight source packages</b>	<b>516</b>
Synopsis	516
Examples	516
Options	516
See also	516
<b>tanzu insight source vulnerabilities</b>	<b>516</b>
Synopsis	516
Examples	517
Options	517
See also	517
<b>tanzu insight version</b>	<b>517</b>
Options	517
See also	517
<b>tanzu insight vulnerabilities</b>	<b>517</b>
Options	517
See also	517
<b>tanzu insight vulnerabilities get</b>	<b>518</b>
Synopsis	518
Examples	518
Options	518
See also	518

<b>tanzu insight vulnerabilities images</b>	<b>518</b>
Synopsis	518
Examples	518
Options	518
See also	519
<b>tanzu insight vulnerabilities packages</b>	<b>519</b>
Synopsis	519
Examples	519
Options	519
See also	519
<b>tanzu insight vulnerabilities sources</b>	<b>519</b>
Synopsis	519
Examples	519
Options	519
See also	520
<b>Overview of Default roles for Tanzu Application Platform</b>	<b>520</b>
Default roles	520
Working with roles using the RBAC CLI plug-in	520
Disclaimer	520
<b>Overview of Default roles for Tanzu Application Platform</b>	<b>520</b>
Default roles	521
Working with roles using the RBAC CLI plug-in	521
Disclaimer	521
<b>Set up authentication for your Tanzu Application Platform deployment</b>	<b>521</b>
Tanzu Kubernetes Grid	522
<b>Set up authentication for your Tanzu Application Platform deployment</b>	<b>522</b>
Tanzu Kubernetes Grid	522
<b>Install Pinniped on Tanzu Application Platform</b>	<b>522</b>
Prerequisites	522
Environment planning	523
Install Pinniped Supervisor by using Let's Encrypt	524
Create Certificates (letsencrypt or cert-manager)	524
Create Ingress resources	525
Create the pinniped-supervisor configuration	525
Apply the resources	526
Switch to production issuer (letsencrypt or cert-manager)	527
Install Pinniped Supervisor Private CA	528

Create Certificate Secret	528
Create Ingress resources	528
Create the pinniped-supervisor configuration	529
Apply the resources	530
Install Pinniped Concierge	531
Log in to the cluster	532
<b>Integrate your Azure Active Directory</b>	<b>532</b>
Integrate Azure AD with a new or existing AKS without Pinniped	532
Prerequisites	532
Set up a platform operator	532
Set up a Tanzu Application Platform default role group	533
Set up kubeconfig	533
Integrate Azure AD with Pinniped	533
Prerequisites	534
Set up the Azure AD app	534
Set up the Tanzu Application Platform default role group	535
Set up kubeconfig	535
<b>Role descriptions for Tanzu Application Platform</b>	<b>536</b>
app-editor	536
app-viewer	536
app-operator	536
service-operator	537
workload	537
deliverable	537
<b>Role descriptions for Tanzu Application Platform</b>	<b>537</b>
app-editor	537
app-viewer	537
app-operator	537
service-operator	538
workload	538
deliverable	538
<b>Detailed role permissions for Tanzu Application Platform</b>	<b>538</b>
Native Kubernetes Resources	538
apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"	538
apps.tanzu.vmware.com/aggregate-to-app-operator: "true"	539
App Accelerator	539
apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"	539
apps.tanzu.vmware.com/aggregate-to-app-operator: "true"	539
Cartographer	539

apps.tanzu.vmware.com/aggregate-to-app-editor: "true"	539
apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"	539
apps.tanzu.vmware.com/aggregate-to-app-viewer-cluster-access: "true"	539
apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access	539
Cloud Native Runtimes	540
apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"	540
apps.tanzu.vmware.com/aggregate-to-app-operator: "true"	540
Convention Service	540
apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"	540
apps.tanzu.vmware.com/aggregate-to-app-viewer-cluster-access: "true"	540
apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access	540
Developer Conventions	541
apps.tanzu.vmware.com/aggregate-to-app-editor: "true"	541
OOTB Templates	541
apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"	541
apps.tanzu.vmware.com/aggregate-to-workload: "true"	542
apps.tanzu.vmware.com/aggregate-to-deliverable: "true"	542
Service Bindings	542
apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"	542
Services Toolkit	542
apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"	543
apps.tanzu.vmware.com/aggregate-to-app-viewer-cluster-access: "true"	543
apps.tanzu.vmware.com/aggregate-to-app-operator: "true"	543
apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access	543
Source Controller	543
apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"	543
Supply Chain Security Tools — Scan	543
apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"	543
apps.tanzu.vmware.com/aggregate-to-app-operator: "true"	543
Tanzu Build Service	543
apps.tanzu.vmware.com/aggregate-to-app-editor: "true"	543
apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"	544
apps.tanzu.vmware.com/aggregate-to-app-viewer-cluster-access: "true"	544
apps.tanzu.vmware.com/aggregate-to-app-operator: "true"	544
apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access	544
Tekton	544
apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"	544
apps.tanzu.vmware.com/aggregate-to-app-viewer-cluster-access: "true"	544
apps.tanzu.vmware.com/aggregate-to-app-operator: "true"	544
apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access	544
<b>Bind a user or group to a default role</b>	<b>545</b>

Prerequisites	545
Install the Tanzu Application Platform RBAC CLI plug-in	545
(Optional) Use a different kubeconfig location	545
Add the specified user or group to a role	546
Get a list of users and groups from a role	546
Remove the specified user or group from a role	546
Error logs	546
Troubleshooting	548
<b>Log in to Tanzu Application Platform by using Pinniped</b>	<b>548</b>
Download the Pinniped CLI	548
Generate and distribute kubeconfig to users	549
Login with the provided kubeconfig	549
<b>Additional resources about Tanzu Application Platform authentication and authorization</b>	<b>549</b>
Install	549
<b>Additional resources about Tanzu Application Platform authentication and authorization</b>	<b>550</b>
Install	550
<b>Install default roles independently for your Tanzu Application Platform</b>	<b>550</b>
Prerequisites	550
Install	550
<b>Overview of API Auto Registration</b>	<b>551</b>
Overview	551
Getting started	551
<b>Overview of API Auto Registration</b>	<b>551</b>
Overview	552
Getting started	552
<b>Key Concepts for API Auto Registration</b>	<b>552</b>
API Auto Registration Architecture	552
APIDescriptor Custom Resource Explained	552
With an Absolute URL	553
With an Object Ref	554
With an HTTPProxy Object Ref	554
With a Knative Service Object Ref	554
With an Ingress Object Ref	554
APIDescriptor Status Fields	555
<b>Install API Auto Registration</b>	<b>555</b>

Tanzu Application Platform prerequisites	555
Using with TLS	555
Install	555
<b>Use API Auto Registration</b>	<b>558</b>
Generate OpenAPI Spec	558
Using a Spring Boot app with a REST service	558
Using App Accelerator Template	559
Using an existing Spring Boot project using springdoc	559
Create APIDescriptor Custom Resource	559
Use Out-Of-The-Box (OOTB) supply chains	559
Using Custom Supply Chains	560
Using other GitOps processes or Manually	561
Additional configuration	561
Setting up CORS for OpenAPI specifications	561
<b>Troubleshoot API Auto Registration</b>	<b>561</b>
Debug API Auto Registration	561
APIDescriptor CRD shows message of connection refused but service is up and running	562
Configure CA Cert Data	562
APIDescriptor CRD shows message of x509: certificate signed by unknown authority but service is running	563
<b>Overview of API Validation and Scoring</b>	<b>563</b>
<b>Overview of API Validation and Scoring</b>	<b>563</b>
<b>Install API Validation and Scoring</b>	<b>563</b>
Prerequisites	564
Resource requirements	564
Relocate images to a registry	564
Add the API Validation and Scoring package repository	565
Install	566
Uninstall	567
<b>Use API Validation and Scoring to score your auto-registered API</b>	<b>567</b>
Use API Validation and Scoring to score your auto-registered API	567
<b>Overview of API portal for VMware Tanzu</b>	<b>568</b>
Getting started	568
<b>Overview of API portal for VMware Tanzu</b>	<b>568</b>
Getting started	569
<b>Install API portal for VMware Tanzu</b>	<b>569</b>

Prerequisites	569
Install	569
Update the installation values for the api-portal package	570
<b>Overview of Application Accelerator</b>	<b>571</b>
Overview	571
Architecture	571
How does Application Accelerator work?	572
Next steps	572
<b>Overview of Application Accelerator</b>	<b>572</b>
Overview	572
Architecture	573
How does Application Accelerator work?	573
Next steps	574
<b>Install Application Accelerator</b>	<b>574</b>
Prerequisites	574
Configure properties and resource use	574
Install	575
Troubleshooting	577
Verify installed packages	577
Look at resource events	577
<b>Configure Application Accelerator</b>	<b>578</b>
Overview	578
Using a Git-Ops style configuration for deploying a set of managed accelerators	578
Functional and Organizational Considerations	579
Examples for creating accelerators	579
A minimal example for creating an accelerator	579
An example for creating an accelerator with customized properties	580
Creating a manifest with multiple accelerators and fragments	580
Configure tap-values.yaml with Git credentials secret	581
Using non-public repositories	582
Examples for a private Git repository	582
Example using http credentials	582
Example using http credentials with self-signed certificate	583
Example using SSH credentials	584
Examples for a private source-image repository	585
Example using image-pull credentials	586
Configure ingress timeouts when some accelerators take longer to generate	586
Configure an ingress timeout overlay secret for each HTTPProxy	587
Apply the timeout overlay secrets in tap-values.yaml	587



Configuring skipping TLS verification for access to Source Controller	588
Enabling TLS for Accelerator Server	588
Configuring skipping TLS verification of Engine calls for Accelerator Server	588
Enabling TLS for Accelerator Engine	589
Next steps	589
<b>Create accelerators</b>	<b>589</b>
Prerequisites	589
Getting started	590
Publishing the new accelerator	590
Using local-path for publishing accelerators	591
Using accelerator fragments	592
Deploying accelerator fragments	593
Next steps	594
<b>Create accelerators</b>	<b>594</b>
Prerequisites	594
Getting started	595
Publishing the new accelerator	595
Using local-path for publishing accelerators	596
Using accelerator fragments	597
Deploying accelerator fragments	598
Next steps	599
<b>Create an accelerator.yaml file in Application Accelerator</b>	<b>599</b>
Accelerator	599
Accelerator metadata	600
Accelerator options	600
DependsOn and multi-value dataType	601
Examples	602
Engine	604
Engine example	604
Engine notation descriptions	605
Advanced accelerator use	605
<b>Use transforms in Application Accelerator</b>	<b>605</b>
Why transforms?	606
Combining transforms	606
Chain	606
Merge	607
Shortened notation	608
A Combo of one?	609
A common pattern with merge transforms	609

Conditional transforms	610
Conditional 'Merge' transform	611
Conditional 'Chain' transform	611
A small gotcha with using conditionals in merge transforms	611
Merge conflict	612
Resolving "merge" conflicts	613
File ordering	613
Next steps	614
<b>Use custom types in Application Accelerator</b>	<b>614</b>
Limitations	616
Interaction with SpEL	617
Interaction with Composition	617
<b>Use fragments in Application Accelerator</b>	<b>617</b>
Introduction	617
Introducing fragments	617
The imports section explained	618
Using the InvokeFragment Transform	619
Back to the imports section	619
Using dependsOn in the imports section	620
Discovering fragments using Tanzu CLI accelerator plug-in	621
<b>Transforms reference</b>	<b>624</b>
Available transforms	624
See also	625
<b>Transforms reference</b>	<b>625</b>
Available transforms	625
See also	625
<b>Combo transform</b>	<b>625</b>
Syntax reference	625
Behavior	626
Examples	627
<b>Include transform</b>	<b>628</b>
Syntax reference	628
Examples	628
See also	628
<b>Exclude transform</b>	<b>628</b>
Syntax reference	628
Examples	629

See also	629
<b>Merge transform</b>	<b>629</b>
Syntax reference	629
See also	629
<b>Chain transform</b>	<b>629</b>
Syntax reference	629
Behavior	630
<b>Let transform</b>	<b>630</b>
Syntax reference	630
Execution	630
See also	631
<b>Loop transform</b>	<b>631</b>
Syntax reference	631
Behavior	631
Examples	631
<b>InvokeFragment transform</b>	<b>632</b>
Syntax reference	633
Behavior	633
Variables	633
</a/>Files	633
Examples	634
See also	635
<b>ReplaceText transform</b>	<b>635</b>
Syntax reference	635
Examples	636
See also	637
<b>RewritePath transform</b>	<b>637</b>
Syntax reference	637
Examples	637
Interaction with Chain and Include	638
See also	638
<b>OpenRewriteRecipe transform</b>	<b>638</b>
Syntax reference	639
Example	639
<b>YTT transform</b>	<b>639</b>
Syntax reference	639

Execution	639
Examples	640
Basic invocation	640
Using extraArgs	640
<b>UseEncoding transform</b>	<b>641</b>
Syntax reference	641
Example use	641
See also	641
<b>UniquePath transform</b>	<b>641</b>
Syntax reference	641
Examples	642
See also	642
<b>Conflict resolution</b>	<b>642</b>
Syntax reference	642
Available strategies	642
See also	643
<b>Use SpEL with Application Accelerator</b>	<b>643</b>
Variables	643
Implicit variables	643
Conditionals	644
Rewrite path concatenation	644
Regular expressions	644
Dealing with string arrays	645
<b>Accelerator custom resource definition</b>	<b>645</b>
<b>The Accelerator custom resource definition (CRD) defines any accelerator resources to be made</b>	<b>645</b>
API definitions	645
Accelerator CRD Spec	646
Fragment CRD Spec	647
Excluding files	648
<b>Test accelerators in Application Accelerator</b>	<b>648</b>
Generating a project from local sources	648
CI/CD Pipeline	649
(Optional) Getting the Tanzu CLI in a CI/CD pipeline	649
<b>Use the Application Accelerator Visual Studio Code extension</b>	<b>650</b>
Dependencies	650
Installation	650

Configure the extension	651
Using the extension	651
Retrieving the URL for the Tanzu Application Platform GUI	652
Known Issues	653
<b>Application Accelerator best practices</b>	<b>653</b>
<b>Best practices for using accelerators</b>	<b>653</b>
Benefits of using an accelerator	653
Design considerations	653
Housekeeping rules	654
Tests	654
Application skeleton	654
<b>Best practices for using fragments</b>	<b>654</b>
Benefits of using Fragment	655
Design considerations	655
Housekeeping rules	655
<b>Troubleshoot Application Accelerator</b>	<b>656</b>
Development issues	656
Failure to generate a new project	656
URI is not absolute error	656
Accelerator authorship issues	657
General tips	657
Speed up the reconciliation of the accelerator	657
Use a source image with local accelerator source directory	657
Expression evaluation errors	657
Operations issues	658
Check status of accelerator resources	658
When Accelerator ready column is blank	658
When Accelerator ready column is false	659
REASON: GitRepositoryResolutionFailed	659
REASON: GitRepositoryResolutionPending	660
REASON: ImageRepositoryResolutionPending	661
<b>Overview of Application Live View</b>	<b>662</b>
Value proposition	662
Intended audience	662
Supported application platforms	662
Multicloud compatibility	662
Deployment	662
<b>Overview of Application Live View</b>	<b>662</b>

Value proposition	663
Intended audience	663
Supported application platforms	663
Multicloud compatibility	663
Deployment	663
<b>Install Application Live View</b>	<b>663</b>
Overview	663
Prerequisites	664
Install Application Live View	664
Install Application Live View back end	664
Install Application Live View connector	667
Install Application Live View conventions	669
Deprecate the sslDisabled key	671
<b>Enabling Spring Boot apps for Application Live View</b>	<b>671</b>
Enable Spring Boot apps	671
Enable Spring Cloud Gateway apps	671
Workload image NOT built with Tanzu Build Service	672
<b>Enabling Spring Boot apps for Application Live View</b>	<b>672</b>
Enable Spring Boot apps	672
Enable Spring Cloud Gateway apps	673
Workload image NOT built with Tanzu Build Service	673
<b>Enable Steeltoe apps for Application Live View</b>	<b>674</b>
Extend .NET Core Apps to Steeltoe Apps	674
Enable Application Live View on Steeltoe Tanzu Application Platform workload	675
<b>Application Live View convention server</b>	<b>675</b>
Role of Application Live View convention	675
Description of metadata labels	676
Verify the applied labels and annotations	676
<b>Custom configuration for the connector</b>	<b>678</b>
Configure the developer workload in Tanzu Application Platform	679
Deploy the workload	679
Verify the label has propagated through the Supply Chain	679
<b>Custom configuration for application actuator endpoints</b>	<b>681</b>
<b>Scaling Knative apps in Tanzu Application Platform</b>	<b>684</b>
Configure the developer workload in Tanzu Application Platform	684
Deploy the workload	685
Verify the annotation has propagated through the Supply Chain	685

Application Live View on OpenShift	686
Support for polyglot apps with Application Live View	687
Application Live View internal architecture	687
Component overview	687
Design flow	688
Troubleshoot Application Live View	688
App is not visible in Application Live View UI	689
App is not visible in Application Live View UI with actuator endpoints enabled	689
The UI does not show any information for an app with actuator endpoints exposed at root	690
No information shown on the Health page	690
Stale information in Application Live View	690
Unable to find CertificateRequests in Application Live View convention	691
No live information for pod with ID	691
Cannot override the actuator path in the labels	691
Cannot configure SSL in appliveview-connector	691
Verify the labels in your workload YAML file	692
Override labels set by the Application Live View convention service	692
Configure labels when management.endpoints.web.base-path and management.server.port are set	693
Uninstall Application Live View	693
Overview of Application Single Sign-On for VMware Tanzu® 3.0.0	693
Overview of Application Single Sign-On for VMware Tanzu® 3.0.0	694
Get started with Application Single Sign-On	694
Prerequisites	694
Key concepts	694
Next steps	695
Get started with Application Single Sign-On	695
Prerequisites	696
Key concepts	696
Next steps	697
Provision an AuthServer	697
Prerequisites	697
Provision an AuthServer	697
The AuthServer spec in detail	699
Metadata	699
TLS & issuer URI	699

Token Signature	700
Identity providers	700
Configuring storage	701
Provision a client registration	701
Prerequisites	701
Creating the ClientRegistration	701
Validating that the credentials are working	702
Deploy an application with Application Single Sign-On	703
Prerequisites	703
Deploy a minimal application	703
Deployment manifest	706
OAuth2-Proxy	706
Application Single Sign-On for Platform Operators	706
Application Single Sign-On for Platform Operators	706
Install Application Single Sign-On	707
What's inside	707
Prerequisites	707
Installation	707
Configure Application Single Sign-On	707
TAP values	707
domain_name	708
domain_template	708
default_authserver_clusterissuer	708
ca_cert_data	708
kubernetes_distribution	709
Configuration schema	709
RBAC for Application Single Sign-On	710
Application Single Sign-On for OpenShift clusters	712
Upgrade Application Single Sign-On	713
Migration guides	713
v2.0.0 to v3.0.0	713
v1.0.0 to v2.0.0	714
Uninstall Application Single Sign-On	715
Application Single Sign-On for Service Operators	715



Application Single Sign-On for Service Operators	715
Annotations and labels for AppSSO	716
Labels	716
Allowing client namespaces	716
Unsafe configuration	717
Unsafe identity provider	717
Unsafe issuer URI	717
Issuer URI and TLS for AppSSO	718
Overview	718
Configure TLS by using a (Cluster)Issuer	719
Configure TLS by using a Certificate	719
Configure TLS by using a Secret	720
Deactivate TLS (unsafe)	721
Allow Workloads to trust a custom CA AuthServer	721
TLS scenario guides for AppSSO	722
Overview	722
Prerequisites	722
Using a default issuer	722
Using a ClusterIssuer	723
Using an Issuer	725
Using an existing Certificate	726
Using an existing TLS certificate	728
Using an existing wildcard TLS certificate	731
CA certificates for AppSSO	734
Configure workloads to trust a custom CA	735
Overview	735
Exporting custom CA certificate Secret	736
Importing custom CA certificate Secret	736
Appending custom CA certificate Secret reference to Workload	737
Identity providers for AppSSO	737
OpenID Connect providers	738
Note for registering a client with the identity provider	739
Supported token signing algorithms	739
LDAP	739
ActiveDirectory group search	740
“Classic” group search	741
Direct group search only	742
Groups in sub-trees	743

Nested group search	744
SAML (experimental)	745
Note for registering a client with the identity provider	745
Internal users	745
Generating a bcrypt hash from a plain-text password	746
Restrictions	746
<b>Token signatures for AppSSO</b>	<b>747</b>
Overview	747
Token signature 101	747
Token signature of an AuthServer	747
Creating keys	748
Using secretgen-controller	749
Using OpenSSL	750
Rotating keys	751
Revoking keys	751
References and further reading	752
<b>Token settings for Application Single Sign-On</b>	<b>752</b>
Token expiry	752
Constraints	753
Verify token settings	753
<b>Storage for AppSSO</b>	<b>757</b>
Overview	757
Securing Data at rest	757
Configuring Redis	757
Configuring Redis Server CA certificate	758
Configuring a Redis Secret	758
Attaching storage to an AuthServer	759
Inspecting storage of an AuthServer	759
Storage provided by default	759
Data types	760
Known limitations of storage providers	760
Redis Cluster	760
<b>AuthServer readiness for AppSSO</b>	<b>761</b>
Client registration check	761
Prerequisites	761
Define and apply a test client	761
Get an access token	762
<b>Scale AuthServer for AppSSO</b>	<b>762</b>

<b>AuthServer audit logs for AppSSO</b>	<b>762</b>
Overview	762
Authentication	763
Token flows	763
<b>Application Single Sign-On for App Operators</b>	<b>764</b>
<b>Application Single Sign-On for App Operators</b>	<b>764</b>
<b>Register a workload</b>	<b>765</b>
Topics	765
Client registration	765
Workloads	765
Prerequisites	765
Configuring a Workload with AppSSO	765
Create and apply a ClientRegistration resource	766
Add a service resource claim to your Workload	766
<b>Configure grant types</b>	<b>768</b>
Topics	768
Client Credentials Grant Type	768
Authorization Code Grant Type	769
<b>Secure a workload</b>	<b>771</b>
Prerequisites	771
Getting started	771
Understanding the sample application	771
The sample application's ClientRegistration	772
Understanding Workloads	773
Deploying the sample application as a Workload	773
Create workload namespace	773
Apply required TAP workload configurations	773
Apply the ClientRegistration	773
Create a ClientRegistration service resource claim for the workload	774
(Optional) Ensure Workload trusts AuthServer	774
Deploy the workload	774
Cleaning up	775
<b>ClientRegistration API for AppSSO</b>	<b>775</b>
Spec	776
Status & conditions	776
Example	778
Configuring public clients	778

<b>ClientRegistration API for AppSSO</b>	<b>779</b>
Spec	779
Status & conditions	780
Example	781
Configuring public clients	782
<b>AuthServer API for AppSSO</b>	<b>782</b>
Spec	783
Status & conditions	785
RBAC	787
Example	787
<b>Troubleshoot Application Single Sign-on</b>	<b>789</b>
Why is my AuthServer not working?	789
Find all AuthServer-related Kubernetes resources	789
Logs of all AuthServers	789
Change propagation	789
My Service is not selecting the authorization server's Deployment	789
Redirect URIs are redirecting to http instead of https with a non-internal identity provider	789
Misconfigured clientSecret	789
Problem:	789
Solution:	790
Misconfigured redirect URI	790
Problem:	790
Solution:	790
Unsupported id_token_signed_response_alg with openid identityProviders	790
Problem:	790
Solution:	790
Misconfigured identity provider clientSecret	790
Problem:	790
Solution:	790
Missing scopes	791
Problem:	791
Solution:	791
Misconfigured sub claim	791
Problem:	791
Solution:	791
<b>Known Issues</b>	<b>791</b>
Unregistration by deletion	791
Limited number of ClientRegistrations per AuthServer	791
LetsEncrypt: domain name for Issuer URI limited to 64 characters maximum	791
Redirect URIs change to http instead of https	792

AuthServer only supports response_type=code	792
<b>Overview of Convention Service for VMware Tanzu</b>	<b>792</b>
Sample conventions	793
<b>Overview of Cartographer Conventions</b>	<b>794</b>
Overview	794
About applying conventions	794
Applying conventions by using image metadata	794
Applying conventions without using image metadata	795
<b>Overview of Cartographer Conventions</b>	<b>795</b>
Overview	795
About applying conventions	796
Applying conventions by using image metadata	796
Applying conventions without using image metadata	796
<b>Install Cartographer Conventions</b>	<b>796</b>
<b>Create conventions with Cartographer Conventions</b>	<b>796</b>
Introduction	796
Convention server	797
How the convention server works	797
Convention controller	798
How the convention controller works	798
Getting started	799
Prerequisites	799
Define convention criteria	799
Define the convention behavior	802
Matching criteria by labels or annotations	802
Matching criteria by environment variables	803
Matching criteria by image metadata	803
Configure and install the convention server	804
Deploy a convention server	806
Next Steps	809
<b>Troubleshoot Cartographer Conventions</b>	<b>809</b>
No server in the cluster	809
Symptoms	809
Cause	809
Solution	809
Server with wrong certificates configured	809
Symptoms	809
Cause	810

Solution	810
Server fails when processing a request	810
Symptoms	810
Cause	811
Solution	811
Connection refused due to unsecured connection	811
Symptoms	811
Cause	812
Solution	812
Self-signed certificate authority (CA) not propagated to the Convention Service	813
Symptoms	813
Cause	813
Solution	813
No imagePullSecrets configured	813
Symptoms	813
Cause	813
Solution	813
<b>Convention Service Resources for Cartographer Conventions</b>	<b>813</b>
Overview	813
API Structure	814
Template Status	814
Chaining Multiple Conventions	814
Collecting Logs from the Controller	814
References	814
<b>Convention Service Resources for Cartographer Conventions</b>	<b>815</b>
Overview	815
API Structure	815
Template Status	815
Chaining Multiple Conventions	815
Collecting Logs from the Controller	815
References	816
<b>ImageConfig for Cartographer Conventions</b>	<b>816</b>
Overview	816
<b>PodConventionContextSpec for Cartographer Conventions</b>	<b>817</b>
Overview	817
<b>PodConventionContextStatus for Cartographer Conventions</b>	<b>818</b>
Overview	818
<b>PodConventionContext for Cartographer Conventions</b>	<b>819</b>

Overview	819
PodConventionContext Objects	819
PodConventionContext Structure	820
ClusterPodConvention for Cartographer Conventions	820
Overview	820
Define conventions	820
PodIntent for Cartographer Conventions	820
Overview	821
BOM for Cartographer Conventions	821
Overview	821
Structure	821
Overview of cert-manager	822
Overview of cert-manager	822
Install cert-manager	822
Overview of Cloud Native Runtimes	825
Overview of Cloud Native Runtimes	825
Install Cloud Native Runtimes	825
Prerequisites	826
Install	826
Overview of Contour	829
Overview of Contour	829
Install Contour	829
Configure Cipher Suites and TLS version in Contour	833
Configure Contour	834
Smaller Clusters	834
Larger Clusters	835
Configuring Envoy as a Deployment	835
Overview of Eventing	835
Overview of Eventing	835
Install Eventing	835
Prerequisites	835

Install	835
<b>Namespace Provisioner</b>	<b>837</b>
Description	837
Component Overview	837
Provisioner Carvel Application	838
Desired Namespaces ConfigMap	838
Example	839
Namespace Provisioner Controller	839
Default Resources Secret	839
Expansion Template ConfigMap	840
<b>Install Namespace Provisioner</b>	<b>840</b>
Install using a Profile	840
Customized Installation	840
<b>Provision namespace resources</b>	<b>841</b>
Using Namespace Provisioner Controller	841
Prerequisites	841
Provision a new developer namespace	842
Using GitOps	843
Prerequisites	843
<b>Customize Namespace Provisioner</b>	<b>843</b>
Data values templating guide	843
GitOps Customizations	844
Extending the default provisioned resources	844
Add the resources required by the Out of the Box Testing and Scanning Supply Chain	845
Customizing the default resources that get provisioned	846
Control the Namespace Provisioner reconcile behavior for specific resources	847
Control the desired-namespaces ConfigMap with GitOps	847
Prerequisites	847
<b>Troubleshoot Namespace Provisioner</b>	<b>848</b>
Controller logs	848
Provisioner application error	848
Common errors	849
Namespace selector malformed	849
Carvel-YTT error in additional_sources	849
Unable to delete namespace	850
<b>Namespace Provisioner Reference Guide</b>	<b>850</b>
Known Limitations	851
Default resources mapping	851



Overview of Spring Boot conventions	852
Overview of Spring Boot conventions	852
Install Spring Boot conventions	853
Prerequisites	853
Install Spring Boot conventions	853
Configure and access Spring Boot actuators in Tanzu Application Platform	854
Workload-level configuration	855
Platform-level configuration	856
Enable Application Live View for Spring Boot applications	856
Verify the applied labels and annotations	857
List of Spring Boot conventions	861
Set a JAVA_TOOL_OPTIONS property for a workload	861
Spring Boot convention	862
Spring boot graceful shut down convention	863
Spring Boot web convention	864
Spring Boot Actuator convention	865
Spring Boot Actuator Probes convention	866
Service intent conventions	867
Example	868
Troubleshoot Spring Boot conventions	869
Collect logs	869
Overview of Service Bindings	870
Supported service binding specifications	870
Overview of Service Bindings	870
Supported service binding specifications	870
Install Service Bindings	871
Prerequisites	871
Install Service Bindings	871
Troubleshoot Service Bindings	872
Collect logs	872
Service Bindings resource specification	874
Overview of Services Toolkit	874
Overview of Services Toolkit	875

<b>Install Services Toolkit</b>	<b>875</b>
Prerequisites	875
Install Services Toolkit	875
<b>Overview of Flux CD Source Controller</b>	<b>876</b>
<b>Overview of Flux CD Source Controller</b>	<b>876</b>
<b>Install Flux CD Source Controller</b>	<b>876</b>
Prerequisites	876
Configuration	877
Installation	877
Try fluxcd-source-controller	878
Documentation	879
<b>Overview of Source Controller</b>	<b>879</b>
<b>Overview of Source Controller</b>	<b>880</b>
<b>Install Source Controller</b>	<b>880</b>
Prerequisites	880
Install	880
<b>Troubleshoot Source Controller</b>	<b>882</b>
Collecting Logs from Source Controller Manager	882
<b>Source Controller reference</b>	<b>883</b>
ImageRepository	883
MavenArtifact	884
<b>Overview of Developer Conventions</b>	<b>885</b>
Prerequisites	885
Features	885
Enabling Live Updates	885
Enabling debugging	885
Next steps	886
<b>Overview of Developer Conventions</b>	<b>886</b>
Prerequisites	886
Features	886
Enabling Live Updates	886
Enabling debugging	887
Next steps	887
<b>Install Developer Conventions</b>	<b>887</b>
Prerequisites	888

Install	888
Resource limits	888
Uninstall	889
<b>Run Developer Conventions on an OpenShift cluster</b>	<b>889</b>
<b>Overview of Learning Center for Tanzu Application Platform</b>	<b>889</b>
Use cases	890
Use case requirements	890
Platform architectural overview	891
Next steps	892
<b>Overview of Learning Center for Tanzu Application Platform</b>	<b>893</b>
Use cases	893
Use case requirements	893
Platform architectural overview	894
Next steps	895
<b>Install Learning Center</b>	<b>896</b>
Prerequisites	896
Install Learning Center	896
Install the Self-Guided Tour Training Portal and Workshop	898
Supported Learning Center Values Configuration	898
<b>About Learning Center workshops</b>	<b>900</b>
<b>Get started with Learning Center</b>	<b>902</b>
Installing Learning Center	902
Get started	903
<b>Get started with Learning Center</b>	<b>903</b>
Installing Learning Center	903
Get started	903
<b>Install and configure the Learning Center operator</b>	<b>903</b>
Installing and setting up Learning Center operator	903
Cluster pod security policies	904
Specifying the ingress domain	904
Set the environment variable manually	905
Enforcing secure connections	905
Configuration YAML	906
Create the TLS secret manually	906
Specifying the ingress class	906
Configuration YAML	907
Set the environment variable manually	907

Trusting unsecured registries	907
<b>Get started with Learning Center workshops</b>	<b>908</b>
Creating the workshop environment	908
Requesting a workshop instance	908
Deleting the workshop instance	909
Deleting the workshop environment	910
<b>Get started with Learning Center training portals</b>	<b>910</b>
Working with multiple workshops	910
Loading the workshop definition	910
Creating the workshop training portal	911
Accessing workshops via the web portal	913
Deleting the workshop training portal	914
<b>Delete Learning Center</b>	<b>914</b>
<b>Local install guides</b>	<b>915</b>
<b>Local install guides</b>	<b>915</b>
<b>Install Learning Center on Kind</b>	<b>915</b>
Prerequisites	915
Kind cluster creation	916
Ingress controller with DNS	916
Install carvel tools	917
Install Tanzu package repository	917
Create a configuration YAML file for Learning Center package	918
Using a nip.io DNS address	918
Install Learning Center package onto a Kubernetes cluster	919
Install workshop tutorial package onto a Kubernetes cluster	919
Run the workshop	919
Trusting insecure registries	919
<b>Install Learning Center on Minikube</b>	<b>920</b>
Trusting insecure registries	921
Prerequisites	921
Ingress controller with DNS	921
Install carvel tools	922
Install Tanzu package repository	922
Create a configuration YAML file for the Learning Center package	923
Using a nip.io DNS address	923
Install Learning Center package onto a minikube cluster	924
Install workshop tutorial package onto a minikube cluster	924

Run the workshop	924
Working with large images	924
Limited resource availability	924
Storage provisioner issue	925
<b>Create workshops for Learning Center</b>	<b>925</b>
<b>Create workshops for Learning Center</b>	<b>925</b>
<b>Configure your Learning Center workshop</b>	<b>926</b>
Specifying structure of the content	926
Specifying the runtime configuration	927
Next steps	928
<b>Create the image for your Learning Center workshop</b>	<b>928</b>
Templates for creating a workshop	928
Workshop content directory layout	929
Directory for workshop exercises	930
<b>Working on your Learning Center workshop content</b>	<b>930</b>
Deactivating reserved sessions	930
Live updates to the content	931
Custom workshop image changes	932
Custom workshop image overlay	932
Changes to workshop definition	933
Local build of workshop image	933
<b>Build an image for your Learning Center workshop</b>	<b>934</b>
Structure of the Dockerfile	934
Custom workshop base images	934
Installing extra system packages	935
Installing third-party packages	935
<b>Writing instructions for your Learning Center workshop</b>	<b>936</b>
Annotation of executable commands	936
Annotation of text to be copied	937
Extensible clickable actions	938
Supported workshop editor	940
Clickable actions for the dashboard	940
Clickable actions for the editor	941
Clickable actions for file download	943
Clickable actions for the examiner	944
Clickable actions for sections	946
Overriding title and description	947

Escaping of code block content	947
Interpolation of data variables	947
Adding custom data variables	948
Passing environment variables	949
Handling embedded URL links	949
Conditional rendering of content	950
Embedding custom HTML content	950
<b>Automate your Learning Center workshop runtime</b>	<b>951</b>
Predefined environment variables	951
Running steps on container start	952
Running background applications	952
Terminal user shell environment	953
Overriding terminal shell command	953
<b>Add presenter slides to your Learning Center workshop</b>	<b>954</b>
Use reveal.js presentation tool	954
Use a PDF file for presenter slides	954
<b>Requirements for Learning Center in an air-gapped environment</b>	<b>954</b>
Workshop yaml changes	954
Self-signed certificates	955
Internet dependencies	955
<b>Define custom resources for Learning Center</b>	<b>955</b>
Workshop definition resource	955
Workshop environment resource	956
Workshop request resource	956
Workshop session resource	957
Training portal resource	957
System profile resource	957
Loading the workshop CRDs	958
<b>Define custom resources for Learning Center</b>	<b>958</b>
Workshop definition resource	958
Workshop environment resource	959
Workshop request resource	960
Workshop session resource	960
Training portal resource	960
System profile resource	961
Loading the workshop CRDs	961
<b>Configure the Workshop resource</b>	<b>961</b>
Workshop title and description	962

Downloading workshop content	963
Container image for the workshop	965
Setting environment variables	966
Overriding the memory available	967
Mounting a persistent volume	967
Resource budget for namespaces	968
Patching workshop deployment	970
Creation of session resources	971
Overriding default role-based access control (RBAC) rules	972
Running user containers as root	974
Creating additional namespaces	974
Shared workshop resources	977
Workshop pod security policy	978
Custom security policies for user containers	980
Defining additional ingress points	981
External workshop instructions	983
Deactivating workshop instructions	984
Enabling the Kubernetes console	984
Enabling the integrated editor	985
Enabling workshop downloads	986
Enabling the test examiner	986
Enabling session image registry	987
Enabling ability to use Docker	989
Enabling WebDAV access to files	990
Customizing the terminal layout	991
Adding custom dashboard tabs	991
<b>Configure the WorkshopEnvironment resource</b>	<b>992</b>
Specifying the workshop definition	993
Overriding environment variables	993
Overriding the ingress domain	994
Controlling access to the workshop	995
Overriding the login credentials	996
Additional workshop resources	996
Creation of workshop instances	997
<b>Configure the WorkshopRequest resource</b>	<b>998</b>
Specifying workshop environment	998
Specifying required access token	999
<b>Configure the TrainingPortal resource</b>	<b>999</b>
Specifying the workshop definitions	999
Limit the number of sessions	1000

Capacity of individual workshops	1000
Set reserved workshop instances	1001
Override initial number of sessions	1001
Setting defaults for all workshops	1002
Set caps on individual users	1002
Expiration of workshop sessions	1003
Updates to workshop environments	1004
Override the ingress domain	1005
Override the portal host name	1006
Set extra environment variables	1007
Override portal credentials	1007
Control registration type	1008
Specify an event access code	1009
Make a list of workshops public	1009
Use an external list of workshops	1010
Override portal title and logo	1010
Allow the portal in an iframe	1011
Collect analytics on workshops	1011
Track using Google Analytics	1013
<b>Configure the SystemProfile resource</b>	<b>1014</b>
Operator default system profile	1014
Defining configuration for ingress	1014
Defining container image registry pull secrets	1015
Defining storage class for volumes	1015
Defining storage group for volumes	1016
Restricting network access	1017
Running Docker daemon rootless	1017
Overriding network packet size	1018
Image registry pull through cache	1019
Setting default access credentials	1020
Overriding the workshop images	1020
Tracking using Google Analytics	1021
Overriding styling of the workshop	1022
Additional custom system profiles	1023
<b>Configure the WorkshopSession resource</b>	<b>1023</b>
Specifying the session identity	1023
Specifying the login credentials	1024
Specifying the ingress domain	1024
Setting the environment variables	1025
<b>Enable anonymous access to a Learning Center training portal</b>	<b>1026</b>



Enabling anonymous access	1026
Triggering workshop creation	1027
<b>Enable anonymous access to a Learning Center training portal</b>	<b>1027</b>
Enabling anonymous access	1028
Triggering workshop creation	1028
<b>Use the Learning Center workshop catalog</b>	<b>1029</b>
Listing available workshops	1029
<b>Use session management for your Learning Center workshops</b>	<b>1030</b>
Deactivating portal user registration	1031
Requesting a workshop session	1031
Associating sessions with a user	1032
Listing all workshop sessions	1033
<b>Use client authentication for Learning Center</b>	<b>1034</b>
Querying the credentials	1034
Requesting an access token	1035
Refreshing the access token	1035
<b>Troubleshoot Learning Center</b>	<b>1036</b>
Training portal stays in pending state	1036
image-policy-webhook-service not found	1036
Updates to Tanzu Application Platform values file not reflected in Learning Center Training Portal	1036
Increase your cluster's resources	1037
Kubernetes Api Timeout error	1037
No URL returned to your trainingportal	1038
<b>Overview of Supply Chain Choreographer for Tanzu</b>	<b>1038</b>
Overview	1038
Out of the Box Supply Chains	1038
<b>Overview of Supply Chain Choreographer for Tanzu</b>	<b>1039</b>
Overview	1039
Out of the Box Supply Chains	1039
<b>Install Supply Chain Choreographer</b>	<b>1039</b>
Prerequisites	1040
Install	1040
<b>Out of the Box Supply Chain Basic</b>	<b>1041</b>
Prerequisites	1041
Developer Namespace	1041

Registries Secrets	1041
ServiceAccount	1042
RoleBinding	1043
Developer workload	1044
<b>Out of the Box Supply Chain Basic</b>	<b>1044</b>
Prerequisites	1044
Developer Namespace	1045
Registries Secrets	1045
ServiceAccount	1046
RoleBinding	1046
Developer workload	1047
<b>Install Out of the Box Supply Chain Basic</b>	<b>1047</b>
Prerequisites	1048
Install	1048
<b>Out of the Box Supply Chain with Testing</b>	<b>1050</b>
Prerequisites	1051
Developer Namespace	1051
Updates to the developer Namespace	1052
Tekton/Pipeline	1052
Allow multiple Tekton pipelines in a namespace	1053
Developer Workload	1054
<b>Out of the Box Supply Chain with Testing</b>	<b>1055</b>
Prerequisites	1055
Developer Namespace	1056
Updates to the developer Namespace	1056
Tekton/Pipeline	1056
Allow multiple Tekton pipelines in a namespace	1057
Developer Workload	1058
<b>Install Out of the Box Supply Chain with Testing</b>	<b>1059</b>
Prerequisites	1059
Install	1059
<b>Out of the Box Supply Chain with Testing and Scanning</b>	<b>1062</b>
Prerequisites	1063
Developer namespace	1063
Updates to the developer namespace	1064
ScanPolicy	1065
ScanTemplate	1066
Enable storing scan results	1066

Allow multiple Tekton pipelines in a namespace	1066
Developer workload	1068
CVE triage workflow	1068
Scan Images using a different scanner	1068
<b>Out of the Box Supply Chain with Testing and Scanning</b>	<b>1069</b>
Prerequisites	1069
Developer namespace	1070
Updates to the developer namespace	1070
ScanPolicy	1071
ScanTemplate	1072
Enable storing scan results	1072
Allow multiple Tekton pipelines in a namespace	1073
Developer workload	1074
CVE triage workflow	1074
Scan Images using a different scanner	1075
<b>Install Out of the Box Supply Chain with Testing and Scanning</b>	<b>1075</b>
Prerequisites	1075
Install	1075
<b>Out of the Box Templates for Supply Chain Choreographer</b>	<b>1078</b>
<b>Out of the Box Templates for Supply Chain Choreographer</b>	<b>1079</b>
<b>Install Out of the Box Templates</b>	<b>1079</b>
Prerequisites	1080
Install	1080
<b>Out of the Box Delivery Basic</b>	<b>1080</b>
Prerequisites	1081
Using Out of the Box Delivery Basic	1081
More information	1081
<b>Out of the Box Delivery Basic</b>	<b>1081</b>
Prerequisites	1082
Using Out of the Box Delivery Basic	1082
More information	1082
<b>Install Out of the Box Delivery Basic</b>	<b>1083</b>
Prerequisites	1083
Install	1083
<b>How-to guides for Supply Chain Choreographer for Tanzu</b>	<b>1084</b>
How-to guides	1084

<b>Out of the Box Supply Chain with testing on Jenkins</b>	<b>1084</b>
Prerequisites	1084
Making a Jenkins test job	1084
Example Jenkins Job	1085
Updates to the developer namespace	1086
Create a secret	1086
Create a Tekton pipeline	1087
Patch the Service Account	1088
Developer Workload	1088
<b>Building container images with Supply Chain Choreographer</b>	<b>1090</b>
Methods for building container images	1090
<b>Building from source with Supply Chain Choreographer</b>	<b>1090</b>
Git source	1091
Private GitRepository	1091
HTTP(S) Basic-authentication and Token-based authentication	1093
SSH authentication	1093
How it works	1094
Workload parameters	1095
Local source	1095
Authentication	1096
Developer	1096
Supply chain components	1096
How it works	1096
Maven Artifact	1097
Maven Repository Secret	1098
<b>Use Dockerfile-based builds with Supply Chain Choreographer</b>	<b>1098</b>
<b>Use Dockerfile-based builds with Supply Chain Choreographer</b>	<b>1099</b>
OpenShift	1100
<b>Tanzu Build Service integration for Supply Chain Choreographer</b>	<b>1100</b>
<b>Use an existing image with Supply Chain Choreographer</b>	<b>1102</b>
Requirements for prebuilt images	1102
Configure your workload to use a prebuilt image	1103
Examples	1103
Using a Dockerfile	1104
Using Spring Boot's build-image Maven target	1105
About Out of the Box Supply Chains	1106
Understanding the supply chain for a prebuilt image	1107

<b>Git authentication</b>	<b>1108</b>
HTTP	1108
SSH	1109
Read more on Git	1111
<b>Author your supply chains</b>	<b>1111</b>
Providing your own supply chain	1111
Providing your own templates	1112
Modifying an Out of the Box Supply Chain	1113
Example	1113
Modifying an Out of the Box Supply template	1115
Example	1115
Live modification of supply chains and templates	1116
Adding custom behavior to Supply Chains	1117
<b>Reference guides for Supply Chain Choreographer for Tanzu</b>	<b>1118</b>
Reference guides	1118
<b>Events reference for Supply Chain Choreographer</b>	<b>1118</b>
Events	1118
StampedObjectApplied	1118
StampedObjectRemoved	1118
ResourceOutputChanged	1118
ResourceHealthyStatusChanged	1119
<b>Workload Reference for Supply Chain Choreographer</b>	<b>1119</b>
Standard Fields	1119
Labels	1119
Parameters	1119
Service Account	1121
<b>Supply chains</b>	<b>1121</b>
Source-to-URL	1121
Purpose	1121
Resources	1121
source-provider	1121
image-provider	1121
Common resources	1122
Parameters provided to all resources	1122
Package	1122
More information	1122
Source-Test-to-URL	1122
Resources	1122

source-provider	1122
source-tester	1122
image-provider	1123
Common resources	1123
Parameters provided to all resources	1123
Package	1123
More information	1123
Source-Test-Scan-to-URL	1123
Resources	1124
source-provider	1124
source-tester	1124
source-scanner	1124
image-provider	1124
image-scanner	1124
Common resources	1124
Parameters provided to all resources	1125
Package	1125
More information	1125
Basic-Image-to-URL	1125
Resources	1125
image-provider	1125
Common resources	1125
Parameters provided to all resources	1125
Package	1126
More information	1126
Testing-Image-to-URL	1126
Resources	1126
image-provider	1126
Common resources	1126
Parameters provided to all resources	1126
Package	1126
More information	1126
Scanning-image-scan-to-URL	1126
Resources	1127
image-provider	1127
image-scanner	1127
Common resources	1127
Parameters provided to all resources	1127
Package	1127
More information	1127
Source-to-URL-Package (experimental)	1127
Purpose	1127

Resources	1127
source-provider	1128
image-provider	1128
carvel-package	1128
package-config-writer	1128
Common resources	1128
Parameters provided to all resources	1128
Package	1129
More information	1129
Basic-Image-to-URL-Package (experimental)	1129
Resources	1129
image-provider	1129
carvel-package	1129
package-config-writer	1129
Common resources	1129
Parameters provided to all resources	1130
Package	1130
More information	1130
Resources common to all OOTB supply chains	1130
config-provider	1130
app-config	1130
service-bindings	1130
api-descriptors	1131
config-writer	1131
deliverable	1131
Parameters provided by all supply chains to all resources	1131
<b>Template reference for Supply Chain Choreographer</b>	<b>1131</b>
source-template	1132
Purpose	1132
Used by	1132
Creates	1132
GitRepository	1132
Parameters	1132
<b>Template reference for Supply Chain Choreographer</b>	<b>1132</b>
More information	1133
ImageRepository	1133
Parameters	1133
More information	1133
MavenArtifact	1134
Parameters	1134

More information	1134
testing-pipeline	1134
Purpose	1135
Used by	1135
Creates	1135
Parameters	1135
More information	1135
source-scanner-template	1135
Purpose	1136
Used by	1136
Creates	1136
Parameters	1136
More information	1136
image-provider-template	1136
Purpose	1136
Used by	1136
Creates	1137
Parameters	1137
More information	1137
kpack-template	1137
Purpose	1137
Used by	1137
Creates	1137
Parameters	1137
More information	1138
kaniko-template	1138
Purpose	1138
Used by	1139
Creates	1139
Parameters	1139
More information	1139
image-scanner-template	1140
Purpose	1140
Used by	1140
Creates	1140
Parameters	1140
More information	1140
convention-template	1140
Purpose	1140
Used by	1140
Creates	1141
Parameters	1141



More information	1142
config-template	1142
Purpose	1142
Used by	1142
Creates	1142
Parameters	1142
More information	1142
worker-template	1143
Purpose	1143
Used by	1143
Creates	1143
Parameters	1143
More information	1143
server-template	1143
Purpose	1143
Used by	1143
Creates	1143
Parameters	1144
More information	1144
service-bindings	1144
Purpose	1144
Used by	1144
Creates	1144
Parameters	1144
More information	1145
api-descriptors	1145
Purpose	1145
Used by	1145
Creates	1145
Parameters	1145
More information	1146
config-writer-template	1146
Purpose	1146
Used by	1146
Creates	1146
Parameters	1146
More information	1148
config-writer-and-pull-requester-template	1148
Purpose	1148
Used by	1148
Creates	1148
Parameters	1148

More information	1150
deliverable-template	1150
Purpose	1150
Used by	1150
Creates	1150
Parameters	1151
More information	1152
external-deliverable-template	1152
Purpose	1152
Used by	1152
Creates	1152
Parameters	1153
More information	1154
delivery-source-template	1154
Purpose	1154
Used by	1154
Creates	1154
GitRepository	1154
Parameters	1155
More information	1155
ImageRepository	1155
Parameters	1155
More information	1155
app-deploy	1156
Purpose	1156
Used by	1156
Creates	1156
Parameters	1156
More information	1156
carvel-package (experimental)	1156
Purpose	1156
Used by	1156
Creates	1157
Parameters	1157
More information	1161
package-config-writer-template (experimental)	1161
Purpose	1161
Used by	1161
Creates	1161
Parameters	1161
More information	1163
package-config-writer-and-pull-requester-template (experimental)	1163

Purpose	1164
Used by	1164
Creates	1164
Parameters	1164
More information	1166
<b>ClusterRunTemplate reference</b>	<b>1166</b>
tekton-source-pipelinerun	1166
Purpose	1166
Used by	1166
Creates	1166
Inputs	1166
<b>ClusterRunTemplate reference</b>	<b>1166</b>
More information	1167
tekton-taskrun	1167
Purpose	1167
Used by	1167
Creates	1167
Inputs	1167
commit-and-pr-pipelinerun	1167
Purpose	1168
Used by	1168
Creates	1168
Inputs	1168
More information	1169
<b>Delivery reference</b>	<b>1169</b>
delivery-basic	1169
Purpose	1169
Resources	1169
source-provider	1169
Deployer	1169
Package	1169
More information	1169
<b>Git</b>	<b>1169</b>
Supported Git Repositories	1170
Related Articles	1170
<b>GitOps versus RegistryOps</b>	<b>1170</b>
GitOps	1170
Examples	1171

Deprecated parameters	1172
Examples	1173
Pull requests	1174
Authentication	1174
Authentication	1175
HTTP(S) Basic-auth or Token-based authentication	1175
SSH	1176
GitOps workload parameters	1177
Read more on Git	1178
RegistryOps	1178
<b>Overview of Supply Chain Security Tools - Scan</b>	<b>1178</b>
Overview	1178
Language support	1179
Use cases	1179
Supply Chain Security Tools - Scan features	1179
A Note on Vulnerability Scanners	1179
Missed CVEs	1179
False positives	1180
<b>Overview of Supply Chain Security Tools - Scan</b>	<b>1181</b>
Overview	1181
Language support	1181
Use cases	1181
Supply Chain Security Tools - Scan features	1181
A Note on Vulnerability Scanners	1181
Missed CVEs	1182
False positives	1182
<b>Install Supply Chain Security Tools - Scan</b>	<b>1183</b>
Prerequisites	1183
Configure properties	1183
Install	1185
Option 1: Install to multiple namespaces with the Namespace Provisioner	1185
Option 2: Install manually to each individual namespace	1185
<b>Upgrade Supply Chain Security Tools - Scan</b>	<b>1189</b>
Prerequisites	1189
General Upgrades for SCST - Scan	1189
Upgrading a scanner in all namespaces	1189
Installation by using Namespace Provisioner	1189
Manual installation	1189
Upgrade to Version v1.2.0	1190

<b>Install another scanner for Supply Chain Security Tools - Scan</b>	<b>1193</b>
Prerequisites	1193
Install	1193
Verify Installation	1195
Install scanner to multiple namespaces	1198
Configure Tanzu Application Platform Supply Chain to use new scanner	1198
Uninstall Scanner	1199
Other Available Scanner Integrations	1199
<b>Prerequisites for Snyk Scanner for Supply Chain Security Tools - Scan (Beta)</b>	<b>1200</b>
Prepare the Snyk Scanner configuration	1200
Supply Chain Security Tools - Store integration	1201
Sample ScanPolicy for Snyk in SPDX JSON format	1202
<b>Prerequisites for Carbon Black Scanner for Supply Chain Security Tools - Scan(Beta)</b>	<b>1203</b>
Prepare the Carbon Black Scanner configuration	1203
Supply Chain Security Tools - Store integration	1204
Using Supply Chain Security Tools - Store Integration	1204
Without Supply Chain Security Tools - Store Integration	1205
Sample ScanPolicy in CycloneDX format	1205
<b>Prerequisites for Prisma Scanner for Supply Chain Security Tools - Scan (Alpha)</b>	<b>1206</b>
Verify the latest alpha package version	1206
Relocate images to a registry	1207
Add the Prisma Scanner package repository	1208
Prepare the Prisma Scanner configuration	1209
Obtain Console url and Access Keys/Token	1209
Access key and secret authentication	1209
Access Token Authentication	1210
Create Prisma Secret	1210
Access Token Authentication	1210
Access Key Authentication	1211
Supply Chain Security Tools - Store integration	1212
Multiple Scanners installed	1213
Prisma Only Scanner Installed	1213
No Store Integration	1214
Prepare the ScanPolicy	1214
Sample ScanPolicy using Prisma Policies	1214
Sample ScanPolicy using Local Policies	1215
Install Prisma Scanner	1216

Self-Signed Registry Certificate	1216
Tanzu Application Platform Values Shared CA	1216
Secret within Developer Namespace	1216
Known Limits	1217
<b>Spec reference</b>	<b>1217</b>
About source and image scans	1217
About policy enforcement around vulnerabilities found	1218
<b>Scan samples for Supply Chain Security Tools - Scan</b>	<b>1218</b>
<b>Scan samples for Supply Chain Security Tools - Scan</b>	<b>1218</b>
<b>Sample public image scan with compliance check for Supply Chain Security Tools - Scan</b>	<b>1218</b>
Public image scan	1218
Define the ScanPolicy and ImageScan	1219
(Optional) Set up a watch	1220
Deploy the resources	1220
View the scan results	1220
Edit the ScanPolicy	1220
Clean up	1220
<b>Sample public source code scan with compliance check for Supply Chain Security Tools - Scan</b>	<b>1221</b>
Public source scan	1221
Run an example public source scan	1221
<b>Sample private image scan for Supply Chain Security Tools - Scan</b>	<b>1223</b>
Define the resources	1223
Set up target image pull secret	1223
Create the private image scan	1224
(Optional) Set up a watch	1224
Deploy the resources	1224
View the scan results	1225
Clean up	1225
View vulnerability reports	1225
<b>Sample private source scan for Supply Chain Security Tools - Scan</b>	<b>1225</b>
Define the resources	1225
(Optional) Set up a watch	1227
Deploy the resources	1227
View the scan status	1227
Clean up	1227
View vulnerability reports	1227

<b>Sample public source scan of a blob for Supply Chain Security Tools - Scan</b>	<b>1227</b>
Define the resources	1228
(Optional) Set up a watch	1228
Deploy the resources	1228
View the scan results	1228
Clean up	1228
View vulnerability reports	1228
<b>Using Grype in air-gapped (offline) environments for Supply Chain Security Tools - Scan</b>	<b>1229</b>
To enable Grype in offline air-gapped environments	1229
Troubleshooting	1229
ERROR failed to fetch latest cli version	1229
Solution	1230
Database is too old	1230
Solution	1230
Grype package overlays are not applied to scantemplates created by Namespace Provisioner	1231
Solution	1232
Debug Grype database in a cluster	1233
<b>Triage and Remediate CVEs for Supply Chain Security Tools - Scan</b>	<b>1234</b>
Confirm that Supply Chain stopped due to failed policy enforcement	1234
Triage	1234
Remediation	1234
Updating the affected component	1234
Amending the scan policy	1235
<b>Observe Supply Chain Security Tools - Scan</b>	<b>1235</b>
Observability	1235
<b>Troubleshoot Supply Chain Security Tools - Scan</b>	<b>1235</b>
Debugging commands	1235
Debugging Scan pods	1235
Debugging SourceScan and ImageScan	1236
Debugging Scanning within a SupplyChain	1236
Viewing the Scan-Controller manager logs	1236
Restarting Deployment	1236
Troubleshooting scanner to MetadataStore configuration	1237
Insight CLI failed to post scan results to metadata store due to failed certificate verification	1237
Troubleshooting issues	1238
Missing target SSH secret	1238
Missing target image pull secret	1238

Deactivate Supply Chain Security Tools (SCST) - Store	1238
Resolving Incompatible Syft Schema Version	1239
Resolving incompatible scan policy	1239
Could not find CA in secret	1239
Blob Source Scan is reporting wrong source URL	1239
Resolving failing scans that block a Supply Chain	1240
Policy not defined in the Tanzu Application Platform GUI	1240
Lookup error when connecting to SCST - Store	1240
Sourcscan error with SCST - Store endpoint without a prefix	1241
Deprecated pre-v1.2 templates	1241
Incorrectly configured self-signed certificate	1241
Unable to pull scan controller and scanner images from a specified registry	1241
Grype database not available	1242
<b>Configure code repositories and image artifacts for Supply Chain Security Tools - Scan</b>	<b>1242</b>
Prerequisite	1242
Deploy scan custom resources	1242
SourceScan	1242
ImageScan	1244
<b>Configure code repositories and image artifacts for Supply Chain Security Tools - Scan</b>	<b>1245</b>
Prerequisite	1245
Deploy scan custom resources	1245
SourceScan	1245
ImageScan	1247
<b>Enforce compliance policy using Open Policy Agent</b>	<b>1248</b>
Writing a policy template	1248
Rego file contract	1248
Define a Rego file for policy enforcement	1249
Further refine the Scan Policy for use	1250
Enable Tanzu Application Platform GUI to view ScanPolicy Resource	1253
Deprecated Rego file Definition	1253
<b>Create a ScanTemplate with Supply Chain Security Tools - Scan</b>	<b>1254</b>
Overview	1254
Output Model	1254
ScanTemplate Structure	1255
Sample Outputs	1255
<b>View scan status conditions for Supply Chain Security Tools - Scan</b>	<b>1256</b>
Viewing scan status	1256



Understanding conditions	1256
Condition types for the scans	1256
Scanning	1256
Succeeded	1256
SendingResults	1257
PolicySucceeded	1257
Understanding CVECount	1257
Understanding MetadataURL	1257
Understanding Phase	1257
Understanding ScannedBy	1258
Understanding ScannedAt	1258
Overview of Supply Chain Security Tools for VMware Tanzu - Policy Controller	1258
Overview of Supply Chain Security Tools for VMware Tanzu - Policy Controller	1259
Install Supply Chain Security Tools - Policy Controller	1260
Prerequisites	1260
Install	1261
Install Sigstore Stack	1265
Download Stack Release Files	1265
Migrate Images onto Internal Registry	1265
Copy Release Files to Cluster Accessible Machine	1267
Prepare Patching Fulcio Release File	1267
OIDCIssuer	1267
Metalssuers	1268
Applying the patch for Fulcio release file	1269
Patch Knative-Serving	1269
Create OIDC Reviewer Binding	1269
Install Trillian	1269
Install Rekor	1270
Install Fulcio	1271
Install Certificate Transparency Log (CTLog)	1271
Install TUF	1272
Update Policy Controller with TUF Mirror and Root	1273
Uninstall Sigstore Stack	1275
Migration From Supply Chain Security Tools - Sign	1275
Enable Policy Controller on Namespaces	1275
Policy Controller ClusterImagePolicy	1275
Excluding Namespaces	1275

Specifying Public Keys	1276
Specifying Image Matching	1277
Specifying policy mode	1278
<b>Configuring Supply Chain Security Tools - Policy</b>	<b>1278</b>
Admission of Images	1278
Including Namespaces	1278
Create a ClusterImagePolicy resource	1279
mode	1279
images	1280
match	1280
authorities	1280
static.action	1282
Provide credentials for the package	1282
Provide secrets for authentication in your policy	1283
Verify your configuration	1283
<b>Overview of Supply Chain Security Tools for VMware Tanzu - Sign</b>	<b>1285</b>
<b>Overview of Supply Chain Security Tools for Tanzu – Store</b>	<b>1285</b>
Overview	1285
Using the Tanzu Insight CLI plug-in	1285
Multicluster configuration	1285
Integrating with Tanzu Application Platform GUI	1285
Additional documentation	1286
<b>Overview of Supply Chain Security Tools for Tanzu – Store</b>	<b>1286</b>
Overview	1286
Using the Tanzu Insight CLI plug-in	1286
Multicluster configuration	1286
Integrating with Tanzu Application Platform GUI	1287
Additional documentation	1287
<b>Configure your target endpoint and certificate for Supply Chain Security Tools - Store</b>	<b>1287</b>
Overview	1287
Using Ingress	1287
<b>Single Cluster setup</b>	<b>1287</b>
<b>Set Target</b>	<b>1288</b>
Next Step	1288
Additional Resources	1288
<b>Configure your access tokens for Supply Chain Security Tools - Store</b>	<b>1288</b>

Setting the Access Token	1288
Additional Resources	1289
<b>Security details for Supply Chain Security Tools - Store</b>	<b>1289</b>
Application security	1289
TLS encryption	1289
Cryptographic algorithms	1289
Access controls	1289
Authentication	1289
Authorization	1289
Container security	1290
Non-root user	1290
Security scanning	1290
Static Application Security Testing (SAST)	1290
Software Composition Analysis (SCA)	1290
<b>Additional documentation for Supply Chain Security Tools - Store</b>	<b>1291</b>
Use and operate	1291
Troubleshooting and logging	1291
Configuration	1291
Access control	1291
Certificates	1291
Database	1291
Other	1292
<b>Additional documentation for Supply Chain Security Tools - Store</b>	<b>1292</b>
Use and operate	1292
Troubleshooting and logging	1292
Configuration	1292
Access control	1292
Certificates	1292
Database	1292
Other	1293
<b>API reference for Supply Chain Security Tools - Store</b>	<b>1293</b>
Information	1293
Version	1293
Content negotiation	1293
URI Schemes	1293
Consumes	1293
Produces	1293
All endpoints	1293
images	1293

Operations	1293
Packages	1293
Sources	1294
vlartifact_groups	1294
viimages	1294
vlpackages	1295
visources	1295
vvulnerabilities	1295
vulnerabilities	1295
Paths	1296
Create an artifact group with specified labels and entity (CreateArtifactGroup)	1296
Parameters	1296
All responses	1296
Responses	1296
201 - ArtifactGroupPostResponse	1296
Schema	1296
400 - ErrorMessage	1296
Schema	1296
Default Response	1296
Schema	1296
Create a new image report. Related packages and vulnerabilities are also created. (CreateImageReport)	1296
Parameters	1297
All responses	1297
Responses	1297
200 - Image	1297
Schema	1297
Default Response	1297
Schema	1297
Create a new source report. Related packages and vulnerabilities are also created. (CreateSourceReport)	1297
Parameters	1297
All responses	1297
Responses	1297
200 - Source	1297
Schema	1298
Default Response	1298
Schema	1298
Search image by ID (GetImageByID)	1298
Parameters	1298
All responses	1298
Responses	1298

200 - Image	1298
Schema	1298
404 - ErrorMessage	1298
Schema	1298
Default Response	1298
Schema	1299
List the packages in an image. (GetImagePackages)	1299
Parameters	1299
All responses	1299
Responses	1299
200 - Package	1299
Schema	1299
Default Response	1299
Schema	1299
List packages of the given image. (GetImagePackagesQuery)	1299
Parameters	1299
All responses	1299
Responses	1300
200 - Package	1300
Schema	1300
Default Response	1300
Schema	1300
List vulnerabilities from the given image. (GetImageVulnerabilities)	1300
Parameters	1300
All responses	1300
Responses	1300
200 - Vulnerability	1300
Schema	1300
Default Response	1301
Schema	1301
Search image by id, name or digest . (GetImages)	1301
All responses	1301
Responses	1301
200 - Image	1301
Schema	1301
Default Response	1301
Schema	1301
Search package by ID (GetPackageByID)	1301
Parameters	1301
All responses	1301
Responses	1302
200 - Package	1302

Schema	1302
404 - ErrorMessage	1302
Schema	1302
Default Response	1302
Schema	1302
List the images that contain the given package. (GetPackageImages)	1302
Parameters	1302
All responses	1302
Responses	1302
200 - Image	1302
Schema	1303
Default Response	1303
Schema	1303
List the sources containing the given package. (GetPackageSources)	1303
Parameters	1303
All responses	1303
Responses	1303
200 - Source	1303
Schema	1303
Default Response	1303
Schema	1303
List vulnerabilities from the given package. (GetPackageVulnerabilities)	1303
Parameters	1304
All responses	1304
Responses	1304
200 - Vulnerability	1304
Schema	1304
Default Response	1304
Schema	1304
Search packages by id, name and/or version. (GetPackages)	1304
Parameters	1304
All responses	1304
Responses	1305
200 - Package	1305
Schema	1305
Default Response	1305
Schema	1305
Search source by ID (GetSourceByID)	1305
Parameters	1305
All responses	1305
Responses	1305
200 - Source	1305

Schema	1305
404 - ErrorMessage	1305
Schema	1306
Default Response	1306
Schema	1306
get source packages (GetSourcePackages)	1306
Parameters	1306
All responses	1306
Responses	1306
200 - Package	1306
Schema	1306
Default Response	1306
Schema	1306
List packages of the given source. (GetSourcePackagesQuery)	1306
Parameters	1307
All responses	1307
Responses	1307
200 - Package	1307
Schema	1307
Default Response	1307
Schema	1307
get source vulnerabilities (GetSourceVulnerabilities)	1307
Parameters	1307
All responses	1307
Responses	1307
200 - Vulnerability	1308
Schema	1308
Default Response	1308
Schema	1308
List vulnerabilities of the given source. (GetSourceVulnerabilitiesQuery)	1308
Parameters	1308
All responses	1308
Responses	1308
200 - Vulnerability	1308
Schema	1308
Default Response	1309
Schema	1309
Search for sources by ID, repository, commit sha and/or organization. (GetSources)	1309
Parameters	1309
All responses	1309
Responses	1309
200 - Source	1309

Schema	1309
Default Response	1309
Schema	1309
Search for vulnerabilities by CVE id. (GetVulnerabilities)	1309
Parameters	1310
All responses	1310
Responses	1310
200 - Vulnerability	1310
Schema	1310
Default Response	1310
Schema	1310
Search vulnerability by ID (GetVulnerabilityByID)	1310
Parameters	1310
All responses	1310
Responses	1311
200 - Vulnerability	1311
Schema	1311
404 - ErrorMessage	1311
Schema	1311
Default Response	1311
Schema	1311
List the images that contain the given vulnerability. (GetVulnerabilityImages)	1311
Parameters	1311
All responses	1311
Responses	1311
200 - Image	1311
Schema	1311
Default Response	1312
Schema	1312
List packages that contain the given CVE id. (GetVulnerabilityPackages)	1312
Parameters	1312
All responses	1312
Responses	1312
200 - Package	1312
Schema	1312
Default Response	1312
Schema	1312
List sources that contain the given vulnerability. (GetVulnerabilitySources)	1312
Parameters	1312
All responses	1313
Responses	1313
200 - Source	1313



Schema	1313
Default Response	1313
Schema	1313
health check (HealthCheck)	1313
All responses	1313
Responses	1313
200	1313
Schema	1313
Default Response	1313
Schema	1313
Query for a list of artifact group that contains image(s) with specified digests, and or source(s) with specified shas. At least one image digest or source sha must be provided. This query can be further refined by matching images and sources with a specific combination of package name and/or cve id. (SearchArtifactGroups)	1314
Parameters	1314
All responses	1314
Responses	1314
200 - PaginatedArtifactGroupResponse	1314
Schema	1314
400 - ErrorMessage	1314
Schema	1314
Default Response	1314
Schema	1314
Search for how many artifact groups are affected by vulnerabilities associated with the specified image(s) digests, and/or source(s) shas. At least one image digest or source sha must be provided. (SearchArtifactGroupsVulnReach)	1315
Parameters	1315
All responses	1315
Responses	1315
200 - PaginatedArtifactGroupVulnReachResponse	1315
Schema	1315
400 - ErrorMessage	1315
Schema	1315
Default Response	1315
Schema	1315
Search for all vulnerabilities associated with an artifact group that contains image(s) with specified digests, and/or source(s) with specified shas. At least one image digest or source sha must be provided. (SearchArtifactGroupsVulnerabilities)	1316
Parameters	1316
All responses	1316
Responses	1316
200 - PaginatedArtifactGroupVulnerabilityResponse	1316
Schema	1316

400 - ErrorMessage	1316
Schema	1316
Default Response	1316
Schema	1316
Query for images. If no parameters are given, this endpoint will return all images. (V1GetImages)	1316
Parameters	1317
All responses	1317
Responses	1317
200 - PaginatedImageResponse	1317
Schema	1317
404 - ErrorMessage	1317
Schema	1317
Default Response	1317
Schema	1318
Query for packages with images parameters. If no parameters are given, this endpoint will return all packages related to images. (V1GetImagesPackages)	1318
Parameters	1318
All responses	1318
Responses	1318
200 - PaginatedPackageResponse	1318
Schema	1318
404 - ErrorMessage	1319
Schema	1319
Default Response	1319
Schema	1319
Query for vulnerabilities with image parameters. If no parameters are give, this endpoint will return all vulnerabilities. (V1GetImagesVulnerabilities)	1319
Parameters	1319
All responses	1319
Responses	1320
200 - PaginatedVulnerabilityResponse	1320
Schema	1320
404 - ErrorMessage	1320
Schema	1320
Default Response	1320
Schema	1320
Query for packages. If no parameters are given, this endpoint will return all packages. (V1GetPackages)	1320
Parameters	1320
All responses	1321
Responses	1321

200 - PaginatedPackageResponse	1321
Schema	1321
404 - ErrorMessage	1321
Schema	1321
Default Response	1321
Schema	1321
Query for sources. If no parameters are given, this endpoint will return all sources. (V1GetSources)	1321
Parameters	1321
All responses	1322
Responses	1322
200 - PaginatedSourceResponse	1322
Schema	1322
404 - ErrorMessage	1322
Schema	1322
Default Response	1322
Schema	1322
Query for packages with source parameters. If no parameters are given, this endpoint will return all packages related to sources. (V1GetSourcesPackages)	1323
All responses	1323
Responses	1323
200 - PaginatedPackageResponse	1323
Schema	1323
404 - ErrorMessage	1323
Schema	1323
Default Response	1323
Schema	1323
Query for vulnerabilities with source parameters. If no parameters are given, this endpoint will return all vulnerabilities. (V1GetSourcesVulnerabilities)	1323
Parameters	1323
All responses	1324
Responses	1324
200 - PaginatedVulnerabilityResponse	1324
Schema	1324
404 - ErrorMessage	1324
Schema	1324
Default Response	1324
Schema	1325
Models	1325
ArtifactGroupPostRequest	1325
ArtifactGroupResponse	1325
ArtifactGroupSearchFilters	1325

ArtifactGroupVulnReachFiltersPostRequest	1326
ArtifactGroupVulnReachPostResponse	1327
ArtifactGroupVulnSearchFilters	1327
DeletedAt	1328
Entity	1328
ErrorMessage	1328
Image	1329
MethodType	1329
Model	1329
NullTime	1330
Package	1330
PaginatedArtifactGroupVulnReachResponse	1330
PaginatedResponse	1330
Rating	1331
RatingResponse	1331
Source	1331
StringArray	1332
VulnResponse	1332
Vulnerability	1332
artifactGroupPostEntity	1333
artifactGroupPostResponse	1333
artifactGroupVulnArtifactGroup	1333
artifactGroupVulnEntity	1333
artifactGroupVulnPackage	1334
artifactGroupVulnResult	1334
paginatedArtifactGroupResponse	1335
paginatedArtifactGroupVulnerabilityResponse	1335
paginatedImageResponse	1335
paginatedPackageResponse	1336
paginatedSourceResponse	1336
paginatedVulnerabilityResponse	1336
responseImage	1336
responsePackage	1337
responseSource	1337
responseVulnerability	1338
<b>API walkthrough for Supply Chain Security Tools - Store</b>	<b>1338</b>
Using curl to post an image report	1338
<b>Connect to the PostgreSQL database</b>	<b>1340</b>
<b>Deployment details and configuration for Supply Chain Security Tools - Store</b>	<b>1341</b>

What is deployed	1341
Deployment configuration	1342
Supported Network Configurations	1342
App service type	1342
Ingress support	1342
Database configuration	1342
Using AWS RDS PostgreSQL database	1342
Using external PostgreSQL database	1343
Custom database password	1343
Service accounts	1343
Exporting certificates	1343
<b>Configure your AWS RDS PostgreSQL configuration</b>	<b>1343</b>
Prerequisites	1343
Setup certificate and configuration	1343
<b>Use external PostgreSQL database for Supply Chain Security Tools - Store</b>	<b>1344</b>
Prerequisites	1344
Set up certificate and configuration	1344
Validation	1345
<b>Database backup recommendations for Supply Chain Security Tools - Store</b>	<b>1345</b>
Backup	1345
Restore	1346
<b>Log configuration and usage for Supply Chain Security Tools - Store</b>	<b>1346</b>
Verbosity levels	1346
Error Logs	1347
Obtaining logs	1347
API endpoint log output	1348
Format	1348
Key-value pairs	1348
Common to all logs	1348
Logging query and path parameter values	1349
API payload log output	1350
SQL Query log output	1350
Format	1350
<b>Connect to the PostgreSQL database</b>	<b>1350</b>
<b>Troubleshooting Supply Chain Security Tools - Store</b>	<b>1351</b>
Querying by insight source returns zero CVEs even though there are CVEs in the source scan	1351

Symptom	1351
Solution	1352
Persistent volume retains data	1352
Symptom	1352
Solution	1352
Missing persistent volume	1352
Symptom	1352
Solution	1353
Builds fail due to volume errors on EKS running Kubernetes v1.23	1353
Symptom	1353
Explanation	1353
Solution	1353
Certificate Expiries	1353
Symptom	1353
Explanation	1354
Solution	1354
Troubleshooting errors from Tanzu Application Platform GUI related to SCST - Store	1354
An error occurred while loading data from the Metadata Store	1354
Symptom	1354
Cause	1354
Solution	1355
<b>Troubleshoot upgrading Supply Chain Security Tools - Store</b>	<b>1355</b>
Database deployment does not exist	1355
Invalid checkpoint record	1355
Upgraded pod hanging	1355
<b>Failover, redundancy, and backups for Supply Chain Security Tools - Store</b>	<b>1356</b>
API Server	1356
Database	1356
<b>Custom certificate configuration for Supply Chain Security Tools - Store</b>	<b>1356</b>
Default configuration	1357
(Optional) Setting up custom ingress TLS certificate	1357
Place the certificates in secret	1357
Update tap-values.yaml	1357
Additional resources	1357
<b>TLS configuration for Supply Chain Security Tools - Store</b>	<b>1357</b>
Setting up custom ingress TLS ciphers	1357
Example custom TLS settings	1358
Additional resources	1358

<b>Certificate rotation for Supply Chain Security Tools - Store</b>	<b>1358</b>
Certificates	1358
Certificate duration setting	1359
<b>Ingress support for Supply Chain Security Tools - Store</b>	<b>1359</b>
Ingress configuration	1359
Get the TLS CA certificate	1361
Additional Resources	1361
<b>Use your LoadBalancer with Supply Chain Security Tools - Store</b>	<b>1361</b>
Configure LoadBalancer	1361
Port forwarding	1362
Edit your /etc/hosts file for Port Forwarding	1362
Configure the Insight plug-in	1362
<b>Use your NodePort with Supply Chain Security Tools - Store</b>	<b>1363</b>
Overview	1363
Edit your /etc/hosts file for Port Forwarding	1363
Configure the Insight plug-in	1363
<b>Multicluster setup for Supply Chain Security Tools - Store</b>	<b>1364</b>
Overview	1364
Prerequisites	1364
Procedure summary	1364
Copy SCST - Store CA certificate from View cluster	1364
Copy SCST - Store authentication token from the View cluster	1365
Apply the CA certificate and authentication token to a new Kubernetes cluster	1365
Install Build profile	1366
More information about how Build profile uses the configuration	1366
Configure developer namespaces	1366
Exporting SCST - Store secrets to a developer namespace in a Tanzu Application Platform multicluster deployment	1366
Additional resources	1367
<b>Developer namespace setup for Supply Chain Security Tools - Store</b>	<b>1367</b>
Overview	1367
Single cluster - Using the Tanzu Application Platform values file	1367
Multicluster - Using SecretExport	1368
Next steps	1368
<b>Retrieve access tokens for Supply Chain Security Tools - Store</b>	<b>1368</b>
Overview	1368
Retrieving the read-write access token	1368
Retrieving the read-only access token	1368

Using an access token	1368
Additional Resources	1369
<b>Retrieve and create service accounts for Supply Chain Security Tools - Store</b>	<b>1369</b>
Overview	1369
Create read-write service account	1369
Create a read-only service account	1370
With a default cluster role	1370
With a custom cluster role	1371
Additional Resources	1371
<b>Create a service account with a custom cluster role for Supply Chain Security Tools - Store</b>	<b>1371</b>
Example service account	1371
Additional Resources	1372
<b>Install Supply Chain Security Tools - Store independent from Tanzu Application Platform profiles</b>	<b>1372</b>
Prerequisites	1372
Install	1373
<b>Overview of Tanzu Developer Tools for Visual Studio</b>	<b>1376</b>
Extension Features	1377
<b>Overview of Tanzu Developer Tools for Visual Studio</b>	<b>1377</b>
Extension Features	1377
<b>Install Tanzu Developer Tools for Visual Studio</b>	<b>1378</b>
Prerequisites	1378
Install	1378
Uninstall	1378
Next steps	1378
<b>Use Tanzu Developer Tools for Visual Studio</b>	<b>1379</b>
Configure settings	1379
Workload Actions	1379
Apply a workload	1379
Delete a workload	1379
Start debugging on the cluster	1379
Live Update	1380
Start Live Update	1380
Stop Live Update	1380
Tanzu Workloads panel	1380
Stop Remote Debug	1380



Extension logs	1380
<b>Troubleshoot Tanzu Developer Tools for Visual Studio</b>	<b>1381</b>
Erroneous WorkloadNotRunningState error message	1381
Symptom	1381
Solution	1381
Live Update fails to update remote app	1381
Symptom	1381
Cause	1381
Solution	1381
Delete workload command fails to delete workload	1382
Symptom	1382
Cause	1382
Solution	1382
Live Update does not work with the Jammy ClusterBuilder	1382
Symptom	1382
Solution	1382
Frequent application restarts	1382
Symptom	1382
Cause	1382
Solution	1382
<b>Overview of Tanzu Developer Tools for VS Code</b>	<b>1383</b>
Extension features	1383
<b>Overview of Tanzu Developer Tools for VS Code</b>	<b>1383</b>
Extension features	1383
<b>Install Tanzu Developer Tools for your VS Code</b>	<b>1384</b>
Prerequisites	1384
Install	1384
Configure	1385
Uninstall	1385
Next steps	1385
<b>Get started with Tanzu Developer Tools for VS Code</b>	<b>1385</b>
Prerequisite	1385
Set up Tanzu Developer Tools	1386
Create the workload.yaml file	1386
Create the catalog-info.yaml file	1387
Create the Tiltfile file	1388
Create a .tanzuignore file	1389
View an example project	1389

Next steps	1390
<b>Use Tanzu Developer Tools for VS Code</b>	<b>1390</b>
Configure for multiple projects in the workspace	1390
Apply a workload	1390
Debugging on the cluster	1391
Start debugging on the cluster	1392
Stop Debugging on the cluster	1392
Debug apps in a microservice repository	1392
Live Update	1392
Start Live Update	1393
Stop Live Update	1393
Deactivate Live Update	1394
Live Update status	1394
Live Update apps in a microservices repository	1394
Delete a workload	1395
Switch namespaces	1395
Tanzu Workloads panel	1396
Working with Microservices in a Monorepo	1396
Recommended structure: Microservices that can be built independently	1397
Alternative structure: Services with build-time interdependencies	1397
<b>Pinniped compatibility</b>	<b>1399</b>
OAuth	1399
LDAP	1399
<b>Integrate Live Hover by using Spring Boot Tools</b>	<b>1399</b>
Prerequisites	1399
Activate the Live Hover feature	1399
Deploy a Workload to the Cluster	1399
<b>Use Memory View in Spring Boot Dashboard</b>	<b>1401</b>
Prerequisites	1401
Deploy a workload	1401
View memory use in Spring Boot Dashboard	1402
<b>Troubleshoot Tanzu Developer Tools for VS Code</b>	<b>1406</b>
Unable to view workloads on the panel when connected to GKE cluster	1406
Symptom	1406
Cause	1406
Solution	1406
Live Update fails with UnsupportedClassVersionError	1406
Symptom	1406

Cause	1406
Solution	1407
Timeout error when Live Updating	1407
Symptom	1407
Cause	1407
Solution	1407
Task-related error when running a Tanzu Debug launch configuration	1407
Symptom	1407
Cause	1407
Solution	1407
Frequent application restarts	1407
Symptom	1407
Cause	1408
Solution	1408
<b>Overview of Tanzu Developer Tools for IntelliJ</b>	<b>1408</b>
Extension features	1408
Next steps	1408
<b>Overview of Tanzu Developer Tools for IntelliJ</b>	<b>1409</b>
Extension features	1409
Next steps	1409
<b>Install Tanzu Developer Tools for IntelliJ</b>	<b>1409</b>
Prerequisites	1409
Install	1410
Update	1410
Uninstall	1410
Next steps	1411
<b>Get Started with Tanzu Developer Tools for IntelliJ</b>	<b>1411</b>
Prerequisite	1411
Run Tanzu Developer Tools for IntelliJ	1411
Set up Tanzu Developer Tools	1411
Create the workload.yaml file	1412
Create the catalog-info.yaml file	1412
Create the Tiltfile file	1413
Create the .tanzuignore file	1414
View an example project	1414
Next steps	1414
<b>Use Tanzu Developer Tools for IntelliJ</b>	<b>1415</b>
Apply a workload	1415

Delete a workload	1415
Debugging on the cluster	1416
Start debugging on the cluster	1416
Stop Debugging on the Cluster	1417
Live Update	1418
Start Live Update	1418
Stop Live Update	1418
Tanzu Workloads panel	1418
Working with microservices in a monorepo	1419
Recommended structure: Microservices that can be built independently	1419
Alternative structure: Services with build-time interdependencies	1420
<b>Glossary of terms</b>	<b>1421</b>
Live Update	1421
Tiltfile	1421
Debugging on the cluster	1421
YAML file format	1422
workload.yaml file	1422
catalog-info.yaml file	1422
Code snippet	1422
Source image	1422
Local path	1422
Kubernetes context	1422
Kubernetes namespace	1422
<b>Troubleshoot Tanzu Developer Tools for IntelliJ</b>	<b>1422</b>
Tanzu Debug re-applies the workload when namespace field is empty	1423
Symptoms	1423
Cause	1423
Solution	1423
Workload is wrongly re-applied because of debug configuration selected from the launch configuration drop-down menu	1423
Symptoms	1423
Cause	1423
Solution	1423
Unable to view workloads on the panel when connected to GKE cluster	1423
Symptom	1423
Cause	1423
Solution	1423
Deactivated launch controls after running a launch configuration	1424
Symptom	1424
Cause	1424
Starting a Tanzu Debug session fails with Unable to open debugger port	1424

Symptom	1424
Cause	1424
Solution	1424
Timeout error when Live Updating	1424
Symptom	1424
Cause	1425
Solution	1425
Tanzu Panel empty when using a GKE cluster on macOS	1425
Symptom	1425
Cause	1425
Solution	1425
The Describe action in the Activity panel fails when used on PodIntent resources	1425
Symptom	1425
Cause	1425
Solution	1426
Tanzu panel shows workloads but doesn't show Kubernetes resources	1426
Symptom	1426
Cause	1426
Solution	1426
Live Update does not work with the Jammy ClusterBuilder	1426
Symptom	1426
Solution	1426
Frequent application restarts	1426
Symptom	1426
Cause	1426
Solution	1426
<b>Overview of Tanzu Application Platform GUI</b>	<b>1426</b>
<b>Overview of Tanzu Application Platform GUI</b>	<b>1428</b>
<b>Install Tanzu Application Platform GUI</b>	<b>1429</b>
Prerequisites	1429
Procedure	1429
<b>Runtime configuration options for Tanzu Application Platform GUI</b>	<b>1431</b>
Identify the Tanzu Application Platform GUI version you have available	1431
Display the possible values options for Tanzu Application Platform GUI	1431
<b>Customize the Tanzu Application Platform GUI portal</b>	<b>1432</b>
Customize branding	1432
Customize the Software Catalog page	1433
Customize the name of the organization	1433

Prevent changes to the software catalog	1433
Customize the Authentication page	1434
Customize the default view	1434
<b>Customize the Tanzu Application Platform GUI portal</b>	<b>1435</b>
Customize branding	1435
Customize the Software Catalog page	1436
Customize the name of the organization	1436
Prevent changes to the software catalog	1437
Customize the Authentication page	1437
Customize the default view	1437
<b>Customize the Support menu</b>	<b>1438</b>
Overview	1438
Customizing	1439
Structure of the support configuration	1439
URL	1439
Items	1440
Title	1440
Icon	1440
Links	1441
<b>Access Tanzu Application Platform GUI</b>	<b>1441</b>
Access with the LoadBalancer method (default)	1441
Access with the shared Ingress method	1441
<b>Catalog operations</b>	<b>1442</b>
Adding catalog entities	1442
Users and groups	1443
Systems	1444
Components	1444
Update software catalogs	1444
Register components	1445
Deregister components	1445
Add or change organization catalog locations	1445
Install demo apps and their catalogs	1446
Yelb system	1446
Install Yelb	1446
Install the Yelb catalog	1447
<b>View resources on multiple clusters in Tanzu Application Platform GUI</b>	<b>1447</b>
Set up a Service Account to view resources on a cluster	1447
Update Tanzu Application Platform GUI to view resources on multiple clusters	1450

View resources on multiple clusters in the Runtime Resources Visibility plug-in	1451
<b>Set up authentication for Tanzu Application Platform GUI</b>	<b>1452</b>
View your Backstage Identity	1452
Configure an authentication provider	1453
(Optional) Allow guest access	1455
(Optional) Customize the login page	1455
<b>View resources on remote clusters</b>	<b>1455</b>
<b>View resources on remote clusters</b>	<b>1456</b>
<b>View resources on remote EKS clusters</b>	<b>1456</b>
Set up the OIDC provider	1456
Configure the Kubernetes cluster with the OIDC provider	1457
Configure the Tanzu Application Platform GUI	1458
Upgrade the Tanzu Application Platform GUI package	1459
<b>View resources on remote GKE clusters</b>	<b>1459</b>
Leverage an external OIDC provider	1459
Set up the OIDC provider	1459
Configure the GKE cluster with the OIDC provider	1460
Configure visibility of the remote cluster	1460
Update the tap-gui package to finish leveraging the external OIDC provider	1461
Leverage Google's OIDC provider	1461
Add redirect configuration on the OIDC side	1461
Configure visibility of the remote GKE cluster	1461
Update the tap-gui package to finish leveraging the Google OIDC provider	1462
<b>View runtime resources on authorization-enabled clusters</b>	<b>1463</b>
Globally-scoped components	1463
Namespace-scoped components	1464
<b>Assign roles and permissions on Kubernetes clusters</b>	<b>1464</b>
Create roles	1465
Cluster-scoped roles	1465
Namespace-scoped roles	1465
Create users	1465
Assign users to their roles	1466
<b>Add Tanzu Application Platform GUI integrations</b>	<b>1466</b>
Add a GitHub provider integration	1466
Add a Git-based provider integration that isn't GitHub	1467
Add a non-Git provider integration	1467
Update the package profile	1467

<b>Configure the Tanzu Application Platform GUI database</b>	<b>1468</b>
Configure a PostgreSQL database	1468
Edit tap-values.yaml	1468
(Optional) Configure extra parameters	1469
Update the package profile	1469
<b>Generate and publish TechDocs</b>	<b>1470</b>
Create an Amazon S3 bucket	1470
Configure Amazon S3 access	1470
Create an AWS IAM user group	1470
Create an AWS IAM user	1471
Find the catalog locations and their entities' namespace, kind, and name	1471
Use the TechDocs CLI to generate and publish TechDocs	1471
Update the techdocs section in app-config.yaml to point to the Amazon S3 bucket	1472
<b>Overview of Tanzu Application Platform GUI plug-ins</b>	<b>1473</b>
<b>Overview of Tanzu Application Platform GUI plug-ins</b>	<b>1474</b>
<b>Runtime resources visibility in Tanzu Application Platform GUI</b>	<b>1474</b>
Prerequisite	1474
If you have a metrics server	1474
Visualize Workloads on Tanzu Application Platform GUI	1475
Navigate to the Runtime Resources Visibility screen	1475
Resources	1475
Resources details page	1476
Overview card	1477
Status card	1478
Ownership card	1479
Annotations and Labels	1479
Selecting completed supply chain pods	1480
Navigating to the pod Details page	1480
Overview of pod metrics	1480
Navigating to Application Live View	1481
Viewing pod logs	1481
Pausing and resuming logs	1482
Filtering by container	1482
Filtering by date and time	1482
Changing log levels	1482
Line wrapping	1483
Downloading logs	1483
Connection interruptions	1483



<b>Application Live View in Tanzu Application Platform GUI</b>	<b>1484</b>
Overview	1484
Entry point to Application Live View plug-in	1484
<b>Application Live View in Tanzu Application Platform GUI</b>	<b>1484</b>
Overview	1484
Entry point to Application Live View plug-in	1485
<b>Application Live View for Spring Boot applications in Tanzu Application Platform GUI</b>	<b>1485</b>
Details page	1485
Health page	1486
Environment page	1486
Log Levels page	1487
Threads page	1488
Memory page	1488
Request Mappings page	1489
HTTP Requests page	1490
Caches page	1491
Configuration Properties page	1491
Conditions page	1492
Scheduled Tasks page	1492
Beans page	1493
Metrics page	1493
Actuator page	1494
Troubleshooting	1494
<b>Application Live View for Spring Cloud Gateway applications in Tanzu Application Platform GUI</b>	<b>1494</b>
API Success Rate page	1495
API Overview page	1495
API Authentications By Path page	1495
Troubleshooting	1496
<b>Application Live View for Steeltoe applications in Tanzu Application Platform GUI</b>	<b>1496</b>
Details page	1496
Health page	1496
Environment page	1497
Log Levels page	1498
Threads page	1499
Memory page	1499
Request Mappings page	1499
HTTP Requests page	1500

Metrics page	1500
Actuator page	1501
Troubleshooting	1501
<b>Application Accelerator in Tanzu Application Platform GUI</b>	<b>1501</b>
Overview	1502
Access Application Accelerator	1502
Configure project generation	1502
Create the project	1503
Develop your code	1503
Next steps	1504
<b>Application Accelerator in Tanzu Application Platform GUI</b>	<b>1504</b>
Overview	1504
Access Application Accelerator	1504
Configure project generation	1505
Create the project	1506
Develop your code	1506
Next steps	1506
<b>Install Application Accelerator</b>	<b>1507</b>
Prerequisites	1507
Configure properties and resource use	1507
Install	1508
Troubleshooting	1510
Verify installed packages	1510
Look at resource events	1510
<b>Create an Application Accelerator Git repository during project creation</b>	<b>1511</b>
Overview	1511
Supported Providers	1511
Configure	1511
(Optional) Deactivate Git repository creation in the Application Accelerator extension for VS Code	1512
Create a Project	1512
<b>API documentation plug-in in Tanzu Application Platform GUI</b>	<b>1513</b>
Overview	1513
Use the API documentation plug-in	1514
Create a new API entry	1515
Manually create a new API entry	1516
Automatically create a new API entry	1517
<b>API documentation plug-in in Tanzu Application Platform GUI</b>	<b>1517</b>

Overview	1517
Use the API documentation plug-in	1518
Create a new API entry	1520
Manually create a new API entry	1520
Automatically create a new API entry	1521
<b>Get started with the API documentation plug-in</b>	<b>1521</b>
API entries	1521
About API entities	1521
Add a demo API entity to the Tanzu Application Platform GUI software catalog	1521
Update your demo API entry	1524
Validation Analysis of API specifications	1525
About the Validation Analysis card	1525
Automatic OpenAPI specification validation	1526
<b>Security Analysis in Tanzu Application Platform GUI</b>	<b>1526</b>
Overview	1526
Installing and configuring	1526
Accessing the plug-in	1526
Viewing vulnerability data	1527
Viewing CVE and package details	1528
<b>Supply Chain Choreographer in Tanzu Application Platform GUI</b>	<b>1529</b>
Overview	1529
Prerequisites	1529
Enable CVE scan results	1530
Enable GitOps Pull Request Flow	1530
Supply Chain Visibility	1530
View Vulnerability Scan Results	1531
<b>Overview of enabling TLS for Tanzu Application Platform GUI</b>	<b>1532</b>
Concepts	1532
Certificate delegation	1532
cert-manager, certificates, and ClusterIssuers	1532
Guides	1533
<b>Overview of enabling TLS for Tanzu Application Platform GUI</b>	<b>1533</b>
Concepts	1533
Certificate delegation	1534
cert-manager, certificates, and ClusterIssuers	1534
Guides	1535
<b>Configure a TLS certificate by using an existing certificate</b>	<b>1535</b>
Prerequisites	1535

Procedure	1536
<b>Configure a TLS certificate by using a self-signed certificate</b>	<b>1537</b>
Prerequisite	1537
Procedure	1537
<b>Configure a TLS certificate by using cert-manager and a ClusterIssuer</b>	<b>1538</b>
Prerequisites	1539
Procedure	1539
<b>Upgrade Tanzu Application Platform GUI</b>	<b>1541</b>
Considerations	1541
Upgrade within a Tanzu Application Platform profile	1541
Upgrade Tanzu Application Platform GUI individually	1541
<b>Troubleshoot Tanzu Application Platform GUI</b>	<b>1542</b>
General issues	1542
Tanzu Developer Portal reports that the port range is not valid	1542
Symptom	1542
Cause	1542
Solution	1542
Tanzu Application Platform GUI does not load the catalog	1543
Symptom	1543
Cause	1543
Solution	1543
Updating a supply chain causes an error (Can not create edge...)	1544
Symptom	1544
Solution	1544
Catalog not found	1544
Symptom	1544
Cause	1544
Solution	1544
Issues updating the values file	1545
Symptom	1545
Solution	1545
Pull logs from Tanzu Application Platform GUI	1545
Symptom	1545
Solution	1546
Runtime Resources tab issues	1546
Error communicating with Tanzu Application Platform web server	1546
Symptom	1546
Causes	1546
Solution	1546

No data available	1546
Symptom	1546
Cause	1546
Solution	1546
Errors retrieving resources	1546
Symptom	1547
Accelerators page issues	1547
No accelerators	1547
Symptom	1547
Cause	1547
Solution	1547
Security Analysis plug-in issues	1548
Empty dashboard after upgrading from Tanzu Application Platform v1.3	1548
Symptom	1548
Cause	1548
Solution	1548
Supply Chain Choreographer plug-in issues	1548
An error occurred while loading data from the Metadata Store	1548
Symptom	1548
Cause	1549
Solution	1549
<b>Overview of Tanzu Application Platform Telemetry</b>	<b>1549</b>
Tanzu Application Platform usage reports	1549
<b>Overview of Tanzu Application Platform Telemetry</b>	<b>1551</b>
Tanzu Application Platform usage reports	1551
<b>Install Tanzu Application Platform Telemetry</b>	<b>1553</b>
Prerequisites	1553
Install	1553
<b>Deployment details and configurations of Tanzu Application Platform Telemetry</b>	<b>1555</b>
What is deployed	1555
Deployment configuration	1555
<b>Overview of Tanzu Build Service</b>	<b>1555</b>
Overview	1556
<b>Overview of Tanzu Build Service</b>	<b>1556</b>
Overview	1556
<b>Install Tanzu Build Service</b>	<b>1556</b>

Before you begin	1556
Prerequisites	1556
Deprecated Features	1557
Install the Tanzu Build Service package	1557
(Optional) Alternatives to plaintext registry credentials	1559
Use Secret references for registry credentials	1559
Use AWS IAM authentication for registry credentials	1560
Install full dependencies	1561
(Optional) Configure automatic dependency updates	1562
(Optional) Deactivate the CNB BOM format	1562
<b>Install Tanzu Build Service on an air-gapped environment</b>	<b>1563</b>
Before you begin	1563
Prerequisites	1563
Deprecated Features	1563
Install the Tanzu Build Service package	1563
Install the Tanzu Build Service dependencies	1564
<b>Configure Tanzu Build Service properties on a workload</b>	<b>1565</b>
Overview	1565
Configure build-time service bindings	1565
Configure environment variables	1566
Configure the service account	1566
Configure the cluster builder	1566
Configure the workload container image registry	1567
Configure custom CA certificates for a single workload using service bindings	1567
Using custom CA certificates for all workloads	1568
<b>Create a signed container image with Tanzu Build Service</b>	<b>1568</b>
Prerequisites	1568
Configure Tanzu Build Service to sign your image builds	1568
<b>Tanzu Build Service Dependencies</b>	<b>1571</b>
How dependencies are installed	1571
View installed dependencies	1571
Bionic and Jammy stacks	1572
Use Jammy stacks for a workload	1572
Default all workloads to Jammy stacks	1572
About lite and full dependencies	1572
Lite dependencies	1573
Lite dependencies: stacks	1573
Lite dependencies: buildpacks	1573
Full dependencies	1573

Full dependencies: stacks	1574
Full dependencies: buildpacks	1574
Dependency comparison	1574
About automatic dependency updates (deprecated)	1575
Descriptors (deprecated)	1575
<b>Security context constraint for OpenShift</b>	<b>1576</b>
<b>Troubleshoot Tanzu Build Service</b>	<b>1577</b>
Builds fail due to volume errors on EKS running Kubernetes v1.23	1577
Symptom	1577
Cause	1578
Solution	1578
Smart-warmer-image-fetcher reports ErrImagePull due to dockerd's layer depth limitation	1578
Symptom	1578
Cause	1578
Solution	1578
Nodes fail due to "trying to send message larger than max" error	1579
Symptom	1579
Cause	1579
Solution	1579
Build platform uses the old build cache after upgrade to new stack	1579
Symptom	1579
Solution	1579
<b>Create a GitHub build action (Alpha)</b>	<b>1579</b>
Prerequisites	1579
Procedure	1580
Developer namespace	1580
Access to Kubernetes API server	1580
Permissions Required	1580
Use the action	1582
Debugging	1582
<b>Overview of Tekton</b>	<b>1582</b>
<b>Overview of Tekton</b>	<b>1583</b>
<b>Install Tekton</b>	<b>1583</b>
Prerequisites	1583
Install Tekton Pipelines	1583
Configure a namespace to use Tekton Pipelines	1584

# Tanzu Application Platform v1.4

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

## Tanzu Application Platform overview

Tanzu Application Platform:

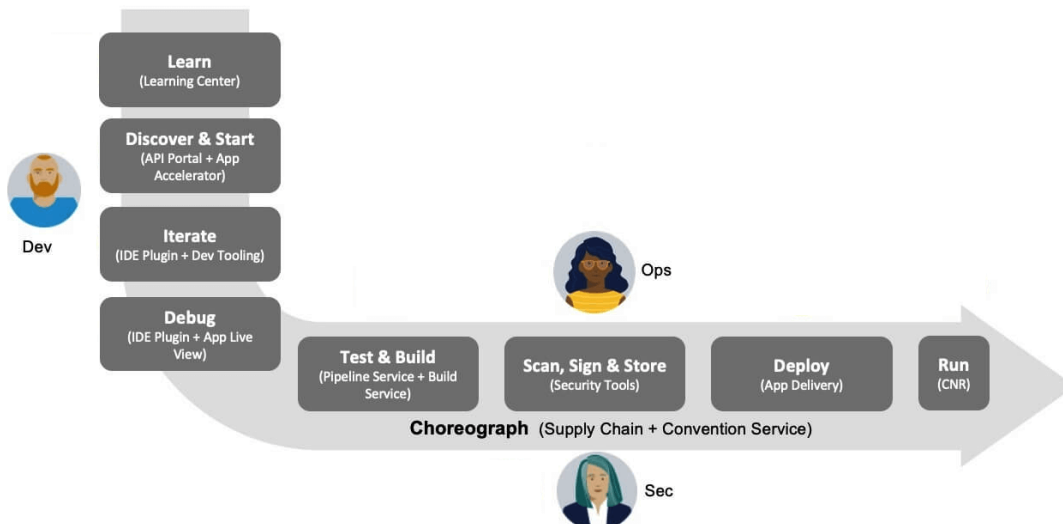
- Delivers a superior developer experience for enterprises building and deploying cloud-native applications on Kubernetes.
- Allows developers to quickly build and test applications regardless of their familiarity with Kubernetes.
- Helps application teams get to production faster by automating source-to-production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

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

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

## Simplified workflows

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





- **Inner Loop**

- The inner loop describes a developer’s development cycle of iterating on code.
- Inner loop activities include coding, testing, and debugging before making a commit.
- On cloud-native or Kubernetes platforms, developers in the inner loop often build container images and connect their apps to all necessary services and APIs to deploy them to a development environment.

- **Outer Loop**

- The outer loop describes how operators deploy apps to production and maintain them over time.
- On a cloud-native platform, outer loop activities include:
  - Building container images.
  - Adding container security.
  - Configuring continuous integration and continuous delivery (CI/CD) pipelines.
- Outer loop activities are challenging in a Kubernetes-based development environment. App delivery platforms are constructed from various third-party and open source components with numerous configuration options.

- **Supply Chains and choreography**

- Tanzu Application Platform uses the choreography pattern inherited from the context of microservices<sup>^1</sup> and applies it to CI/CD to create a path to production.<sup>^2</sup>

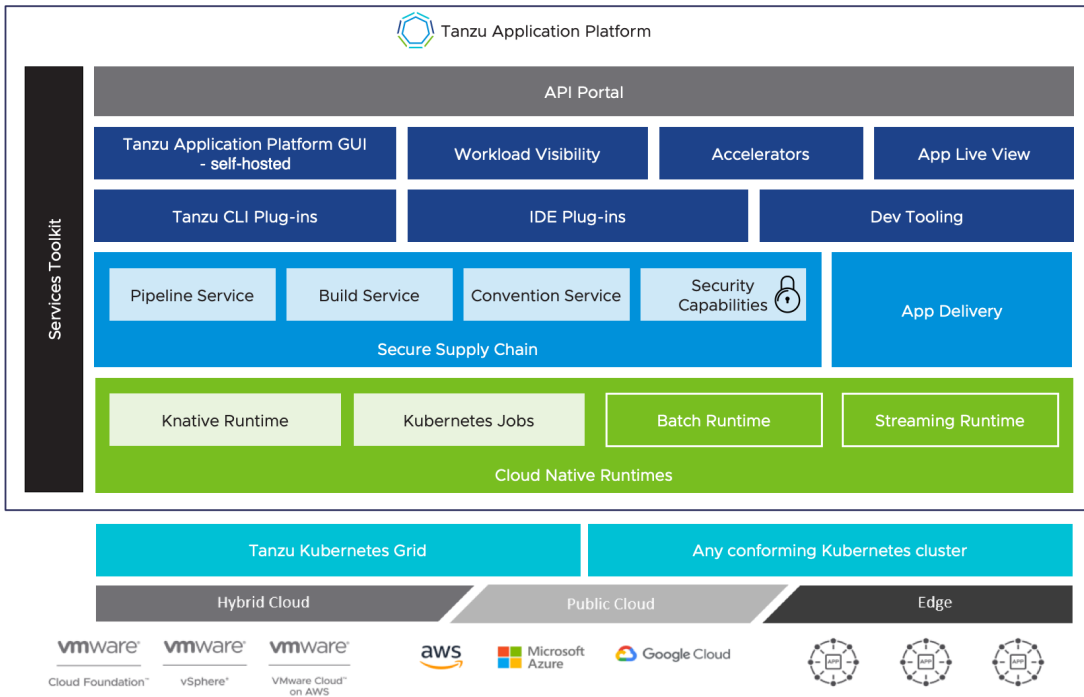
Supply chains provide a way of codifying all of the steps of your path to production, or what is more commonly known as CI/CD. A supply chain differs from CI/CD in that with a supply chain, you can add every step necessary for an application to reach production or a lower environment.



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

Instead of having separate tools that are loosely coupled to each other for testing and building, security, deploying, and running apps, a path to production defines all four tools in a single, unified layer of abstraction. Where tools typically can’t integrate with one another and additional scripting or webhooks are necessary, a unified automation tool codifies all interactions between each of the tools.

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



For more information about Tanzu Application Platform components, see [Components and installation profiles](#).

## Notice of telemetry collection for Tanzu Application Platform

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

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

# Tanzu Application Platform release notes

This topic describes the changes in Tanzu Application Platform (commonly known as TAP) v1.4.

## v1.4.13

**Release Date:** 09 January 2024

### v1.4.13 Security fixes

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

Package Name	Vulnerabilities Resolved
cnrs.tanzu.vmware.com	<ul style="list-style-type: none"><li>• CVE-2023-24534</li><li>• CVE-2023-24536</li><li>• CVE-2023-29403</li><li>• CVE-2023-44487</li></ul>

### v1.4.13 Known issues

This release introduces no new known issues.

## v1.4.12

**Release Date:** 12 December 2023

### v1.4.12 Security fixes

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

Package Name	Vulnerabilities Resolved
api-portal.tanzu.vmware.com	<p data-bbox="782 230 1002 253">▼ Expand to see the list</p> <ul data-bbox="829 264 1091 1059" style="list-style-type: none"><li data-bbox="829 264 1075 286">• GHSAs: qppj-fm5r-hxr3</li><li data-bbox="829 309 1086 331">• GHSAs: 6mjq-h674-j845</li><li data-bbox="829 353 1091 376">• GHSAs: 2wrh-6pvc-2jm9</li><li data-bbox="829 398 1018 421">• CVEs: 2023-5363</li><li data-bbox="829 443 1011 465">• CVEs: 2023-3817</li><li data-bbox="829 488 1018 510">• CVEs: 2023-3446</li><li data-bbox="829 533 1027 555">• CVEs: 2023-31484</li><li data-bbox="829 577 1018 600">• CVEs: 2023-2975</li><li data-bbox="829 622 1027 645">• CVEs: 2023-29491</li><li data-bbox="829 667 1018 689">• CVEs: 2023-2650</li><li data-bbox="829 712 1018 734">• CVEs: 2023-2603</li><li data-bbox="829 757 1018 779">• CVEs: 2023-2602</li><li data-bbox="829 801 1027 824">• CVEs: 2023-22049</li><li data-bbox="829 846 1027 869">• CVEs: 2023-22045</li><li data-bbox="829 891 1027 913">• CVEs: 2023-22044</li><li data-bbox="829 936 1027 958">• CVEs: 2023-22041</li><li data-bbox="829 981 1027 1003">• CVEs: 2023-22036</li><li data-bbox="829 1025 1027 1048">• CVEs: 2023-22006</li><li data-bbox="829 1070 1011 1093">• CVEs: 2023-1255</li></ul>

---

Package Name	Vulnerabilities Resolved
buildservice.tanzu.vmware.com	<p data-bbox="766 224 1267 257">▼ Expand to see the list</p> <ul data-bbox="766 257 1267 2042" style="list-style-type: none"><li data-bbox="766 257 1267 291">• GHSA-m8cg-xc2p-r3fc</li><li data-bbox="766 291 1267 324">• GHSA-hp87-p4gw-j4gq</li><li data-bbox="766 324 1267 358">• GHSA-g2j6-57v7-gm8c</li><li data-bbox="766 358 1267 392">• GHSA-4374-p667-p6c8</li><li data-bbox="766 392 1267 425">• CVE-2023-5981</li><li data-bbox="766 425 1267 459">• CVE-2023-5363</li><li data-bbox="766 459 1267 492">• CVE-2023-5197</li><li data-bbox="766 492 1267 526">• CVE-2023-4921</li><li data-bbox="766 526 1267 560">• CVE-2023-4911</li><li data-bbox="766 560 1267 593">• CVE-2023-4881</li><li data-bbox="766 593 1267 627">• CVE-2023-4623</li><li data-bbox="766 627 1267 660">• CVE-2023-4622</li><li data-bbox="766 660 1267 694">• CVE-2023-45871</li><li data-bbox="766 694 1267 728">• CVE-2023-44487</li><li data-bbox="766 728 1267 761">• CVE-2023-44466</li><li data-bbox="766 761 1267 795">• CVE-2023-42756</li><li data-bbox="766 795 1267 828">• CVE-2023-42755</li><li data-bbox="766 828 1267 862">• CVE-2023-42753</li><li data-bbox="766 862 1267 896">• CVE-2023-42752</li><li data-bbox="766 896 1267 929">• CVE-2023-4273</li><li data-bbox="766 929 1267 963">• CVE-2023-4244</li><li data-bbox="766 963 1267 996">• CVE-2023-4208</li><li data-bbox="766 996 1267 1030">• CVE-2023-4207</li><li data-bbox="766 1030 1267 1064">• CVE-2023-4206</li><li data-bbox="766 1064 1267 1097">• CVE-2023-4194</li><li data-bbox="766 1097 1267 1131">• CVE-2023-4155</li><li data-bbox="766 1131 1267 1164">• CVE-2023-4132</li><li data-bbox="766 1164 1267 1198">• CVE-2023-4016</li><li data-bbox="766 1198 1267 1232">• CVE-2023-3866</li><li data-bbox="766 1232 1267 1265">• CVE-2023-3865</li><li data-bbox="766 1265 1267 1299">• CVE-2023-3863</li><li data-bbox="766 1299 1267 1332">• CVE-2023-38546</li><li data-bbox="766 1332 1267 1366">• CVE-2023-38545</li><li data-bbox="766 1366 1267 1400">• CVE-2023-38432</li><li data-bbox="766 1400 1267 1433">• CVE-2023-38429</li><li data-bbox="766 1433 1267 1467">• CVE-2023-38428</li><li data-bbox="766 1467 1267 1500">• CVE-2023-38426</li><li data-bbox="766 1500 1267 1534">• CVE-2023-3817</li><li data-bbox="766 1534 1267 1568">• CVE-2023-3772</li><li data-bbox="766 1568 1267 1601">• CVE-2023-36054</li><li data-bbox="766 1601 1267 1635">• CVE-2023-35829</li><li data-bbox="766 1635 1267 1668">• CVE-2023-35828</li></ul>

Package Name	Vulnerabilities Resolved
	<ul style="list-style-type: none"><li>• CVE-2023-35824</li><li>• CVE-2023-35823</li><li>• CVE-2023-3446</li><li>• CVE-2023-34319</li><li>• CVE-2023-34256</li><li>• CVE-2023-3338</li><li>• CVE-2023-3268</li><li>• CVE-2023-32248</li><li>• CVE-2023-3212</li><li>• CVE-2023-31484</li><li>• CVE-2023-3141</li><li>• CVE-2023-31085</li><li>• CVE-2023-31084</li><li>• CVE-2023-31083</li><li>• CVE-2023-2975</li><li>• CVE-2023-29491</li><li>• CVE-2023-2898</li><li>• CVE-2023-2650</li><li>• CVE-2023-2603</li><li>• CVE-2023-2602</li><li>• CVE-2023-25775</li><li>• CVE-2023-23004</li><li>• CVE-2023-2269</li><li>• CVE-2023-2235</li><li>• CVE-2023-2163</li><li>• CVE-2023-2156</li><li>• CVE-2023-21255</li><li>• CVE-2023-2124</li><li>• CVE-2023-1255</li><li>• CVE-2023-1206</li><li>• CVE-2023-1192</li><li>• CVE-2023-0465</li><li>• CVE-2023-0464</li><li>• CVE-2022-48502</li><li>• CVE-2022-48425</li><li>• CVE-2022-40982</li><li>• CVE-2022-3996</li></ul>

---

Package Name	Vulnerabilities Resolved
cnrs.tanzu.vmware.com	<ul style="list-style-type: none"> <li>▼ Expand to see the list <ul style="list-style-type: none"> <li>• <a href="#">CVE-2023-39319</a></li> <li>• <a href="#">CVE-2023-39318</a></li> <li>• <a href="#">CVE-2023-29409</a></li> <li>• <a href="#">CVE-2023-29406</a></li> <li>• <a href="#">CVE-2023-29403</a></li> <li>• <a href="#">CVE-2023-24536</a></li> <li>• <a href="#">CVE-2023-24534</a></li> <li>• <a href="#">CVE-2023-24532</a></li> </ul> </li> </ul>
developer-conventions.tanzu.vmware.com	<ul style="list-style-type: none"> <li>▼ Expand to see the list <ul style="list-style-type: none"> <li>• <a href="#">GHSA-qppj-fm5r-hxr3</a></li> <li>• <a href="#">GHSA-4374-p667-p6c8</a></li> <li>• <a href="#">GHSA-2wrh-6pvc-2jm9</a></li> <li>• <a href="#">CVE-2023-5363</a></li> <li>• <a href="#">CVE-2023-3817</a></li> <li>• <a href="#">CVE-2023-3446</a></li> <li>• <a href="#">CVE-2023-2975</a></li> <li>• <a href="#">CVE-2023-2650</a></li> <li>• <a href="#">CVE-2023-1255</a></li> <li>• <a href="#">CVE-2023-0465</a></li> <li>• <a href="#">CVE-2023-0464</a></li> <li>• <a href="#">CVE-2022-3996</a></li> </ul> </li> </ul>
eventing.tanzu.vmware.com	<ul style="list-style-type: none"> <li>▼ Expand to see the list <ul style="list-style-type: none"> <li>• <a href="#">GHSA-m425-mq94-257g</a></li> <li>• <a href="#">GHSA-4374-p667-p6c8</a></li> <li>• <a href="#">GHSA-2wrh-6pvc-2jm9</a></li> </ul> </li> </ul>
metadata-store.apps.tanzu.vmware.com	<ul style="list-style-type: none"> <li>▼ Expand to see the list <ul style="list-style-type: none"> <li>• <a href="#">CVE-2023-5363</a></li> <li>• <a href="#">CVE-2023-4911</a></li> <li>• <a href="#">CVE-2023-3817</a></li> <li>• <a href="#">CVE-2023-3446</a></li> <li>• <a href="#">CVE-2023-2975</a></li> <li>• <a href="#">CVE-2023-1255</a></li> <li>• <a href="#">CVE-2022-3996</a></li> </ul> </li> </ul>

## v1.4.12 Known issues

This release introduces no new known issues.

## v1.4.11

**Release Date:** 14 November 2023

## v1.4.11 Security fixes

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

Package Name	Vulnerabilities Resolved
contour.tanzu.vmware.com	▼ Expand to see the list <ul style="list-style-type: none"> <li><a href="#">GHSA-qqpj-fm5r-hxr3</a></li> <li><a href="#">GHSA-m425-mq94-257g</a></li> <li><a href="#">GHSA-4374-p667-p6c8</a></li> <li><a href="#">GHSA-2wrh-6pvc-2jm9</a></li> </ul>
eventing.tanzu.vmware.com	▼ Expand to see the list <ul style="list-style-type: none"> <li><a href="#">GHSA-hp87-p4gw-j4gq</a></li> <li><a href="#">CVE-2023-2650</a></li> <li><a href="#">CVE-2023-1255</a></li> <li><a href="#">CVE-2023-0465</a></li> <li><a href="#">CVE-2023-0464</a></li> <li><a href="#">CVE-2022-3996</a></li> </ul>

## v1.4.11 Known issues

This release introduces no new known issues.

## v1.4.10

**Release Date:** 10 October 2023

## v1.4.10 Security fixes

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

Package Name	Vulnerabilities Resolved
accelerator.apps.tanzu.vmware.com	▼ Expand to see the list <ul style="list-style-type: none"> <li><a href="#">CVE-2023-43642</a></li> </ul>
buildservice.tanzu.vmware.com	▼ Expand to see the list <ul style="list-style-type: none"> <li><a href="#">CVE-2022-48064</a></li> </ul>
learningcenter.tanzu.vmware.com	▼ Expand to see the list <ul style="list-style-type: none"> <li><a href="#">CVE-2022-48064</a></li> <li><a href="#">CVE-2022-45919</a></li> <li><a href="#">CVE-2022-45887</a></li> <li><a href="#">CVE-2021-3712</a></li> </ul>
services-toolkit.tanzu.vmware.com	▼ Expand to see the list <ul style="list-style-type: none"> <li><a href="#">GHSA-hp87-p4gw-j4gq</a></li> </ul>



Package Name	Vulnerabilities Resolved
tekton.tanzu.vmware.com	▼ Expand to see the list <ul style="list-style-type: none"> <li>CVE-2022-48566</li> <li>CVE-2022-48565</li> <li>CVE-2022-48564</li> <li>CVE-2022-48560</li> <li>CVE-2022-48064</li> <li>CVE-2022-45919</li> <li>CVE-2022-45887</li> </ul>
workshops.learningcenter.tanzu.vmware.com	▼ Expand to see the list <ul style="list-style-type: none"> <li>CVE-2022-48064</li> <li>CVE-2022-45919</li> <li>CVE-2022-45887</li> <li>CVE-2021-3712</li> </ul>

## v1.4.10 Known issues

This release introduces no new known issues.

## v1.4.9

**Release Date:** 12 September 2023

### v1.4.9 Security fixes

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

Package Name	Vulnerabilities Resolved
api-portal.tanzu.vmware.com	▼ Expand to see the list <ul style="list-style-type: none"> <li>GHSA-pmhc-2g4f-85cg</li> <li>CVE-2023-34478</li> </ul>

Package Name	Vulnerabilities Resolved
buildservice.tanzu.vmware.com	<p>▼ Expand to see the list</p> <ul style="list-style-type: none"><li>• <a href="#">GHSA-69cg-p879-7622</a></li><li>• <a href="#">CVE-2023-35788</a></li><li>• <a href="#">CVE-2023-35001</a></li><li>• <a href="#">CVE-2023-3439</a></li><li>• <a href="#">CVE-2023-3390</a></li><li>• <a href="#">CVE-2023-3389</a></li><li>• <a href="#">CVE-2023-32233</a></li><li>• <a href="#">CVE-2023-3220</a></li><li>• <a href="#">CVE-2023-31436</a></li><li>• <a href="#">CVE-2023-31248</a></li><li>• <a href="#">CVE-2023-3117</a></li><li>• <a href="#">CVE-2023-3090</a></li><li>• <a href="#">CVE-2023-30456</a></li><li>• <a href="#">CVE-2023-2985</a></li><li>• <a href="#">CVE-2023-2612</a></li><li>• <a href="#">CVE-2023-25012</a></li><li>• <a href="#">CVE-2023-2283</a></li><li>• <a href="#">CVE-2023-1667</a></li><li>• <a href="#">CVE-2023-1380</a></li><li>• <a href="#">CVE-2023-0361</a></li><li>• <a href="#">CVE-2022-48303</a></li><li>• <a href="#">CVE-2022-4415</a></li><li>• <a href="#">CVE-2022-3821</a></li></ul>
learningcenter.tanzu.vmware.com	<p>▼ Expand to see the list</p> <ul style="list-style-type: none"><li>• <a href="#">GHSA-cf7p-gm2m-833m</a></li><li>• <a href="#">CVE-2023-4147</a></li><li>• <a href="#">CVE-2023-4015</a></li><li>• <a href="#">CVE-2023-4004</a></li><li>• <a href="#">CVE-2023-3995</a></li><li>• <a href="#">CVE-2023-3777</a></li><li>• <a href="#">CVE-2023-3635</a></li><li>• <a href="#">CVE-2023-3610</a></li><li>• <a href="#">CVE-2023-3609</a></li><li>• <a href="#">CVE-2022-45886</a></li></ul>
tap-gui.tanzu.vmware.com	<p>▼ Expand to see the list</p> <ul style="list-style-type: none"><li>• <a href="#">GHSA-cchq-frgv-rjh5</a></li><li>• <a href="#">GHSA-g644-9gfx-q4q4</a></li></ul>
tekton.tanzu.vmware.com	<p>▼ Expand to see the list</p> <ul style="list-style-type: none"><li>• <a href="#">CVE-2022-45886</a></li></ul>

Package Name	Vulnerabilities Resolved
workshops.learningcenter.tanzu.vmware.com	▼ Expand to see the list <ul style="list-style-type: none"> <li>• <a href="#">CVE-2023-4147</a></li> <li>• <a href="#">CVE-2023-4015</a></li> <li>• <a href="#">CVE-2023-4004</a></li> <li>• <a href="#">CVE-2023-3995</a></li> <li>• <a href="#">CVE-2023-3777</a></li> <li>• <a href="#">CVE-2023-3635</a></li> <li>• <a href="#">CVE-2023-3610</a></li> <li>• <a href="#">CVE-2023-3609</a></li> <li>• <a href="#">CVE-2022-45886</a></li> </ul>

## v1.4.9 Resolved issues

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

### v1.4.9 Resolved issues: Tanzu CLI and plug-ins

- This release includes Tanzu CLI v1.0.0 and a set of installable plug-in groups that are versioned so that the CLI is compatible with all supported Tanzu Application Platform versions. For more information see [Install Tanzu CLI](#).

## v1.4.9 Known issues

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

### v1.4.9 Known issues: Tanzu Build Service

- Tanzu Application Platform installation temporarily fails with this error related to the `deactivate_smart_warmer` key: `waiting on reconcile packageinstall/buildservice. Overlaying data values (in following order: ca-cert.yaml, cert-injection-webhook/upstream/imagevalues.yaml, cert-injection-webhook/values.yaml, additional data values)` . This issue resolves after a few minutes.

## v1.4.8

**Release Date:** 15 August 2023

### v1.4.8 Security fixes

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

Package Name	Vulnerabilities Resolved
ootb-templates.tanzu.vmware.com	<ul style="list-style-type: none"> <li>▼ Expand to see the list                             <ul style="list-style-type: none"> <li>• <a href="#">GHSA-m8cg-xc2p-r3fc</a></li> <li>• <a href="#">GHSA-hw7c-3rfg-p46j</a></li> <li>• <a href="#">GHSA-g2j6-57v7-gm8c</a></li> <li>• <a href="#">CVE-2023-31484</a></li> <li>• <a href="#">CVE-2023-3138</a></li> <li>• <a href="#">CVE-2023-29491</a></li> <li>• <a href="#">CVE-2023-29007</a></li> <li>• <a href="#">CVE-2023-2650</a></li> <li>• <a href="#">CVE-2023-2603</a></li> <li>• <a href="#">CVE-2023-2602</a></li> <li>• <a href="#">CVE-2023-25815</a></li> <li>• <a href="#">CVE-2023-25652</a></li> <li>• <a href="#">CVE-2023-2283</a></li> <li>• <a href="#">CVE-2023-1667</a></li> <li>• <a href="#">CVE-2023-1255</a></li> <li>• <a href="#">CVE-2023-0465</a></li> <li>• <a href="#">CVE-2023-0464</a></li> <li>• <a href="#">CVE-2022-3996</a></li> </ul> </li> </ul>
buildservice.tanzu.vmware.com	<ul style="list-style-type: none"> <li>▼ Expand to see the list                             <ul style="list-style-type: none"> <li>• <a href="#">CVE-2023-0466</a></li> <li>• <a href="#">CVE-2022-44034</a></li> <li>• <a href="#">CVE-2022-44033</a></li> <li>• <a href="#">CVE-2022-44032</a></li> <li>• <a href="#">CVE-2022-41848</a></li> <li>• <a href="#">CVE-2022-3114</a></li> <li>• <a href="#">CVE-2022-29458</a></li> <li>• <a href="#">CVE-2021-45261</a></li> <li>• <a href="#">CVE-2021-39801</a></li> <li>• <a href="#">CVE-2021-32078</a></li> <li>• <a href="#">CVE-2021-26934</a></li> <li>• <a href="#">CVE-2018-6952</a></li> <li>• <a href="#">CVE-2018-20657</a></li> <li>• <a href="#">CVE-2018-12928</a></li> <li>• <a href="#">CVE-2017-13693</a></li> <li>• <a href="#">CVE-2017-13165</a></li> <li>• <a href="#">CVE-2017-11164</a></li> <li>• <a href="#">CVE-2016-20013</a></li> <li>• <a href="#">CVE-2015-8985</a></li> <li>• <a href="#">CVE-2009-5155</a></li> </ul> </li> </ul>

---

## v1.4.8 Resolved issues

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

#### v1.4.8 Resolved issues: Tanzu Build Service

- Fixed an issue where some buildpacks caused the builder image to become excessively large.

#### v1.4.8 Known issues

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

#### v1.4.8 Known issues: Tanzu Build Service

- Tanzu Application Platform installation fails if the automatic dependency updater is used with a Kubernetes secret ref, that is, using the fields `buildservice.tanzunet_secret.name` and `buildservice.tanzunet_secret.name` in the `tap-values.yaml` file. For a workaround, use plaintext secrets by using the fields `buildservice.tanzunet_username` and `buildservice.tanzunet_password` in the `tap-values.yaml` file.

## v1.4.7

**Release Date:** 11 July 2023

### v1.4.7 Security fixes

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

Package Name	Vulnerabilities Resolved
sso.apps.tanzu.vmware.com	▼ Expand to see the list <ul style="list-style-type: none"> <li>• <a href="#">GHSA-m8cg-xc2p-r3fc</a></li> <li>• <a href="#">GHSA-g2j6-57v7-gm8c</a></li> <li>• <a href="#">GHSA-f3fp-gc8g-vw66</a></li> <li>• <a href="#">CVE-2023-2650</a></li> <li>• <a href="#">CVE-2023-20863</a></li> <li>• <a href="#">CVE-2023-20861</a></li> <li>• <a href="#">CVE-2023-1255</a></li> <li>• <a href="#">CVE-2023-0466</a></li> <li>• <a href="#">CVE-2023-0465</a></li> <li>• <a href="#">CVE-2023-0464</a></li> <li>• <a href="#">CVE-2022-3996</a></li> <li>• <a href="#">CVE-2022-3821</a></li> </ul>
learningcenter.tanzu.vmware.com	▼ Expand to see the list <ul style="list-style-type: none"> <li>• <a href="#">CVE-2023-2004</a></li> </ul>
workshops.learningcenter.tanzu.vmware.com	▼ Expand to see the list <ul style="list-style-type: none"> <li>• <a href="#">CVE-2023-2004</a></li> </ul>

### v1.4.7 Known issues

This release introduces no new known issues.

## v1.4.6

**Release Date:** 13 June 2023

### v1.4.6 Security fixes

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

Package Name	Vulnerabilities Resolved
api-portal.tanzu.vmware.com	▼ Expand to see the list <ul style="list-style-type: none"> <li><a href="#">CVE-2022-41881</a></li> </ul>
buildservice.tanzu.vmware.com	▼ Expand to see the list <ul style="list-style-type: none"> <li><a href="#">CVE-2023-1829</a></li> <li><a href="#">CVE-2023-1281</a></li> <li><a href="#">CVE-2023-0386</a></li> </ul>
cert-manager.tanzu.vmware.com	▼ Expand to see the list <ul style="list-style-type: none"> <li><a href="#">GHSA-vvpx-j8f3-3w6h</a></li> </ul>
sso.apps.tanzu.vmware.com	▼ Expand to see the list <ul style="list-style-type: none"> <li><a href="#">CVE-2023-31484</a></li> </ul>
tap-gui.tanzu.vmware.com	▼ Expand to see the list <ul style="list-style-type: none"> <li><a href="#">GHSA-f9xv-q969-pqx4</a></li> </ul>

### v1.4.6 Resolved issues

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

#### v1.4.6 Resolved issues: Tanzu Developer Tools for IntelliJ

- Resolved permission-denied errors encountered during Live Update when operating against platforms configured to use the Jammy build stack.

#### v1.4.6 Resolved issues: Tanzu Developer Tools for Visual Studio

- Resolved permission-denied errors encountered during Live Update when operating against platforms configured to use the Jammy build stack.

#### v1.4.6 Resolved issues: Tanzu Developer Tools for VS Code

- Resolved permission-denied errors encountered during Live Update when operating against platforms configured to use the Jammy build stack.

### v1.4.6 Known issues

This release introduces no new known issues.

## v1.4.5

**Release Date:** 09 May 2023

## v1.4.5 Security fixes

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

Package Name	Vulnerabilities Resolved
accelerator.apps.tanzu.vmware.com	▼ Expand to see the list <ul style="list-style-type: none"> <li><a href="#">CVE-2023-21930</a></li> </ul>
api-portal.tanzu.vmware.com	▼ Expand to see the list <ul style="list-style-type: none"> <li><a href="#">GHSA-493p-pfq6-5258</a></li> <li><a href="#">CVE-2023-21930</a></li> <li><a href="#">CVE-2023-20860</a></li> </ul>
sso.apps.tanzu.vmware.com	▼ Expand to see the list <ul style="list-style-type: none"> <li><a href="#">CVE-2023-21930</a></li> </ul>
buildservice.tanzu.vmware.com	▼ Expand to see the list <ul style="list-style-type: none"> <li><a href="#">CVE-2023-0468</a></li> <li><a href="#">CVE-2023-0461</a></li> <li><a href="#">CVE-2023-0179</a></li> </ul>
ootb-templates.tanzu.vmware.com	▼ Expand to see the list <ul style="list-style-type: none"> <li><a href="#">GHSA-vvpx-j8f3-3w6h</a></li> </ul>

## v1.4.4

Release Date: 12 April 2023

## v1.4.4 Security fixes

This release has the following security fixes, listed by package name and vulnerabilities.

Package Name	Vulnerabilities Resolved
buildservice.tanzu.vmware.com	▼ Expand to see the list <ul style="list-style-type: none"> <li><a href="#">GHSA-fxg5-wq6x-vr4w</a></li> </ul>
eventing.tanzu.vmware.com	▼ Expand to see the list <ul style="list-style-type: none"> <li><a href="#">GHSA-fxg5-wq6x-vr4w</a></li> <li><a href="#">GHSA-69ch-w2m2-3vjp</a></li> <li><a href="#">GHSA-69cg-p879-7622</a></li> </ul>

Package Name	Vulnerabilities Resolved
learningcenter.tanzu.vmware.com	<ul style="list-style-type: none"> <li>▼ Expand to see the list               <ul style="list-style-type: none"> <li>• <a href="#">GHSA-x4qr-2fvf-3mr5</a></li> <li>• <a href="#">GHSA-ppp9-7jff-5vj2</a></li> <li>• <a href="#">GHSA-fxg5-wq6x-vr4w</a></li> <li>• <a href="#">GHSA-83g2-8m93-v3w7</a></li> <li>• <a href="#">GHSA-69ch-w2m2-3vjp</a></li> <li>• <a href="#">GHSA-3vm4-22fp-5rfm</a></li> <li>• <a href="#">GHSA-2hrw-hx67-34x6</a></li> <li>• <a href="#">CVE-2023-24329</a></li> <li>• <a href="#">CVE-2023-23919</a></li> <li>• <a href="#">CVE-2023-0461</a></li> </ul> </li> </ul>
policy.apps.tanzu.vmware.com	<ul style="list-style-type: none"> <li>▼ Expand to see the list               <ul style="list-style-type: none"> <li>• <a href="#">GHSA-fxg5-wq6x-vr4w</a></li> </ul> </li> </ul>
snyk.scanning.apps.tanzu.vmware.com	<ul style="list-style-type: none"> <li>▼ Expand to see the list               <ul style="list-style-type: none"> <li>• <a href="#">CVE-2023-24329</a></li> </ul> </li> </ul>
tap-gui.tanzu.vmware.com	<ul style="list-style-type: none"> <li>▼ Expand to see the list               <ul style="list-style-type: none"> <li>• <a href="#">CVE-2023-23919</a></li> <li>• <a href="#">CVE-2023-23918</a></li> <li>• <a href="#">CVE-2023-0361</a></li> <li>• <a href="#">CVE-2023-0286</a></li> <li>• <a href="#">CVE-2023-0215</a></li> <li>• <a href="#">CVE-2022-4450</a></li> </ul> </li> </ul>
workshops.learningcenter.tanzu.vmware.com	<ul style="list-style-type: none"> <li>▼ Expand to see the list               <ul style="list-style-type: none"> <li>• <a href="#">GHSA-ppp9-7jff-5vj2</a></li> <li>• <a href="#">GHSA-fxg5-wq6x-vr4w</a></li> <li>• <a href="#">GHSA-83g2-8m93-v3w7</a></li> <li>• <a href="#">GHSA-69ch-w2m2-3vjp</a></li> <li>• <a href="#">GHSA-3vm4-22fp-5rfm</a></li> <li>• <a href="#">CVE-2023-24329</a></li> <li>• <a href="#">CVE-2023-23919</a></li> <li>• <a href="#">CVE-2023-0461</a></li> </ul> </li> </ul>

## v1.4.4 Resolved issues

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

### v1.4.4 Resolved issues: Grype Scanner

- Updated Syft version to fix image scans failing with panic. This fixes an issue that caused the scanner to fail with an index out of range error. This happened when parsing APK metadata to identify the installed OS packages if a package's list of provided files is empty.

### v1.4.4 Resolved issues: Source Controller



- Updated imgpkg API to v0.36.0 to fix file permissions after extracting the source tarball. File permissions were stripped from source files while using IMGPKG v0.25.0. This issue is fixed in IMGPKG v0.29.0 and later.

## v1.4.4 Known issues

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

### v1.4.4 Known issues: API Auto Registration

- Users cannot update their APIs through API Auto Registration due to a issue with the ID used to retrieve APIs. This issue causes errors in the API Descriptor CRD similar to the following: `Unable to find API entity's uid within TAP GUI. Retrying the sync.`

### v1.4.4 Known issues: Grype Scanner

- **Scanning Java source code that uses Gradle package manager might not reveal vulnerabilities:**

For most languages, source code scanning only scans files present in the source code repository. Except for support added for Java projects using Maven, no network calls fetch dependencies. For languages using dependency lock files, such as go and Node.js, Grype uses the lock files to verify dependencies for vulnerabilities.

For Java using Gradle, dependency lock files are not guaranteed, so Grype uses dependencies present in the built binaries, such as `.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.

## v1.4.2

**Release Date:** 06 March 2023

### v1.4.2 Security fixes

This release has the following security fixes, listed by package name and vulnerabilities.

- **accelerator.apps.tanzu.vmware.com:** GHSA-69cg-p879-7622 and [CVE-2023-0286](#)
- **api-portal.tanzu.vmware.com:** [CVE-2023-0286](#)
- **apis.apps.tanzu.vmware.com:** [CVE-2023-0286](#)
- **buildservice.tanzu.vmware.com:** GHSA-69cg-p879-7622, [CVE-2023-0286](#), and GHSA-69ch-w2m2-3vjp
- **cartographer.tanzu.vmware.com:** [CVE-2023-0286](#) and GHSA-fxg5-wq6x-vr4w
- **cert-manager.tanzu.vmware.com:** GHSA-69cg-p879-7622, GHSA-69ch-w2m2-3vjp, and GHSA-83g2-8m93-v3w7
- **cnrs.tanzu.vmware.com:** GHSA-69cg-p879-7622, [CVE-2023-0286](#), GHSA-fxg5-wq6x-vr4w, and GHSA-69ch-w2m2-3vjp
- **controller.conventions.apps.tanzu.vmware.com:** GHSA-69cg-p879-7622, [CVE-2023-0286](#), GHSA-fxg5-wq6x-vr4w, and GHSA-69ch-w2m2-3vjp
- **controller.source.apps.tanzu.vmware.com:** [CVE-2023-0286](#) and GHSA-fxg5-wq6x-vr4w

- **conventions.appliveview.tanzu.vmware.com:** GHSA-69cg-p879-7622 and GHSA-fxg5-wq6x-vr4w
- **developer-conventions.tanzu.vmware.com:** [CVE-2023-0286](#)
- **eventing.tanzu.vmware.com:** [CVE-2023-0286](#)
- **external-secrets.apps.tanzu.vmware.com:** [CVE-2023-0286](#)
- **fluxcd.source.controller.tanzu.vmware.com:** [CVE-2023-0286](#)
- **metadata-store.apps.tanzu.vmware.com:** GHSA-69cg-p879-7622, [CVE-2023-0286](#), GHSA-fxg5-wq6x-vr4w, GHSA-69ch-w2m2-3vjp, GHSA-8c26-wmh5-6g9v, and GHSA-r48q-9g5r-8q2h
- **namespace-provisioner.apps.tanzu.vmware.com:** GHSA-fxg5-wq6x-vr4w
- **ootb-templates.tanzu.vmware.com:** GHSA-69cg-p879-7622, [CVE-2023-0286](#), GHSA-fxg5-wq6x-vr4w, GHSA-69ch-w2m2-3vjp, GHSA-3vm4-22fp-5rfm, GHSA-8c26-wmh5-6g9v, GHSA-gwc9-m7rh-j2ww, GHSA-83g2-8m93-v3w7, and GHSA-ppp9-7jff-5vj2
- **policy.apps.tanzu.vmware.com:** [CVE-2023-0286](#)
- **services-toolkit.tanzu.vmware.com:** GHSA-69cg-p879-7622 and GHSA-fxg5-wq6x-vr4w
- **spring-boot-conventions.tanzu.vmware.com:** GHSA-69cg-p879-7622 and GHSA-fxg5-wq6x-vr4w
- **sso.apps.tanzu.vmware.com:** [CVE-2023-0286](#), [CVE-2022-4450](#), and [CVE-2023-0215](#)
- **tekton.tanzu.vmware.com:** [CVE-2023-0286](#), [CVE-2022-45061](#), [CVE-2022-42703](#), and [CVE-2022-4378](#)

## v1.4.2 Resolved issues

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

### v1.4.2 Resolved issues: Tanzu Build Service

- Fixed an issue that prevented the Cloud Native Buildpacks lifecycle component from upgrading with Tanzu Build Service.
  - Outdated lifecycle components can be built with older versions of Golang containing CVEs in the standard library.
  - Upgrading to Tanzu Application Platform v1.4.2 ensures the lifecycle component is updated to the latest version.

## v1.4.2 Known issues

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

### v1.4.2 Known issues: Grype scanner

- **Scanning Java source code that uses Gradle package manager might not reveal vulnerabilities:**

For most languages, source code scanning only scans files present in the source code repository. Except for support added for Java projects using Maven, no network calls fetch dependencies. For languages using dependency lock files, such as Golang and Node.js, Grype uses the lock files to check dependencies for vulnerabilities.

For Java using Gradle, dependency lock files are not guaranteed, so Grype uses dependencies present in the built binaries, such as `.jar` or `.war` files.

Because VMware discourages committing binaries to source code repositories, Gype 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.

- **Scanning some Alpine-based container images fails with a panic:**

An issue in Syft causes the scanner to crash with index out of range, while parsing APK metadata to identify installed OS packages if a package's list of provided files is empty. This problem is resolved in SCST - Scan (Gype) version [1.4.1](#) or in the Tanzu Application Platform version [1.5.0+](#).

---

## v1.4.1

**Release Date:** 16 February 2023

### v1.4.1 Security fixes

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

#### v1.4.1 Security fixes: Tanzu Application Platform GUI

- [GHSA-3xq5-wjfh-ppjc](#)
  - [GHSA-36fh-84j7-cv5h](#)
- 

### v1.4.1 Resolved issues

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

#### v1.4.1 Resolved issues: Source Controller

- Fixed an issue that caused some registries, including DockerHub, to incur higher than expected pulls because all HTTP GET calls are considered to be pulls. With this fix, HTTP requests use HEAD operations instead of GET operations, which reduces the number of pulls while checking updated image versions.

#### v1.4.1 Resolved issues: Tanzu Application Platform GUI

- Fixed SVG icons that appeared overly large on the sidebar.
- Added catalog graph cards and diagram defaults to align with upstream Backstage.

#### v1.4.1 Resolved issues: Tanzu Application Platform plug-ins

##### API Validation and Scoring Toolkit plug-in

- Better error-handling for when a scoring value is missing in the API YAML.
- Adjusted some styles of the components to meet requirements.

##### Application Accelerator plug-in

- Fixed the rendering of options that share an identical `dependsOn` array value. Added system property configuration for the Git repository creation feature.
- Added a workflow scope to the Git repository creation feature.

##### Application Live View plug-in

- Fixed the reset button in the root logger of the Application Live View log levels page.

#### Out of the Box Supply Chain Templates plug-in

- Fixed deliverable content written into ConfigMaps in a multicluster setup. Added labels to attribute the deliverable content with the supply chain and the template. This was done to be consistent with the delivery on a non-Build profile cluster. For more information, see [Multicluster Tanzu Application Platform overview](#).

#### Security Analysis plug-in

- Updated the data model for the response of `metadata-store`.
- Changed the table's position on the index page.
- Updated the filter for workloads with no associated policy.
- Updated the bar graph for workloads with big and small values for different severities.
- Fixed a discrepancy between the widget and the information in **Workload Build Vulnerabilities**.

#### Supply Chain Choreographer plug-in

- The **Generation** box now shows the correct amended scan policy version. Clicking the scan policy link displays the amended policy.
- The **Approve a Request** button now appears in the **Stage Details** section of the Supply Chain view when the **Config Writer** stage is selected and the GitOps PR flow is configured.
- When an error occurs and the scan policy documentation link appears, the link now targets the latest version of the Tanzu Application Platform documentation.

#### Supply Chain Security Tools plug-in

- Fixed view approval failing to display in the **Config Writer** stage.
- Fixed the check box status in the Table filter.
- Updated the Scan policy documentation URL.
- Fixed the Generation number displayed after the scan policy is updated.

---

## v1.4.1 Known issues

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

### v1.4.1 Known issues: Grype scanner

- **Scanning Java source code that uses Gradle package manager might not reveal vulnerabilities:**

For most languages, source code scanning only scans files present in the source code repository. Except for support added for Java projects using Maven, no network calls fetch dependencies. For languages using dependency lock files, such as Golang and Node.js, Grype uses the lock files to check dependencies for vulnerabilities.

For Java using Gradle, dependency lock files are not guaranteed, so Grype uses dependencies present in the built binaries, such as `.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.

- **Scanning some Alpine-based container images fails with a panic:**

An issue in Syft causes the scanner to crash with index out of range, while parsing APK metadata to identify installed OS packages if a package's list of provided files is empty. This problem is resolved in SCST - Scan (Grype) version [1.4.1](#) or in the Tanzu Application Platform version [1.5.0+](#).

#### v1.4.1 Known issues: Security Analysis GUI

- After upgrading to Tanzu Application Platform v1.4 from v1.3, the Security Analysis GUI dashboard might appear empty because the dashboard now displays information from the Metadata Store. To repopulate the dashboard, see [Troubleshooting](#).

## v1.4.0

**Release Date:** 10 January 2023

### v1.4.0 Tanzu Application Platform new features

This release includes the following platform-wide enhancements.

- [Shared Ingress Issuer](#) for secure ingress communication by default. [CNRs](#), [AppSSO](#), and [Tanzu Application Platform GUI](#) use this issuer to secure ingress. Over future releases, VMware plans to incrementally update all Tanzu Application Platform components to support the shared ingress issuer.
- [Namespace Provisioner](#) provides a secure, automated way for Platform Operators to provision namespaces with the resources and proper namespace-level privileges required for their workloads to function as intended.
- [Tanzu Application Platform Telemetry Reports](#) offers the option to enroll in a usage reporting program that provides a usage summary of your Tanzu Application Platform.
- [Tanzu Developer Tools for Visual Studio](#) is an IDE extension for Visual Studio to help you develop, providing the rapid iteration experience for .NET Core apps in Tanzu Application Platform.

### v1.4.0 New features by component and area

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

#### v1.4.0 Features: API Validation and Scoring Toolkit

- API Validation and Scoring focuses on scanning and validating an OpenAPI specification. The API specification is generated from the API Auto Registration of Tanzu Application Platform. See [API Validation and Scoring](#) for more information.

#### v1.4.0 Features: Application Single Sign-On (AppSSO)

- Added ability to configure custom Redis storage for an [AuthServer](#) by using a ProvisionedService-style API. For more information, see [Storage](#).
- Added package field `default_authserver_clusterissuer` that inherits the `shared.ingress_issuer` value from Tanzu Application Platform if not set. For more

information, see [IssuerURI](#) and [TLS](#).

- Added `AuthServer.spec.tls.deactivated` to deprecate `AuthServer.spec.tls.disabled`.
- `AuthServer.spec.tokenSignatures` is now a required field.
- In addition to globally trusted CA certificates, granular trust can be extended with `AuthServer.spec.caCerts`.
- LDAP is now a supported identity provider protocol. For more information, see [LDAP](#).
  - LDAP bind is validated on `AuthServer` creation when an LDAP identity provider is defined.
  - Introduced `identityProviders.ldap.url` in `AuthServer.spec`.
  - Introduced `identityProviders.ldap.group.search`.
  - `identityProviders.ldap.group` is now optional in `AuthServer.spec`.

#### v1.4.0 Features: Application Accelerator

- Optional Git repository creation during project generation is supported in the Application Accelerator extension for VS Code.
  - The parameter `.app_config.gitProviders.active` in `tap-values.yaml` was added to deactivate optional Git repository creation in the VS Code extension during project creation.
- Added `custom types`, which allows for 1-N number of repeating sets of options in an accelerator's form during project creation.
- Added the `Loop transform` to allow for transforms to be applied on list options.
- Added `generate-from-local` command to the Application Accelerator plug-in for the Tanzu CLI to generate projects from local assets without needing to commit code.
- Additional Application Accelerator plug-ins for Tanzu CLI improvements:
  - `fragment create` now supports `--local-path` and `--source-image`.
  - `fragment update` now supports `--source-image`.
- Application Accelerator Samples:
  - **Accelerators** - the following accelerators were added:
    - [Angular Frontend Accelerator](#)
      - Quickly bootstrap an Angular-based Single Page App.
    - [React Frontend Accelerator](#)
      - Quickly bootstrap a React-based Single Page App.
  - **Fragments** - the following fragments were added:
    - [Spring Boot H2 Fragment](#)
    - [Spring Boot MySQL Fragment](#)
    - [Spring Boot PostgreSQL Fragment](#)

#### v1.4.0 Features: Application Live View

- Users can now activate or deactivate the automatic configuration of Spring Boot actuators on Tanzu Application Platform and on workloads. For more information, see [Configure and access Spring Boot actuators in Tanzu Application Platform](#).

- Added Application Live View Memory View panel in Visual Studio Code as part of Spring Boot Dashboard extension.
- Added support for Spring Boot 3. Spring Boot 3 Native Image is not supported.
- Added new Application Live View pages [HTTP Requests](#) and [Request Mappings](#) for Steeltoe workloads.
- Added `appliveview_connector.backend.sslDeactivated` to deprecate `appliveview_connector.backend.sslDisabled`.

#### v1.4.0 Features: Apps plug-in for Tanzu CLI

- Added `--update-strategy` flag to allow you to change `tanzu apps workload apply` behavior when contents from file are applied. See [How-to-guides](#) section for use and examples.
- Added ability to pass URL for `--file` flag.
- Show fully qualified resource name in the resources column of Supply chain and Delivery sections of the `tanzu apps workload get` command. Example output is found in [tanzu apps workload get](#) command description.
- Added new shorthand flag aliases: `-a` for `--app`, `-e` for `--env`, `-i` for `--image`, `-l` for `label`, `-p` for `--param`, and `-t` for `--type`. For more information, see [Tanzu apps workload apply flags list](#).
- Added emojis to `tanzu apps workload create/apply/delete` commands.
- Do not print emojis when `--no-color` flag is set.
- Added namespace to `tanzu apps workload get` command's overview section.
- Added progress bar to provide feedback to users when uploading source code to registry.
- Removed color from tail command output when `--no-color` flag is passed.

#### v1.4.0 Features: cert-manager

- `cert-manager.tap.tanzu.vmware.com` can optionally install self-signed `ClusterIssuerS`.

#### v1.4.0 Features: Eventing

- Upgraded Knative Eventing version from 1.6 to 1.8.
- Added a Kubernetes tracing attribute to `ApiServerSource`.
- The `ApiServerSource` is a Knative Eventing Kubernetes custom resource that listens for events emitted by the Kubernetes API server. For example, pod creation, deployment updates, and so on. It then forwards them as `CloudEvents` to a sink.

#### v1.4.0 Features: External Secrets Operator (alpha)

- The External Secrets Operator is a Kubernetes operator that integrates with external secret management systems. With this release, Tanzu Application Platform repackages this open source Kubernetes operator into a Carvel bundle that ships with Tanzu Application Platform. **External Secrets Operator is currently in alpha and is intended for evaluation and test purposes only. Do not use in a production environment.**

#### v1.4.0 Features: Services Toolkit

- Added new `ClassClaim` API that allows claims for service instances to be created by referring to a `ClusterInstanceClass`. For more information, see [When to use ClassClaim vs](#)

[ResourceClaim](#) and [Introducing different service implementations in different environments](#).

- Added corresponding `tanzu services class-claims` CLI plug-in command.
- Added support for OpenShift v4.11.
- Added support for Kubernetes v1.25.
- All containers are now configured with read-only root file systems.

#### v1.4.0 Features: Tanzu Application Platform GUI plug-ins

##### Security Analysis Plug-in

- **Understand the total number of affected packages and vulnerabilities on the Security Analysis Dashboard:** The Vulnerabilities by Severity widget and the Workload Build Vulnerabilities table have updated logic to count all CVEs and packages, providing a better idea of the discrete, affected packages. Previously, the logic counted unique CVEs, even if a particular CVE affected multiple packages.
- **Quickly identify all affected workloads for a specific CVE, package, or dependency:** The CVE and Package details pages include a new table that shows all affected workloads for a specific CVE or package. You access the CVE and Package details page on the source or image scan stage in the Supply Chain Choreographer Plug-in.

##### Supply Chain Choreographer plug-in

- Events are now emitted when resources are applied and when their output or health status changes. See [Events reference](#).
- Source Tester stage now includes a link to the Jenkins job when Jenkins is configured for use in the supply chain.
- `spec.source.git.url` is added to the Overview section of the Source Provider stage in the supply chain.
- Added support to include current and historical Kaniko build logs in the Stage Details section of the supply chain when Kaniko is used as the build service in the Image Provider stage.
- Scanning stages now include a `Show Unique CVEs` filter so that the scan results show one CVE per ID as opposed to each CVE per package. This allows better alignment between the data in the Supply Chain Choreographer plug-in and the Security Analysis plug-in.

#### v1.4.0 Features: Supply Chain Security Tools - Policy

- Added ability to [configure action when no policy matches the admitting image digest](#).
- Added ability to [allow empty authorities for ClusterImagePolicies](#).
- Added ability to [specify which resources a ClusterImagePolicy should enforce](#).

#### v1.4.0 Features: Supply Chain Security Tools - Scan

- Users no longer need to create a package overlay to enable Grype in offline and air-gapped environments. See [Using Grype in offline and air-gapped environments](#).
- Increased compatibility with customers' existing environments by supporting custom certificate authorities (CAs) by using the `tap-values.yml` for both Grype and Snyk scanners.
- Alpha release of Prisma Scanner integration. See [Install Prisma Scanner](#).



#### v1.4.0 Features: Tanzu Developer Tools for IntelliJ

- The developer sandbox enables developers to Live Update their code and simultaneously debug the updated code, without having to deactivate Live Update when debugging.
- An Activity pane was added in the Tanzu Panel that allows developers to visualize the supply chain, delivery, and running application pods. It displays detailed error messages on each resource and enables developers to describe and view logs on these resources from within their IDE.
- Tanzu workload `apply` and `delete` actions were added to IntelliJ.
- Code snippets to create `workload.yaml` and `catalog-info.yaml` files were added to IntelliJ.

#### v1.4.0 Features: Tanzu Developer Tools for Visual Studio

- **See code updates running on-cluster in seconds:** By using Live Update facilitated by Tilt, deploy your workload once, save changes to the code, and then in seconds see those changes reflected in the workload running on the cluster. All Live Update output is filtered to its own output pane window within Visual Studio.
- **Debug workloads directly on the cluster:** Debug your application in a production-like environment by debugging on your Kubernetes cluster that has Tanzu Application Platform. An environment's similarity to production relies on keeping dependencies updated, among other variables.

#### v1.4.0 Features: Tanzu Developer Tools for Visual Studio Code

- The developer sandbox enables developers to Live Update their code, and simultaneously debug the updated code, without having to deactivate Live Update when debugging.

---

## v1.4.0 Breaking changes

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

#### v1.4.0 Breaking changes: Application Single Sign-On (AppSSO)

- Removed `AuthServer.spec.identityProvider.ldap.group.search{Filter,Base,Depth,SubTree}` and introduced `ldap.group.search: {}`.
  - If `ldap.group` is defined and `ldap.group.search` is not defined, the LDAP is considered an ActiveDirectory style LDAP and groups are loaded from the user's `memberOf` attribute.
  - If `ldap.group` and `ldap.group.search` are both defined, the LDAP is considered a Classic LDAP and group search is done by searching in the `ldap.group.search.base`.
  - There used to be a mixed mode, when both searches were attempted every time.
- Removed `AuthServer.spec.identityProviders.ldap.server` field.
- Removed `AuthServer.status.deployments.authServer.lastParentGenerationWithRestart` field.
- Removed deprecated field `AuthServer.spec.issuerURI`. For more information, see [IssuerURI and TLS](#).

#### v1.4.0 Breaking changes: Out of the Box Supply Chain Templates

- In a multicluster setup, when a Deliverable is created on a Build profile cluster, the ConfigMap it's in is renamed from `<workload-name>` to `<workload-name>-deliverable`. Any automation that depends on obtaining the Deliverable content by the former name must be updated with the new name. For more information, see [Multicluster Tanzu Application Platform overview](#).

#### v1.4.0 Breaking changes: Supply Chain Security Tools - Image Policy Webhook

- The Image Policy Webhook component is removed in Tanzu Application Platform v1.4. This component is deprecated in favor of the [Policy Controller](#).

#### v1.4.0 Breaking changes: Supply Chain Security Tools - Policy Controller

- Policy Controller no longer initializes TUF by default. TUF is required to support the keyless authorities in `ClusterImagePolicy`. To continue to use keyless authorities, provide the value `policy.tuf_enabled: true` by using the `tap-values.yaml` file while upgrading. By default, the public Sigstore The Update Framework (TUF) server is used. To target an alternative Sigstore stack, specify `policy.tuf_mirror` and `policy.tuf_root`.

#### v1.4.0 Breaking changes: Tanzu Application Platform GUI

- **Ingress URL:** As mentioned in the [new features section](#), Tanzu Application Platform GUI participates in the shared ingress issuer feature. You might need to change your scheme from http to https. For more information, see [Troubleshooting](#).
- **Communication with Supply Chain Security Tools - Store:** In previous versions of Tanzu Application Platform, you configured Tanzu Application Platform GUI to use the read-only access token to communicate with Supply  
  
In v1.4, you must use the read-write access token to use new features in the Security Analysis GUI plug-in. If upgrading from v1.3, update your Tanzu Application Platform GUI configuration accordingly. See the updated instructions in [Enable CVE scan results](#).

#### v1.4.0 Breaking changes: Tanzu Developer Tools for IntelliJ

- IntelliJ IDEA v2022.2 to v2022.3 is required to install the extension.

#### v1.4.0 Breaking changes: Tanzu Developer Tools for Visual Studio Code

- `Tanzu Debug` no longer port forwards the application port (8080).

---

## v1.4.0 Security fixes

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

#### v1.4.0 Security fixes: API Auto Registration

- Base image updated to use the latest Paketo Jammy Base image.

#### v1.4.0 Security fixes: Contour

- Update to [Contour v1.22.3](#). Includes an update to [go v1.19.4](#), which contains security fixes to the `net/http` and `os` packages.

#### v1.4.0 Security fixes: Supply Chain Security Tools - Grype

- `python` is updated to `3.7.5-22.ph3`.

#### v1.4.0 Security fixes: Remediated vulnerabilities

The following is a list of vulnerabilities remediated with this release:

- GHSA-7hfm-57qf-j43q, GHSA-crv7-7245-f45f, GHSA-mc84-pj99-q6hh, GHSA-xqfj-vm6h-2x34, CVE-2022-42003, CVE-2022-42004, GHSA-jjih-jjxp-wpff, GHSA-rgv9-q543-rqg4, GHSA-3mc7-4q67-w48m, GHSA-36p3-wjmg-h94x, CVE-2022-23960, CVE-2022-43945, GHSA-crp2-qrr5-8pq7, GHSA-7qw8-847f-pggm, GHSA-c3xm-pvg7-gh7r, GHSA-f524-rf33-2jir, CVE-2022-2509, CVE-2022-3171, CVE-2022-3509, CVE-2022-3510, GHSA-4gg5-vx3j-xwc7, GHSA-g5ww-5jh7-63cx, GHSA-66x3-6cw3-v5gj, CVE-2022-3515, CVE-2022-2602, CVE-2022-41222, CVE-2022-32212, CVE-2022-35255, CVE-2021-27478, CVE-2021-27482, CVE-2021-27498, CVE-2021-27500, CVE-2019-12900, CVE-2021-28861, CVE-2021-3737, CVE-2022-0391, GHSA-4w2j-2rg4-5mjw, CVE-2022-2586, CVE-2022-2588, CVE-2022-34918, GHSA-4wf5-vphf-c2xc, CVE-2022-42916, CVE-2022-43551, CVE-2022-43552, CVE-2021-3999, GHSA-m974-647v-whv7

#### Note about CVE-2022-4378

- [CVE-2022-4378](#) is a high severity, exploitable stack overflow flaw found in the Linux kernel's SYSCTL subsystem. At this time, there is no available patch from Canonical in their upstream Ubuntu distribution. Once there is a patch available for the 22.04 release line, Tanzu Application Platform will release a patched base stack image. The current status for patching this vulnerability in the Jammy stack is available on [Ubuntu's security page](#).

It is important for customers to understand CVE-2022-4378 is a kernel exploit, and the kernel runs on the customers' container host VM, not the Tanzu Application Platform container image. Even with a patched image, the vulnerability will not be mitigated until customers deploy their containers on a host with a patched OS. An unpatched host OS may be exploitable if the base image is deployed allowing users to modify SYSCTL parameters.

RedHat has published a [potential mitigation](#) preventing regular users from accessing sysctl files and increasing privileges until a patch becomes available.

## v1.4.0 Resolved issues

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

### v1.4.0 Resolved issues: API Auto Registration

- API Auto Registration periodically checks the original API specification from the defined location to find changes and registers any changes into the `API Descriptor`. This triggers reconciliation into the Tanzu Application Platform GUI catalog. This synchronization period or frequency is configurable through the new value `sync_period`. The default value is 5 minutes.

### v1.4.0 Resolved issues: Application Single Sign-On (AppSSO)

- Fixed infinite redirect loops for an `AuthServer` configured with a single OIDC or SAML identity provider.
- Authorization Code request rejected audit event from anonymous users logging proper IP address.
- `AuthServer` no longer attempts to configure Redis event listeners.

- OpenShift: custom `SecurityContextConstraint` resource is created for Kubernetes platforms versions 1.23.x and lower.
- LDAP error log now contains proper error message.

#### v1.4.0 Resolved issues: Out of the Box Supply Chain Templates

Fixed deliverable content written into ConfigMaps in multicluster setup. ConfigMap is renamed to avoid conflict with `config-template`.

For more information, see [Multicluster Tanzu Application Platform overview](#).

#### v1.4.0 Resolved issues: Tanzu CLI Apps Plug-in

- Fixed `tanzu apps workload tail` command output, which was displaying extra init container log lines.
- Fixed `tanzu apps workload tail` command not including all logs.

#### v1.4.0 Resolved issues: Tanzu Application Platform GUI plug-ins

- **Immediate entity provider back-end plug-in**
  - The entity provider, used mainly by API Auto Registration, now allows a body size of `5Mb` to accept larger API specifications.
  - Considering the restriction of Backstage for [Entity Provider mutations](#), whenever an existing entity is intended for a mutation through this plug-in, and its origin is a different entity provider, a `409 Conflict` error is returned.

#### v1.4.0 Resolved issues: Supply Chain Choreographer plug-in

- The UI no longer shows the error `Unable to retrieve details from Image Provider Stage` when the Builder is not available or configured. It now correctly shows the same error as the CLI, `Builder default is not ready`.
  - Build logs are now displayed when the **Image Provider** stage fails
- 

## v1.4.0 Known issues

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

#### v1.4.0 Known issues: API Auto Registration

- Unknown Certificate Authority and connection refused errors from the APIDescriptor CRD. For more information, see [Troubleshoot API Auto Registration](#).

#### v1.4.0 Known issues: Application Accelerator for Visual Studio Code

- When using custom types, if there is a check box in the list of attributes then re-ordering the inputs doesn't work.

#### v1.4.0 Known issues: Cloud Native Runtimes for VMware Tanzu

- Knative Serving: Certain app name, namespace, and domain combinations produce invalid HTTPProxy resources. See [Cloud Native Runtimes Troubleshooting](#).

- **Knative Serving and Cert Manager:** When auto-tls is enabled, the default in Tanzu Application Platform v1.4.0, Knative services fail with `certificateNotReady` if workload name, namespace, and domain are more than 64 bytes. See [Cloud Native Runtimes Troubleshooting](#).

#### v1.4.0 Known issues: Grype scanner

- **Scanning Java source code that uses Gradle package manager might not reveal vulnerabilities:**

For most languages, Source Code Scanning only scans files present in the source code repository. Except for support added for Java projects using Maven, no network calls fetch dependencies. For languages using dependency lock files, such as Golang and Node.js, Grype uses the lock files to check dependencies for vulnerabilities.

For Java using Gradle, dependency lock files are not guaranteed, so Grype uses dependencies present in the built binaries, such as `.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.

- **Scanning some Alpine-based container images fails with a panic:**

An issue in Syft causes the scanner to crash with index out of range, while parsing APK metadata to identify installed OS packages if a package's list of provided files is empty. This problem is resolved in SCST - Scan (Grype) version `1.4.1` or in the Tanzu Application Platform version `1.5.0+`.

#### v1.4.0 Known issues: Namespace Provisioner

- Applying the label selector used by the namespace provisioner controller to the developer namespace, which is configured at deployment time under the `grype` package values, will cause the `provisioner Carvel app` to crash due to ownership issues. This is because it's trying to install Grype in a namespace where it's already been installed.

#### v1.4.0 Known issues: Out of the Box Supply Chain Templates

In a Build profile cluster, Deliverables will be created with the labels to associate them with their Workload missing. As a workaround, they will have to be manually injected. For more information, see [Multicluster Tanzu Application Platform overview](#).

#### v1.4.0 Known issues: Tanzu Application Platform GUI plug-ins

##### Security Analysis plug-in

- The **No Associated Policy** tab in Workload Build Vulnerabilities does not show workloads that lack associated scan policies.
- The CVEs bar graph in Workload Build Vulnerabilities sometimes cuts numbers off.

##### Supply Chain Choreographer plug-in

- The `Generation` field and scan policy link in the **Overview** section does not update when you amend a scan policy. The correct version and details of the policy are shown in the CLI.
- Customizing the `Source Tester` stage in an Out Of the Box supply chain does not show details in the **Stage Details** section.

- When a GitOps PR flow is configured, the **Approve a Request** link no longer appears in the supply chain graph.

#### v1.4.0 Known issues: Tanzu Developer Tools for IntelliJ

- If a workload is deployed onto a namespace by using Live Update, you must set that namespace as the namespace of the current context of your kubeconfig file. Otherwise, if you run Tanzu Debug, it causes the workload to re-deploy. For more information, see [Troubleshooting](#).
- On macOS, Tanzu Panel might be empty when using a GKE cluster. For more information, see [Troubleshooting](#).
- The **Describe** action in the pop-up menu in the Activity panel can fail when used on PodIntent resources. For more information, see [Troubleshooting](#).
- The Tanzu panel might show workloads without showing Kubernetes resources in the center panel of the activity pane. For more information, see [Troubleshooting](#).
- The Details table and Messages pane in the activity panel can show stale data because these views only refresh when the selection in the Resource tree is changed. As a workaround, make the views refresh by clicking somewhere in the Resource tree to change the current selection.
- Live Update does not work when using the Jammy `ClusterBuilder`.

#### v1.4.0 Known issues: Tanzu Developer Tools for Visual Studio

- The `Tanzu: Delete Workload` command can fail with the extension erroneously reporting that the workload isn't running. For more information, see [Troubleshooting](#).
- The `Tanzu: Start Live Update` command can fail because the specified path was not found. For more information, see [Troubleshooting](#).
- Live Update does not work when using the Jammy `ClusterBuilder`.

#### v1.4.0 Known issues: Tanzu Developer Tools for Visual Studio Code

- `Could not find the task 'tanzuManagement: Kill Port Forward fortune-service'`. You might see this error message if an app was deployed with a previous version of the Visual Studio Code extension. For more information, see [Troubleshooting](#).

---

## Deprecations

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

### Application Live View deprecations

- `appliveview_connector.backend.sslDisabled` is deprecated and marked for removal in Tanzu Application Platform 1.7.0. For more information on the migration, see [Deprecate the sslDisabled key](#).

### Application Single Sign-On (AppSSO) deprecations

- `AuthServer.spec.tls.disabled` is deprecated and marked for removal in the next release. For more information about how to migrate to `AuthServer.spec.tls.deactivated`, see [Migration guides](#).

## Services Toolkit deprecations

- The `tanzu services claims` CLI plug-in command is now deprecated. It is hidden from help text output, but continues to work until officially removed after the deprecation period. The new `tanzu services resource-claims` command provides the same functionality.

## Supply Chain Security Tools - Image Policy Webhook deprecations

- The Image Policy Webhook component is removed in Tanzu Application Platform v1.4. This component is deprecated in favor of the [Policy Controller](#).

## Supply Chain Security Tools - Scan deprecations

- Removed deprecated ScanTemplates:
  - Deprecated Grype ScanTemplates shipped with versions prior to Tanzu Application Platform 1.2.0 are removed and no longer supported. Use Grype ScanTemplates v1.2 and later.
  - `docker` field and related sub-fields used in Supply Chain Security Tools - Scan are deprecated and marked for removal in Tanzu Application Platform 1.7.0.
  - The deprecation impacts the following components: Scan Controller, Grype Scanner, and Snyk Scanner. Carbon Black Scanner is not impacted.
  - For information about the migration path, see [Troubleshooting](#).

## Supply Chain Security Tools - Sign deprecations

- [Supply Chain Security Tools - Sign](#) is deprecated. For migration information, see [Migration From Supply Chain Security Tools - Sign](#).

## Tanzu Build Service deprecations

- The Ubuntu Bionic stack is deprecated: Ubuntu Bionic stops receiving support in April 2023. VMware recommends you migrate builds to Jammy stacks in advance. For how to migrate builds, see [Use Jammy stacks for a workload](#).
- The Cloud Native Buildpack Bill of Materials (CNB BOM) format is deprecated. It is still activated by default in Tanzu Application Platform v1.3 and v1.4. VMware plans to deactivate this format by default in Tanzu Application Platform v1.6 and remove support in Tanzu Application Platform v1.8. To manually deactivate legacy CNB BOM support, see [Deactivate the CNB BOM format](#).

## Tanzu CLI Apps plug-in deprecations

- The default value for the `--update-strategy` flag will change from `merge` to `replace` in Tanzu Application Platform v1.7.0.
- The `tanzu apps workload update` command is deprecated and marked for removal in Tanzu Application Platform 1.5.0. Use `tanzu apps workload apply` instead.

## Linux Kernel CVEs

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

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



# Components and installation profiles for Tanzu Application Platform

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

## Tanzu Application Platform components

- **API Auto Registration**

When users deploy a [workload](#) that exposes an API, they want that API to automatically show in Tanzu Application Platform GUI without requiring any added manual steps.

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

- **API portal 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 appear in a single instance.

- **API Scoring and Validation**

API Validation and Scoring focuses on scanning and validating an OpenAPI specification. The API specification is generated from the [API Auto Registration](#). After an API is registered, the API specification goes through static scan analysis and is validated. Based on the validation, a scoring is provided to indicate the quality and health of the API specification as it relates to Documentation, OpenAPI best practices, and Security.

- **Application Accelerator for VMware Tanzu**

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

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

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

- **Application Live View**

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

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

- **Application Single Sign-On for VMware Tanzu**

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

- **Cloud Native Runtimes for VMware Tanzu**

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

- **cert-manager**

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

- **Contour**

Contour is an ingress controller for Kubernetes that supports dynamic configuration updates and multi-team ingress delegation. It provides the control plane for the Envoy edge and service proxy. For more information about Contour, see the [Contour documentation](#).

- **Convention Service for VMware Tanzu**

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

- **Default roles for Tanzu Application Platform**

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

Default roles only apply to a user interacting with the cluster by using kubectl and Tanzu CLI. Tanzu Application Platform GUI support for default roles is planned for a future release.

- **Developer Conventions**

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

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

- **Eventing for VMware Tanzu**

Eventing for VMware Tanzu focuses on providing tooling and patterns for Kubernetes applications to manage event-triggered systems through Knative Eventing. For information about Knative, see the [Knative documentation](#).

- **Flux CD Source Controller**

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

- **Grype**

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

- **Namespace Provisioner**

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

- **Services Toolkit for VMware Tanzu**

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

- **Source Controller**

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

- ImageRepository
- MavenArtifact

- **Spring Boot conventions**

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

- **Supply Chain Choreographer for VMware Tanzu**

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

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

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

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

To enable this, Supply Chain Security Tools - Scan provides scanning and gate keeping capabilities that Application and DevSecOps teams can incorporate earlier in their path to production. This is an established industry best practice for reducing security risk and ensuring more efficient remediation.

- **Supply Chain Security Tools - Sign (Deprecated)**

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. SCST - Sign works with [cosign signature format](#) and allows for fine-tuned configuration based on image source patterns.

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

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

- **Supply Chain Security Tools - Store**

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

- **Tanzu Application Platform GUI**

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

- **Tanzu Build Service**

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

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

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

- **Tanzu Developer Tools for IntelliJ**

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

- **Tanzu Developer Tools for Visual Studio**

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

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

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

- **Tanzu Learning Center**

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

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

- **Tekton Pipelines**

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

- **Tanzu Application Platform Telemetry**

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



#### Note

You can opt out of telemetry collection by following the instructions in [Opting out of telemetry collection](#).

## Installation profiles in Tanzu Application Platform v1.4

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

The following profiles are available in Tanzu Application Platform:

- **Full** (`full`): Contains all of the Tanzu Application Platform packages.
- **Iterate** (`iterate`): Intended for iterative application development.
- **Build** (`build`): Intended for the transformation of source revisions to workload revisions. Specifically, hosting workloads and SupplyChains.
- **Run** (`run`): Intended for the transformation of workload revisions to running pods. Specifically, hosting deliveries and deliverables.
- **View** (`view`): Intended for instances of applications related to centralized developer experiences. Specifically, Tanzu Application Platform GUI and Metadata Store.

The following tables list the packages contained in each profile. For a diagram showing the packages contained in each profile, see [Overview of multicluster Tanzu Application Platform](#).

### Packages: A to C

Package Name	Full	Iterate	Build	Run	View
API Auto Registration	✓	✓		✓	
API portal	✓				✓
Application Accelerator	✓				✓

Package Name	Full	Iterate	Build	Run	View
Application Live View back end	✓				✓
Application Live View connector	✓	✓		✓	
Application Live View conventions	✓	✓	✓		
Application Single Sign-On	✓	✓		✓	
cert-manager	✓	✓	✓	✓	✓
Cloud Native Runtimes	✓	✓		✓	
Contour	✓	✓	✓	✓	✓
Convention controller	✓	✓	✓		

## Packages: D to R

Package Name	Full	Iterate	Build	Run	View
Default Roles	✓	✓	✓	✓	
Developer Conventions	✓	✓			
Eventing	✓	✓		✓	
Flux Source Controller	✓	✓	✓	✓	✓
Grype	✓		✓		
Learning Center	✓				✓
Namespace Provisioner	✓	✓	✓	✓	
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	✓	✓	✓	✓	

## Packages: S to Z

Package Name	Full	Iterate	Build	Run	View
Service Bindings	✓	✓		✓	
Services Toolkit	✓	✓		✓	
Source Controller	✓	✓	✓	✓	✓
Spring Boot conventions	✓	✓	✓		
Supply Chain Choreographer	✓	✓	✓	✓	
SCST - Policy Controller	✓	✓		✓	
SCST - Scan	✓		✓		
SCST - Sign (deprecated)	✓	✓		✓	
SCST - Store	✓				✓

Package Name	Full	Iterate	Build	Run	View
Tanzu Build Service	✓	✓	✓		
Tanzu Application Platform GUI	✓				✓
Tekton Pipelines	✓	✓	✓		
Telemetry	✓	✓	✓	✓	✓



#### Note

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

## Language and framework support in Tanzu Application Platform

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

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

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

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

## Installing Tanzu Application Platform

For more information about installing Tanzu Application Platform, see [Installing Tanzu Application Platform](#).

# Install Tanzu Application Platform

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

- [Install Tanzu Application Platform online](#). For Tanzu Application Platform on a Kubernetes cluster with internet access.
- [Install Tanzu Application Platform in an air-gapped environment](#). For Tanzu Application Platform on a Kubernetes cluster air-gapped from external traffic.
- [Install Tanzu Application Platform in AWS](#). For installing Tanzu Application platform using AWS Cloud Services.
- [Install Tanzu Application Platform on OpenShift](#). For Tanzu Application Platform on an OpenShift cluster with internet access.

## Install Tanzu Application Platform

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

- [Install Tanzu Application Platform online](#). For Tanzu Application Platform on a Kubernetes cluster with internet access.
- [Install Tanzu Application Platform in an air-gapped environment](#). For Tanzu Application Platform on a Kubernetes cluster air-gapped from external traffic.
- [Install Tanzu Application Platform in AWS](#). For installing Tanzu Application platform using AWS Cloud Services.
- [Install Tanzu Application Platform on OpenShift](#). For Tanzu Application Platform on an OpenShift cluster with internet access.

## Prerequisites for installing Tanzu Application Platform

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

### VMware Tanzu Network and container image registry requirements

Installation requires:

- Access to VMware Tanzu Network:
  - A [Tanzu Network](#) account to download Tanzu Application Platform packages.
  - Network access to <https://registry.tanzu.vmware.com>.
- Cluster-specific registry:
  - A container image registry, such as [Harbor](#) or [Docker Hub](#) for application images, base images, and runtime dependencies. When available, VMware recommends



using a paid registry account to avoid potential rate-limiting associated with some free registry offerings.

- o Recommended storage space for container image registry:
  - 1 GB of available storage if installing Tanzu Build Service with the `lite` set of dependencies.
  - 10 GB of available storage if installing Tanzu Build Service with the `full` set of dependencies, which are suitable for offline environments.



#### Note

For production environments, `full` dependencies are recommended to optimize security and performance. For more information about Tanzu Build Service dependencies, see [About lite and full dependencies](#).

- Registry credentials with read and write access available to Tanzu Application Platform to store images.
- Network access to your chosen container image registry.

## DNS Records

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

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

## Tanzu Application Platform GUI

For Tanzu Application Platform GUI, you must have:

- Latest version of Chrome, Firefox, or Edge. Tanzu Application Platform GUI currently does not support Safari browser.

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

## Kubernetes cluster requirements

Installation requires Kubernetes cluster v1.23, v1.24 or v1.25 on one of the following Kubernetes providers:

- Azure Kubernetes Service.
- Amazon Elastic Kubernetes Service.
  - containerd must be used as the Container Runtime Interface (CRI). Some versions of EKS default to Docker as the container runtime and must be changed to containerd.
  - EKS clusters on Kubernetes version 1.23 and above require the [Amazon EBS CSI Driver](#) due to [CSIMigrationAWS](#) is enabled by default in Kubernetes version 1.23 and above.
    - Users currently on EKS Kubernetes version 1.22 must install the Amazon EBS CSI Driver before upgrading to Kubernetes version 1.23 and above. See [AWS documentation](#) for more information.
  - AWS Fargate is not supported.
- Google Kubernetes Engine.
  - GKE Autopilot clusters do not have the required features enabled.
  - GKE clusters that are set up in zonal mode might detect Kubernetes API errors when the GKE control plane is resized after traffic increases. Users can mitigate this by creating a regional cluster with three control-plane nodes right from the start.

- Minikube.
  - Reference the [resource requirements](#) in the following section.
  - Hyperkit driver is supported on macOS only. Docker driver is not supported.
- Red Hat OpenShift Container Platform v4.10 or v4.11.
  - vSphere
  - Baremetal
- Tanzu Kubernetes Grid multicloud.
- vSphere with Tanzu v7.0 U3f or later.  
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](#).

For more information about the supported Kubernetes versions, see [Kubernetes version support for Tanzu Application Platform](#).

## Resource requirements

- To deploy Tanzu Application Platform packages iterate profile on local Minikube cluster, your cluster must have at least:
  - 8 vCPUs for i9 (or equivalent) available to Tanzu Application Platform components on Mac OS.
  - 12 vCPUs for i7 (or equivalent) available to Tanzu Application Platform components on Mac OS.
  - 8 vCPUs available to Tanzu Application Platform components on Linux and Windows.
  - 12 GB of RAM available to Tanzu Application Platform components on Mac OS, Linux and Windows.
  - 70 GB of disk space available per node.
- To deploy Tanzu Application Platform packages full profile, your cluster must have at least:
  - 8 GB of RAM available per node to Tanzu Application Platform.
  - 16 vCPUs available across all nodes to Tanzu Application Platform.
  - 100 GB of disk space available per node.
- To deploy Tanzu Application Platform packages build, run and iterate (shared) profile, your cluster must have at least:
  - 8 GB of RAM available per node to Tanzu Application Platform.
  - 12 vCPUs available across all nodes to Tanzu Application Platform.
  - 100 GB of disk space available per node.
- To deploy Tanzu Application Platform packages view profile, your cluster must have at least:

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

For more information about pod security policies, see [Kubernetes documentation](#).

## Tools and CLI requirements

Installation requires:

- The Kubernetes CLI (kubectl) v1.23, v1.24, or v1.25 installed and authenticated with admin rights for your target cluster. See [Install Tools](#) in the Kubernetes documentation.

## Next steps

- [Accept Tanzu Application Platform EULAs and installing the Tanzu CLI](#)

## Kubernetes version support for Tanzu Application Platform

The following is a matrix table providing details of the compatible Kubernetes cluster versions for Tanzu Application Platform v1.4.

Kubernetes Cluster	Support Information	Notes
Kubernetes	v1.23, v1.24, v1.25	
VMware Tanzu Kubernetes Grid	v1.6	
Tanzu Kubernetes releases (vSphere with Tanzu)	TKr v1.25.7 for vSphere v8.x, TKr v1.24.9 for vSphere v8.x, TKr v1.23.8 for vSphere v7.x (Photon)	Support for TKr v1.25.7 begins with Tanzu Application Platform v1.4.8  Support for TKr v1.24.9 begins with Tanzu Application Platform v1.4.4
OpenShift	v4.10, v4.11	OpenShift v4.10 reached its end of life on September 10, 2023, which means it no longer receives support for Tanzu Application Platform v1.4.9, and v1.4.10
Azure Kubernetes Service	Supported	
Elastic Kubernetes Service	Supported	
Google Kubernetes Engine	Supported	

## Install Tanzu CLI

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

- [Install Tanzu CLI](#)
  - [Accept the End User License Agreements](#)
  - [Example of accepting the Tanzu Application Platform EULA](#)
  - [Set the Kubernetes cluster context](#)
  - [Install or update the Tanzu CLI and plug-ins](#)
  - [Install the Tanzu CLI](#)
  - [Install Tanzu CLI Plug-ins](#)
    - [List the versions of each plug-in group available across Tanzu](#)
    - [List the versions of the Tanzu Application Platform specific plug-in group](#)
    - [Install the version of the Tanzu Application Platform plug-in group matching your target environment](#)
    - [Verify the plug-in group list against the plug-ins that were installed](#)
  - [Next steps](#)

## Accept the End User License Agreements

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

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

## Example of accepting the Tanzu Application Platform EULA

To accept the Tanzu Application Platform EULA:

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



GET EMAIL ALERTS

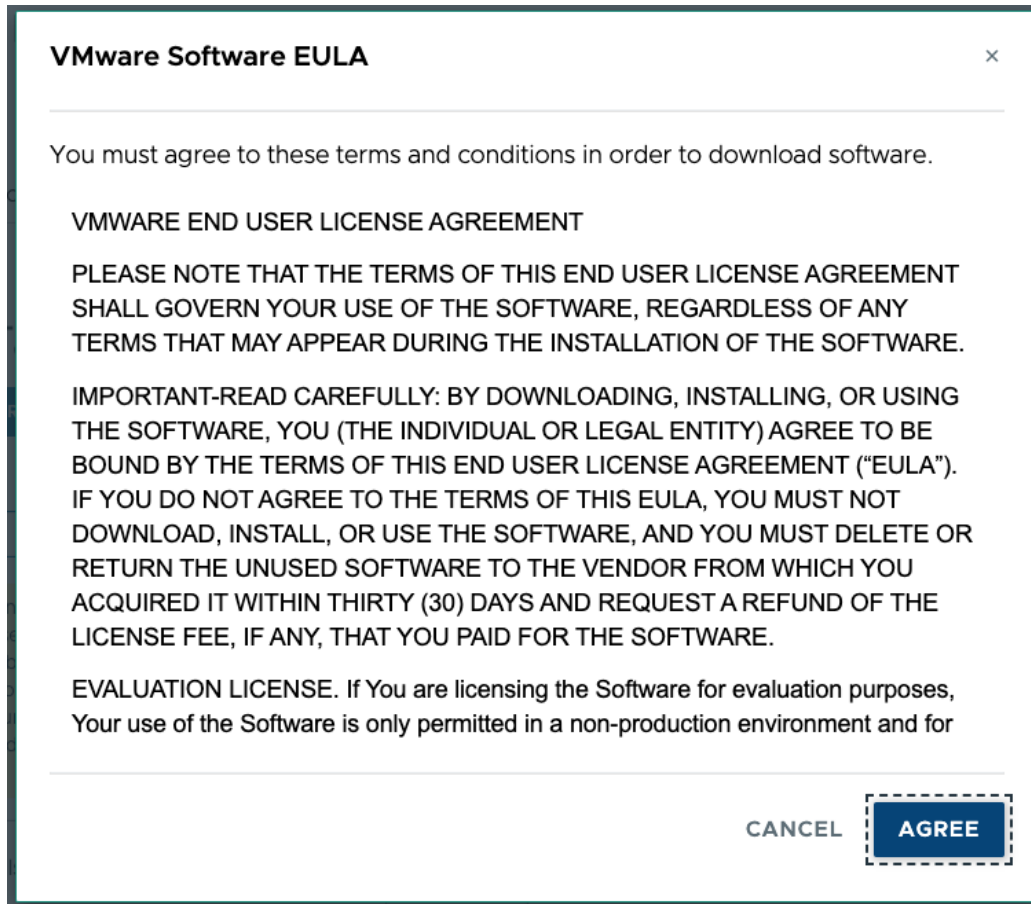
RELEASE: 1.2.1

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

	Tanzu Developer Tools for Visual Studio Code	60.9 MB 0.7.1+build.1	•
	learning-center-workshop-samples.zip	111 KB 1.0.1	•
	Tanzu App Accelerator Extension for Visual Studio Code	201 KB 0.1.2	•
	Tanzu Developer Tools for IntelliJ	24.1 MB 0.1.0	•
	tap-gui-catalogs-latest	2 Files	>
	tanzu-cli-v0.11.6	3 Files	>
	Artifact References	238 Artifacts	>

<b>Release Details</b>	
Release Date	2022-08-09
Release Type	Patch Release
End of General Support	2023-08-31
<b>Release Description</b>	
Patch release includes bug fixes within the Supply Chain, Scanning, TAP GUI, and App Accelerator components.	
<b>Depends On</b>	
Products in the "Depends On" section must be installed prior to installing or upgrading to VMware Tanzu Application Platform 1.2.1. Please install or upgrade these products to one of the listed versions.	
Cluster Essentials for VMware Tanzu 1.2.0, 1.1.0 or 1.0.0	
<b>Upgrades From</b>	
VMware Tanzu Application Platform versions in the "Upgrades From" section can be directly upgraded to VMware Tanzu Application Platform 1.2.1. If your current version of VMware Tanzu Application Platform is not on this list, please contact <a href="#">Tanzu Customer Service</a> for assistance.	
1.1.* or 1.2.0	
For any assistance with upgrades please use the <a href="#">Tanzu Upgrade Planner Tool</a> .	
<b>License Files</b>	
<a href="#">VMWARE TANZU APPLICATION PLATFORM 1.2.1 OPEN SOURCE LICENSE</a>	
<b>RELEASE NOTES*</b>	
<a href="#">END USER LICENSE AGREEMENT</a>	
*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.



## Set the Kubernetes cluster context

For information about the supported Kubernetes cluster providers and versions, see [Kubernetes cluster requirements](#).

To set the Kubernetes cluster context:

1. List the existing contexts by running:

```
kubectl config get-contexts
```

For example:

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

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

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

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

For example:

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

## Install or update the Tanzu CLI and plug-ins

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

### Install the Tanzu CLI

The Tanzu CLI core v1.0.0 distributed with Tanzu Application Platform is forward and backward compatible with all supported Tanzu Application Platform versions. Run a single command to install the plug-in group version that matches the Tanzu Application Platform version on any target environment. For more information, see [Install Plugins](#).

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

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



#### Note

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

#### Install using a package manager

To install the Tanzu CLI using a package manager:

1. Follow the instructions for your package manager below. This installs the latest version of the CLI available in the package registry.

- o **Homebrew (MacOS):**

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

- o **Chocolatey (Windows):**

```
choco install tanzu-cli
```

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

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

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

For example:

```
choco install tanzu-cli --version 1.1.0
```

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



```

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

```

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

```

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

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

```

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

```

tanzu version
version: v1.1.0
...

```

### Install from a binary release

To install the Tanzu CLI from a binary release:

1. Download the Tanzu CLI binary from one of the following locations:

- o **VMware Tanzu Network**

1. Go to [VMware Tanzu Network](#).
2. Choose the 1.4.13 release from the Release dropdown menu.
3. Click the tanzu-core-cli-binaries item from the result set.
4. Download the Tanzu CLI binary for your operating system.

- o **VMware Customer Connect**

1. Go to [VMware Customer Connect](#).
2. Download the Tanzu CLI binary for your operating system.

- o **GitHub**

1. Go to [Tanzu CLI release v1.1.0 on GitHub](#).
2. Download the Tanzu CLI binary for your operating system, for example, [tanzu-cli-windows-amd64.tar.gz](#).

2. Use an extraction tool to unpack the binary file:

- o **macOS:**

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

- o **Linux:**

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

- o **Windows:**

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

3. Make the CLI available to the system:

- o cd to the directory containing the extracted CLI binary

- o **macOS:**

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

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

- o **Linux:**

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

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

- o **Windows:**

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

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

```
tanzu version
version: v1.1.0
...
```

## Install Tanzu CLI Plug-ins

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

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

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

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

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

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

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

```
tanzu plugin install --group vmware-tap/default:v1.4.13
```

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

```
tanzu plugin group get vmware-tap/default:v1.4.13
```

```
tanzu plugin list
```

```
For air-gapped installation, see the [Installing the Tanzu CLI in Internet-Restricted Environments] (https://docs.vmware.com/en/VMware-Tanzu-CLI/1.1/tanzu-cli/index.html#internet-restricted-install) section of the Tanzu CLI documentation.
```

```
## <a id='next-steps'></a>Next steps
```

```
For online installation:
```

```
- [Deploy Cluster Essentials*] (https://docs.vmware.com/en/Cluster-Essentials-for-VMware-Tanzu/1.4/cluster-essentials/deploy.html)
- [Install the Tanzu Application Platform package and profiles] (install-online/profile.hbs.md)
```

```
For air-gapped installation:
```

```
- [Deploy Cluster Essentials] (https://docs.vmware.com/en/Cluster-Essentials-for-VMware-Tanzu/1.4/cluster-essentials/deploy.html)
- [Install Tanzu Application Platform in an air-gapped environment] (install-offline/profile.hbs.md)
```

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

## Install Tanzu Application Platform (online)

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

Step	Task	Link
1.	Review the prerequisites to ensure you have met all requirements before installing.	<a href="#">Prerequisites</a>

Step	Task	Link
2.	Accept Tanzu Application Platform EULAs and install the Tanzu CLI.	<a href="#">Accept Tanzu Application Platform EULAs and installing the Tanzu CLI</a>
3.	Install Cluster Essentials for Tanzu*.	<a href="#">Deploy Cluster Essentials</a>
4.	Add the Tanzu Application Platform package repository, prepare your Tanzu Application Platform profile, and install the profile to the cluster.	<a href="#">Install the Tanzu Application Platform package and profiles</a>
5.	(Optional) Install any additional packages that were not in the profile.	<a href="#">Install individual packages</a>
6.	Set up developer namespaces to use your installed packages.	<a href="#">Set up developer namespaces to use your installed packages</a>
7.	Install developer tools into your integrated development environment (IDE).	<a href="#">Install Tanzu Developer Tools for your VS Code</a>

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

After installing Tanzu Application Platform on to your Kubernetes clusters, proceed with [Get started with Tanzu Application Platform](#).

## Install Tanzu Application Platform (online)

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

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

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

After installing Tanzu Application Platform on to your Kubernetes clusters, proceed with [Get started with Tanzu Application Platform](#).

## Install Tanzu Application Platform package and profiles

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

Before installing the packages, ensure you have:

- Completed the [Prerequisites](#).
- Configured and verified the cluster.

- [Accepted Tanzu Application Platform EULA and installed Tanzu CLI with any required plug-ins.](#)

## Relocate images to a registry

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

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

- [Harbor documentation](#)
- [Google Container Registry documentation](#)
- [Quay.io documentation](#)

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

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

```
docker login MY-REGISTRY
```

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

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

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

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

```
export INSTALL_REGISTRY_USERNAME=MY-REGISTRY-USER
export INSTALL_REGISTRY_PASSWORD=MY-REGISTRY-PASSWORD
export INSTALL_REGISTRY_HOSTNAME=MY-REGISTRY
export TAP_VERSION=VERSION-NUMBER
export INSTALL_REPO=TARGET-REPOSITORY
```

Where:

- `MY-REGISTRY-USER` is the user with write access to `MY-REGISTRY`.
- `MY-REGISTRY-PASSWORD` is the password for `MY-REGISTRY-USER`.
- `MY-REGISTRY` is your own container registry.
- `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.4.13`.
- `TARGET-REPOSITORY` is your target repository, a folder/repository on `MY-REGISTRY` that serves as the location for the installation files for Tanzu Application Platform.

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

5. [Install the Carvel tool imgpkg CLI.](#)

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

```
imgpkg tag list -i registry.tanzu.vmware.com/tanzu-application-platform/tap-packages | grep -v sha | sort -V
```

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

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

## Add the Tanzu Application Platform package repository

Tanzu CLI packages are available on repositories. Adding the Tanzu Application Platform package repository makes Tanzu Application Platform and its packages available for installation.

Relocate images to a registry is strongly recommended but not required for installation. If you skip this step, you can use the following values to replace the corresponding variables:

- `INSTALL_REGISTRY_HOSTNAME` is `registry.tanzu.vmware.com`
- `INSTALL_REPO` is `tanzu-application-platform`
- `INSTALL_REGISTRY_USERNAME` and `INSTALL_REGISTRY_PASSWORD` are the credentials to run `docker login registry.tanzu.vmware.com`
- `TAP_VERSION` is your Tanzu Application Platform version. For example, `1.4.13`

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

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

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

This namespace keeps the objects grouped together logically.

2. Create a registry secret by running:

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

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

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

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

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

For example:

```
$ tanzu package repository get tanzu-tap-repository --namespace tap-install
- Retrieving repository tap...
NAME:          tanzu-tap-repository
```

```

VERSION:      16253001
REPOSITORY:   tapmdc.azurecr.io/mdc/1.4.0/tap-packages
TAG:          1.4.13
STATUS:       Reconcile succeeded
REASON:

```



### Note

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

## 5. List the available packages by running:

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

For example:

```

$ tanzu package available list --namespace tap-install
/ Retrieving available packages...
NAME                                DISPLAY-NAME
SHORT-DESCRIPTION
  accelerator.apps.tanzu.vmware.com  Application Accelerator
  for VMware Tanzu                    Used to create new projects a
  nd configurations.
  api-portal.tanzu.vmware.com        API portal
  A unified user interface for API discovery and exploration at scale.
  apis.apps.tanzu.vmware.com         API Auto Registration fo
  r VMware Tanzu                      A TAP component to automatica
  lly register API exposing workloads as API entities
  in TAP GUI.
  backend.appliveview.tanzu.vmware.com Application Live View fo
  r VMware Tanzu                      App for monitoring and troubl
  eshooting running apps
  buildservice.tanzu.vmware.com      Tanzu Build Service
  Tanzu Build Service enables the building and automation of containerized
  software workflows securely and at scale.
  carbonblack.scanning.apps.tanzu.vmware.com VMware Carbon Black for
  Supply Chain Security Tools - Scan  Default scan templates using
  VMware Carbon Black
  cartographer.tanzu.vmware.com       Cartographer
  Kubernetes native Supply Chain Choreographer.
  cnrs.tanzu.vmware.com              Cloud Native Runtimes
  Cloud Native Runtimes is a serverless runtime based on Knative
  connector.appliveview.tanzu.vmware.com Application Live View Co
  nconnector for VMware Tanzu         App for discovering and regis
  tering running apps
  controller.conventions.apps.tanzu.vmware.com Convention Service for V
  Mware Tanzu                         Convention Service enables ap
  p operators to consistently apply desired runtime
  configurations to fleets of workloads.
  controller.source.apps.tanzu.vmware.com Tanzu Source Controller
  Tanzu Source Controller enables workload create/update from source code.
  conventions.appliveview.tanzu.vmware.com Application Live View Co
  nventions for VMware Tanzu         Application Live View convent
  ion server
  developer-conventions.tanzu.vmware.com Tanzu App Platform Devel
  oper Conventions                   Developer Conventions
  eventing.tanzu.vmware.com           Eventing
  Eventing is an event-driven architecture platform based on Knative Eventing
  external-secrets.apps.tanzu.vmware.com External Secrets Operato

```

r	External Secrets Operator is
a Kubernetes operator that integrates external	
secret management systems.	
fluxcd.source.controller.tanzu.vmware.com	Flux Source Controller
The source-controller is a Kubernetes operator, specialised in artifacts	
acquisition from external sources such as Git, Helm repositories and S3 bucket	
s.	
grype.scanning.apps.tanzu.vmware.com	Grype for Supply Chain S
ecurity Tools - Scan	Default scan templates using
Anchore Grype	
learningcenter.tanzu.vmware.com	Learning Center for Tanz
u Application Platform	Guided technical workshops
metadata-store.apps.tanzu.vmware.com	Supply Chain Security To
ols - Store	Post SBOMs and query for imag
e, package, and vulnerability metadata.	
namespace-provisioner.apps.tanzu.vmware.com	Namespace Provisioner
Automatic Provisioning of Developer Namespaces.	
ootb-delivery-basic.tanzu.vmware.com	Tanzu App Platform Out o
f The Box Delivery Basic	Out of The Box Delivery Basi
c.	
ootb-supply-chain-basic.tanzu.vmware.com	Tanzu App Platform Out o
f The Box Supply Chain Basic	Out of The Box Supply Chain B
asic.	
ootb-supply-chain-testing-scanning.tanzu.vmware.com	Tanzu App Platform Out o
f The Box Supply Chain with Testing and Scanning	Out of The Box Supply Chain w
ith Testing and Scanning.	
ootb-supply-chain-testing.tanzu.vmware.com	Tanzu App Platform Out o
f The Box Supply Chain with Testing	Out of The Box Supply Chain w
ith Testing.	
ootb-templates.tanzu.vmware.com	Tanzu App Platform Out o
f The Box Templates	Out of The Box Templates.
policy.apps.tanzu.vmware.com	Supply Chain Security To
ols - Policy Controller	Policy Controller enables def
ining of a policy to restrict unsigned container	
images.	
scanning.apps.tanzu.vmware.com	Supply Chain Security To
ols - Scan	Scan for vulnerabilities and
enforce policies directly within Kubernetes native	
Supply Chains.	
service-bindings.labs.vmware.com	Service Bindings for Kub
ernetes	Service Bindings for Kubernet
es implements the Service Binding Specification.	
services-toolkit.tanzu.vmware.com	Services Toolkit
The Services Toolkit enables the management, lifecycle, discoverability and	
connectivity of Service Resources (databases, message queues, DNS records,	
etc.).	
snyk.scanning.apps.tanzu.vmware.com	Snyk for Supply Chain Se
curity Tools - Scan	Default scan templates using
Snyk	
spring-boot-conventions.tanzu.vmware.com	Tanzu Spring Boot Conven
tions Server	Default Spring Boot conventio
n server.	
sso.apps.tanzu.vmware.com	AppSSO
Application Single Sign-On for Tanzu	
tap-auth.tanzu.vmware.com	Default roles for Tanzu
Application Platform	Default roles for Tanzu Appli
cation Platform	
tap-gui.tanzu.vmware.com	Tanzu Application Platfo
rm GUI	web app graphical user interf
ace for Tanzu Application Platform	



tap-telemetry.tanzu.vmware.com	Telemetry Collector for
Tanzu Application Platform	Tanzu Application Platform Telemetry
tap.tanzu.vmware.com	Tanzu Application Platform
rm	Package to install a set of TAP components to get you started based on your use case.
tekton.tanzu.vmware.com	Tekton Pipelines
Tekton Pipelines is a framework for creating CI/CD systems.	
workshops.learningcenter.tanzu.vmware.com	Workshop Building Tutorial
al	Workshop Building Tutorial

## Install your Tanzu Application Platform profile

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

For more information about profiles, see [Components and installation profiles](#).

To prepare to install a profile:

1. List version information for the package by running:

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

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

Keep the values file for future configuration use.



### Note

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

3. [View possible configuration settings for your package](#)

## Full profile

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

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

  image_registry:
    project_path: "SERVER-NAME/REPO-NAME"
    username: "KP-DEFAULT-REPO-USERNAME"
    password: "KP-DEFAULT-REPO-PASSWORD"

  kubernetes_distribution: "openshift" # To be passed only for OpenShift. Defaults to "".
```

```

kubernetes_version: "K8S-VERSION"

ca_cert_data: | # To be passed if using custom certificates.
-----BEGIN CERTIFICATE-----
MIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIB3DQEBDQUAMEY...
-----END CERTIFICATE-----

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

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

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

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

supply_chain: basic # Can take testing, testing_scanning.

ootb_supply_chain_basic: # Based on supply_chain set above, can be changed to ootb_sup
ply_chain_testing, ootb_supply_chain_testing_scanning.
  registry:
    server: "SERVER-NAME" # Takes the value from shared section above by default, but
can be overridden by setting a different value.
    repository: "REPO-NAME" # Takes the value from shared section above by default, bu
t can be overridden by setting a different value.
  gitops:
    ssh_secret: "SSH-SECRET-KEY" # Takes "" as value by default; but can be overridden
by setting a different value.

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

buildservice:
  kp_default_repository: "KP-DEFAULT-REPO"
  kp_default_repository_username: "KP-DEFAULT-REPO-USERNAME"
  kp_default_repository_password: "KP-DEFAULT-REPO-PASSWORD"

tap_gui:
  app_config:
    catalog:
      locations:
        - type: url
          target: https://GIT-CATALOG-URL/catalog-info.yaml

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

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

grype:
  namespace: "MY-DEV-NAMESPACE"
  targetImagePullSecret: "TARGET-REGISTRY-CREDENTIALS-SECRET"

policy:
  tuf_enabled: false # By default, TUF initialization and keyless verification are dea
ctivated.
tap_telemetry:

```

```
customer_entitlement_account_number: "CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER" # (Optional) Identify data for creating Tanzu Application Platform usage reports.
```

#### Where:

- `INGRESS-DOMAIN` is the subdomain for the host name that you point at the `tanzu-shared-ingress` service's External IP address. It is not required to know the External IP address or set up the DNS record while installing. Installing the Tanzu Application Platform package creates the `tanzu-shared-ingress` and its External IP address. You can create the DNS record after completing the installation.
- `KP-DEFAULT-REPO` is a writable repository in your registry. Tanzu Build Service dependencies are written to this location. Examples:
  - Harbor has the form `kp_default_repository: "my-harbor.io/my-project/build-service"`.
  - Docker Hub has the form `kp_default_repository: "my-dockerhub-user/build-service"` or `kp_default_repository: "index.docker.io/my-user/build-service"`.
  - Google Cloud Registry has the form `kp_default_repository: "gcr.io/my-project/build-service"`.
- `KP-DEFAULT-REPO-USERNAME` is the user name that can write to `KP-DEFAULT-REPO`. You can `docker push` to this location with this credential.
  - For Google Cloud Registry, use `kp_default_repository_username: _json_key`.
  - Alternatively, you can configure this credential as a [secret reference](#).
- `KP-DEFAULT-REPO-PASSWORD` is the password for the user that can write to `KP-DEFAULT-REPO`. You can `docker push` to this location with this credential.
  - For Google Cloud Registry, use the contents of the service account JSON file.
  - Alternatively, you can configure this credential as a [secret reference](#).
- `K8S-VERSION` is the Kubernetes version used by your OpenShift cluster. It must be in the form of `1.23.x` or `1.24.x`, where `x` stands for the patch version. Examples:
  - Red Hat OpenShift Container Platform v4.10 uses the Kubernetes version `1.23.3`.
  - Red Hat OpenShift Container Platform v4.11 uses the Kubernetes version `1.24.1`.
- `SERVER-NAME` is the host name of the registry server. Examples:
  - Harbor has the form `server: "my-harbor.io"`.
  - Docker Hub has the form `server: "index.docker.io"`.
  - Google Cloud Registry has the form `server: "gcr.io"`.
- `REPO-NAME` is where workload images are stored in the registry. If this key is passed through the shared section earlier and AWS ECR registry is used, you must ensure that the `SERVER-NAME/REPO-NAME/buildservice` and `SERVER-NAME/REPO-NAME/workloads` exist. AWS ECR expects the paths to be pre-created. Images are written to `SERVER-NAME/REPO-NAME/workload-name`. Examples:
  - Harbor has the form `repository: "my-project/supply-chain"`.
  - Docker Hub has the form `repository: "my-dockerhub-user"`.
  - Google Cloud Registry has the form `repository: "my-project/supply-chain"`.
- `SSH-SECRET-KEY` is the SSH secret key in the developer namespace for the supply chain to fetch source code from and push configuration to. This field is only required if you use a private repository, otherwise, leave it empty. See [Git authentication](#) for more information.

- `GIT-CATALOG-URL` is the path to the `catalog-info.yaml` catalog definition file. You can download either a blank or populated catalog file from the [Tanzu Application Platform product page](#). Otherwise, you can use a Backstage-compliant catalog you've already built and posted on the Git infrastructure.
- `MY-DEV-NAMESPACE` is the name of the developer namespace. SCST - Store exports secrets to the namespace, and SCST - Scan deploys the `ScanTemplates` there. This allows the scanning feature to run in this namespace. If there are multiple developer namespaces, use `ns_for_export_app_cert: "*"`  to export the SCST - Store CA certificate to all namespaces. To install Grype in multiple namespaces, use a namespace provisioner. For more information, see [Namespace Provisioner](#).
- `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the secret that contains the credentials to pull an image from the registry for scanning.
- `CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER` (optional) refers to the Entitlement Account Number (EAN), which is a unique identifier VMware assigns to its customers. Tanzu Application Platform telemetry uses this number to identify data that belongs to a particular customers and prepare usage reports. See [Locating the Entitlement Account number for new orders](#) for more information about identifying the Entitlement Account Number.

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

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

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

## CEIP policy disclosure

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

See [Opt out of telemetry collection](#) for more information.

## (Optional) Additional Build Service configurations

The following tasks are optional during the Tanzu Application Platform installation process:

- [\(Optional\) Configure your profile with full dependencies](#)
- [\(Optional\) Configure your profile with the Jammy stack only](#)

### (Optional) Configure your profile with full dependencies

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

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

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

```
buildservice:
  kp_default_repository: "KP-DEFAULT-REPO"
  kp_default_repository_username: "KP-DEFAULT-REPO-USERNAME"
  kp_default_repository_password: "KP-DEFAULT-REPO-PASSWORD"
  exclude_dependencies: true
```

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

#### (Optional) Configure your profile with the Jammy stack only

Tanzu Application Platform v1.3.0 supports building applications with the [Ubuntu 22.04 \(Jammy\) stack](#). By default, workloads are built with Ubuntu 18.04 (Bionic) stack. However, if you do not need access to the Bionic stack, you can install Tanzu Application Platform without the Bionic stack and all workloads are built with the Jammy stack by default.

To install Tanzu Application Platform with Jammy as the only available stack, include the `stack_configuration: jammy-only` field under the `buildservice:` section in `tap-values.yaml`.

## Install your Tanzu Application Platform package

Follow these steps to install the Tanzu Application Platform package:

1. Install the package by running:

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

2. Verify the package install by running:

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

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

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

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

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



#### Important

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

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

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

## Install the full dependencies package

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

For more information about the differences between `lite` and `full` dependencies, see [About lite and full dependencies](#).

To install the `full` dependencies package:

1. If you have not done so already, add the key-value pair `exclude_dependencies: true` to your `tap-values.yaml` file under the `buildservice` section. For example:

```
buildservice:
  kp_default_repository: "KP-DEFAULT-REPO"
  kp_default_repository_username: "KP-DEFAULT-REPO-USERNAME"
  kp_default_repository_password: "KP-DEFAULT-REPO-PASSWORD"
  exclude_dependencies: true
  ...
```

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

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

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

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

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

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

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

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

5. Install the full dependencies package by running:

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

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

## Access Tanzu Application Platform GUI

To access Tanzu Application Platform GUI, you can use the host name that you configured earlier. This host name is pointed at the shared ingress. To configure LoadBalancer for Tanzu Application Platform GUI, see [Access Tanzu Application Platform GUI](#).

You're now ready to start using Tanzu Application Platform GUI. Proceed to the [Getting Started](#) topic or the [Tanzu Application Platform GUI - Catalog Operations](#) topic.

## Exclude packages from a Tanzu Application Platform profile

To exclude packages from a Tanzu Application Platform profile:

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

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

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

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



### Important

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

## Next steps

- (Optional) [Install individual packages](#)
- [Set up developer namespaces to use your installed packages](#)

## 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
```



### Note

The `tap.tanzu.vmware.com` package does not show all configuration settings for packages it plans to install. The package only shows top-level keys. You can view individual package configuration settings with the same `tanzu package available get` command. For example, to find the keys for Cloud Native Runtimes, you must first identify the version of the package with `tanzu package installed list -n tap-install`, which lists all the installed packages versions. Then run the command `tanzu package available get -n tap-install cnrs.tanzu.vmware.com/CNRS-VERSION --values-schema` by using the package version listed for Cloud Native Runtimes.

```

profile: full

# ...

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

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

```

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

Package	Top-level Key
<i>see table below</i>	<code>shared</code>
API Auto Registration	<code>api_auto_registration</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>
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>
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>scanning</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 [Install individual packages](#).



## Install individual packages

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

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

## Install pages for individual Tanzu Application Platform packages

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

- [Install Telemetry](#)

## Verify the installed packages

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

1. List the installed packages by running:

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

For example:

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

## Next steps

- [Set up developer namespaces to use your installed packages](#)

## Set up developer namespaces to use your installed packages

To automatically set up your developer namespaces, see [Provision namespace resources](#).

For more information about Namespace Provisioner, see [Namespace Provisioner overview](#).

## Additional configuration for testing and scanning

If you plan to install or have already installed Out of the Box Supply Chains with Testing and Scanning, you can use Namespace Provisioner to set up the required resources. For more information, see [Add the resources required by Out of the Box Testing and Scanning Supply Chain](#).

## Legacy namespace setup

To use the legacy manual process for setting up developer namespaces, see [Legacy namespace setup](#).

## Next steps

- [Install Tanzu Developer Tools for your VS Code](#)

## Legacy manual developer namespace setup instructions

Using [Namespace Provisioner](#) is the recommended best practice for setting up developer namespaces on Tanzu Application Platform.

To provision namespaces manually, complete the following steps:

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

## Enable single user access

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

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

Where:

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

If you observe the following issue:

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

Use kubectl to create the secret instead:

```
kubectl create secret docker-registry registry-credentials --docker-server=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.

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

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

```
subjects:
  - kind: ServiceAccount
    name: default
EOF
```



### Note

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

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

## Enable additional users with Kubernetes RBAC

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

1. Before you begin, ensure that you have [enabled single user access](#).
2. Choose either of the following options to give developers namespace-level access and view access to the appropriate cluster-level resources:
  - o **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 (Azure AD) with your cluster, see [Integrate Azure Active Directory](#).

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

Run the following to apply the RBAC policy:

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

Where:

- `YOUR-NAMESPACE` is the name you give to the developer namespace.

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

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

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

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

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

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

## Additional configuration for testing and scanning

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

## Install Tanzu Developer Tools for your VS Code

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

### Prerequisites

Before installing the extension, you must have:

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

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

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

### Install

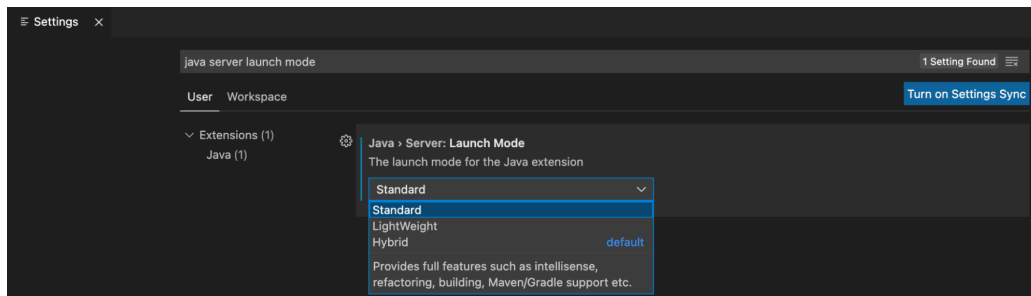
To install the extension:

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:
  - o [Debugger for Java](#)
  - o [Language Support for Java\(™\) by Red Hat](#)
  - o [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:
  - o **Confirm Delete:** This controls whether the extension asks for confirmation when deleting a workload.
  - o **Enable Live Hover:** For more information, see [Integrating Live Hover by using Spring Boot Tools](#). Reload VS Code for this change to take effect.
  - o **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.
  - o **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.
  - o **Namespace:** (Optional) This is the namespace that workloads are deployed into. The namespace set in `kubeconfig` is the default.

## Uninstall



To uninstall VMware Tanzu Developer Tools for VS Code:

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

## Next steps

Proceed to [Getting started with Tanzu Developer Tools for Visual Studio Code](#).

## Install Tanzu Application Platform (offline)

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

Step	Task	Link
1.	Review the prerequisites to ensure you have met all requirements before installing.	<a href="#">Prerequisites</a>
2.	Accept Tanzu Application Platform EULAs and install the Tanzu CLI.	<a href="#">Accept Tanzu Application Platform EULAs and installing the Tanzu CLI</a>
3.	Install Cluster Essentials for Tanzu*.	<a href="#">Deploy Cluster Essentials</a>
4.	Add the Tanzu Application Platform package repository, prepare your Tanzu Application Platform profile, and install the profile to the cluster.	<a href="#">Install Tanzu Application Platform in an air-gapped environment</a>
5.	Install Tanzu Build Service full dependencies.	<a href="#">Install the Tanzu Build Service dependencies</a>
6.	Configure custom certificate authorities for Tanzu Application Platform GUI.	<a href="#">Configure custom certificate authorities for Tanzu Application Platform GUI</a>
7.	Add the certificate for the private Git repository in the Accelerator system namespace.	<a href="#">Configure Application Accelerator</a>
8.	Apply patch to Grype.	<a href="#">Use Grype in offline and air-gapped environments</a>
9.	Set up developer namespaces to use your installed packages.	<a href="#">Set up developer namespaces to use your installed packages</a>

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

After installing Tanzu Application Platform on to your air-gapped cluster, you can start creating workloads that run in your air-gapped containers. For instructions, see [Deploy an air-gapped workload](#).

## Install Tanzu Application Platform (offline)

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

Step	Task	Link
1.	Review the prerequisites to ensure you have met all requirements before installing.	<a href="#">Prerequisites</a>
2.	Accept Tanzu Application Platform EULAs and install the Tanzu CLI.	<a href="#">Accept Tanzu Application Platform EULAs and installing the Tanzu CLI</a>

Step	Task	Link
3.	Install Cluster Essentials for Tanzu*.	<a href="#">Deploy Cluster Essentials</a>
4.	Add the Tanzu Application Platform package repository, prepare your Tanzu Application Platform profile, and install the profile to the cluster.	<a href="#">Install Tanzu Application Platform in an air-gapped environment</a>
5.	Install Tanzu Build Service full dependencies.	<a href="#">Install the Tanzu Build Service dependencies</a>
6.	Configure custom certificate authorities for Tanzu Application Platform GUI.	<a href="#">Configure custom certificate authorities for Tanzu Application Platform GUI</a>
7.	Add the certificate for the private Git repository in the Accelerator system namespace.	<a href="#">Configure Application Accelerator</a>
8.	Apply patch to Grype.	<a href="#">Use Grype in offline and air-gapped environments</a>
9.	Set up developer namespaces to use your installed packages.	<a href="#">Set up developer namespaces to use your installed packages</a>

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

After installing Tanzu Application Platform on to your air-gapped cluster, you can start creating workloads that run in your air-gapped containers. For instructions, see [Deploy an air-gapped workload](#).

## Install Tanzu Application Platform in your air-gapped environment

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

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

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

## Relocate images to a registry

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

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

```
export IMGPKG_REGISTRY_HOSTNAME_0=registry.tanzu.vmware.com
export IMGPKG_REGISTRY_USERNAME_0=MY-TANZUNET-USERNAME
export IMGPKG_REGISTRY_PASSWORD_0=MY-TANZUNET-PASSWORD
export IMGPKG_REGISTRY_HOSTNAME_1=MY-REGISTRY
export IMGPKG_REGISTRY_USERNAME_1=MY-REGISTRY-USER
export IMGPKG_REGISTRY_PASSWORD_1=MY-REGISTRY-PASSWORD
export TAP_VERSION=VERSION-NUMBER
export REGISTRY_CA_PATH=PATH-TO-CA
```

Where:

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

- o `MY-REGISTRY-USER` is the user with write access to `MY-REGISTRY`.
  - o `MY-REGISTRY-PASSWORD` is the password for `MY-REGISTRY-USER`.
  - o `MY-TANZUNET-USERNAME` is the user with access to the images in the VMware Tanzu Network registry `registry.tanzu.vmware.com`
  - o `MY-TANZUNET-PASSWORD` is the password for `MY-TANZUNET-USERNAME`.
  - o `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.4.13`
2. Copy the images into a `.tar` file from the VMware Tanzu Network onto an external storage device with the Carvel tool `imgpkg` by running:

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

Where:

- o `TANZUNET-REGISTRY-USERNAME` is your username of the VMware Tanzu Network.
  - o `TANZUNET-REGISTRY-PASSWORD` is your password of the VMware Tanzu Network.
3. Relocate the images with the Carvel tool `imgpkg` by running:

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

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

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

This namespace keeps the objects grouped together logically.

5. Create a registry secret by running:

```
tanzu secret registry add tap-registry \
  --server $IMGPKG_REGISTRY_HOSTNAME \
  --username $IMGPKG_REGISTRY_USERNAME \
  --password $IMGPKG_REGISTRY_PASSWORD \
  --namespace tap-install \
  --export-to-all-namespaces \
  --yes
```

6. Create a internal registry secret by running:

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

Where:

- o `MY_REGISTRY` is where the workload images and the Tanzu Build Service dependencies are stored.

- `MY_REGISTRY_USER` is the user with write access to `MY_REGISTRY`.
- `MY_REGISTRY_PASSWORD` is the password for `MY_REGISTRY_USER`.

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

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

Where:

- `$TAP_VERSION` is the Tanzu Application Platform version environment variable you defined earlier.
  - `TARGET-REPOSITORY` is the necessary repository.
8. Get the status of the Tanzu Application Platform package repository, and ensure the status updates to `Reconcile succeeded` by running:

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



#### Note

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

9. List the available packages by running:

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

For example:

```
$ tanzu package available list --namespace tap-install
/ Retrieving available packages...
NAME                                DISPLAY-NAME
SHORT-DESCRIPTION
  accelerator.apps.tanzu.vmware.com  Application Accelerator
for VMware Tanzu                    Used to create new projects a
nd configurations.
  api-portal.tanzu.vmware.com        API portal
A unified user interface for API discovery and exploration at scale.
  apis.apps.tanzu.vmware.com         API Auto Registration fo
r VMware Tanzu                       A TAP component to automatica
lly register API exposing workloads as API entities
in TAP GUI.
  backend.appliveview.tanzu.vmware.com Application Live View fo
r VMware Tanzu                        App for monitoring and troubl
eshooting running apps
  buildservice.tanzu.vmware.com      Tanzu Build Service
Tanzu Build Service enables the building and automation of containerized
software workflows securely and at scale.
  carbonblack.scanning.apps.tanzu.vmware.com VMware Carbon Black for
Supply Chain Security Tools - Scan    Default scan templates using
VMware Carbon Black
  cartographer.tanzu.vmware.com      Cartographer
Kubernetes native Supply Chain Choreographer.
  cnrs.tanzu.vmware.com              Cloud Native Runtimes
Cloud Native Runtimes is a serverless runtime based on Knative
  connector.appliveview.tanzu.vmware.com Application Live View Co
```

connector for VMware Tanzu tering running apps	App for discovering and regis tering running apps
controller.conventions.apps.tanzu.vmware.com Mware Tanzu	Convention Service for V Convention Service enables ap p operators to consistently apply desired runtime
configurations to fleets of workloads.	
controller.source.apps.tanzu.vmware.com	Tanzu Source Controller
Tanzu Source Controller enables workload create/update from source code.	
conventions.appliveview.tanzu.vmware.com	Application Live View Co
ventions for VMware Tanzu	Application Live View convent
ion server	
developer-conventions.tanzu.vmware.com	Tanzu App Platform Devel
oper Conventions	Developer Conventions
eventing.tanzu.vmware.com	Eventing
Eventing is an event-driven architecture platform based on Knative Eventing	
external-secrets.apps.tanzu.vmware.com	External Secrets Operato
r	External Secrets Operator is
a Kubernetes operator that integrates external	
secret management systems.	
fluxcd.source.controller.tanzu.vmware.com	Flux Source Controller
The source-controller is a Kubernetes operator, specialised in artifacts	
acquisition from external sources such as Git, Helm repositories and S3 bucket s.	
grype.scanning.apps.tanzu.vmware.com	Grype for Supply Chain S
ecurity Tools - Scan	Default scan templates using
Anchore Grype	
learningcenter.tanzu.vmware.com	Learning Center for Tanz
u Application Platform	Guided technical workshops
metadata-store.apps.tanzu.vmware.com	Supply Chain Security To
ols - Store	Post SBOMs and query for imag
e, package, and vulnerability metadata.	
namespace-provisioner.apps.tanzu.vmware.com	Namespace Provisioner
Automatic Provisioning of Developer Namespaces.	
ootb-delivery-basic.tanzu.vmware.com	Tanzu App Platform Out o
f The Box Delivery Basic	Out of The Box Delivery Basi
c.	
ootb-supply-chain-basic.tanzu.vmware.com	Tanzu App Platform Out o
f The Box Supply Chain Basic	Out of The Box Supply Chain B
asic.	
ootb-supply-chain-testing-scanning.tanzu.vmware.com	Tanzu App Platform Out o
f The Box Supply Chain with Testing and Scanning	Out of The Box Supply Chain w
ith Testing and Scanning.	
ootb-supply-chain-testing.tanzu.vmware.com	Tanzu App Platform Out o
f The Box Supply Chain with Testing	Out of The Box Supply Chain w
ith Testing.	
ootb-templates.tanzu.vmware.com	Tanzu App Platform Out o
f The Box Templates	Out of The Box Templates.
policy.apps.tanzu.vmware.com	Supply Chain Security To
ols - Policy Controller	Policy Controller enables def
ining of a policy to restrict unsigned container	
images.	
scanning.apps.tanzu.vmware.com	Supply Chain Security To
ols - Scan	Scan for vulnerabilities and
enforce policies directly within Kubernetes native	
Supply Chains.	
service-bindings.labs.vmware.com	Service Bindings for Kub
ernetes	Service Bindings for Kubernet
es implements the Service Binding Specification.	
services-toolkit.tanzu.vmware.com	Services Toolkit
The Services Toolkit enables the management, lifecycle, discoverability and	

connectivity of Service Resources (databases, message queues, DNS records, etc.).	
<code>snyk.scanning.apps.tanzu.vmware.com</code>	Snyk for Supply Chain Security Tools - Scan Snyk
<code>spring-boot-conventions.tanzu.vmware.com</code>	Tanzu Spring Boot Conventions Server
<code>sso.apps.tanzu.vmware.com</code>	AppSSO
<code>tap-auth.tanzu.vmware.com</code>	Default roles for Tanzu Application Platform
<code>tap-gui.tanzu.vmware.com</code>	Tanzu Application Platform GUI
<code>tap-telemetry.tanzu.vmware.com</code>	Telemetry Collector for Tanzu Application Platform
<code>tap.tanzu.vmware.com</code>	Tanzu Application Platform Package to install a set of TAP components to get you started based on your use case.
<code>tekton.tanzu.vmware.com</code>	Tekton Pipelines
<code>workshops.learningcenter.tanzu.vmware.com</code>	Workshop Building Tutorial

## Prepare Sigstore Stack for air-gapped policy controller

**Important**

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

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

The Sigstore Stack consists of:

- [Trillian](#)
- [Rekor](#)
- [Fulcio](#)
- [Certificate Transparency Log \(CTLog\)](#)
- [The Update Framework \(TUF\)](#)

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

## Install your Tanzu Application Platform profile

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

For more information about profiles, see [Components and installation profiles](#).

To prepare to install a profile:

1. List version information for the package by running:

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

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

Keep the values file for future configuration use.

## Full Profile

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

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



### Important

Tanzu Build Service is installed by default with `lite` dependencies. When installing Tanzu Build Service in an air-gapped environment, the lite dependencies are not available because they require Internet access. You must install the `full` dependencies by setting `exclude_dependencies` to `true`.

```
shared:
  ingress_domain: "INGRESS-DOMAIN"
  image_registry:
    project_path: "SERVER-NAME/REPO-NAME"
  secret:
    name: "KP-DEFAULT-REPO-SECRET"
    namespace: "KP-DEFAULT-REPO-SECRET-NAMESPACE"
  ca_cert_data: |
    -----BEGIN CERTIFICATE-----
    MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...
    -----END CERTIFICATE-----
profile: full
ceip_policy_disclosed: true
buildservice:
  kp_default_repository: "KP-DEFAULT-REPO"
  kp_default_repository_secret: # Takes the value from the shared section by default,
  but can be overridden by setting a different value.
  name: "KP-DEFAULT-REPO-SECRET"
  namespace: "KP-DEFAULT-REPO-SECRET-NAMESPACE"
  exclude_dependencies: true
supply_chain: basic
scanning:
  metadataStore:
    url: ""
contour:
  infrastructure_provider: aws
envoy:
  service:
    type: LoadBalancer
```

```

    annotations:
      # This annotation is for air-gapped AWS only.
      service.kubernetes.io/aws-load-balancer-internal: "true"

ootb_supply_chain_basic:
  registry:
    server: "SERVER-NAME" # Takes the value from the shared section by default, but
    can be overridden by setting a different value.
    repository: "REPO-NAME" # Takes the value from the shared section by default, bu
    t can be overridden by setting a different value.
  gitops:
    ssh_secret: "SSH-SECRET"
  maven:
    repository:
      url: https://MAVEN-URL
      secret_name: "MAVEN-CREDENTIALS"

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

appliveview:
  ingressEnabled: true

appliveview_connector:
  backend:
    ingressEnabled: true
    sslDeactivated: false
    host: appliveview.INGRESS-DOMAIN
    caCertData: |-
      -----BEGIN CERTIFICATE-----
      MIIGMzCCBBUGAwIBAgIJALHHzQjxM6wMMA0GCSqGSIb3DQEEDQUAMGcxCzAJBgNV
      BAQMAk1OMRQWEgYDVQQHDAcNaW5uZWVwb2xpczEPMA0GA1UECgwGVk13YXJlMRMw
      -----END CERTIFICATE-----

tap_gui:
  app_config:
    kubernetes:
      serviceLocatorMethod:
        type: multiTenant
      clusterLocatorMethods:
        - type: config
      clusters:
        - url: https://${KUBERNETES_SERVICE_HOST}:${KUBERNETES_SERVICE_PORT}
          name: host
          authProvider: serviceAccount
          serviceAccountToken: ${KUBERNETES_SERVICE_ACCOUNT_TOKEN}
          skipTLSVerify: false
          caData: B64_ENCODED_CA

    catalog:
      locations:
        - type: url
          target: https://GIT-CATALOG-URL/catalog-info.yaml
      #Example Integration for custom GitLab:
    integrations:
      gitlab:
        - host: GITLAB-URL
          token: GITLAB-TOKEN
          apiBaseUrl: https://GITLABURL/api/v4/

  backend:
    reading:

```



```

    allow:
      - host: GITLAB-URL # Example URL: gitlab.example.com

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

grype:
  namespace: "MY-DEV-NAMESPACE"
  targetImagePullSecret: "TARGET-REGISTRY-CREDENTIALS-SECRET"

```

#### Where:

- `INGRESS-DOMAIN` is the subdomain for the host name that you point at the `tanzu-shared-ingress` service's External IP address.
- `KP-DEFAULT-REPO` is a writable repository in your registry. Tanzu Build Service dependencies are written to this location. Examples:
  - Harbor has the form `kp_default_repository: "my-harbor.io/my-project/build-service"`.
  - Docker Hub has the form `kp_default_repository: "my-dockerhub-user/build-service"` or `kp_default_repository: "index.docker.io/my-user/build-service"`.
  - Google Cloud Registry has the form `kp_default_repository: "gcr.io/my-project/build-service"`.
- `KP-DEFAULT-REPO-SECRET` is the user name that can write to `KP-DEFAULT-REPO`. You can `docker push` to this location with this credential.
  - For Google Cloud Registry, use `kp_default_repository_username: _json_key`.
  - You must create the secret before the installation. For example, you can use the `registry-credentials` secret created earlier.
- `KP-DEFAULT-REPO-SECRET-NAMESPACE` is the namespace where `KP-DEFAULT-REPO-SECRET` is created.
- `SERVER-NAME` is the host name of the registry server. Examples:
  - Harbor has the form `server: "my-harbor.io"`.
  - Docker Hub has the form `server: "index.docker.io"`.
  - Google Cloud Registry has the form `server: "gcr.io"`.
- `REPO-NAME` is where workload images are stored in the registry. If this key is passed through the shared section earlier and AWS ECR registry is used, you must ensure that the `SERVER-NAME/REPO-NAME/buildservice` and `SERVER-NAME/REPO-NAME/workloads` exist. AWS ECR expects the paths to be pre-created.
- Images are written to `SERVER-NAME/REPO-NAME/workload-name`. Examples:
  - Harbor has the form `repository: "my-project/supply-chain"`.
  - Docker Hub has the form `repository: "my-dockerhub-user"`.
  - Google Cloud Registry has the form `repository: "my-project/supply-chain"`.
- `SSH-SECRET` is the secret name for https authentication, certificate authority, and SSH authentication. See [Git authentication](#) for more information.
- `MAVEN-CREDENTIALS` is the name of [the secret with maven creds](#). This secret must be in the developer namespace. You can create it after the fact.
- `GIT-CATALOG-URL` is the path to the `catalog-info.yaml` catalog definition file. You can download either a blank or populated catalog file from the [Tanzu Application Platform](#)

[product page](#). Otherwise, you can use a Backstage-compliant catalog you've already built and posted on the Git infrastructure.

- `GITLABURL` is the host name of your GitLab instance.
- `GITLAB-USER` is the user name of your GitLab instance.
- `GITLAB-PASSWORD` is the password for the `GITLAB-USER` of your GitLab instance. This can also be the `GITLAB-TOKEN`.
- `GITLAB-TOKEN` is the API token for your GitLab instance.
- `MY-DEV-NAMESPACE` is the name of the developer namespace. SCST - Store exports secrets to the namespace, and SCST - Scan deploys the `ScanTemplates` there. This allows the scanning feature to run in this namespace. If there are multiple developer namespaces, use `ns_for_export_app_cert: "*"`  to export the SCST - Store CA certificate to all namespaces.

**Note:** To install Grype in multiple namespaces, use a namespace provisioner. See [Namespace Provisioner](#).

- `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the secret that contains the credentials to pull an image from the registry for scanning.



#### Note

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

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

The `ingressEnabled` key is set to `false` by default. Set this key to `true` for the Application Live View back end to be exposed on the ingress domain. This creates a HTTPProxy object in the cluster.

You must create the `app-live-view` namespace and the TLS secret `appliveview-cert` for the domain before installing the Tanzu Application Platform packages on the cluster so that the HTTPProxy is updated with the TLS secret. To create a TLS secret, run:

```
kubectl create -n app-live-view secret tls appliveview-cert --cert=CRT-FILE --key=KEY-FILE
```

To verify the HTTPProxy object with the secret, run:

```
kubectl get httpproxy -A
```

Expected output:

```

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

```

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

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

## Install your Tanzu Application Platform package

Follow these steps to install the Tanzu Application Platform package:

1. Install the package by running:

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

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

2. Verify the package install by running:

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

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

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

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

## Next steps

- [Install the Tanzu Build Service dependencies](#)

## Install the Tanzu Build Service dependencies

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

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

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

To install `full` dependencies:

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

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

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

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

Where:

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

- `TARGET-REPOSITORY` is your target repository.

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

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

Where:

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

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

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

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

## Next steps

- [Configure custom CAs for Tanzu Application Platform GUI](#)

## Configure custom certificate authorities for Tanzu Application Platform GUI

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

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

### Add a custom CA

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

For example:

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

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

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

```

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

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

```

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

### Deactivate all SSL verification

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

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

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

```

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

```

Where `NAMESPACE` is the namespace in which your Tanzu Application Platform GUI instance is deployed. For example, `tap-gui`.

## Next steps

- [Configure Application Accelerator](#)

## Configure Application Accelerator

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

public repositories and in air-gapped environments.

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

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

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

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

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

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

Where:

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

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

## Functional and Organizational Considerations

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

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

## Examples for creating accelerators

### A minimal example for creating an accelerator

A minimal example might look like the following manifest:

`spring-cloud-serverless.yaml`

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

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

`accelerator.yaml`

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

To create this accelerator with `kubectl`, run:

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

Or, you can use the Tanzu CLI and run:

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

## An example for creating an accelerator with customized properties

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

`my-spring-cloud-serverless.yaml`

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

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

To create this accelerator with kubectl, run:

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

To use the Tanzu CLI, run:

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



#### Note

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

## Creating a manifest with multiple accelerators and fragments

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

`accelerator-collection.yaml`

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



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

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



### Note

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

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

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

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

Where:

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

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

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

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

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

```

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

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

```

## Using non-public repositories

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

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

## Examples for a private Git repository

### Example using http credentials

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



#### Note

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

```

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

```

Verify that your secret was created by running:

```

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

```

The output is similar to:

```

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

```

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

After you created and verified the secret, you can create the accelerator by using the `spec.git.secretRef.name` property:

`private-acc.yaml`

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: private-acc
spec:
  displayName: private
  description: Accelerator using a private repository
  git:
    url: REPOSITORY-URL
    ref:
      branch: main
    secretRef:
      name: https-credentials
```

For https credentials, the `REPOSITORY-URL` must use `https://` as the URL scheme.

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

#### Example using http credentials with self-signed certificate

To create an accelerator using a private Git repository with a self-signed certificate, create a secret with the HTTP credentials and the certificate.



#### Note

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

```
kubectl create secret generic https-ca-credentials \
  --namespace accelerator-system \
  --from-literal=username=<user> \
  --from-literal=password=<access-token> \
  --from-file=caFile=<path-to-CA-file>
```

Verify that your secret was created by running:

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

The output is similar to:

```
apiVersion: v1
kind: Secret
metadata:
  name: https-ca-credentials
  namespace: accelerator-system
type: Opaque
data:
  username: <BASE64>
  password: <BASE64>
  caFile: <BASE64>
```

After you have the secret created, you can create the accelerator by using the `spec.git.secretRef.name` property:

`private-acc.yaml`

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: private-acc
spec:
  displayName: private
  description: Accelerator using a private repository
  git:
    url: REPOSITORY-URL
    ref:
      branch: main
    secretRef:
      name: https-ca-credentials
```



### Important

For https credentials, the `REPOSITORY-URL` must use `https://` as the URL scheme.

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

### Example using SSH credentials

To create an accelerator using a private Git repository, create a secret with the SSH credentials such as this example:

```
ssh-keygen -q -N "" -f ./identity
ssh-keyscan github.com > ./known_hosts
kubectl create secret generic ssh-credentials \
  --namespace accelerator-system \
  --from-file=./identity \
  --from-file=./identity.pub \
  --from-file=./known_hosts
```

If you have a key file already created, skip the `ssh-keygen` and `ssh-keyscan` steps and replace the values for the `kubectl create secret` command. Such as:

- `--from-file=identity=<path to your identity file>`
- `--from-file=identity.pub=<path to your identity.pub file>`
- `--from-file=known_hosts=<path to your know_hosts file>`

Verify that your secret was created by running:

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

The output is similar to :

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

```
identity: <BASE64>
identity.pub: <BASE64>
known_hosts: <BASE64>
```

To use this secret when creating an accelerator, provide the secret name in the `spec.git.secretRef.name` property:

`private-acc-ssh.yaml`

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: private-acc
spec:
  displayName: private
  description: Accelerator using a private repository
  git:
    url: REPOSITORY-URL
    ref:
      branch: main
    secretRef:
      name: ssh-credentials
```

When using SSH credentials, the `REPOSITORY-URL` must include the user name as part of the URL. For example: `ssh://user@example.com:22/repository.git`. For more information, see [Flux documentation](#).

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

## Examples for a private source-image repository

If your registry uses a self-signed certificate then you must add the CA certificate data to the configuration for the “Tanzu Application Platform Source Controller” component. Add it under `source_controller.ca_cert_data` in your `tap-values.yaml` file that is used during installation.

`tap-values.yaml`

```
source_controller:
  ca_cert_data: |-
    -----BEGIN CERTIFICATE-----
    .
    .
    . < certificate data >
    .
    .
    -----END CERTIFICATE-----
```

### Example using image-pull credentials

To create an accelerator using a private source-image repository, create a secret with the image-pull credentials:

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

Verify that your secret was created by running:

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

The output is similar to:

```
apiVersion: v1
kind: Secret
metadata:
  name: registry-credentials
  namespace: accelerator-system
type: Opaque
data:
  username: <BASE64>
  password: <BASE64>
```

After you have the secret created, you can create the accelerator by using the `spec.git.secretRef.name` property:

`private-acc.yaml`

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: private-acc
spec:
  displayName: private
  description: Accelerator using a private repository
  source:
    image: "registry.example.com/test/private-acc-src:latest"
    imagePullSecrets:
      - name: registry-credentials
```

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

## Configure ingress timeouts when some accelerators take longer to generate

If Tanzu Application Platform is configured to use an ingress for Tanzu Application Platform GUI and the Accelerator Server, then it might detect a timeout during accelerator generation. This can happen if the accelerator takes a longer time to generate than the default timeout. When this happens, Tanzu Application Platform GUI appears to continue to run for an indefinite period. In the IDE extension, it shows a 504 error. To mitigate this, you can increase the timeout value for the HTTPProxy resources used for the ingress by applying secrets with overlays to edit the HTTPProxy resources.

### Configure an ingress timeout overlay secret for each HTTPProxy

For Tanzu Application Platform GUI, create the following overlay secret in the `tap-install` namespace:

```
apiVersion: v1
kind: Secret
metadata:
  name: patch-tap-gui-timeout
  namespace: tap-install
stringData:
  patch.yaml: |
    #@ load("@ytt:overlay", "overlay")
    #@overlay/match by=overlay.subset({"kind": "HTTPProxy", "metadata": {"name": "tap-gui"}})
    ---
    spec:
      routes:
```

```

#@overlay/match by=overlay.subset({"services": [{"name": "server"}]})
#@overlay/match-child-defaults missing_ok=True
- timeoutPolicy:
  idle: 30s
  response: 30s

```

For Accelerator Server (used for IDE extension), create the following overlay secret in the `tap-install` namespace:

```

apiVersion: v1
kind: Secret
metadata:
  name: patch-accelerator-timeout
  namespace: tap-install
stringData:
  patch.yaml: |
    #@ load("@ytt:overlay", "overlay")
    #@overlay/match by=overlay.subset({"kind": "HTTPProxy", "metadata": {"name": "accelerator"}})
    ---
    spec:
      routes:
        #@overlay/match by=overlay.subset({"services": [{"name": "acc-server"}]})
        #@overlay/match-child-defaults missing_ok=True
        - timeoutPolicy:
            idle: 30s
            response: 30s

```

## Apply the timeout overlay secrets in tap-values.yaml

Add the following `package_overlays` section to `tap-values.yaml` before installing or updating Tanzu Application Platform:

```

package_overlays:
- name: tap-gui
  secrets:
  - name: patch-tap-gui-timeout
- name: accelerator
  secrets:
  - name: patch-accelerator-timeout

```

## Configuring skipping TLS verification for access to Source Controller

You can configure the Flux or Tanzu Application Platform Source Controller to use Transport Layer Security (TLS) and use custom certificates. In that case, configure the Accelerator System to skip the TLS verification for calls to access the sources by providing the following property in the `accelerator` section of the `tap-values.yaml` file:

```

sources:
  skip_tls_verify: true

```

## Enabling TLS for Accelerator Server

To enable TLS for the Accelerator Server, the following properties must be provided in the `accelerator` section of the `tap-values.yaml` file:

```

server:
  tls:

```

```
enabled: true
key: SERVER-PRIVATE-KEY
crt: SERVER-CERTIFICATE
```

Where:

- `SERVER-PRIVATE-KEY` is the pem encoded server private key.
- `SERVER-CERTIFICATE` is the pem encoded server certificate.

Here is a sample `tap-values.yaml` configuration with TLS enabled for Accelerators Server:

```
server:
  tls:
    enabled: true
    key: |
      -----BEGIN PRIVATE KEY-----
      .
      . < private key data >
      .
      -----END PRIVATE KEY-----
    crt: |
      -----BEGIN CERTIFICATE-----
      .
      . < certificate data >
      .
      -----END CERTIFICATE-----
```

## Configuring skipping TLS verification of Engine calls for Accelerator Server

If you configure the Accelerator Engine to use TLS and use custom certificates, then you can configure the Accelerator Server to skip the TLS verification for calls to the Engine by providing the following property in the `accelerator` section of the `tap-values.yaml` file:

```
server:
  engine_skip_tls_verify: true
```

## Enabling TLS for Accelerator Engine

To enable TLS for the Accelerator Engine, the following properties are provided in the `accelerator` section of the `tap-values.yaml` file:

```
engine:
  tls:
    enabled: true
    key: ENGINE-PRIVATE-KEY
    crt: ENGINE-CERTIFICATE
```

Where:

- `ENGINE-PRIVATE-KEY` is the pem encoded acc-engine private key.
- `ENGINE-CERTIFICATE` is the pem encoded acc-engine certificate.

Here is a sample `tap-values.yaml` configuration with TLS enabled for Accelerators Engine:

```
engine:
  tls:
    enabled: true
    key: |
      -----BEGIN PRIVATE KEY-----
```



```

.
. < private key data >
.
-----END PRIVATE KEY-----
cert: |
-----BEGIN CERTIFICATE-----
.
. < certificate data >
.
-----END CERTIFICATE-----

```

## Next steps

- [Using Grype in offline and air-gapped environments](#)

## Use Grype in offline and air-gapped environments

The `grype` CLI attempts to perform two over the Internet calls:

- One to verify for later versions of the CLI.
- One to update the vulnerability database before scanning.

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

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

For information about setting up an offline vulnerability database, see the [Anchore Grype README](#) in GitHub.

## To enable Grype in offline air-gapped environments

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

```

grype:
  db:
    dbUpdateUrl: INTERNAL-VULN-DB-URL

```

Where `INTERNAL-VULN-DB-URL` is the URL that points to the internal file server.

2. Update Tanzu Application Platform:

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



### Note

If you are using the Namespace Provisioner to provision a new developer namespace and want to apply a package overlay for Grype, you must complete additional configuration steps. See [Grype package overlays are not applied to scantemplates created by Namespace Provisioner](#).

## Troubleshooting

### ERROR failed to fetch latest cli version



### Note

This message is a warning and the Grype scan still runs with this message.

The Grype CLI checks for later versions of the CLI by contacting the anchore endpoint over the Internet.

```
ERROR failed to fetch latest version: Get "https://toolbox-data.anchore.io/grype/releases/latest/VERSION": dial tcp: lookup toolbox-data.anchore.io on [::1]:53: read udp [::1]:65010->[::1]:53: read: connection refused
```

## Solution

To deactivate this check, set the environment variable `GRYPE_CHECK_FOR_APP_UPDATE` to `false` by using a package overlay with the following steps:

1. Create a secret that contains the ytt overlay to add the Grype environment variable to the ScanTemplates.

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

    #@overlay/match by=overlay.subset({"kind":"ScanTemplate"}),expects="1+"
    ---
    spec:
      template:
        initContainers:
          #@overlay/match by=overlay.subset({"name": "scan-plugin"}), expects
          ="1+"
          - name: scan-plugin
            #@overlay/match missing_ok=True
            env:
              #@overlay/append
              - name: GRYPE_CHECK_FOR_APP_UPDATE
                value: "false"
```

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

```
package_overlays:
  - name: "grype"
    secrets:
      - name: "grype-airgap-deactivate-cli-check-overlay"
```

3. Update Tanzu Application Platform:

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

## Database is too old

```
1 error occurred:
 * db could not be loaded: the vulnerability database was built N days/weeks ago (max allowed age is 5 days)
```

Grype needs up-to-date vulnerability information to provide accurate matches. By default, it fails to run if the local database was not built in the last 5 days.

## Solution

Two options to resolve this:

1. Stale databases weaken your security posture. VMware recommends updating the database daily as the first recommended solution.
2. If updating the database daily is not an option, the data staleness check is configurable by using the environment variable `GRYPE_DB_MAX_ALLOWED_BUILT_AGE` and is addressed using a package overlay with the following steps:
  1. Create a secret that contains the ytt overlay to add the Grype environment variable to the ScanTemplates.

```

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

    #@overlay/match by=overlay.subset({"kind":"ScanTemplate"}),expects="1
+
---
spec:
  template:
    initContainers:
      #@overlay/match by=overlay.subset({"name": "scan-plugin"}), exp
ects="1+"
    - name: scan-plugin
      #@overlay/match missing_ok=True
      env:
        #@overlay/append
        - name: GRYPE_DB_MAX_ALLOWED_BUILT_AGE #! see note on best
practices
          value: "120h"

```



### Note

The default maximum allowed built age of Grype's vulnerability database is 5 days. This means that scanning with a 6 day old database causes the scan to fail. You can use the `GRYPE_DB_MAX_ALLOWED_BUILT_AGE` parameter to override the default in accordance with your security posture.

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

```

package_overlays:
  - name: "grype"
    secrets:
      - name: "grype-airgap-override-stale-db-overlay"

```

3. Update Tanzu Application Platform:

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

## Grype package overlays are not applied to scantemplates created by Namespace Provisioner

If you used the Namespace Provisioner to provision a new developer namespace and want to apply a package overlay for Grype, see [Import overlay secrets](#).

```
scan-pod[scan-plugin] 1 error occurred:
scan-pod[scan-plugin] * failed to load vulnerability db: vulnerability database is in
valid (run db update to correct): database metadata not found: /.cache/grype/db/5
```

### Solution

Examine the `listing.json` file you created. This matches the format of the listing file. The listing file is located at Anchore Grype's public endpoint. See the [Grype README.md](#) in GitHub.

Here is an example of a properly formatted `listing.json`:

```

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

#@ def matchGrypeScanners(index, left, right):
  #@ if left["apiVersion"] != "packaging.carvel.dev/v1alpha1" or left["kind"] != "Pack
ageInstall":
    #@ return False
  #@ end

  #@ return left["metadata"]["name"].startswith("grype-scanner")
#@ end

#@ def customize():

#@overlay/match by=matchGrypeScanners, expects="0+"
---
metadata:
  annotations:
    #@overlay/match missing_ok=True
    ext.packaging.carvel.dev/ytt-paths-from-secret-name.0: SECRET-NAME
#@ end

```

Where:

- `5` refers to the Grype's vulnerability database schema.
- `built` is the build timestamp in the format `yyyy-MM-ddTHH:mm:ssZ`.
- `url` is the download URL for the tarball containing the database. This points at your internal endpoint. The tarball contains the following files:
  - `vulnerability.db` is an SQLite file that is Grype's vulnerability database. Each time the data shape of the vulnerability database changes, a new schema is created. Different Grype versions require specific database schema versions. For example, Grype `v0.54.0` requires database schema version `v5`.
  - `metadata.json` file
- `checksum` is the SHA used to verify the database's integrity.

Verify these possible reasons why the vulnerability database is not valid:

1. The database schema is invalid. Confirm that the required database schema for the installed Grype version is used. Confirm that the top level version key matches the nested `version`.

For example, the top level version `1` in the following snippet does not match the nested `version: 5`.

```
{
  "available": {
    "1": [{
      "built": "2023-02-08T08_17_20Z",
      "version": 5,
      "url": "https://INTERNAL-ENDPOINT/PATH-TO-TARBALL/vulnerability-db_v
5_2023-02-08T08_17_20Z_6ef73016d160043c630f.tar.gz",
      "checksum": "sha256:aab8d369933c845878ef1b53bb5c26ee49b91ddc5cd87c9e
b57ffb203a88a72f"
    }]
  }
}
```

Where `PATH-TO-TARBALL` is the path to the tarball containing the vulnerability database.

As stale databases weaken your security posture, VMware recommends using the newest entry of the relevant schema version in the `listing.json` file. See Anchore's [grype-db](#) in GitHub.

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

### Debug Grype database in a cluster

- Describe the failed source scan or image scan to verify the name of the ScanTemplate being used.
  - For `sourcescan`, run:

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

- For `imagescan`, run:

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

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

- Edit the ScanTemplate's `scan-plugin` container to include a "sleep" entrypoint which allows you to troubleshoot inside the container:

```
- name: scan-plugin
  volumeMounts:
    ...
  image: #@ data.values.scanner.image
  imagePullPolicy: IfNotPresent
  env:
    ...
  command: ["/bin/bash"]
  args:
    - "sleep 1800" # insert 30 min sleep here
```

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

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

5. Get a shell to the container. See the [Kubernetes documentation](#):

```
kubectl exec --stdin --tty SCAN-PLUGIN-POD -c step-scan-plugin -- /bin/bash
```

Where `SCAN-PLUGIN-POD` is the name of the `scan-plugin` pod.

6. Inside the container, run Grype CLI commands to report database status and verify connectivity from cluster to mirror. See the [Grype documentation](#) in GitHub.
  - o Report current status of Grype's database (location, build date, and checksum):

```
grype db status
```

7. Ensure that the built parameters in the listing.json has timestamps in this proper format `yyyy-MM-ddTHH:mm:ssZ`.

## Set up developer namespaces to use your installed packages

To automatically set up your developer namespaces, see [Provision namespace resources](#).

For more information about Namespace Provisioner, see [Namespace Provisioner overview](#).

## Additional configuration for testing and scanning

If you plan to install or have already installed Out of the Box Supply Chains with Testing and Scanning, you can use Namespace Provisioner to set up the required resources. For more information, see [Add the resources required by Out of the Box Testing and Scanning Supply Chain](#).

## Legacy namespace setup

To use the legacy manual process for setting up developer namespaces, see [Legacy namespace setup](#).

## Next steps

- [Deploy an air-gapped workload](#)

## Install Tanzu Application Platform (AWS)

You can install Tanzu Application Platform (commonly known as TAP) on [Amazon Elastic Kubernetes Services \(EKS\)](#) by using [Amazon Elastic Container Registry \(ECR\)](#).

To install, take the following steps.

Step	Task	Link
1.	Review the prerequisites to ensure that you have set up everything required before beginning the installation	<a href="#">Prerequisites</a>
2.	Accept Tanzu Application Platform EULAs and install the Tanzu CLI	<a href="#">Accept Tanzu Application Platform EULAs and installing the Tanzu CLI</a>
3.	Create AWS Resources (EKS Cluster, roles, etc)	<a href="#">Create AWS Resources</a>
4.	Install Cluster Essentials for Tanzu	<a href="#">Deploy Cluster Essentials</a>

Step	Task	Link
5.	Add the Tanzu Application Platform package repository, prepare your Tanzu Application Platform profile, and install the profile to the cluster	<a href="#">Install the Tanzu Application Platform package and profiles</a>
6.	(Optional) Install any additional packages that were not in the profile	<a href="#">Install individual packages</a>
7.	Set up developer namespaces to use your installed packages	<a href="#">Set up developer namespaces to use your installed packages</a>
8.	Install developer tools into your integrated development environment (IDE)	<a href="#">Install Tanzu Developer Tools for your VS Code</a>

After installing Tanzu Application Platform on your Kubernetes clusters, [get started with Tanzu Application Platform](#) and create your ECR repositories for your workload, such as [tanzu-application-platform/tanzu-java-web-app-default](#), [tanzu-application-platform/tanzu-java-web-app-default-bundle](#), and [tanzu-application-platform/tanzu-java-web-app-default-source](#).

## Install Tanzu Application Platform (AWS)

You can install Tanzu Application Platform (commonly known as TAP) on [Amazon Elastic Kubernetes Services \(EKS\)](#) by using [Amazon Elastic Container Registry \(ECR\)](#).

To install, take the following steps.

Step	Task	Link
1.	Review the prerequisites to ensure that you have set up everything required before beginning the installation	<a href="#">Prerequisites</a>
2.	Accept Tanzu Application Platform EULAs and install the Tanzu CLI	<a href="#">Accept Tanzu Application Platform EULAs and installing the Tanzu CLI</a>
3.	Create AWS Resources (EKS Cluster, roles, etc)	<a href="#">Create AWS Resources</a>
4.	Install Cluster Essentials for Tanzu	<a href="#">Deploy Cluster Essentials</a>
5.	Add the Tanzu Application Platform package repository, prepare your Tanzu Application Platform profile, and install the profile to the cluster	<a href="#">Install the Tanzu Application Platform package and profiles</a>
6.	(Optional) Install any additional packages that were not in the profile	<a href="#">Install individual packages</a>
7.	Set up developer namespaces to use your installed packages	<a href="#">Set up developer namespaces to use your installed packages</a>
8.	Install developer tools into your integrated development environment (IDE)	<a href="#">Install Tanzu Developer Tools for your VS Code</a>

After installing Tanzu Application Platform on your Kubernetes clusters, [get started with Tanzu Application Platform](#) and create your ECR repositories for your workload, such as [tanzu-application-platform/tanzu-java-web-app-default](#), [tanzu-application-platform/tanzu-java-web-app-default-bundle](#), and [tanzu-application-platform/tanzu-java-web-app-default-source](#).

## Create AWS Resources for Tanzu Application Platform

To install Tanzu Application Platform (commonly known as TAP) within the Amazon Web Services (AWS) Ecosystem, you must create several AWS resources. Use this topic to learn how to create:

- An Amazon Elastic Kubernetes Service (EKS) cluster to install Tanzu Application Platform.
- Identity and Access Management (IAM) roles to allow authentication and authorization to read and write from Amazon Elastic Container Registry (ECR).
- ECR Repositories for the Tanzu Application Platform container images.

Creating these resources enables Tanzu Application Platform to use an IAM role bound to a Kubernetes service account for authentication, rather than the typical username and password stored in a Kubernetes secret strategy. For more information, see this [AWS documentation](#).

This is important when using ECR because authenticating to ECR is a two-step process:

1. Retrieve a token using your AWS credentials.
2. Use the token to authenticate to the registry.

To increase security, the token has a lifetime of 12 hours. This makes storing it as a secret for a service impractical because it has to be refreshed every 12 hours.

Using an IAM role on a service account mitigates the need to retrieve the token at all because it is handled by credential helpers within the services.

## Prerequisites

Before installing Tanzu Application Platform on AWS, you need:

- An AWS Account. You need to create all of your resources within Amazon Web Services, so you need an Amazon account. For more information, see [How do I create and activate a new AWS account?](#). You need your account ID for this walkthrough.
- AWS CLI. This walkthrough uses the AWS CLI to both query and configure resources in AWS, such as IAM roles. For more information, see this [AWS documentation](#).
- `eksctl` command line. The `eksctl` command line helps you manage the life cycle of EKS clusters. This guide uses it to create clusters. To install `eksctl`, see the [eksctl documentation](#).

## Export environment variables

Variables are used throughout this guide. To simplify the process and minimize the opportunity for errors, export these variables:

```
export AWS_ACCOUNT_ID=012345678901
export AWS_REGION=us-west-2
export EKS_CLUSTER_NAME=tap-on-aws
```

Where:

Variable	Description
AWS_ACCOUNT_ID	Your AWS account ID
AWS_REGION	The AWS region you are going to deploy to
EKS_CLUSTER_NAME	The name of your EKS Cluster

## Create an EKS cluster

To create an EKS cluster in the specified region, run:

```
eksctl create cluster --name $EKS_CLUSTER_NAME --managed --region $AWS_REGION --instance-types t3.xlarge --version 1.23 --with-oidc -N 5
```

Creating the control plane and node group can take anywhere from 30-60 minutes.



### Note



This step is optional if you already have an existing EKS Cluster v1.23 or later with OpenID Connect (OIDC) authentication enabled. For more information about how to enable the OIDC provider, see [AWS documentation](#).

## Install EBS CSI driver

As a requirement for Tanzu Application Platform, EBS CSI driver is no longer installed by default starting from EKS 1.23. For more information about how to install EBS CSI driver, see [AWS documentation](#).

## Create the container repositories

ECR requires that the container repositories are already created. For Tanzu Application Platform, you need to create two repositories:

- A repository to store the Tanzu Application Platform service container images.
- A repository to store Tanzu Build Service Base OS and Buildpack container images.

To create these repositories, run:

```
aws ecr create-repository --repository-name tap-images --region $AWS_REGION
aws ecr create-repository --repository-name tap-build-service --region $AWS_REGION
```

Name the repositories any name you want, but remember the names for when you later build the configuration.

## Create the workload container repositories

Similar to the two repositories created earlier for the platform, you must create repositories for each workload that Tanzu Application Platform creates before creating any workloads so that a repository is available to upload container images and workload bundles. This is because AWS ECR does not support automatically creating container repositories on initial push. For more information, see the [AWS repository](#) in GitHub.

When installing Tanzu Application Platform, you must specify a prefix for all workload registries. This topic uses `tanzu-application-platform` as the default value, but you can customize this value in the profile configuration created in [Install Tanzu Application Platform package and profiles on AWS](#).

To use the default value, create two workload repositories for each workload with the following format:

```
tanzu-application-platform/WORKLOADNAME-NAMESPACE
tanzu-application-platform/WORKLOADNAME-NAMESPACE-bundle
```

For example, to create these repositories for the sample workload `tanzu-java-web-app` in the `default` namespace, you can run the following ECR command:

```
aws ecr create-repository --repository-name tanzu-application-platform/tanzu-java-web-app-default --region $AWS_REGION
aws ecr create-repository --repository-name tanzu-application-platform/tanzu-java-web-app-default-bundle --region $AWS_REGION
```



### Note

The default Supply Chain Choreographer method of storing Kubernetes configuration is RegistryOps, which requires the `bundle` repository. If you enabled the GitOps capability, this repository is not required. For more information about the differences between RegistryOps and GitOps, see [Use GitOps or RegistryOps with Supply Chain Choreographer](#).

## Create IAM roles

By default, the EKS cluster is provisioned with an EC2 instance profile that provides read-only access for the entire EKS cluster to the ECR registry within your AWS account. For more information, see this [AWS documentation](#).

However, some of the services within Tanzu Application Platform require write access to the container repositories. To provide that access, create IAM roles and add the ARN to the Kubernetes service accounts that those services use. This ensures that only the required services have access to write container images to ECR, rather than a blanket policy that applies to the entire cluster.

You must create two IAM Roles:

- **Tanzu Build Service:** Gives write access to the repository to allow the service to automatically upload new images. This is limited in scope to the service account for `kpack` and the dependency updater.
- **Workload:** Gives write access to the entire ECR registry with a prepended path. This prevents you from having to update the policy for each new workload created.

To create the roles, you must establish two policies:

- **Trust Policy:** Limits the scope to the OIDC endpoint for the Kubernetes cluster and the Kubernetes service account you attach the role to.
- **Permission Policy:** Limits the scope of actions the role can take on resources.



### Note

These policies attempt to achieve a least privilege model. Review them to confirm they adhere to your organization's policies.

To simplify this walkthrough, use a script to create these policy documents and the roles. This script outputs the files and then creates the IAM roles by using the policy documents.

Run:

```
# Retrieve the OIDC endpoint from the Kubernetes cluster and store it for use in the policy.
export OIDCPROVIDER=$(aws eks describe-cluster --name $EKS_CLUSTER_NAME --region $AWS_REGION --output json | jq '.cluster.identity.oidc.issuer' | tr -d '"' | sed 's/https://\\\\\\\\\\\\\\\\')
cat << EOF > build-service-trust-policy.json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::${AWS_ACCOUNT_ID}:oidc-provider/${OIDCPROVIDER}"
      }
    }
  ],
}
```

```

    "Action": "sts:AssumeRoleWithWebIdentity",
    "Condition": {
      "StringEquals": {
        "${OIDCPROVIDER}:aud": "sts.amazonaws.com"
      },
      "StringLike": {
        "${OIDCPROVIDER}:sub": [
          "system:serviceaccount:kpack:controller",
          "system:serviceaccount:build-service:dependency-updater-controller-serviceaccount"
        ]
      }
    }
  ]
}
}
EOF

cat << EOF > build-service-policy.json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ecr:DescribeRegistry",
        "ecr:GetAuthorizationToken",
        "ecr:GetRegistryPolicy",
        "ecr:PutRegistryPolicy",
        "ecr:PutReplicationConfiguration",
        "ecr>DeleteRegistryPolicy"
      ],
      "Resource": "*",
      "Effect": "Allow",
      "Sid": "TAPEcrBuildServiceGlobal"
    },
    {
      "Action": [
        "ecr:DescribeImages",
        "ecr:ListImages",
        "ecr:BatchCheckLayerAvailability",
        "ecr:BatchGetImage",
        "ecr:BatchGetRepositoryScanningConfiguration",
        "ecr:DescribeImageReplicationStatus",
        "ecr:DescribeImageScanFindings",
        "ecr:DescribeRepositories",
        "ecr:GetDownloadUrlForLayer",
        "ecr:GetLifecyclePolicy",
        "ecr:GetLifecyclePolicyPreview",
        "ecr:GetRegistryScanningConfiguration",
        "ecr:GetRepositoryPolicy",
        "ecr:ListTagsForResource",
        "ecr:TagResource",
        "ecr:UntagResource",
        "ecr:BatchDeleteImage",
        "ecr:BatchImportUpstreamImage",
        "ecr:CompleteLayerUpload",
        "ecr:CreatePullThroughCacheRule",
        "ecr:CreateRepository",
        "ecr>DeleteLifecyclePolicy",
        "ecr>DeletePullThroughCacheRule",
        "ecr>DeleteRepository",
        "ecr:InitiateLayerUpload",
        "ecr:PutImage",
        "ecr:PutImageScanningConfiguration",
        "ecr:PutImageTagMutability",
        "ecr:PutLifecyclePolicy",

```

```

        "ecr:PutRegistryScanningConfiguration",
        "ecr:ReplicateImage",
        "ecr:StartImageScan",
        "ecr:StartLifecyclePolicyPreview",
        "ecr:UploadLayerPart",
        "ecr>DeleteRepositoryPolicy",
        "ecr:SetRepositoryPolicy"
    ],
    "Resource": [
        "arn:aws:ecr:${AWS_REGION}:${AWS_ACCOUNT_ID}:repository/tap-build-service",
        "arn:aws:ecr:${AWS_REGION}:${AWS_ACCOUNT_ID}:repository/tap-images"
    ],
    "Effect": "Allow",
    "Sid": "TAPEcrBuildServiceScoped"
}
]
}
EOF

cat << EOF > workload-policy.json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "ecr:DescribeRegistry",
                "ecr:GetAuthorizationToken",
                "ecr:GetRegistryPolicy",
                "ecr:PutRegistryPolicy",
                "ecr:PutReplicationConfiguration",
                "ecr>DeleteRegistryPolicy"
            ],
            "Resource": "*",
            "Effect": "Allow",
            "Sid": "TAPEcrWorkloadGlobal"
        },
        {
            "Action": [
                "ecr:DescribeImages",
                "ecr:ListImages",
                "ecr:BatchCheckLayerAvailability",
                "ecr:BatchGetImage",
                "ecr:BatchGetRepositoryScanningConfiguration",
                "ecr:DescribeImageReplicationStatus",
                "ecr:DescribeImageScanFindings",
                "ecr:DescribeRepositories",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetLifecyclePolicy",
                "ecr:GetLifecyclePolicyPreview",
                "ecr:GetRegistryScanningConfiguration",
                "ecr:GetRepositoryPolicy",
                "ecr:ListTagsForResource",
                "ecr:TagResource",
                "ecr:UntagResource",
                "ecr:BatchDeleteImage",
                "ecr:BatchImportUpstreamImage",
                "ecr:CompleteLayerUpload",
                "ecr>CreatePullThroughCacheRule",
                "ecr>CreateRepository",
                "ecr>DeleteLifecyclePolicy",
                "ecr>DeletePullThroughCacheRule",
                "ecr>DeleteRepository",
                "ecr:InitiateLayerUpload",
                "ecr:PutImage",
                "ecr:PutImageScanningConfiguration",
            ]
        }
    ]
}

```

```

        "ecr:PutImageTagMutability",
        "ecr:PutLifecyclePolicy",
        "ecr:PutRegistryScanningConfiguration",
        "ecr:ReplicateImage",
        "ecr:StartImageScan",
        "ecr:StartLifecyclePolicyPreview",
        "ecr:UploadLayerPart",
        "ecr:DeleteRepositoryPolicy",
        "ecr:SetRepositoryPolicy"
    ],
    "Resource": [
        "arn:aws:ecr:${AWS_REGION}:${AWS_ACCOUNT_ID}:repository/tap-build-service",
        "arn:aws:ecr:${AWS_REGION}:${AWS_ACCOUNT_ID}:repository/tanzu-application-platform/tanzu-java-web-app",
        "arn:aws:ecr:${AWS_REGION}:${AWS_ACCOUNT_ID}:repository/tanzu-application-platform/tanzu-java-web-app-bundle",
        "arn:aws:ecr:${AWS_REGION}:${AWS_ACCOUNT_ID}:repository/tanzu-application-platform",
        "arn:aws:ecr:${AWS_REGION}:${AWS_ACCOUNT_ID}:repository/tanzu-application-platform/*"
    ],
    "Effect": "Allow",
    "Sid": "TAPEcrWorkloadScoped"
}
]
}
EOF

cat << EOF > workload-trust-policy.json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::${AWS_ACCOUNT_ID}:oidc-provider/${OIDCPROVIDER}"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "${OIDCPROVIDER}:sub": "system:serviceaccount:default:default",
                    "${OIDCPROVIDER}:aud": "sts.amazonaws.com"
                }
            }
        }
    ]
}
EOF

# Create the Tanzu Build Service Role
aws iam create-role --role-name tap-build-service --assume-role-policy-document file://build-service-trust-policy.json
# Attach the Policy to the Build Role
aws iam put-role-policy --role-name tap-build-service --policy-name tapBuildServicePolicy --policy-document file://build-service-policy.json

# Create the Workload Role
aws iam create-role --role-name tap-workload --assume-role-policy-document file://workload-trust-policy.json
# Attach the Policy to the Workload Role
aws iam put-role-policy --role-name tap-workload --policy-name tapWorkload --policy-document file://workload-policy.json

```

## Install Tanzu Application Platform package and profiles on AWS

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

Before installing the packages, ensure you have:

- Completed the [Prerequisites](#).
- Created [AWS Resources](#).
- [Accepted Tanzu Application Platform EULA](#) and installed [Tanzu CLI](#) with any required plugins.
- Installed [Cluster Essentials for Tanzu](#).

### Relocate images to a registry

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

This section describes how to relocate images to the `tap-images` repository created in [Amazon ECR](#). See [Creating AWS Resources](#) for more information.

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

1. Install Docker if it is not already installed.
2. Log in to your ECR image registry by following the [AWS documentation](#).



#### Note

This is a one time copy of images from the VMware Tanzu Network to ECR, so the ECR token expiring in 12 hours is not a concern.

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

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

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

```
export AWS_ACCOUNT_ID=MY-AWS-ACCOUNT-ID
export AWS_REGION=TARGET-AWS-REGION
export TAP_VERSION=VERSION-NUMBER
export INSTALL_REGISTRY_HOSTNAME=$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com
export INSTALL_REPO=tap-images
```

Where:

- `MY-AWS-ACCOUNT-ID` is the account ID you depoloy Tanzu Application Platform in. No dashes and must be in the format `012345678901`.
- `TARGET-AWS-REGION` is the region you deploy the Tanzu Application Platform to.
- `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.4.13`.

5. Install the Carvel tool `imgpkg` CLI.
6. Relocate the images with the `imgpkg` CLI by running:

```
imgpkg copy --concurrency 1 -b registry.tanzu.vmware.com/tanzu-application-platform/tap-packages:${TAP_VERSION} --to-repo ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}
```

7. 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.

8. (Optional) If you haven't relocated the images to ECR, create a secret to your registry by running:

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

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

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

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

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

For example:

```
$ tanzu package repository get tanzu-tap-repository --namespace tap-install
- Retrieving repository tap...
NAME:          tanzu-tap-repository
VERSION:       16253001
REPOSITORY:    123456789012.dkr.ecr.us-west-2.amazonaws.com/tap-images
TAG:           1.4.13
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.

11. 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 for VMware Tanzu and configurations.	Application Accelerator Used to create new projects a
api-portal.tanzu.vmware.com A unified user interface for API discovery and exploration at scale.	API portal
apis.apps.tanzu.vmware.com for VMware Tanzu	API Auto Registration fo A TAP component to automatica
lly register API exposing workloads as API entities in TAP GUI.	
backend.appliveview.tanzu.vmware.com for VMware Tanzu eshooting running apps	Application Live View fo App for monitoring and troubl
buildservice.tanzu.vmware.com Tanzu Build Service enables the building and automation of containerized	Tanzu Build Service
software workflows securely and at scale.	
carbonblack.scanning.apps.tanzu.vmware.com Supply Chain Security Tools - Scan VMware Carbon Black	VMware Carbon Black for Default scan templates using
cartographer.tanzu.vmware.com Kubernetes native Supply Chain Choreographer.	Cartographer
cnrs.tanzu.vmware.com Cloud Native Runtimes is a serverless runtime based on Knative	Cloud Native Runtimes
connector.appliveview.tanzu.vmware.com connector for VMware Tanzu tering running apps	Application Live View Co App for discovering and regis
controller.conventions.apps.tanzu.vmware.com Mware Tanzu	Convention Service for V Convention Service enables ap
p operators to consistently apply desired runtime configurations to fleets of workloads.	
controller.source.apps.tanzu.vmware.com Tanzu Source Controller enables workload create/update from source code.	Tanzu Source Controller
conventions.appliveview.tanzu.vmware.com nventions for VMware Tanzu ion server	Application Live View Co Application Live View convent
developer-conventions.tanzu.vmware.com oper Conventions	Tanzu App Platform Devel Developer Conventions
eventing.tanzu.vmware.com Eventing is an event-driven architecture platform based on Knative Eventing	Eventing
external-secrets.apps.tanzu.vmware.com r a Kubernetes operator that integrates external	External Secrets Operato External Secrets Operator is
secret management systems.	
fluxcd.source.controller.tanzu.vmware.com The source-controller is a Kubernetes operator, specialised in artifacts	Flux Source Controller
acquisition from external sources such as Git, Helm repositories and S3 bucket s.	
grype.scanning.apps.tanzu.vmware.com ecurity Tools - Scan Anchore Grype	Grype for Supply Chain S Default scan templates using
learningcenter.tanzu.vmware.com u Application Platform	Learning Center for Tanz Guided technical workshops
metadata-store.apps.tanzu.vmware.com ols - Store e, package, and vulnerability metadata.	Supply Chain Security To Post SBOMs and query for imag
namespace-provisioner.apps.tanzu.vmware.com Automatic Provisioning of Developer Namespaces.	Namespace Provisioner
ootb-delivery-basic.tanzu.vmware.com f The Box Delivery Basic c.	Tanzu App Platform Out o Out of The Box Delivery Basi
ootb-supply-chain-basic.tanzu.vmware.com	Tanzu App Platform Out o



ootb-supply-chain-testing-scanning.tanzu.vmware.com	Out of The Box Supply Chain Basic.
ootb-supply-chain-testing-scanning.tanzu.vmware.com	Tanzu App Platform Out of The Box Supply Chain with Testing and Scanning.
ootb-supply-chain-testing.tanzu.vmware.com	Out of The Box Supply Chain with Testing and Scanning.
ootb-templates.tanzu.vmware.com	Tanzu App Platform Out of The Box Templates.
policy.apps.tanzu.vmware.com	Supply Chain Security Tools - Policy Controller
scanning.apps.tanzu.vmware.com	Policy Controller enables defining of a policy to restrict unsigned container images.
scanning.apps.tanzu.vmware.com	Supply Chain Security Tools - Scan
service-bindings.labs.vmware.com	Scan for vulnerabilities and enforce policies directly within Kubernetes native Supply Chains.
services-toolkit.tanzu.vmware.com	Service Bindings for Kubernetes implements the Service Binding Specification.
snyk.scanning.apps.tanzu.vmware.com	Services Toolkit
spring-boot-conventions.tanzu.vmware.com	The Services Toolkit enables the management, lifecycle, discoverability and connectivity of Service Resources (databases, message queues, DNS records, etc.).
snyk.scanning.apps.tanzu.vmware.com	Snyk for Supply Chain Security Tools - Scan
spring-boot-conventions.tanzu.vmware.com	Default scan templates using Snyk
sso.apps.tanzu.vmware.com	Tanzu Spring Boot Conventions Server
tap-auth.tanzu.vmware.com	Default Spring Boot convention server.
tap-gui.tanzu.vmware.com	AppSSO
tap-telemetry.tanzu.vmware.com	Application Single Sign-On for Tanzu Application Platform
tap.tanzu.vmware.com	Default roles for Tanzu Application Platform
tekton.tanzu.vmware.com	Tanzu Application Platform GUI
workshops.learningcenter.tanzu.vmware.com	web app graphical user interface for Tanzu Application Platform
workshops.learningcenter.tanzu.vmware.com	Telemetry Collector for Tanzu Application Platform
workshops.learningcenter.tanzu.vmware.com	Tanzu Application Platform Telemetry
workshops.learningcenter.tanzu.vmware.com	Tanzu Application Platform
workshops.learningcenter.tanzu.vmware.com	Package to install a set of TAP components to get you started based on your use case.
workshops.learningcenter.tanzu.vmware.com	Tekton Pipelines
workshops.learningcenter.tanzu.vmware.com	Tekton Pipelines is a framework for creating CI/CD systems.
workshops.learningcenter.tanzu.vmware.com	Workshop Building Tutorial
workshops.learningcenter.tanzu.vmware.com	Workshop Building Tutorial

## Install your Tanzu Application Platform profile

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

For more information about profiles, see [Components and installation profiles](#).

To prepare to install a profile:

1. List version information for the package by running:

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

2. Create a `tap-values.yaml` file by using the [Full Profile \(AWS\)](#), which contains the minimum configurations required to deploy Tanzu Application Platform on AWS. The sample values file contains the necessary defaults for:
  - The meta-package, or parent Tanzu Application Platform package.
  - Subordinate packages, or individual child packages.

Keep the values file for future configuration use.



#### Note

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

3. [View possible configuration settings for your package](#)

## Full profile (AWS)

The following command generates the YAML file sample for the full-profile on AWS by using the ECR repositories you created earlier. The `profile:` field takes `full` as the default value, but you can also set it to `iterate`, `build`, `run`, or `view`. Refer to [Install multicluster Tanzu Application Platform profiles](#) for more information.

```
cat << EOF > tap-values.yaml
shared:
  ingress_domain: "INGRESS-DOMAIN"

ceip_policy_disclosed: true

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

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

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

supply_chain: basic # Can take testing, testing_scanning.

ootb_supply_chain_basic: # Based on supply_chain set above, can be changed to ootb_sup
ply_chain_testing, ootb_supply_chain_testing_scanning.
registry:
  server: ${AWS_ACCOUNT_ID}.dkr.ecr.${AWS_REGION}.amazonaws.com
  # The prefix of the ECR repository. Workloads will need
  # two repositories created:
  #
  # tanzu-application-platform/<workloadname>-<namespace>
  # tanzu-application-platform/<workloadname>-<namespace>-bundle
  repository: tanzu-application-platform

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

buildservice:
  kp_default_repository: ${AWS_ACCOUNT_ID}.dkr.ecr.${AWS_REGION}.amazonaws.com/tap-bui
```

```

ld-service
  # Enable the build service k8s service account to bind to the AWS IAM Role
  kp_default_repository_aws_iam_role_arn: "arn:aws:iam::${AWS_ACCOUNT_ID}:role/tap-bui
ld-service"

ootb_templates:
  # Enable the config writer service to use cloud based iaas authentication
  # which are retrieved from the developer namespace service account by
  # default
  iaas_auth: true

tap_gui:
  app_config:
    catalog:
      locations:
        - type: url
          target: https://GIT-CATALOG-URL/catalog-info.yaml

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

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

tap_telemetry:
  customer_entitlement_account_number: "CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER" # (Option
al) Identify data for creating Tanzu Application Platform usage reports.
EOF

```

#### Where:

- `INGRESS-DOMAIN` is the subdomain for the host name that you point at the `tanzu-shared-ingress` service's External IP address.
- `kp_default_repository_aws_iam_role_arn` is the ARN that was created to write to the ECR repository for the build service. This value is generated by the script, but you can modify it manually.
- `GIT-CATALOG-URL` is the path to the `catalog-info.yaml` catalog definition file. You can download either a blank or populated catalog file from the [Tanzu Application Platform product page](#). Otherwise, you can use a Backstage-compliant catalog you've already built and posted on the Git infrastructure.
- `MY-DEV-NAMESPACE` is the name of the developer namespace. SCST - Store exports secrets to the namespace, and SCST - Scan deploys the `ScanTemplates` there. This allows the scanning feature to run in this namespace. If there are multiple developer namespaces, use `ns_for_export_app_cert: "*"`  to export the SCST - Store CA certificate to all namespaces.
- `CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER` (optional) refers to the Entitlement Account Number (EAN), which is a unique identifier VMware assigns to its customers. Tanzu Application Platform telemetry uses this number to identify data that belongs to a particular customers and prepare usage reports. See [Kubernetes Grid documentation](#) for more information about identifying the Entitlement Account Number.

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

```

contour:
  infrastructure_provider: aws
envoy:
  service:

```

```
aws:
  LBType: nlb
```

## (Optional) Configure your profile with full dependencies

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

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

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

```
buildservice:
  kp_default_repository: ${AWS_ACCOUNT_ID}.dkr.ecr.${AWS_REGION}.amazonaws.com/tap-build-service
  exclude_dependencies: true
```

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

## Install your Tanzu Application Platform package

Follow these steps to install the Tanzu Application Platform package:

1. Install the package by running:

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

2. Verify the package install by running:

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

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

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

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

4. If you configured `full` dependencies in your `tbs-values.yaml` file, install the `full` dependencies by following the procedure in [Install full dependencies](#).

After installing the Full profile on your cluster, you can install the Tanzu Developer Tools for VS Code Extension to help you develop against it. For instructions, see [Install Tanzu Developer Tools for your VS Code](#).



### Note

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

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

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

## Install the full dependencies package

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

For more information about the differences between `lite` and `full` dependencies, see [About lite and full dependencies](#).

To install the `full` dependencies package:

1. If you have not done so already, add the key-value pair `exclude_dependencies: true` to your `tap-values.yaml` file under the `buildservice` section. For example:

```
buildservice:
  kp_default_repository: ${AWS_ACCOUNT_ID}.dkr.ecr.${AWS_REGION}.amazonaws.com/
  tap-build-service
  exclude_dependencies: true
  ...
```

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

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

3. Create an ECR repository for Tanzu Build Service full dependencies by running:

```
aws ecr create-repository --repository-name tbs-full-deps --region ${AWS_REGIO
N}
```

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

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

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

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

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

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

6. Install the full dependencies package by running:

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

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

## Access Tanzu Application Platform GUI

To access Tanzu Application Platform GUI, you can use the host name that you configured earlier. This host name is pointed at the shared ingress. To configure LoadBalancer for Tanzu Application Platform GUI, see [Access Tanzu Application Platform GUI](#).

You're now ready to start using Tanzu Application Platform GUI. Proceed to the [Getting Started](#) topic or the [Tanzu Application Platform GUI - Catalog Operations](#) topic.

## Exclude packages from a Tanzu Application Platform profile

To exclude packages from a Tanzu Application Platform profile:

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

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

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

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



### Important

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

## Next steps

- (Optional) [Install Individual Packages](#)
- [Set up developer namespaces to use your installed packages](#)

## 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
```



### Note

The `tap.tanzu.vmware.com` package does not show all configuration settings for packages it plans to install. The package only shows top-level keys. You can view individual package configuration settings with the same `tanzu package available get` command. For example, to find the keys for Cloud Native Runtimes, you must first identify the version of the package with `tanzu package installed list -n tap-install`, which lists all the installed packages versions. Then run the command `tanzu package available get -n tap-install cnrs.tanzu.vmware.com/CNRS-VERSION --values-schema` by using the package version listed for Cloud Native Runtimes.

```

profile: full

# ...

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

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

```

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

Package	Top-level Key
<i>see table below</i>	<code>shared</code>
API Auto Registration	<code>api_auto_registration</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>
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>
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>scanning</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 [Install individual packages](#).

## Install individual packages

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

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

## Install pages for individual Tanzu Application Platform packages

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



- [Install Telemetry](#)

## Verify the installed packages

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

1. List the installed packages by running:

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

For example:

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

## Next steps

- [Set up developer namespaces to use your installed packages](#)

## Set up developer namespaces to use your installed packages

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. (Optional) If the variable `AWS_ACCOUNT_ID` environment is not set during the installation process, export the AWS Account ID.

```
export AWS_ACCOUNT_ID=MY-AWS-ACCOUNT-ID
```

2. Add a service account to execute the supply chain and RBAC rules to authorize the service account to the developer namespace.

```
cat <<EOF | kubectl -n YOUR-NAMESPACE apply -f -
apiVersion: v1
kind: ServiceAccount
metadata:
  name: default
  annotations:
    eks.amazonaws.com/role-arn: "arn:aws:iam::${AWS_ACCOUNT_ID}:role/tap-workload"
---
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
```

Where `YOUR-NAMESPACE` is your developer namespace.

3. (Optional) If you haven't relocated the images to ECR, add a placeholder secret for gathering the credentials used for pulling container images.

```
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
  annotations:
    eks.amazonaws.com/role-arn: "arn:aws:iam::${AWS_ACCOUNT_ID}:role/tap-workload"
imagePullSecrets:
  - name: tap-registry
EOF
```

Where `YOUR-NAMESPACE` is your developer namespace.

## Enable additional users access with Kubernetes RBAC

Follow these steps to enable additional users by using Kubernetes RBAC to submit jobs to the Supply Chain:

1. [Enable single user access.](#)
2. Choose either of the following options to give developers namespace-level access and view access to appropriate cluster-level resources:

- **Option 1:** Use the [Tanzu Application Platform RBAC CLI plug-in \(beta\)](#).

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

```
tanzu rbac binding add -g GROUP-FOR-APP-VIEWER -n YOUR-NAMESPACE -r app-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.

## Next steps

- [Install Tanzu Developer Tools for your VS Code](#)

## Install Tanzu Developer Tools for your VS Code

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

## Prerequisites

Before installing the extension, you must have:

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

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

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

## Install

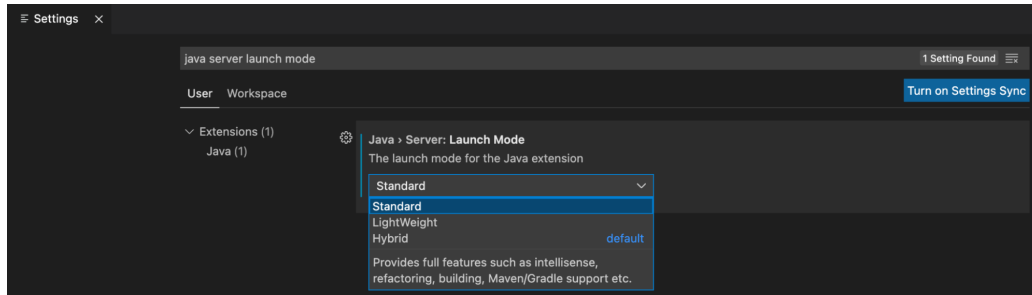
To install the extension:

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](#)

- o [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:
  - o **Confirm Delete:** This controls whether the extension asks for confirmation when deleting a workload.
  - o **Enable Live Hover:** For more information, see [Integrating Live Hover by using Spring Boot Tools](#). Reload VS Code for this change to take effect.
  - o **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.
  - o **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.
  - o **Namespace:** (Optional) This is the namespace that workloads are deployed into. The namespace set in `kubeconfig` is the default.

## Uninstall

To uninstall VMware Tanzu Developer Tools for VS Code:

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

## Next steps

Proceed to [Getting started with Tanzu Developer Tools for Visual Studio Code](#).

## Install Tanzu Application Platform (OpenShift)

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

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

After installing Tanzu Application Platform on to your OpenShift clusters, proceed with [Get started with Tanzu Application Platform](#).

## Install Tanzu Application Platform (OpenShift)

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

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

After installing Tanzu Application Platform on to your OpenShift clusters, proceed with [Get started with Tanzu Application Platform](#).

## Install Tanzu Application Platform on your OpenShift clusters

This topic tells you how to install Tanzu Application Platform (commonly known as TAP) packages on your OpenShift clusters.

Before installing the packages, ensure you have:

- Completed the [Prerequisites](#).

- Configured and verified the cluster.
- [Accepted Tanzu Application Platform EULA and installed Tanzu CLI](#) with any required plug-ins.

## Relocate images to a registry

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

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

- [Harbor documentation](#)
- [Google Container Registry documentation](#)
- [Quay.io documentation](#)

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

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

```
docker login MY-REGISTRY
```

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

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

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

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

```
export INSTALL_REGISTRY_USERNAME=MY-REGISTRY-USER
export INSTALL_REGISTRY_PASSWORD=MY-REGISTRY-PASSWORD
export INSTALL_REGISTRY_HOSTNAME=MY-REGISTRY
export TAP_VERSION=VERSION-NUMBER
export INSTALL_REPO=TARGET-REPOSITORY
```

Where:

- `MY-REGISTRY-USER` is the user with write access to `MY-REGISTRY`.
- `MY-REGISTRY-PASSWORD` is the password for `MY-REGISTRY-USER`.
- `MY-REGISTRY` is your own container registry.
- `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.4.13`.
- `TARGET-REPOSITORY` is your target repository, a folder/repository on `MY-REGISTRY` that serves as the location for the installation files for Tanzu Application Platform.

5. [Install the Carvel tool `imgpkg` CLI](#).
6. Relocate the images with the `imgpkg` CLI by running:

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



```
kages
```

7. 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.

8. Create a registry secret by running:

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

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

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

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

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

For example:

```
$ tanzu package repository get tanzu-tap-repository --namespace tap-install
- Retrieving repository tap...
NAME:          tanzu-tap-repository
VERSION:       16253001
REPOSITORY:    tapmdc.azurecr.io/mdc/1.4.0/tap-packages
TAG:           1.4.13
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.

11. List the available packages by running:

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

For example:

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

r VMware Tanzu A TAP component to automatically register API exposing workloads as API entities

in TAP GUI.

backend.appliveview.tanzu.vmware.com Application Live View for VMware Tanzu

App for monitoring and troubleshooting running apps

buildservice.tanzu.vmware.com Tanzu Build Service

Tanzu Build Service enables the building and automation of containerized software workflows securely and at scale.

carbonblack.scanning.apps.tanzu.vmware.com VMware Carbon Black for Supply Chain Security Tools - Scan

Default scan templates using VMware Carbon Black

cartographer.tanzu.vmware.com Cartographer

Kubernetes native Supply Chain Choreographer.

cnrs.tanzu.vmware.com Cloud Native Runtimes

Cloud Native Runtimes is a serverless runtime based on Knative

connector.appliveview.tanzu.vmware.com Application Live View Connector for VMware Tanzu

App for discovering and registering running apps

controller.conventions.apps.tanzu.vmware.com Convention Service for VMware Tanzu

Convention Service enables app operators to consistently apply desired runtime configurations to fleets of workloads.

controller.source.apps.tanzu.vmware.com Tanzu Source Controller

Tanzu Source Controller enables workload create/update from source code.

conventions.appliveview.tanzu.vmware.com Application Live View Conventions for VMware Tanzu

Application Live View convention server

developer-conventions.tanzu.vmware.com Tanzu App Platform Developer Conventions

Developer Conventions

eventing.tanzu.vmware.com Eventing

Eventing is an event-driven architecture platform based on Knative Eventing

external-secrets.apps.tanzu.vmware.com External Secrets Operator

External Secrets Operator is a Kubernetes operator that integrates external secret management systems.

fluxcd.source.controller.tanzu.vmware.com Flux Source Controller

The source-controller is a Kubernetes operator, specialised in artifacts acquisition from external sources such as Git, Helm repositories and S3 buckets.

grype.scanning.apps.tanzu.vmware.com Grype for Supply Chain Security Tools - Scan

Default scan templates using Anchore Grype

learningcenter.tanzu.vmware.com Learning Center for Tanzu Application Platform

Guided technical workshops

metadata-store.apps.tanzu.vmware.com Supply Chain Security Tools - Store

Post SBOMs and query for image, package, and vulnerability metadata.

namespace-provisioner.apps.tanzu.vmware.com Namespace Provisioner

Automatic Provisioning of Developer Namespaces.

ootb-delivery-basic.tanzu.vmware.com Tanzu App Platform Out of The Box Delivery Basic

Out of The Box Delivery Basic.

ootb-supply-chain-basic.tanzu.vmware.com Tanzu App Platform Out of The Box Supply Chain Basic

Out of The Box Supply Chain Basic.

ootb-supply-chain-testing-scanning.tanzu.vmware.com Tanzu App Platform Out of The Box Supply Chain with Testing and Scanning

Out of The Box Supply Chain with Testing and Scanning.

ootb-supply-chain-testing.tanzu.vmware.com Tanzu App Platform Out of The Box Supply Chain with Testing

Out of The Box Supply Chain with Testing

ith Testing.	
ootb-templates.tanzu.vmware.com	Tanzu App Platform Out o
f The Box Templates	Out of The Box Templates.
policy.apps.tanzu.vmware.com	Supply Chain Security To
ols - Policy Controller	Policy Controller enables def
ining of a policy to restrict unsigned container	
images.	
scanning.apps.tanzu.vmware.com	Supply Chain Security To
ols - Scan	Scan for vulnerabilities and
enforce policies directly within Kubernetes native	
Supply Chains.	
service-bindings.labs.vmware.com	Service Bindings for Kub
ernetes	Service Bindings for Kubernet
es implements the Service Binding Specification.	
services-toolkit.tanzu.vmware.com	Services Toolkit
The Services Toolkit enables the management, lifecycle, discoverability and	
connectivity of Service Resources (databases, message queues, DNS records,	
etc.).	
snyk.scanning.apps.tanzu.vmware.com	Snyk for Supply Chain Se
curity Tools - Scan	Default scan templates using
Snyk	
spring-boot-conventions.tanzu.vmware.com	Tanzu Spring Boot Conven
tions Server	Default Spring Boot conventio
n server.	
sso.apps.tanzu.vmware.com	AppSSO
Application Single Sign-On for Tanzu	
tap-auth.tanzu.vmware.com	Default roles for Tanzu
Application Platform	Default roles for Tanzu Appli
cation Platform	
tap-gui.tanzu.vmware.com	Tanzu Application Platfo
rm GUI	web app graphical user interf
ace for Tanzu Application Platform	
tap-telemetry.tanzu.vmware.com	Telemetry Collector for
Tanzu Application Platform	Tanzu Application Platform Te
lemetry	
tap.tanzu.vmware.com	Tanzu Application Platfo
rm	Package to install a set of T
AP components to get you started based on your use	
case.	
tekton.tanzu.vmware.com	Tekton Pipelines
Tekton Pipelines is a framework for creating CI/CD systems.	
workshops.learningcenter.tanzu.vmware.com	Workshop Building Tutori
al	Workshop Building Tutorial

## Install your Tanzu Application Platform profile

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

For more information about profiles, see [Components and installation profiles](#).

To prepare to install a profile:

1. List version information for the package by running:

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

2. Create a `tap-values.yaml` file by using the [Full Profile sample](#) in the following section as a guide. These samples have the minimum configuration required to deploy Tanzu Application

Platform. The sample values file contains the necessary defaults for:

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

Keep the values file for future configuration use.



#### Note

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

### 3. View possible configuration settings for your package

## Full profile

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

```
shared:
  ingress_domain: "INGRESS-DOMAIN"
  image_registry:
    project_path: "SERVER-NAME/REPO-NAME"
    username: "KP-DEFAULT-REPO-USERNAME"
    password: "KP-DEFAULT-REPO-PASSWORD"
  kubernetes_distribution: "openshift" # To be passed only for OpenShift. Defaults to
  ".
  kubernetes_version: "K8S-VERSION"
  ca_cert_data: | # To be passed if using custom certificates.
    -----BEGIN CERTIFICATE-----
    MIIIFXzCCA0egAwIBAgIJAjYm37SFocjlMA0GCSqGSIb3DQEEDQUAMEY...
    -----END CERTIFICATE-----

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

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

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

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

supply_chain: basic # Can take testing, testing_scanning.

ootb_supply_chain_basic: # Based on supply_chain set above, can be changed to ootb_sup
ply_chain_testing, ootb_supply_chain_testing_scanning.
  registry:
    server: "SERVER-NAME" # Takes the value from shared section above by default, but
can be overridden by setting a different value.
    repository: "REPO-NAME" # Takes the value from shared section above by default, bu
t can be overridden by setting a different value.
  gitops:
    ssh_secret: "SSH-SECRET-KEY" # Takes "" as value by default; but can be overridden
by setting a different value.

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

```

buildservice:
  kp_default_repository: "KP-DEFAULT-REPO"
  kp_default_repository_username: "KP-DEFAULT-REPO-USERNAME"
  kp_default_repository_password: "KP-DEFAULT-REPO-PASSWORD"

tap_gui:
  app_config:
    catalog:
      locations:
        - type: url
          target: https://GIT-CATALOG-URL/catalog-info.yaml

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

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

grype:
  namespace: "MY-DEV-NAMESPACE"
  targetImagePullSecret: "TARGET-REGISTRY-CREDENTIALS-SECRET"

```

#### Where:

- `INGRESS-DOMAIN` is the subdomain for the host name that you point at the `tanzu-shared-ingress` service's External IP address.
- `KP-DEFAULT-REPO` is a writable repository in your registry. Tanzu Build Service dependencies are written to this location. Examples:
  - Harbor has the form `kp_default_repository: "my-harbor.io/my-project/build-service"`.
  - Docker Hub has the form `kp_default_repository: "my-dockerhub-user/build-service"` Or `kp_default_repository: "index.docker.io/my-user/build-service"`.
  - Google Cloud Registry has the form `kp_default_repository: "gcr.io/my-project/build-service"`.
- `KP-DEFAULT-REPO-USERNAME` is the user name that can write to `KP-DEFAULT-REPO`. You can `docker push` to this location with this credential.
  - For Google Cloud Registry, use `kp_default_repository_username: _json_key`.
  - Alternatively, you can configure this credential as a [secret reference](#).
- `KP-DEFAULT-REPO-PASSWORD` is the password for the user that can write to `KP-DEFAULT-REPO`. You can `docker push` to this location with this credential.
  - For Google Cloud Registry, use the contents of the service account JSON file.
  - Alternatively, you can configure this credential as a [secret reference](#).
- `K8S-VERSION` is the Kubernetes version used by your OpenShift cluster. It must be in the form of `1.23.x` or `1.24.x`, where `x` stands for the patch version. Examples:
  - Red Hat OpenShift Container Platform v4.10 uses the Kubernetes version `1.23.3`.
  - Red Hat OpenShift Container Platform v4.11 uses the Kubernetes version `1.24.1`.
- `SERVER-NAME` is the host name of the registry server. Examples:
  - Harbor has the form `server: "my-harbor.io"`.
  - Docker Hub has the form `server: "index.docker.io"`.

- Google Cloud Registry has the form `server: "gcr.io"`.
- `REPO-NAME` is where workload images are stored in the registry. If this key is passed through the shared section earlier and AWS ECR registry is used, you must ensure that the `SERVER-NAME/REPO-NAME/buildservice` and `SERVER-NAME/REPO-NAME/workloads` exist. AWS ECR expects the paths to be pre-created. Images are written to `SERVER-NAME/REPO-NAME/workload-name`. Examples:
  - Harbor has the form `repository: "my-project/supply-chain"`.
  - Docker Hub has the form `repository: "my-dockerhub-user"`.
  - Google Cloud Registry has the form `repository: "my-project/supply-chain"`.
- `SSH-SECRET-KEY` is the SSH secret key in the developer namespace for the supply chain to fetch source code from and push configuration to. This field is only required if you use a private repository, otherwise, leave it empty. See [Git authentication](#) for more information.
- `GIT-CATALOG-URL` is the path to the `catalog-info.yaml` catalog definition file. You can download either a blank or populated catalog file from the [Tanzu Application Platform product page](#). Otherwise, you can use a Backstage-compliant catalog you've already built and posted on the Git infrastructure.
- `MY-DEV-NAMESPACE` is the name of the developer namespace. SCST - Store exports secrets to the namespace, and SCST - Scan deploys the `ScanTemplates` there. This allows the scanning feature to run in this namespace. If there are multiple developer namespaces, use `ns_for_export_app_cert: "*"`  to export the SCST - Store CA certificate to all namespaces.
- `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the secret that contains the credentials to pull an image from the registry for scanning.

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

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

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

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

## (Optional) Additional Build Service configurations

The following tasks are optional during the Tanzu Application Platform installation process:

- [\(Optional\) Configure your profile with full dependencies](#)
- [\(Optional\) Configure your profile with the Jammy stack only](#)

**(Optional) Configure your profile with full dependencies**

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

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

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

```
buildservice:
  kp_default_repository: "KP-DEFAULT-REPO"
  kp_default_repository_username: "KP-DEFAULT-REPO-USERNAME"
  kp_default_repository_password: "KP-DEFAULT-REPO-PASSWORD"
  exclude_dependencies: true
```

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

#### (Optional) Configure your profile with the Jammy stack only

Tanzu Application Platform v1.3.0 supports building applications with the [Ubuntu 22.04 \(Jammy\) stack](#). By default, workloads are built with Ubuntu 18.04 (Bionic) stack. However, if you do not need access to the Bionic stack, you can install Tanzu Application Platform without the Bionic stack and all workloads are built with the Jammy stack by default.

To install Tanzu Application Platform with Jammy as the only available stack, include the `stack_configuration: jammy-only` field under the `buildservice:` section in `tap-values.yaml`.

## Security Context Constraints

Security Context Constraints (SCC) define a set of rules that a pod must satisfy to be created. Tanzu Application Platform components use the built-in `nonroot-v2` or `restricted-v2` SCC.

In Red Hat OpenShift, SCC are used to restrict privileges for pods. In Tanzu Application Platform v1.4 there is no custom SCC.

Tanzu Application Platform packages reconcile without any issues when using OpenShift v4.11 with `restricted-v2` or `nonroot-v2`.

#### (Optional) Exclude components that require RedHat OpenShift privileged SCC

Learning Center package uses privileged SCC. To exclude this package, update your `tap-values` file with a section listing the exclusions:

```
...
excluded_packages:
  - learningcenter.tanzu.vmware.com
  - workshops.learningcenter.tanzu.vmware.com
...
```

See [Exclude packages from a Tanzu Application Platform profile](#) for more information.

## Install your Tanzu Application Platform package

Follow these steps to install the Tanzu Application Platform package:

1. Install the package by running:

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

2. Verify the package install by running:

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

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

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

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

4. If you configured `full` dependencies in your `tbs-values.yaml` file, install the `full` dependencies by following the procedure in [Install full dependencies](#).

After installing the Full profile on your cluster, you can install the Tanzu Developer Tools for VS Code Extension to help you develop against it. For instructions, see [Install Tanzu Developer Tools for your VS Code](#).



#### Note

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

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

## Install the full dependencies package

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

For more information about the differences between `lite` and `full` dependencies, see [About lite and full dependencies](#).

To install the `full` dependencies package:

1. If you have not done so already, add the key-value pair `exclude_dependencies: true` to your `tap-values.yaml` file under the `buildservice` section. For example:

```
buildservice:
  kp_default_repository: "KP-DEFAULT-REPO"
  kp_default_repository_username: "KP-DEFAULT-REPO-USERNAME"
  kp_default_repository_password: "KP-DEFAULT-REPO-PASSWORD"
  exclude_dependencies: true
  ...
```

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

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

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

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



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

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

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

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

5. Install the full dependencies package by running:

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

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

## Access Tanzu Application Platform GUI

To access Tanzu Application Platform GUI, you can use the host name that you configured earlier. This host name is pointed at the shared ingress. To configure LoadBalancer for Tanzu Application Platform GUI, see [Access Tanzu Application Platform GUI](#).

You're now ready to start using Tanzu Application Platform GUI. Proceed to the [Getting Started](#) topic or the [Tanzu Application Platform GUI - Catalog Operations](#) topic.

## Exclude packages from a Tanzu Application Platform profile

To exclude packages from a Tanzu Application Platform profile:

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

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

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

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



### Important

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

## View possible configuration settings for your package

To view possible configuration settings for a package, run:

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

```
ace tap-install
```



**Note**

The `tap.tanzu.vmware.com` package does not show all configuration settings for packages it plans to install. The package only shows top-level keys. You can view individual package configuration settings with the same `tanzu package available get` command. For example, to find the keys for Cloud Native Runtimes, you must first identify the version of the package with `tanzu package installed list -n tap-install`, which lists all the installed packages versions. Then run the command `tanzu package available get -n tap-install cnrs.tanzu.vmware.com/CNRS-VERSION --values-schema` by using the package version listed for Cloud Native Runtimes.

```
profile: full

# ...

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

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

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

Package	Top-level Key
<i>see table below</i>	<code>shared</code>
API Auto Registration	<code>api_auto_registration</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>
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>

Package	Top-level Key
Supply Chain Security Tools - Scan (Grype Scanner)	<code>grype</code>
Supply Chain Security Tools - Store	<code>metadata_store</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>scanning</code> , <code>ta</code>	Optional: PEM-encoded certificate data to trust TLS connections with a private CA

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

## Install individual packages

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

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

## Install pages for individual Tanzu Application Platform packages

- [Install API Auto Registration](#)
- [Install API portal](#)
- [Install Application Accelerator](#)
- [Install Application Live View](#)
- [Install Application Single Sign-On](#)
- [Install cert-manager](#)
- [Install Cloud Native Runtimes](#)
- [Install Contour](#)
- [Install default roles for Tanzu Application Platform](#)
- [Install Developer Conventions](#)
- [Install Eventing](#)
- [Install Flux CD Source Controller](#)
- [Install Learning Center for Tanzu Application Platform](#)
- [Install Out of the Box Templates](#)
- [Install Out of the Box Supply Chain with Testing](#)

- [Install Out of the Box Supply Chain with Testing and Scanning](#)
- [Install Service Bindings](#)
- [Install Services Toolkit](#)
- [Install Source Controller](#)
- [Install Spring Boot conventions](#)
- [Install Supply Chain Choreographer](#)
- [Install Supply Chain Security Tools - Store](#)
- [Install Supply Chain Security Tools - Policy Controller](#)
- [Install Supply Chain Security Tools - Scan](#)
- [Install Tanzu Application Platform GUI](#)
- [Install Tanzu Build Service](#)
- [Install Tekton](#)
- [Install Telemetry](#)

## Verify the installed packages

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

1. List the installed packages by running:

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

For example:

```
$ tanzu package installed list --namespace tap-install
\ Retrieving installed packages...
NAME                PACKAGE-NAME                PAC
KAGE-VERSION  STATUS
api-portal        api-portal.tanzu.vmware.com    1.
0.3              Reconcile succeeded
app-accelerator   accelerator.apps.tanzu.vmware.com  1.
0.0              Reconcile succeeded
app-live-view     appliveview.tanzu.vmware.com    1.
0.2              Reconcile succeeded
appliveview-conventions build.appliveview.tanzu.vmware.com  1.
0.2              Reconcile succeeded
cartographer      cartographer.tanzu.vmware.com    0.
1.0              Reconcile succeeded
cloud-native-runtimes cnrs.tanzu.vmware.com    1.
0.3              Reconcile succeeded
convention-controller controller.conventions.apps.tanzu.vmware.com  0.
7.0              Reconcile succeeded
developer-conventions developer-conventions.tanzu.vmware.com  0.
3.0-build.1      Reconcile succeeded
grype-scanner     grype.scanning.apps.tanzu.vmware.com  1.
0.0              Reconcile succeeded
image-policy-webhook image-policy-webhook.signing.apps.tanzu.vmware.com  1.
1.2              Reconcile succeeded
metadata-store    metadata-store.apps.tanzu.vmware.com  1.
0.2              Reconcile succeeded
ootb-supply-chain-basic ootb-supply-chain-basic.tanzu.vmware.com  0.
5.1              Reconcile succeeded
ootb-templates    ootb-templates.tanzu.vmware.com    0.
5.1              Reconcile succeeded
scan-controller   scanning.apps.tanzu.vmware.com    1.
```

```

0.0          Reconcile succeeded
service-bindings      service-bindings.labs.vmware.com           0.
5.0          Reconcile succeeded
services-toolkit     services-toolkit.tanzu.vmware.com         0.
8.0          Reconcile succeeded
source-controller   controller.source.apps.tanzu.vmware.com   0.
2.0          Reconcile succeeded
sso4k8s-install     sso.apps.tanzu.vmware.com                 1.
0.0-beta.2-31 Reconcile succeeded
tap-gui             tap-gui.tanzu.vmware.com                  0.
3.0-rc.4          Reconcile succeeded
tekton-pipelines    tekton.tanzu.vmware.com                   0.3
0.0              Reconcile succeeded
tbs                 buildservice.tanzu.vmware.com             1.
5.0              Reconcile succeeded

```

## Next steps

- [Set up developer namespaces to use your installed packages](#)

## Set up developer namespaces to use your installed packages

To automatically set up your developer namespaces, see [Provision namespace resources](#).

For more information about Namespace Provisioner, see [Namespace Provisioner overview](#).

## Additional configuration for testing and scanning

If you plan to install or have already installed Out of the Box Supply Chains with Testing and Scanning, you can use Namespace Provisioner to set up the required resources. For more information, see [Add the resources required by Out of the Box Testing and Scanning Supply Chain](#).

## Legacy namespace setup

To use the legacy manual process for setting up developer namespaces, see [Legacy namespace setup](#).

## Next steps

- [Install Tanzu Developer Tools for your VS Code](#)

## Install Tanzu Developer Tools for your VS Code

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

## Prerequisites

Before installing the extension, you must have:

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

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

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

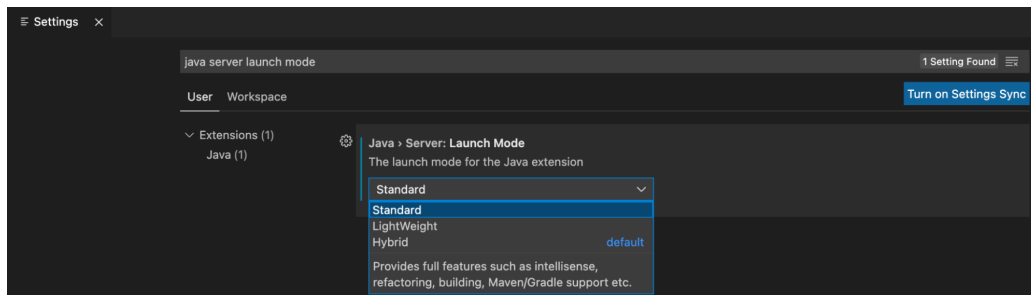
## Install

To install 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.

- **Enable Live Hover:** For more information, see [Integrating Live Hover by using Spring Boot Tools](#). Reload VS Code for this change to take effect.
- **Source Image:** (Required) The registry location for publishing local source code. For example, `registry.io/yourapp-source`. This must include both a registry and a project name.
- **Local Path:** (Optional) The path on the local file system to a directory of source code to build. This is the current directory by default.
- **Namespace:** (Optional) This is the namespace that workloads are deployed into. The namespace set in `kubeconfig` is the default.

## Uninstall

To uninstall VMware Tanzu Developer Tools for VS Code:

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

## Next steps

Proceed to [Getting started with Tanzu Developer Tools for Visual Studio Code](#).

## Custom Security Context Constraint details for Tanzu Application Platform

Custom Security Context Constraint (commonly known as SCC) details for Tanzu Application Platform (commonly known as TAP) components are as follows:

- [Application Accelerator on OpenShift cluster](#)
- [Application Live View on OpenShift](#)
- [Application Single Sign-On for OpenShift cluster](#)
- [Contour for OpenShift cluster](#)
- [Developer Conventions for OpenShift cluster](#)
- [Tanzu Build Service for OpenShift cluster](#)

## Application Accelerator on OpenShift

On OpenShift clusters, Application Accelerator must run with a custom SecurityContextConstraint (SCC) to enable compliance with restricted Kubernetes pod security standards. Tanzu Application Platform configures the following SCC for Application Accelerator when you configure the `kubernetes_distribution: openshift` key in the `tap-values.yaml` file.

Specification follows:

```
#@ load("@ytt:data", "data")
#@ load("@ytt:assert", "assert")

#@ kubernetes_distribution = data.values.kubernetes_distribution
#@ validDistributions = [None, "", "openshift"]
#@ if kubernetes_distribution not in validDistributions:
#@   assert.fail("{} not in {}".format(kubernetes_distribution, validDistributions))
#@ end

#@ if kubernetes_distribution == "openshift":
```

```

---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: accelerator-system-nonroot-scc
  namespace: accelerator-system
rules:
- apiGroups:
  - security.openshift.io
  resourceNames:
  - nonroot
  resources:
  - securitycontextconstraints
  verbs:
  - use
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: accelerator-system-nonroot-scc
  namespace: accelerator-system
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: accelerator-system-nonroot-scc
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: system:serviceaccounts:accelerator-system
#@ end

```

## Application Live View on OpenShift

Application Live View must run with a custom SecurityContextConstraint (SCC) to enable compliance with restricted Kubernetes Pod Security Standards on OpenShift. Tanzu Application Platform configures the following SCC for Application Live View back end, Application Live View connector, and Application Live View convention service when you configure the `kubernetes_distribution: openshift` key in the `tap-values.yaml` file.

The following is a `SecurityContextConstraints` specification for Application Live View connector:

```

---
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  name: appliveview-connector-restricted-with-seccomp
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: false
allowPrivilegedContainer: false
allowedCapabilities: null
defaultAddCapabilities: null
fsGroup:
  type: MustRunAs
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities:
- ALL
runAsUser:
  type: MustRunAsNonRoot
seLinuxContext:

```



```

  type: MustRunAs
supplementalGroups:
  type: RunAsAny
volumes:
- configMap
- downwardAPI
- emptyDir
- persistentVolumeClaim
- projected
- secret
seccompProfiles:
- runtime/default

```

The preceding `SecurityContextConstraints` specification is applicable to Application Live View back end and Application Live View convention service as well.

## Application Single Sign-On for OpenShift cluster

On OpenShift clusters, AppSSO must run with a custom `SecurityContextConstraint` (SCC) to enable compliance with restricted Kubernetes Pod Security Standards. Tanzu Application Platform configures the following SCC for AppSSO controller and its `AuthServer` managed resources when you configure the `kubernetes_distribution: openshift` key in the `tap-values.yaml` file.

Specification follows:

```

---
kind: SecurityContextConstraints
apiVersion: security.openshift.io/v1
metadata:
  name: appssso-scc
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: false
allowPrivilegedContainer: false
allowedCapabilities: null
defaultAddCapabilities: null
fsGroup:
  type: MustRunAs
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities:
- KILL
- MKNOD
- SETUID
- SETGID
runAsUser:
  type: MustRunAsNonRoot
seLinuxContext:
  type: MustRunAs
volumes:
- configMap
- downwardAPI
- emptyDir
- persistentVolumeClaim
- projected
- secret
seccompProfiles:
- 'runtime/default'

```

AppSSO controller's `ServiceAccount` is given the following additional permissions, including a `use` permission for AppSSO SCC, so `AuthServer` can use the custom SCC:

```
- apiGroups:
  - security.openshift.io
resources:
  - securitycontextconstraints
verbs:
  - "get"
  - "list"
  - "watch"
```

```
- apiGroups:
  - security.openshift.io
resourceNames:
  - appssso-scc
resources:
  - securitycontextconstraints
verbs:
  - "use"
```

## Contour for OpenShift cluster

On OpenShift clusters, Contour must run with a custom `SecurityContextConstraint` (SCC) to enable compliance with restricted Kubernetes Pod Security Standards. Tanzu Application Platform configures the following SCC for the service accounts in the `tanzu-system-ingress` namespace, which applies to Contour's controller and Envoy pods, when you configure the `kubernetes_distribution: openshift` key in the `tap-values.yaml` file.

Specification follows:

```
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  annotations:
    include.release.openshift.io/ibm-cloud-managed: "true"
    include.release.openshift.io/self-managed-high-availability: "true"
    include.release.openshift.io/single-node-developer: "true"
    kubernetes.io/description: nonroot provides all features of the restricted SCC
      but allows users to run with any non-root UID. The user must specify the UID
      or it must be specified on the by the manifest of the container runtime. On
      top of the legacy 'nonroot' SCC, it also requires to drop ALL capabilities and
      does not allow privilege escalation binaries. It will also default the seccomp
      profile to runtime/default if unset, otherwise this seccomp profile is required.
    name: contour-seccomp-nonroot-v2
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: false
allowPrivilegedContainer: false
allowedCapabilities:
- NET_BIND_SERVICE
defaultAddCapabilities: null
fsGroup:
  type: RunAsAny
groups: []
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities:
- ALL
```

```

runAsUser:
  type: MustRunAsNonRoot
seLinuxContext:
  type: MustRunAs
seccompProfiles:
- runtime/default
supplementalGroups:
  type: RunAsAny
users: []
volumes:
- configMap
- downwardAPI
- emptyDir
- persistentVolumeClaim
- projected
- secret

```

The SCC is bound to the service accounts by using the following Role and RoleBinding:

```

---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: contour-seccomp-nonroot-v2
  namespace: tanzu-system-ingress
rules:
- apiGroups:
  - security.openshift.io
  resourceNames:
  - contour-seccomp-nonroot-v2
  resources:
  - securitycontextconstraints
  verbs:
  - use
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: contour-seccomp-nonroot-v2
  namespace: tanzu-system-ingress
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: contour-seccomp-nonroot-v2
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: system:serviceaccounts:tanzu-system-ingress

```

## Developer Conventions for OpenShift cluster

On OpenShift clusters, Developer Conventions must run with a custom SecurityContextConstraint (SCC) to enable compliance with restricted Kubernetes pod security standards. Tanzu Application Platform configures the following SCC for the Developer Convention's webhook when you configure the `kubernetes_distribution: openshift` key in the `tap-values.yaml` file.

Specification follows:

```

---
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:

```

```

  name: developer-conventions-scc
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: false
allowPrivilegedContainer: false
defaultAddCapabilities: null
fsGroup:
  type: RunAsAny
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities: null
runAsUser:
  type: MustRunAsNonRoot
seLinuxContext:
  type: MustRunAs
supplementalGroups:
  type: RunAsAny
volumes:
  - secret
seccompProfiles: []
groups:
  - system:serviceaccounts:developer-conventions

```

## Tanzu Build Service for OpenShift cluster

On OpenShift clusters Tanzu Build Service must run with a custom [Security Context Constraint](#) (SCC) to enable compliance. Tanzu Application Platform configures the following SCC for Tanzu Build Service when you configure the `kubernetes_distribution: openshift` key in the `tap-values.yaml` file.

```

---
kind: SecurityContextConstraints
apiVersion: security.openshift.io/v1
metadata:
  name: tbs-restricted-scc-with-seccomp
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: false
allowPrivilegedContainer: false
allowedCapabilities:
  - NET_BIND_SERVICE
defaultAddCapabilities: null
fsGroup:
  type: RunAsAny
groups: []
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities:
  - ALL
runAsUser:
  type: MustRunAsNonRoot
seLinuxContext:
  type: MustRunAs
seccompProfiles:
  - runtime/default
supplementalGroups:
  type: RunAsAny
users: []

```

```
volumes:
  - configMap
  - downwardAPI
  - emptyDir
  - persistentVolumeClaim
  - projected
  - secret
```

It also applies the following RBAC to allow Tanzu Build Service services to use the SCC:

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    apps.tanzu.vmware.com/aggregate-to-workload: "true"
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  name: system:tbs:scc:restricted-with-seccomp
rules:
  - apiGroups:
    - security.openshift.io
    resourceNames:
    - tbs-restricted-scc-with-seccomp
    resources:
    - securitycontextconstraints
    verbs:
    - use
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: system:tbs:scc:restricted-with-seccomp
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:tbs:scc:restricted-with-seccomp
subjects:
  - kind: ServiceAccount
    namespace: build-service
    name: dependency-updater-serviceaccount
  - kind: ServiceAccount
    namespace: build-service
    name: dependency-updater-controller-serviceaccount
  - kind: ServiceAccount
    namespace: build-service
    name: secret-syncer-service-account
  - kind: ServiceAccount
    namespace: build-service
    name: warmer-service-account
  - kind: ServiceAccount
    namespace: build-service
    name: build-service-daemonset-serviceaccount
  - kind: ServiceAccount
    namespace: cert-injection-webhook
    name: cert-injection-webhook-sa
  - kind: ServiceAccount
    namespace: kpack
    name: kp-default-repository-serviceaccount
  - kind: ServiceAccount
    namespace: kpack
    name: kpack-pull-lifecycle-serviceaccount
  - kind: ServiceAccount
    namespace: kpack
    name: controller
  - kind: ServiceAccount
```

```
namespace: kpack
name: webhook
- kind: ServiceAccount
  namespace: stacks-operator-system
  name: controller-manager
```

## Customize your package installation

You can customize your package configuration that is not exposed through data values by using annotations and ytt overlays.

You can customize a package that was [installed manually](#) or that was [installed by using a Tanzu Application Platform profile](#).

## Customize a package that was manually installed

To customize a package that was installed manually:

1. Create a `secret.yml` file with a `Secret` that contains your ytt overlay. For example:

```
apiVersion: v1
kind: Secret
metadata:
  name: tap-overlay
  namespace: tap-install
stringData:
  custom-package-overlay.yml: |
    CUSTOM-OVERLAY
```

For more information about ytt overlays, see the [Carvel documentation](#).

2. Apply the `Secret` to your cluster by running:

```
kubectl apply -f secret.yml
```

3. Update your `PackageInstall` to include the `ext.packaging.carvel.dev/ytt-paths-from-secret-name.x` annotation to reference your new overlay `Secret`. For example:

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



### Note

You can suffix the extension annotation with `.x`, where `x` is a number, to apply multiple overlays. For more information, see the [Carvel documentation](#).

## Customize a package that was installed by using a profile

To add an overlay to a package that was installed by using a Tanzu Application Platform profile:

1. Create a `Secret` with your ytt overlay. For more information about ytt overlays, see the [Carvel documentation](#).
2. Update your values file to include a `package_overlays` field:

```
package_overlays:
- name: PACKAGE-NAME
  secrets:
  - name: SECRET-NAME
```

Where `PACKAGE-NAME` is the target package for the overlay. For example, `tap-gui`.

3. Update Tanzu Application Platform by running:

```
tanzu package installed update tap -p tap.tanzu.vmware.com -v 1.4.13 --values-file tap-values.yaml -n tap-install
```

For information about Tanzu Application Platform profiles, see [Installing Tanzu Application Platform package and profiles](#).

## Upgrade your Tanzu Application Platform

This document tells you how to upgrade your Tanzu Application Platform (commonly known as TAP).

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

## Prerequisites

Before you upgrade Tanzu Application Platform:

- Verify that you meet all the [prerequisites](#) of the target Tanzu Application Platform version. If the target Tanzu Application Platform version does not support your existing Kubernetes version, VMware recommends upgrading to a supported version before proceeding with the upgrade.
- For information about installing your Tanzu Application Platform, see [Install your Tanzu Application Platform profile](#).
- Ensure that Tanzu CLI is updated to the version recommended by the target Tanzu Application Platform version. For information about installing or updating the Tanzu CLI and plug-ins, see [Install or update the Tanzu CLI and plug-ins](#).
- For information about Tanzu Application Platform GUI considerations, see [Tanzu Application Platform GUI Considerations](#).
- Verify all packages are reconciled by running `tanzu package installed list -A`.
- To avoid the temporary warning state that is described in [Update the new package repository](#), upgrade to Cluster Essentials v1.4. See [Cluster Essentials documentation](#) for more information about the upgrade procedures.

## Update the new package repository

Follow these steps to update the new package repository:

1. Relocate the latest version of Tanzu Application Platform images by following step 1 through step 6 in [Relocate images to a registry](#).

**Important**

Make sure to update the `TAP_VERSION` to the target version of Tanzu Application Platform you are migrating to. For example, `1.4.13`.

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

**Cluster Essentials 1.2 or above**

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

**Cluster Essentials 1.1 or 1.0**

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

Expect to see the installed Tanzu Application Platform packages in a temporary “Reconcile Failed” state, following a “Package not found” warning. These warnings will disappear after you upgrade the installed Tanzu Application Platform packages to version 1.2.0.

3. Verify you have added the new package repository by running:

```
tanzu package repository get TAP-REPO-NAME --namespace tap-install
```

Where `TAP-REPO-NAME` is the package repository name. It must match with either `NEW-TANZU-TAP-REPOSITORY` or `tanzu-tap-repository` in the previous step.

## Perform the upgrade of Tanzu Application Platform

The following sections describe how to upgrade in different scenarios.

### Upgrade instructions for Profile-based installation

The following changes affect the upgrade procedures:

- **Keyless support deactivated by default**

In Tanzu Application Platform v1.4.0, keyless support is deactivated by default. For more information, see [Install Supply Chain Security Tools - Policy Controller](#).

To support the keyless authorities in `ClusterImagePolicy`, Policy Controller no longer initializes TUF by default. To continue using keyless authorities, you must set the `policy.tuf_enabled` field to `true` in the `tap-values.yaml` file during the upgrade process.

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

- **Image Policy Webhook no longer in use**

Tanzu Application Platform v1.4.0 removes Image Policy Webhook. If you use Image Policy Webhook in the previous version of Tanzu Application Platform, you must migrate the



`ClusterImagePolicy` resource from Image Policy Webhook to Policy Controller. For more information, see [Migration From Supply Chain Security Tools - Sign](#).

- **CVE results require a read-write service account**

Tanzu Application Platform v1.3.0 uses a read-only service account. In Tanzu Application Platform v1.4.0, enabling CVE results for the Supply Chain Choreographer and Security Analysis GUI plug-ins requires a read-write service account. For more information, see [Enable CVE scan results](#).

If you installed Tanzu Application Platform by using a profile, you can perform the upgrade by running the following command in the directory where the `tap-values.yaml` file resides:

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

## Upgrade the full dependencies package

If you installed the [full dependencies package](#), you can upgrade the package by following these steps:

1. After upgrading Tanzu Application Platform, retrieve the latest version of the Tanzu Build Service package by running:

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

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

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

Where `VERSION` is the version of the Tanzu Build Service package you retrieved in the previous step.

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

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

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

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

## Multicluster upgrade order

Upgrading a [multicluster deployment](#) requires updating multiple clusters with different profiles. If upgrades are not performed at the exact same time, different clusters have different versions of profiles installed temporarily. This might cause a temporary API mismatch that leads to errors. Those errors eventually disappear when the versions are consistent across all clusters.

To reduce the likelihood of temporary failures, follow these steps to upgrade your multicluster deployment:

1. Upgrade the view-profile cluster.
2. Upgrade the remaining clusters in any order.

## Upgrade instructions for component-specific installation

For information about upgrading Tanzu Application Platform GUI, see [Upgrade Tanzu Application Platform GUI](#). For information about upgrading Supply Chain Security Tools - Scan, see [Upgrade Supply Chain Security Tools - Scan](#).

## Verify the upgrade

Verify the versions of packages after the upgrade by running:

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

Your output is similar, but probably not identical, to the following example output:

```
- Retrieving installed packages...
NAME                                PACKAGE-NAME
PACKAGE-VERSION  STATUS
accelerator      accelerator.apps.tanzu.vmware.com
1.3.0            Reconcile succeeded
api-auto-registration  apis.apps.tanzu.vmware.com
0.1.1           Reconcile succeeded
api-portal       api-portal.tanzu.vmware.com
1.2.2           Reconcile succeeded
appliveview      backend.appliveview.tanzu.vmware.com
1.3.0           Reconcile succeeded
appliveview-connector  connector.appliveview.tanzu.vmware.com
1.3.0           Reconcile succeeded
appliveview-conventions  conventions.appliveview.tanzu.vmware.com
1.3.0           Reconcile succeeded
appsso          sso.apps.tanzu.vmware.com
2.0.0           Reconcile succeeded
buildservice    buildservice.tanzu.vmware.com
1.7.1           Reconcile succeeded
cartographer    cartographer.tanzu.vmware.com
0.5.3           Reconcile succeeded
cert-manager    cert-manager.tanzu.vmware.com
1.7.2+tap.1     Reconcile succeeded
cnrs            cnrs.tanzu.vmware.com
2.0.1           Reconcile succeeded
contour         contour.tanzu.vmware.com
1.22.0+tap.3    Reconcile succeeded
conventions-controller  controller.conventions.apps.tanzu.vmware.com
0.7.1           Reconcile succeeded
developer-conventions  developer-conventions.tanzu.vmware.com
0.8.0           Reconcile succeeded
eventing        eventing.tanzu.vmware.com
2.0.1           Reconcile succeeded
fluxcd-source-controller  fluxcd.source.controller.tanzu.vmware.com
0.27.0+tap.1    Reconcile succeeded
grype           grype.scanning.apps.tanzu.vmware.com
1.3.0           Reconcile succeeded
image-policy-webhook  image-policy-webhook.signing.apps.tanzu.vmware.c
om 1.1.7        Reconcile succeeded
learningcenter   learningcenter.tanzu.vmware.com
0.2.3           Reconcile succeeded
learningcenter-workshops  workshops.learningcenter.tanzu.vmware.com
0.2.2           Reconcile succeeded
metadata-store   metadata-store.apps.tanzu.vmware.com
1.3.3           Reconcile succeeded
ootb-delivery-basic  ootb-delivery-basic.tanzu.vmware.com
0.10.2          Reconcile succeeded
ootb-supply-chain-testing-scanning  ootb-supply-chain-testing-scanning.tanzu.vmware.
com 0.10.2      Reconcile succeeded
ootb-templates   ootb-templates.tanzu.vmware.com
```

```

0.10.2      Reconcile succeeded
  policy-controller      policy.apps.tanzu.vmware.com
1.1.1      Reconcile succeeded
  scanning               scanning.apps.tanzu.vmware.com
1.3.0      Reconcile succeeded
  service-bindings      service-bindings.labs.vmware.com
0.8.0      Reconcile succeeded
  services-toolkit      services-toolkit.tanzu.vmware.com
0.8.0      Reconcile succeeded
  source-controller     controller.source.apps.tanzu.vmware.com
0.5.0      Reconcile succeeded
  spring-boot-conventions
spring-boot-conventions.tanzu.vmware.com
0.5.0      Reconcile succeeded
  tap                   tap.tanzu.vmware.com
1.3.0      Reconcile succeeded
  tap-auth              tap-auth.tanzu.vmware.com
1.1.0      Reconcile succeeded
  tap-gui               tap-gui.tanzu.vmware.com
1.3.0      Reconcile succeeded
  tap-telemetry         tap-telemetry.tanzu.vmware.com
0.3.1      Reconcile succeeded
  tekton-pipelines     tekton.tanzu.vmware.com
0.39.0+tap.2      Reconcile succeeded

```

## Opt out of telemetry collection

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



### Note

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

Follow these steps to turn off telemetry collection:

### kubectl

To turn off telemetry collection on your Tanzu Application Platform by using `kubectl`:

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

```

- If you already have Tanzu Application Platform installed, restart the telemetry collector to pick up the change:

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

### Tanzu CLI

The Tanzu CLI provides a telemetry plugin enabled by the Tanzu Framework v0.25.0, which has been included in Tanzu Application Platform since v1.3.

To turn off telemetry collection on your Tanzu Application Platform by using the Tanzu CLI:

```
$ tanzu telemetry update --CEIP-opt-out
*no output*
```

To learn more about how to update the telemetry settings:

```
$ tanzu telemetry update --help
Update tanzu telemetry settings

Usage:
  tanzu telemetry update [flags]

Examples:

  # opt into ceip
  tanzu telemetry update --CEIP-opt-in
  # opt out of ceip
  tanzu telemetry update --CEIP-opt-out
  # update shared configuration settings
  tanzu telemetry update --env-is-prod "true" --entitlement-account-number "1234"
  --csp-org-id "XXXX"

Flags:
  --CEIP-opt-in           opt into VMware's CEIP program
  --CEIP-opt-out         opt out of VMware's CEIP program
  --csp-org-id string    Accepts a string and sets a cluster-wide
                        CSP
                        org ID.
                        Empty string is equivalent to unsetting
                        this value.
  --entitlement-account-number string  Accepts a string and sets a cluster-wide
                        entitlement account number. Empty string is
                        equivalent to unsetting this value
  --env-is-prod string    Accepts a boolean and sets a cluster-wide
                        value denoting whether the target is a
                        production cluster or not.
  -h, --help             help for update
```

At this point, your Tanzu Application Platform deployment no longer emits telemetry, and you are opted out of the CEIP.

# Overview of security and compliance in Tanzu Application Platform

Security is a primary focus for Tanzu Application Platform (commonly known as TAP).

This section describes:

[Secure Ingress certificates in Tanzu Application Platform](#)

[\[Use custom CA certificates in Tanzu Application Platform\(custom-ca-certificates.hbs.md\)](#)

[Assess Tanzu Application Platform against the NIST 800-53 Moderate Assessment](#)

## Overview of security and compliance in Tanzu Application Platform

Security is a primary focus for Tanzu Application Platform (commonly known as TAP).

This section describes:

[Secure Ingress certificates in Tanzu Application Platform](#)

[\[Use custom CA certificates in Tanzu Application Platform\(custom-ca-certificates.hbs.md\)](#)

[Assess Tanzu Application Platform against the NIST 800-53 Moderate Assessment](#)

## Overview of security and compliance in Tanzu Application Platform

Security is a primary focus for Tanzu Application Platform (commonly known as TAP).

This section describes:

[Secure Ingress certificates in Tanzu Application Platform](#)

[\[Use custom CA certificates in Tanzu Application Platform\(custom-ca-certificates.hbs.md\)](#)

[Assess Tanzu Application Platform against the NIST 800-53 Moderate Assessment](#)

## Secure Ingress certificates in Tanzu Application Platform

This topic tells you about securing ingress communication in Tanzu Application Platform (commonly known as TAP).

By default, Tanzu Application Platform installs and uses a self-signed CA for issuing TLS certificates to components for the purpose of securing ingress communication.

The ingress issuer is a self-signed `cert-manager.io/v1/ClusterIssuer` and is provided by Tanzu Application Platform's `cert-manager` package. Its default name is `tap-ingress-selfsigned`.

To understand how each component uses the ingress issuer, see [Component documentation](#).

As of v1.4.0, not all components use the ingress issuer. For an overview of the components that participate, see the [Release notes](#).

## Replacing the default ingress issuer

Tanzu Application Platform's default ingress issuer can be replaced by any other `cert-manager.io/v1/ClusterIssuer`.

To replace the default ingress issuer, create a `ClusterIssuer` and set `shared.ingress_issuer` to the name of the issuer. After the configuration is applied, components eventually obtain certificates from the new issuer and serve them.

Tanzu Application Platform's `cert-manager` package must be present for the `ClusterIssuer` API to be available. This means you can only provide your own `ClusterIssuer` after the initial installation. You can, however, reference your issuer in the initial installation.

For example, to use `Let's Encrypts` production API to issue TLS certificates:

1. Update and apply Tanzu Application Platform's installation values such that `shared.ingress_issuer` denotes the bespoke issuer:

```
#!/ my-tap-values.yaml
#! ...
shared:
  ingress_issuer: letsencrypt-production
#! ...
```

2. Create a `ClusterIssuer` for `Let's Encrypts` production API:

```
---
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-production
spec:
  acme:
    email: certificate-notices@my-company.com
    privateKeySecretRef:
      name: letsencrypt-production
    server: https://acme-v02.api.letsencrypt.org/directory
    solvers:
      - http01:
          ingress:
            class: contour
```

`Let's Encrypts` production API has [rate limits](#).

For more information about the possible configurations of `ClusterIssuer`, see [cert-manager documentation](#).

## Deactivating TLS for ingress

Although VMware discourages this, you can deactivate the ingress issuer by setting `shared.ingress_issuer: ""`. As a result, components consider TLS for ingress to be deactivated.

## Overriding TLS for components

You can override TLS settings for each component. In your Tanzu Application Platform installation values, set the component's values that you want, and they take precedence over `shared` values. See [components](#) to understand its settings for ingress and TLS.

## Use custom CA certificates in Tanzu Application Platform

This topic tells you about configuring custom CA certificates in Tanzu Application Platform (commonly known as TAP).

You configure trust for custom CAs. This is helpful if any Tanzu Application Platforms components are connecting to services that serve certificates issued by private certificate authorities.

The `shared.ca_cert_data` installation value can contain a PEM-encoded CA bundle. Each component then trusts the CAs contained in the bundle.

You can also configure trust per component by providing a CA bundle in the component's installation values. The component then trusts those CAs and the CAs configured in `shared.ca_cert_data`. For more information, see [components](#).

## Use External Secrets Operator in Tanzu Application Platform (alpha)

The [External Secrets Operator](#) is a Kubernetes operator that integrates with external secret management systems, for example, Google Secrets Manager and Hashicorp Vault. It reads information from external APIs and automatically injects the values into a Kubernetes secret. Starting with v1.4.0, Tanzu Application Platform repackages the External Secrets Operator v0.6.1 into a Carvel bundle.



### Caution

External Secrets Operator is in alpha and is intended for evaluation and test purposes only. Do not use in a production environment. This package does not constitute an entire solution. VMware expects later Tanzu Application Platform releases to have a more comprehensive secret management solution.

## Installing the External Secrets Operator

Tanzu Application Platform packages a version of the External Secrets Operator that you can install in the `tap-install` namespace. The External Secrets Operator is an optional Tanzu Application Platform component. It does not come installed with any of the default Tanzu Application Platform profiles.

```
ESO_VERSION=0.6.1+tap.2
TAP_NAMESPACE=tap-install

tanzu package install external-secrets \
  --package-name external-secrets.apps.tanzu.vmware.com \
  --version "$ESO_VERSION" \
  --namespace "$TAP_NAMESPACE"
```

## Using the External Secrets Operator

The following example demonstrates a use case for configuring external secrets in a Supply Chain.

### Connecting to a secret manager

#### Example : Google Secret Manager

To connect to a Google Secret Manager instance, you need a service account with the appropriate permissions. The following steps detail how to create a `ClusterStore` resource that uses this service account to connect to your Secret Manager.

1. Create a secret that holds the service account. Run:

```
ytt -f google-secrets-manager-secret.yaml \
  --data-value gcp_secret_name=google-secrets-manager-secret \
  --data-value secrets_namespace=external-secrets \
  --data-value-file service_account_json=<path_to_service_account_json> | kubectl apply -f-
```

Where:

```
> cat google-secrets-manager-secret.yaml
#@ load("@ytt:data", "data")
---
apiVersion: v1
kind: Secret
metadata:
  labels:
    type: gcpsm
  name: #@ data.values.gcp_secret_name
  namespace: #@ data.values.secrets_namespace
stringData:
  secret-access-credentials: #@ data.values.service_account_json
type: Opaque
```

2. Create a `ClusterStoreSecret` resource that references the earlier secret. Run:

```
export GOOGLE_PROJECT_ID="project-id-name"
ytt -f google-secrets-store.yaml \
  --data-value cluster_store_name=google-secrets-manager-store-secret \
  --data-value secret_name=google-secrets-manager-secret-store \
  --data-value google_project_id="#{GOOGLE_PROJECT_ID}" | kubectl delete -f-
```

Where:

```
> cat google-secrets-store.yaml
#@ load("@ytt:data", "data")
---
apiVersion: external-secrets.io/v1beta1
kind: ClusterSecretStore
metadata:
  name: #@ data.values.cluster_store_name
spec:
  provider:
    gcpsm:
      auth:
        secretRef:
          secretAccessKeySecretRef:
            key: secret-access-credentials
            name: #@ data.values.secret_name
            namespace: external-secrets
      projectID: #@ data.values.google_project_id
```

3. To verify that you have correctly authenticated to the Secret Manager, run:

```
tanzu external-secrets stores list -A
NAMESPACE NAME PROVIDER STAT
US
Cluster google-secrets-manager-cluster-store Google Secrets Manager Valid
```

For more information about using the External Secrets Operator, see the [External Secrets Operator site](#).



## Create a synchronized secret

The secret to be synchronized must exist on your Secret Manager. Alternatively, you can create one. For example, say you want to access your secret to access a maven repository stored in a secret `maven-registry-credential` in your Secret Manager. First, create an external secrets resource with a reference to the secret to be synchronized and the `Valid ClusterStore` configured earlier. Run:

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

Where:

```
---
apiVersion: external-secrets.io/v1beta1
kind: ExternalSecret
metadata:
  name: git-secret
  namespace: my-apps
spec:
  data:
    # SecretStoreRef defines which SecretStore to use when fetching the secret data
    - remoteRef:
        conversionStrategy: Default
        key: maven-registry-credentials
        secretKey: apitoken
  refreshInterval: 1m
  secretStoreRef:
    kind: ClusterSecretStore
    name: google-secrets-manager-cluster-store
  # the target describes the secret that shall be created
  # there can only be one target per ExternalSecret
  target:
    creationPolicy: Owner
    deletionPolicy: Retain
    name: maven-registry-credentials
    template:
      metadata:
        annotations:
          tekton.dev/git-0: https://github.com
      type: kubernetes.io/basic-auth
      data:
        password: '{{ .apitoken | toString }}'
        username: "_json_key"
      engineVersion: v2
```

## Using a synchronized secret

The following example uses the synchronized secret to access a maven artifact:

```
local readonly workload_name='java-web-app'

tanzu apps wld apply $workload_name \
  --service-account supply-chain-service-account \
  --param-yaml maven='{"artifactId": "spring-petclinic", "version": "3.7.4", "groupId": "org.springframework.samples"}' \
  --param maven_repository_secret_name="maven-registry-credentials" \
  --type web \
  --app spring-petclinic -y -n my-apps
```

## Assess Tanzu Application Platform against the NIST 800-53 Moderate Assessment

This topic provides you with an assessment of Tanzu Application Platform (commonly known as TAP) against the NIST SP 800-53 Revision 4 Moderate baseline controls. This translates to FISMA Moderate and CNSSI 1253 Mod/Mod/Mod for use in US Federal systems accreditation.

The Moderate baseline applies to only technical controls. Organizational policy controls, physical security, media policies, and similar are excluded as they are not applicable to Tanzu Application Platform. These excluded controls are still relevant to the system at large and must be inherited from existing accreditations or otherwise addressed.

The initial iteration of this assessment delineates responsible parties. Incremental updates will add more details about implementation and updates to 800-53 Revision 5.

Name	Title	Responsible Party	Notes
AC-2(1)	Automated System Account Management	Customer	Implemented on customer identity store. The customer must employ automated mechanisms to support the management of information system accounts used to access their Tanzu Application Platform installation.
AC-2(2)	Removal of Temporary / Emergency Accounts	Customer	Implemented on customer identity store. If the customer chooses to use temporary or emergency accounts, they must ensure that the system automatically deactivates or removes the account following an organization-defined time period.
AC-2(3)	Deactivate Inactive Accounts	Customer	Implemented on customer identity store. The customer must automatically deactivate inactive accounts used to access their Tanzu Application Platform installation following an organization-defined time period of inactivity.
AC-2(4)	Automated Audit Actions	Customer	Implemented on customer identity store. The customer must automatically audit account creation, modification, enabling, deactivating, and removal actions associated with accounts used to access their Tanzu Application Platform installation and must notify an organization-defined personnel or role.
AC-3	Access Enforcement	Customer	The customer must federate their IdP with Tanzu Application Platform to enforce approved access authorizations to their Tanzu Application Platform installation.
AC-4	Information Flow Enforcement	Customer	The customer is responsible for enforcing approved authorizations for controlling the flow of information between Tanzu Application Platform and interconnected systems, based on organization-defined information flow control policies, for example, a SIEM. Tanzu Application Platform does not restrict intra-service or inter-system communication. Future versions of Tanzu Application Platform will include this feature using service mesh architecture or similar methods.
AC-6	Least Privilege	Shared	The customer is responsible for enforcing least privilege by ensuring Tanzu Application Platform users have the minimum permissions necessary to perform their job function. Tanzu Application Platform is responsible for providing RBAC functionality to enforce least privilege.
AC-6(1)	Authorize Access to Security Functions	Shared	The customer is responsible for explicitly authorizing access to organization-defined security functions and security-relevant information as it relates to their Tanzu Application Platform installation. Tanzu Application Platform is responsible for providing the RBAC functionality necessary to restrict which users can access security functions and security-related information.
AC-6(5)	Privileged Accounts	Shared	The customer must restrict privileged Tanzu Application Platform accounts to organization-defined personnel or roles. Tanzu Application Platform is responsible for providing the RBAC functionality for customers to restrict privileged Tanzu Application Platform accounts to organization-defined personnel or roles.

Name	Title	Responsible Party	Notes
AC-6(9)	Auditing Use of Privileged Functions	Shared	The customer is responsible for configuring Tanzu Application Platform and underlying Kubernetes to send log streams to their SIEM tool for log analysis to be capable of auditing the execution of privileged functions. Tanzu Application Platform is responsible for generating logs pertaining to the execution of privileged functions that can be ingested by the customer SIEM tool for analysis.
AC-6(10)	Prohibit Non-Privileged Users from Executing Privileged Functions	Tanzu Application Platform	This functionality is inherent to Tanzu Application Platform/Kubernetes RBAC and can't be configured otherwise.
AC-7 AC-7a AC-7b	Unsuccessful Logon Attempts	Customer	Implemented on customer identity provider. The customer is responsible for configuring their IdP to enforce a limit of consecutive invalid logon attempts by a user during an organization-defined time period which locks the user's account for an organization-defined time period, or until released by an administrator.
AC-8 AC-8a AC-8a.1 AC-8a.2 AC-8a.3 AC-8a.4 AC-8b AC-8c AC-8c.1 AC-8c.2 AC-8c.3	System Use Notification	Customer	Implemented on customer identity provider. Customer must configure their IdP to display the system use notification banner before login.
AC-11 AC-11a AC-11b	Session Lock	Customer	The customer must configure sessions locks on user workstations used to access their Tanzu Application Platform installation. Tanzu Application Platform does not have a concept of session locks and relies on sessions locks applied by the user's workstation. Tanzu Application Platform provides logout functionality in place of session locking.
AC-11(1)	Pattern-Hiding Displays	Customer	The customer must configure sessions locks on user workstations used to access their Tanzu Application Platform installation. This includes hiding the user's private session with a publicly available image. Tanzu Application Platform does not have a concept of session locks and relies on sessions locks applied by the user's workstation. Tanzu Application Platform provides logout functionality in place of session locking.
AC-12	Session Termination	Shared	Implemented on customer identity provider. The customer is responsible for configuring IdP token TTL and refresh policies that apply to Tanzu Application Platform sessions. Tanzu Application Platform enforces token policies and cannot be configured otherwise.
AC-14 AC-14a	Permitted Actions Without Identification or Authentication	Shared	The customer is responsible for identifying organization-defined user actions that can be performed on the information system without identification or authentication consistent with organizational missions/business functions. For production installations, Tanzu Application Platform GUI must be configured with OIDC authentication and guest access deactivated.
AC-17(1)	Automated Monitoring / Control	Customer	"Remote Access" is defined as outside-the-org endpoints like remote workers over VPN. This is outside the scope of Tanzu Application Platform. The customer is responsible for all aspects regarding "remote access" to Tanzu Application Platform.

Name	Title	Responsible Party	Notes
AC-17(2)	Protection of Confidentiality / Integrity Using Encryption	Customer	“Remote Access” is defined as outside-the-org endpoints like remote workers over VPN. This is outside the scope of Tanzu Application Platform. The customer is responsible for implementing cryptographic mechanisms to protect the confidentiality and integrity of “remote access” sessions to Tanzu Application Platform.
AC-17(3)	Managed Access Control Points	Customer	“Remote Access” is defined as outside-the-org endpoints like remote workers over VPN. This is outside the scope of Tanzu Application Platform. The customer is responsible for routing all “remote accesses” to Tanzu Application Platform through an organization-defined number of managed network access control points.
AC-19 AC-19 AC-19b	Access Control for Mobile Devices	Customer	The customer is responsible for all aspects regarding mobile devices which grant access to Tanzu Application Platform.
AU-3	Content of Audit Records	Tanzu Application Platform	The Tanzu Application Platform application must be capable of generating audit logs that contain the minimum content required by the customer consuming the application.
AU-3(1)	Additional Audit Information	Customer	Implemented on customer SIEM. The customer is responsible for parsing Tanzu Application Platform logs on their SIEM to extract organization-defined extra information.
AU-4	Audit Storage Capacity	Customer	Implemented on customer Kubernetes. Tanzu Application Platform logs are all captured by Kubernetes logging. The customer is responsible for configuring their Kubernetes hosts with record storage capacity to ensure that there is adequate storage of logs generated by Tanzu Application Platform clusters.
AU-5 AU-5a AU-5b	Response to Audit Processing Failures	Customer	Implemented on customer Kubernetes. Tanzu Application Platform audit records are collected and managed by Kubernetes and are out of Tanzu Application Platform scope. The customer is responsible for configuring their Kubernetes hosts to account for audit processing failures and to alert the appropriate personnel responsible to take appropriate action.
AU-7 AU-7a AU-7b	Audit Reduction and Report Generation	Customer	Implemented on customer Kubernetes and SIEM Tanzu Application Platform audit records are collected and managed by Kubernetes. The customer is responsible for ensuring that Kubernetes ships Tanzu Application Platform audit records to a central SIEM for review and analysis.
AU-7(1)	Automatic Processing	Customer	Implemented on customer Kubernetes and SIEM Tanzu Application Platform audit records are collected and managed by Kubernetes. The customer is responsible for ensuring that Kubernetes ships Tanzu Application Platform audit records to a central SIEM for review and analysis.
AU-8 AU-8a AU-8b	Time Stamps	Tanzu Application Platform	Tanzu Application Platform components pull their system time from the container OS and the Kubernetes host and cannot be configured otherwise. Tanzu Application Platform components log statements include UTC timestamps and cannot be configured otherwise.
AU-8(1) AU-8(1) (a) AU-8(1) (b)	Synchronization With Authoritative Time Source	Customer	The customer is responsible for configuring authoritative time sources on K8 clusters.
AU-9	Protection of Audit Information	Customer	Tanzu Application Platform audit records are collected and managed by Kubernetes. The customer is responsible for protecting Kubernetes and Kubernetes logging configurations from unauthorized access, modification, and deletion.

Name	Title	Responsible Party	Notes
AU-12 AU-12a AU-12b AU-12c	Audit Generation	Shared	Tanzu Application Platform audit records are collected and managed by Kubernetes. The customer is responsible for ensuring that Kubernetes ships Tanzu Application Platform audit records to a central SIEM for review and analysis. Tanzu Application Platform cannot be configured to audit specific information. Tanzu Application Platform logs verbosely and lets the customer filter out what is relevant to them using their SIEM. Tanzu Application Platform logging cannot be deactivated.
CM-7 CM-7a CM-7b	Least Functionality	Shared	The customer is responsible for configuring Tanzu Application Platform to provide only essential capabilities. Tanzu Application Platform is responsible for providing customers with the capability to deactivate non-essential features not required by the customer. The customer must restrict the use of functions, ports, protocols, and services for the Tanzu Application Platform installation. Tanzu Application Platform is responsible for ensuring that functions, ports, protocols, and services are limited to those explicitly required for the application to operate.
CM-7(2)	Prevent Program Execution	Tanzu Application Platform	As an extension of CM-7, Least Functionality, this control is a responsibility of Tanzu Application Platform. Tanzu Application Platform only consists of containers with purposeful services with no extra programs running or bloat. This cannot be configured by the customer.
CM-7(4) (b)	Unauthorized Software/De nylisting	Tanzu Application Platform	Tanzu Application Platform service containers do not implement a deny-by-exception policy to prohibit the execution of unauthorized software programs. Tanzu Application Platform service containers are built to provide stripped-down services and do not include extra programs or bloat. Tanzu Application Platform can provide a SBOM to compare against customer organization policies on disallowed software.
IA-2	Identification and Authenticati on (Organizatio nal Users)	Shared	The customer is responsible for configuring Tanzu Application Platform to use their IdP which is capable of uniquely identifying and authenticating organizational users. Tanzu Application Platform is responsible for providing customers with the capability to integrate their IdP to allow Tanzu Application Platform to uniquely identify organizational users.
IA-2(1)	Network Access to Privileged Accounts	Customer	Implemented on customer identity provider. The customer is responsible for implementing multifactor authentication on their IdP for network access to privileged accounts.
IA-2(2)	Network Access to Non- Privileged Accounts	Customer	Implemented on customer identity provider. The customer is responsible for implementing multifactor authentication on their IdP for network access to non-privileged accounts.
IA-2(3)	Local Access to Privileged Accounts	N/A	Tanzu Application Platform does not use local accounts. All access occurs over a network connection.
IA-2(8)	Network Access to Privileged Accounts - Replay Resistant	Tanzu Application Platform	Tanzu Application Platform is responsible for ensuring that all connections to the customer IdP are over TLS 1.2+.
IA-2(11)	Remote Access - Separate Device	Customer	The customer is responsible for all aspects of MFA and MFA devices used to authenticate to their Tanzu Application Platform installation, including using remote access.

Name	Title	Responsible Party	Notes
IA-2(12)	Acceptance of Piv Credentials	Customer	Implemented on customer identity provider. The customer is responsible for implementing CAC/PIV credentials with their IdP.
IA-3	Device Identification and Authentication	Customer	The customer is responsible for uniquely identifying and authenticating organization-defined specific and/or types of devices before establishing a local, remote, or network connection.
IA-4e	Identifier Management	Customer	Implemented on customer identity provider. The customer is responsible for configuring IdP token TTL and refresh policies that apply to Tanzu Application Platform sessions. Tanzu Application Platform enforces token policies and cannot be configured otherwise.
IA-5(1) IA-5(1)(a) IA-5(1)(b) IA-5(1)(c) IA-5(1)(d) IA-5(1)(e) IA-5(1)(f)	Password-Based Authentication	Customer	Implemented on customer identity store. The customer is responsible for all aspects of password-based authentication to their IdP, using their identity store. Tanzu Application Platform does not employ password-based authentication itself.
IA-5(2) IA-5(2)(a) IA-5(2)(b) IA-5(2)(c) IA-5(2)(d)	PKI-Based Authentication	Customer	Implemented on customer identity provider. The customer is responsible for all aspects of PKI-based authentication on the IdP used to access their Tanzu Application Platform installation.
IA-5(11)	Hardware Token-Based Authentication	Customer	The customer is responsible for ensuring hardware token-based authentication employs mechanisms that satisfy organization-defined token quality requirements.
IA-6	Authenticator Feedback	Customer	Implemented on customer identity provider. The customer is responsible for ensuring their IdP obscures feedback of authentication information during the authentication process.
IA-7	Cryptographic Module Authentication	Customer	Implemented on customer identity provider. The customer is responsible for ensuring their IdP implements FIPS 140-2 validated cryptographic modules.
IA-8	Identification and Authentication(Non-Organizational Users)	Customer	Implemented on customer identity provider. The customer is responsible for ensuring that their IdP uniquely identifies and authenticates non-organizational Tanzu Application Platform users, or processes acting on behalf of non-organizational users.
IA-8(1)	Acceptance of Piv Credentials from Other Agencies	Customer	Implemented on customer identity provider. The customer is responsible for configuring their IdP to accept and electronically verify Personal Identity Verification(PIV) credentials from other federal agencies.
IA-8(2)	Acceptance of Third-Party Credentials	Customer	Implemented on customer identity provider. The customer is responsible for configuring their IdP to accept only FICAM-approved third-party credentials.
IA-8(3)	Use of FICAM-Approved Products	Customer	Implemented on customer identity provider. The customer is responsible for employing only FICAM-approved information system components on their IdP to accept third-party credentials.

Name	Title	Responsible Party	Notes
IA-8(4)	Use of FICAM-Issued Profiles	Customer	Implemented on customer identity provider. The customer is responsible for ensuring their IdP conforms to FICAM-issued profiles.
SC-2	Application Partitioning	Tanzu Application Platform	Tanzu Application Platform does not isolate user and management functionality on separate network interfaces, instances, CPUs, or similar. Tanzu Application Platform relies on different roles and Kubernetes RBAC to keep user and management functionality distinct.
SC-4	Information in Shared Resources	Tanzu Application Platform	Tanzu Application Platform creates dedicated Kubernetes namespaces upon deployment. Kubernetes namespaces prevent unauthorized and unintended information transfer using shared system resources.
SC-5	Denial of Service Protection	Customer	The customer is responsible for ensuring that organizational DoS protections at the network layer include the Tanzu Application Platform installation.
SC-7 SC-7a SC-7b SC-7c	Boundary Protection	Customer	The customer is responsible for the configuration and management of boundary protection devices.
SC-7(4)(c)	External Telecommunications Services	Customer	The customer is responsible for external telecommunication services used to establish connections to their Tanzu Application Platform installation.
SC-7(5)	Deny by Default / Allow by Exception	Shared	Tanzu Application Platform does not implement “deny by default” network policies. This might be mitigated by network-level access controls configured by the customer.
SC-7(7)	Prevent Split Tunneling for Remote Devices	Customer	The customer is responsible for all configuration of remote devices used to access Tanzu Application Platform.
SC-8	Transmission Confidentiality and Integrity	Tanzu Application Platform	Tanzu Application Platform is responsible for ensuring all communications occur over TLS 1.2+.
SC-8(1)	Cryptographic or Alternate Physical Protection	Tanzu Application Platform	Tanzu Application Platform is responsible for ensuring all communications occur over TLS 1.2+.
SC-10	Network Disconnect	Tanzu Application Platform	Tanzu Application Platform tears down TCP connections and deallocates system resources following the expiration of a session token and cannot be configured otherwise.
SC-12	Cryptographic Key Establishment and Management	Tanzu Application Platform	Tanzu Application Platform is responsible for providing customers with the ability to manage trust stores.
SC-13	Cryptographic Protection	Tanzu Application Platform	Tanzu Application Platform is responsible for implementing FIPS 140 validated cryptographic modules and providing the customer with a means to enable “FIPS Mode”.

Name	Title	Responsible Party	Notes
SC-21	Secure Name / Address Resolution Service (Recursive or Caching Resolver)	Customer	Tanzu Application Platform inherits the DNSSEC capabilities of the organization resolvers it is configured to use. The customer is responsible for configuring the Tanzu Application Platform and Kubernetes infrastructure to use DNSSEC-capable resolvers.
SC-23	Session Authenticity	Tanzu Application Platform	Tanzu Application Platform is responsible for ensuring all communications occur over TLS 1.2+.
SC-28	Protection of Information at Rest	Customer	Tanzu Application Platform does not natively provide encryption for data at rest, but instead relies on the underlying Kubernetes persistent volumes for appropriate cryptographic protections. The customer is responsible for deploying Tanzu Application Platform to Kubernetes with persistent volumes for appropriate cryptographic protections.
SC-39	Process Isolation	Tanzu Application Platform	Tanzu Application Platform container OS enforces the use of separate execution domains for each executing process and cannot be configured otherwise. The underlying Kubernetes host isolates each container from the other.
SI-2c	Flaw Remediation	Tanzu Application Platform	The customer is responsible for keeping the Tanzu Application Platform installation up to date, to within org-defined standards. Tanzu Application Platform does not automatically update itself.
SI-3(2)	Automatic Updates	N/A	Tanzu Application Platform does not include malicious code protection mechanisms therefore automatic update to such mechanisms does not apply.
SI-7(1)	Integrity Checks	Tanzu Application Platform	Tanzu Application Platform performs a hash check when images are downloaded, and a cryptographic signature validation at runtime. This cannot be configured otherwise.
SI-10	Information Input Validation	Tanzu Application Platform	Tanzu Application Platform is responsible for performing input validation of user-supplied input to Tanzu Application Platform.
SI-11 SI-11a SI-11b	Error Handling	Tanzu Application Platform	Tanzu Application Platform limits error message verbosity but does display errors to users. Given the development/coding nature of Tanzu Application Platform, deployment errors and similar must be raised to the user so they can be corrected.
SI-16	Memory Protection	Tanzu Application Platform	Tanzu Application Platform container OS protects its memory from unauthorized code execution and cannot be configured otherwise. The underlying Kubernetes host also isolates container memory pages.



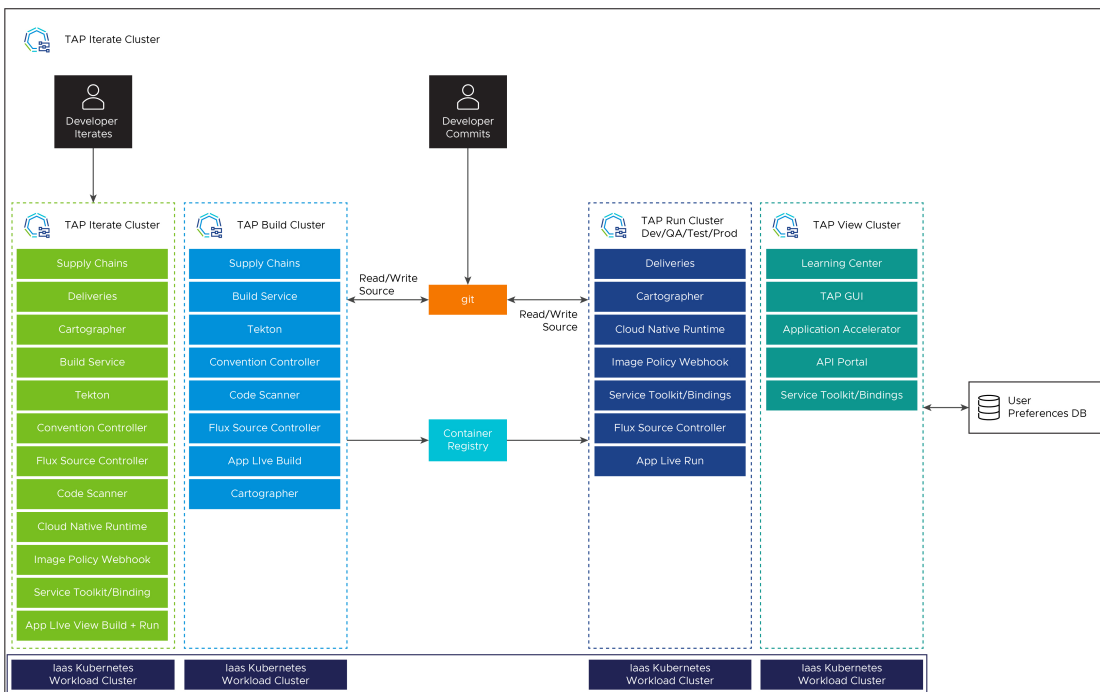
# Overview of multicluster Tanzu Application Platform

You can install Tanzu Application Platform (commonly known as TAP) in various topologies to reflect your existing landscape. VMware has tested and recommends a multicluster topology for production use. Because flexibility and choice are core to Tanzu Application Platform’s design, none of the implementation recommendations are set in stone.

The multicluster topology uses the [profile capabilities](#) supported by Tanzu Application Platform. Each cluster adopts one of following multicluster-aligned profiles:

- **Iterate:** Intended for inner-loop iterative application development.
- **Build:** Transforms source revisions to workload revisions; specifically, hosting workloads and supply chains.
- **Run:** Transforms workload revisions to running pods; specifically, hosting deliveries and deliverables.
- **View:** For applications related to centralized developer experiences; specifically, Tanzu Application Platform GUI and metadata store.

The following diagram illustrates this topology.



## Next steps

To get started with installing a multicluster topology, see [Install multicluster Tanzu Application Platform profiles](#).

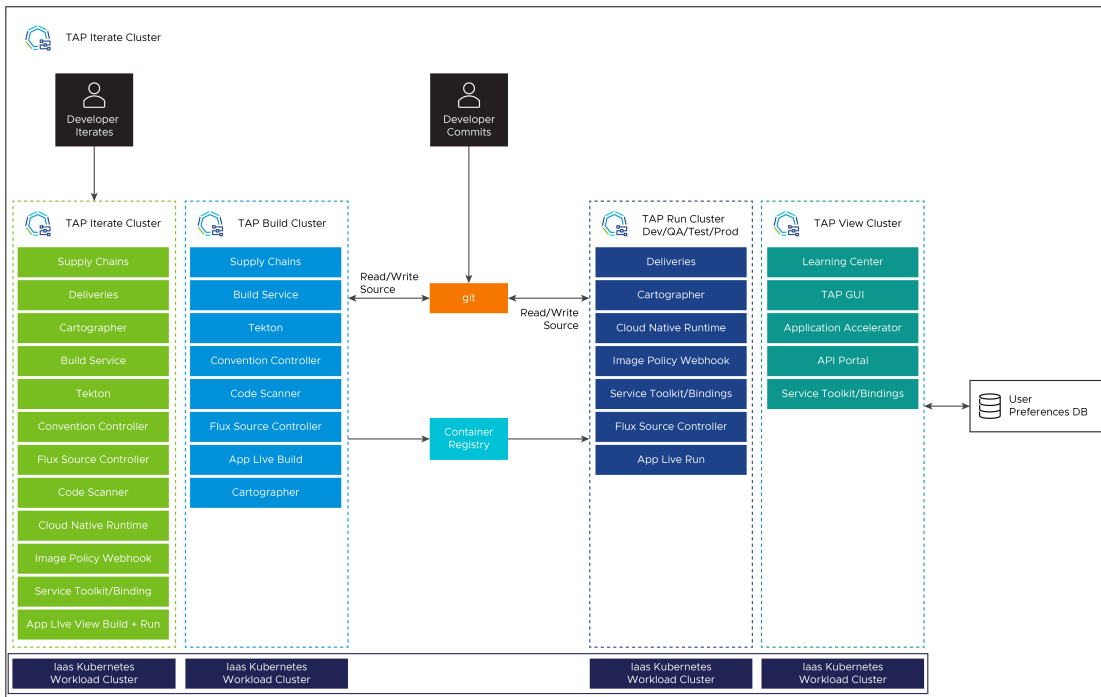
## Overview of multicluster Tanzu Application Platform

You can install Tanzu Application Platform (commonly known as TAP) in various topologies to reflect your existing landscape. VMware has tested and recommends a multicluster topology for production use. Because flexibility and choice are core to Tanzu Application Platform’s design, none of the implementation recommendations are set in stone.

The multicluster topology uses the [profile capabilities](#) supported by Tanzu Application Platform. Each cluster adopts one of following multicluster-aligned profiles:

- **Iterate:** Intended for inner-loop iterative application development.
- **Build:** Transforms source revisions to workload revisions; specifically, hosting workloads and supply chains.
- **Run:** Transforms workload revisions to running pods; specifically, hosting deliveries and deliverables.
- **View:** For applications related to centralized developer experiences; specifically, Tanzu Application Platform GUI and metadata store.

The following diagram illustrates this topology.



### Next steps

To get started with installing a multicluster topology, see [Install multicluster Tanzu Application Platform profiles](#).

### Install multicluster Tanzu Application Platform profiles

This topic tells you how to install a multicluster topology for your Tanzu Application Platform (commonly known as TAP).

### Prerequisites

Before installing multicluster Tanzu Application Platform profiles, you must meet the following prerequisites:

- All clusters must satisfy all the requirements to install Tanzu Application Platform. See [Prerequisites](#).
- [Accept Tanzu Application Platform EULA and install Tanzu CLI](#) with any required plug-ins.
- Install Tanzu Cluster Essentials on all clusters. For more information, see [Deploy Cluster Essentials](#).

## Multicluster Installation Order of Operations

The installation order is flexible given the ability to update the installation with a modified values file using the `tanzu package installed update` command. The following is an example of the order of operations to be used:

1. [Install View profile cluster](#).
2. [Install Build profile cluster](#).
3. [Install Run profile cluster](#).
4. [Install Iterate profile cluster](#).
5. [Add Build, Run and Iterate clusters to Tanzu Application Platform GUI](#).
6. Update the View cluster's installation values file with the previous information and run the following command to pass the updated config values to Tanzu Application Platform GUI:

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

Where `TAP-VERSION` is the Tanzu Application Platform version you've installed.

## Install View cluster

Install the View profile cluster first, because some components must exist before installing the Run clusters. For example, the Application Live View back end must be present before installing the Run clusters. For more information about profiles, see [About Tanzu Application Platform package profiles](#).

To install the View cluster:

1. Follow the steps described in [Installing the Tanzu Application Platform package and profiles](#) by using a reduced values file as shown in [View profile](#).
2. Verify that you can access Tanzu Application Platform GUI by using the ingress that you set up. The address must follow this format: `https://tap-gui.INGRESS-DOMAIN`, where `INGRESS-DOMAIN` is the DNS domain you set in `shared.ingress_domain` which points to the shared Contour installation in the `tanzu-system-ingress` namespace with the service `envoy`.
3. Deploy Supply Chain Security Tools (SCST) - Store. See [Multicluster setup](#) for more information.

## Install Build clusters

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

## Install Run clusters

To install the Run profile cluster:

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



#### Note

The default configuration of `shared.ingress_domain` points to the local Run cluster, rather than the View cluster, as a result, `shared.ingress_domain` must be set explicitly.

## Install Iterate clusters

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

## Add Build, Run and Iterate clusters to Tanzu 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 four profiles, you're ready to run a workload by using the supply chain. See [Get started with multicluster Tanzu Application Platform](#).

## Get started with multicluster Tanzu Application Platform

This topic tells you how to validate the implementation of a multicluster topology by taking a sample workload and passing it by using the supply chains on the Build and Run clusters.

Use this topic to build an application on the Build profile clusters and run the application on the Run profile clusters.

You can view the workload and associated objects from Tanzu Application Platform GUI (commonly known as TAP GUI) interface on the View profile cluster.

You can take various approaches to configuring the supply chain in this topology, but the following procedures validate the most basic capabilities.

## Prerequisites

Before implementing a multicluster topology, complete the following:

1. Complete all [installation steps for the four profiles](#): Build, Run, View and Iterate.
2. For the sample workload, VMware uses the same Application Accelerator - Tanzu Java Web App in the non-multicluster [Get Started](#) guide. You can download this accelerator to your own Git infrastructure of choice. You might need to configure additional permissions. Alternatively, you can also use the [application-accelerator-samples GitHub repository](#).
3. The two supply chains are `ootb-supply-chain-basic` on the Build/Iterate profile and `ootb-delivery-basic` on the Run profile. For the Build/Iterate and Run profiled clusters, perform the steps described in [Setup Developer Namespace](#). This guide assumes that you use the `default` namespace.
4. To set the value of `DEVELOPER_NAMESPACE` to the namespace you setup in the previous step, run:

```
export GIT_PROJECT_URL=GIT-URL-TO-PROJECT-REPO
export GIT_BRANCH=GIT-PROJECT-BRANCH
export DEVELOPER_NAMESPACE=YOUR-DEVELOPER-NAMESPACE
```

Where:

- `GIT-URL-TO-PROJECT-REPO` is the git URL of the repository you uploaded your source code to. For example, <https://github.com/my-org/repository.git>.
- `GIT-PROJECT-BRANCH` is the git branch of the project. For example, `main`.
- `YOUR-DEVELOPER-NAMESPACE` is the namespace you set up in [Set up developer namespaces to use your installed packages](#). `default` is used in this example.

## Start the workload on the Build profile cluster

The Build cluster starts by building the necessary bundle for the workload that is delivered to the Run cluster.

1. Use the Tanzu CLI to start the workload down the first supply chain:

```
tanzu apps workload create tanzu-java-web-app \
--git-repo ${GIT_PROJECT_URL} \
--git-branch ${GIT_BRANCH} \
--sub-path tanzu-java-web-app \
--git-branch main \
--type web \
--label app.kubernetes.io/part-of=tanzu-java-web-app \
--yes \
--namespace ${DEVELOPER_NAMESPACE}
```

2. To monitor the progress of this process, run:

```
tanzu apps workload tail tanzu-java-web-app --since 10m --timestamp --namespace
${DEVELOPER_NAMESPACE}
```

3. To exit the monitoring session, press **CTRL + C**.
4. Verify that your supply chain has produced the necessary `ConfigMap` containing `Deliverable` content produced by the `Workload`:

```
kubectl get configmap tanzu-java-web-app-deliverable --namespace ${DEVELOPER_NAMESPACE} -o go-template='{{.data.deliverable}}'
```

The output resembles the following:

```

apiVersion: carto.run/v1alpha1
kind: Deliverable
metadata:
  name: tanzu-java-web-app-deliverable
  labels:
    apis.apps.tanzu.vmware.com/register-api: "true"
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/workload-type: web
    app.kubernetes.io/component: deliverable
    app.tanzu.vmware.com/deliverable-type: web
spec:
  params:
    - name: gitops_ssh_secret
      value: ""
  source:
    git:
      url: http://git-server.default.svc.cluster.local/app-namespace/tanzu-java
      -web-app
      ref:
        branch: main

```

5. Store the `Deliverable` content, which you can take to the Run profile clusters from the `ConfigMap` by running:

```

kubectl get configmap tanzu-java-web-app-deliverable -n ${DEVELOPER_NAMESPACE}
-o go-template='{{.data.deliverable}}' > deliverable.yaml

```

6. (v1.4.0 only) Patch the `Deliverable` created on the Run profile cluster to add missing labels. See [known issues](#).

```

kubectl patch deliverable tanzu-java-web-app \
-n ${DEVELOPER_NAMESPACE} \
--type merge \
--patch '{"metadata":{"labels":{"carto.run/workload-name":"tanzu-java-we
b-app","carto.run/workload-namespace":"${DEVELOPER_NAMESPACE}"}}}'

```

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

```

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

```

8. Verify that this `Deliverable` is started and `Ready` by running:

```

kubectl get deliverables --namespace ${DEVELOPER_NAMESPACE}

```

The output resembles the following:

```

kubectl get deliverables --namespace default
NAME                               SOURCE
DELIVERY    READY    REASON    AGE
tanzu-java-web-app    tapmulticloud.azurecr.io/tap-multi-build-dev/tanzu-java-we
b-app-default-bundle:xxxx-xxxx-xxxx-xxxx-1a7beafd6389    delivery-basic    True
Ready    7m2s

```

9. To test the application, query the URL for the application. Look for the `httpProxy` by running:

```

kubectl get httpproxy --namespace ${DEVELOPER_NAMESPACE}

```

The output resembles the following:

```
kubectl get httpproxy --namespace default
NAME                                     FQDN
TLS SECRET   STATUS   STATUS DESCRIPTION
tanzu-java-web-app-contour-a98df54e3629c5ae9c82a395501ee1fdtanz  tanzu-java-we
b-app.default.svc.cluster.local          valid      Valid HTTP
roxy
tanzu-java-web-app-contour-e1d997a9ff9e7dfb6c22087e0ce6fd7ftanz  tanzu-java-we
b-app.default.apps.run.multi.kapplegate.com  valid      Valid HTTP
roxy
tanzu-java-web-app-contour-tanzu-java-web-app.default             tanzu-java-we
b-app.default          valid      Valid HTTP
roxy
tanzu-java-web-app-contour-tanzu-java-web-app.default.svc       tanzu-java-we
b-app.default.svc     valid      Valid HTTP
roxy
```

Select the URL that corresponds to the domain you specified in your Run cluster's profile and enter it into a browser. Expect to see the message "Greetings from Spring Boot + Tanzu!".

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

## Install Tanzu Application Platform Build profile

This topic tells you how to install Build profile cluster by using a reduced values file.

### Prerequisites

Before installing the Build profile, follow all the steps in [Install View cluster](#).

### Example values.yaml

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

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

shared:
  ingress_domain: "INGRESS-DOMAIN"
  kubernetes_distribution: "openshift" # To be passed only for Openshift. Defaults to
  "".
  kubernetes_version: "K8S-VERSION"
  image_registry:
    project_path: "SERVER-NAME/REPO-NAME" # To be used by Build Service by appending
"/buildservice" and used by Supply chain by appending "/workloads".
    username: "KP-DEFAULT-REPO-USERNAME"
    password: "KP-DEFAULT-REPO-PASSWORD"
  ca_cert_data: | # To be passed if using custom certificates.
  -----BEGIN CERTIFICATE-----
  MIIFXzCCA0egAwIBAgIJAJYm37SFocj1MA0GCSqGSIb3DQEEDQUAMEY...
  -----END CERTIFICATE-----

# The above shared keys can be overridden in the below section.

buildservice: # Optional if the corresponding shared keys are provided.
  kp_default_repository: "KP-DEFAULT-REPO"
  kp_default_repository_username: "KP-DEFAULT-REPO-USERNAME"
  kp_default_repository_password: "KP-DEFAULT-REPO-PASSWORD"
supply_chain: testing_scanning
otb_supply_chain_testing_scanning: # Optional if the corresponding shared keys are pr
```

```

provided.
  registry:
    server: "SERVER-NAME"
    repository: "REPO-NAME"
  gitops:
    ssh_secret: "SSH-SECRET-KEY" # (Optional) Defaults to "".
  grype:
    namespace: "MY-DEV-NAMESPACE" # (Optional) Defaults to default namespace.
    targetImagePullSecret: "TARGET-REGISTRY-CREDENTIALS-SECRET"
  metadataStore:
    url: METADATA-STORE-URL-ON-VIEW-CLUSTER
    caSecret:
      name: store-ca-cert
      importFromNamespace: metadata-store-secrets
    authSecret:
      name: store-auth-token
      importFromNamespace: metadata-store-secrets
  scanning:
    metadataStore:
      url: "" # Configuration is moved, so set this string to empty.
  tap_telemetry:
    customer_entitlement_account_number: "CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER" # (Optional) Identify data for creating Tanzu Application Platform usage reports.

```

#### Where:

- `K8S-VERSION` is the Kubernetes version used by your OpenShift cluster. It must be in the form of `1.23.x` or `1.24.x`, where `x` stands for the patch version. Examples:
  - Red Hat OpenShift Container Platform v4.10 uses the Kubernetes version `1.23.3`.
  - Red Hat OpenShift Container Platform v4.11 uses the Kubernetes version `1.24.1`.
- `KP-DEFAULT-REPO` is a writable repository in your registry. Tanzu Build Service dependencies are written to this location. Examples:
  - Harbor has the form `kp_default_repository: "my-harbor.io/my-project/build-service"`
  - Docker Hub has the form `kp_default_repository: "my-dockerhub-user/build-service"` or `kp_default_repository: "index.docker.io/my-user/build-service"`
  - Google Cloud Registry has the form `kp_default_repository: "gcr.io/my-project/build-service"`
- `KP-DEFAULT-REPO-USERNAME` is the user name that can write to `KP-DEFAULT-REPO`. You can `docker push` to this location with this credential.
  - For Google Cloud Registry, use `kp_default_repository_username: _json_key`
- `KP-DEFAULT-REPO-PASSWORD` is the password for the user that can write to `KP-DEFAULT-REPO`. You can `docker push` to this location with this credential. This credential can also be configured by using a Secret reference. For more information, see [Install Tanzu Build Service](#) for details.
  - For Google Cloud Registry, use the contents of the service account JSON file.
- `SERVER-NAME` is the host name of the registry server. Examples:
  - Harbor has the form `server: "my-harbor.io"`.
  - Docker Hub has the form `server: "index.docker.io"`.
  - Google Cloud Registry has the form `server: "gcr.io"`.
- `REPO-NAME` is where workload images are stored in the registry. Images are written to `SERVER-NAME/REPO-NAME/workload-name`. Examples:
  - Harbor has the form `repository: "my-project/supply-chain"`.



- Docker Hub has the form `repository: "my-dockerhub-user"`.
- Google Cloud Registry has the form `repository: "my-project/supply-chain"`.
- `SSH-SECRET-KEY` is the SSH secret key in the developer namespace for the supply chain to fetch source code from and push configuration to. See [Git authentication](#) for more information.
- `METADATA-STORE-URL-ON-VIEW-CLUSTER` is the URL of the Supply Chain Security Tools (SCST) - Store deployed on the View cluster. For example, <https://metadata-store.example.com>
- `MY-DEV-NAMESPACE` is the name of the developer namespace. SCST - Scan deploys the `ScanTemplates` there. This allows the scanning feature to run in this namespace.
- `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the Secret that contains the credentials to pull an image from the registry for scanning.
- `CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER` (optional) refers to the Entitlement Account Number (EAN), which is a unique identifier VMware assigns to its customers. Tanzu Application Platform telemetry uses this number to identify data that belongs to a particular customer and prepare usage reports. See the [Tanzu Kubernetes Grid documentation](#) for more information about identifying the Entitlement Account Number.

When you install Tanzu Application Platform, it is bootstrapped with the `lite` set of dependencies, including buildpacks and stacks, for application builds. For more information about buildpacks, see the [VMware Tanzu Buildpacks Documentation](#). You can find the buildpack and stack artifacts installed with Tanzu Application Platform on [Tanzu Network](#). You can update dependencies by [upgrading Tanzu Application Platform](#) to the latest patch, or by using an [automatic update process \(deprecated\)](#).

See [Multicluster setup](#) for more information about the value settings of `grype.metadataStore`.

You must set the `scanning.metadatastore.url` to an empty string if you're installing Grype Scanner v1.2.0 and later or Snyk Scanner to deactivate the embedded SCST - Store integration.

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

## Install Tanzu Application Platform Run profile

This topic tells you how to install Run profile cluster by using a reduced values file.

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

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

shared:
  ingress_domain: INGRESS-DOMAIN
  kubernetes_distribution: "openshift" # To be passed only for Openshift. Defaults to ""
  kubernetes_version: "K8S-VERSION"
  ca_cert_data: | # To be passed if using custom certificates.
    -----BEGIN CERTIFICATE-----
    MIIFXzCCA0egAwIBAgIJAJYm37SFocj1MA0GCSqGSIb3DQEBAQUAMEY...
    -----END CERTIFICATE-----
supply_chain: basic

contour:
  envoy:
    service:
```

```

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

appliveview_connector:
  backend:
    sslDeactivated: TRUE-OR-FALSE-VALUE
    ingressEnabled: true
    host: appliveview.VIEW-CLUSTER-INGRESS-DOMAIN

tap_telemetry:
  customer_entitlement_account_number: "CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER" # (Optional) Identify data for creating Tanzu Application Platform usage reports.

```

Where:

- `INGRESS-DOMAIN` is the subdomain for the host name that you point at the `tanzu-shared-ingress` service's external IP address.
- `K8S-VERSION` is the Kubernetes version used by your OpenShift cluster. It must be in the form of `1.23.x` or `1.24.x`, where `x` stands for the patch version. Examples:
  - Red Hat OpenShift Container Platform v4.10 uses the Kubernetes version `1.23.3`.
  - Red Hat OpenShift Container Platform v4.11 uses the Kubernetes version `1.24.1`.
- `VIEW-CLUSTER-INGRESS-DOMAIN` is the subdomain you setup on the View profile cluster. This matches the value key `appliveview.ingressDomain` or `shared.ingress_domain` on the view cluster. Include the default host name `appliveview.` ahead of the domain.
- `CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER` (optional) refers to the Entitlement Account Number (EAN), which is a unique identifier VMware assigns to its customers. Tanzu Application Platform telemetry uses this number to identify data that belongs to a particular customer and prepare usage reports. See the [Tanzu Kubernetes Grid documentation](#) for more information about identifying the Entitlement Account Number.

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

If you set `shared.ingress_domain` in run profile, the `appliveview_connector.backend.host` is automatically configured as `host: appliveview.INGRESS-DOMAIN`. To override the shared ingress for Application Live View to connect to the view cluster, set the `appliveview_connector.backend.host` key to `appliveview.VIEW-CLUSTER-INGRESS-DOMAIN`.

## Install Tanzu Application Platform View profile

This topic tells you how to install View profile cluster by using a reduced values file.

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

```

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

shared:
  ingress_domain: "INGRESS-DOMAIN"
  kubernetes_distribution: "openshift" # To be passed only for Openshift. Defaults to "".
  kubernetes_version: "K8S-VERSION"
  ca_cert_data: | # To be passed if using custom certificates.
    -----BEGIN CERTIFICATE-----
    MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...
    -----END CERTIFICATE-----

```

```

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

tap_gui:
  app_config:
    catalog:
      locations:
        - type: url
          target: https://GIT-CATALOG-URL/catalog-info.yaml
    kubernetes:
      serviceLocatorMethod:
        type: 'multiTenant'
      clusterLocatorMethods:
        - type: 'config'
      clusters:
        - url: CLUSTER-URL
          name: CLUSTER-NAME # Build profile cluster can go here.
          authProvider: serviceAccount
          serviceAccountToken: CLUSTER-TOKEN
          skipTLSVerify: TRUE-OR-FALSE-VALUE
        - url: CLUSTER-URL
          name: CLUSTER-NAME # Run profile cluster can go here.
          authProvider: serviceAccount
          serviceAccountToken: CLUSTER-TOKEN
          skipTLSVerify: TRUE-OR-FALSE-VALUE

appliveview:
  ingressEnabled: true

tap_telemetry:
  customer_entitlement_account_number: "CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER" # (Optional) Identify data for creating Tanzu Application Platform usage reports.

```

#### Where:

- `INGRESS-DOMAIN` is the subdomain for the host name that you point at the `tanzu-shared-ingress` service's external IP address.
- `K8S-VERSION` is the Kubernetes version used by your OpenShift cluster. It must be in the form of `1.23.x` or `1.24.x`, where `x` stands for the patch version. Examples:
  - Red Hat OpenShift Container Platform v4.10 uses the Kubernetes version `1.23.3`.
  - Red Hat OpenShift Container Platform v4.11 uses the Kubernetes version `1.24.1`.
- `GIT-CATALOG-URL` is the path to the `catalog-info.yaml` catalog definition file. You can download either a blank or populated catalog file from the [Tanzu Application Platform product page](#). Otherwise, use a Backstage-compliant catalog you've already built and posted on the Git infrastructure in the Integration section.
- `CLUSTER-URL`, `CLUSTER-NAME` and `CLUSTER-TOKEN` are described in the [View resources on multiple clusters in Tanzu Application Platform GUI](#). Observe the [order of operations](#) laid out in the previous steps.
- `CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER` (optional) refers to the Entitlement Account Number (EAN), which is a unique identifier VMware assigns to its customers. Tanzu Application Platform telemetry uses this number to identify data that belongs to a particular customer and prepare usage reports. See the [Tanzu Kubernetes Grid documentation](#) for more information about identifying the Entitlement Account Number.

If you use custom CA certificates, you must provide one or more PEM-encoded CA certificates under the `ca_cert_data` key. If you configured `shared.ca_cert_data`, Tanzu Application Platform

component packages inherit that value by default.

The `appliveview.ingressEnabled` key is set to `false` by default. In a multicluster setup, `ingressEnabled` key must be set to `true`. If the `shared.ingress_domain` key is set, the Application Live View back end is automatically exposed through the shared ingress.

## Install Tanzu Application Platform Iterate profile

This topic tells you how to install Iterate profile cluster by using a reduced values file.

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

```
profile: iterate

shared:
  ingress_domain: "INGRESS-DOMAIN"
  kubernetes_distribution: "openshift" # To be passed only for OpenShift. Defaults to ""
  kubernetes_version: "K8S-VERSION"
  image_registry:
    project_path: "SERVER-NAME/REPO-NAME" # To be used by Build Service by appending "/buildservice" and used by Supply chain by appending "/workloads"
    username: "KP-DEFAULT-REPO-USERNAME"
    password: "KP-DEFAULT-REPO-PASSWORD"
  ca_cert_data: | # To be passed if using custom certificates
  -----BEGIN CERTIFICATE-----
  MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...
  -----END CERTIFICATE-----

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

# The above shared keys may be overridden in the below section.

buildservice: # Optional if the corresponding shared keys are provided.
  kp_default_repository: "KP-DEFAULT-REPO"
  kp_default_repository_username: "KP-DEFAULT-REPO-USERNAME"
  kp_default_repository_password: "KP-DEFAULT-REPO-PASSWORD"

supply_chain: basic
ootb_supply_chain_basic: # Optional if the shared above mentioned shared keys are provided.
  registry:
    server: "SERVER-NAME"
    repository: "REPO-NAME"
  gitops:
    ssh_secret: "SSH-SECRET-KEY" # (Optional) Defaults to ""

image_policy_webhook:
  allow_unmatched_tags: true

contour:
  envoy:
    service:
      type: LoadBalancer # (Optional) Defaults to LoadBalancer.

cnrs:
  domain_name: "TAP-ITERATE-CNRS-DOMAIN" # Optional if the shared.ingress_domain is provided.

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

```
tap_telemetry:
  customer_entitlement_account_number: "CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER" # (Optional) Identify data for creating Tanzu Application Platform usage reports.
```

Where:

- `K8S-VERSION` is the Kubernetes version used by your OpenShift cluster. It must be in the form of `1.23.x` or `1.24.x`, where `x` stands for the patch version. Examples:
  - Red Hat OpenShift Container Platform v4.10 uses the Kubernetes version `1.23.3`.
  - Red Hat OpenShift Container Platform v4.11 uses the Kubernetes version `1.24.1`.
- `KP-DEFAULT-REPO` is a writable repository in your registry. Tanzu Build Service dependencies are written to this location. Examples:
  - Harbor has the form `kp_default_repository: "my-harbor.io/my-project/build-service"`.
  - Docker Hub has the form `kp_default_repository: "my-dockerhub-user/build-service"` or `kp_default_repository: "index.docker.io/my-user/build-service"`.
  - Google Cloud Registry has the form `kp_default_repository: "gcr.io/my-project/build-service"`.
- `KP-DEFAULT-REPO-USERNAME` is the user name that can write to `KP-DEFAULT-REPO`. You can `docker push` to this location with this credential.
  - For Google Cloud Registry, use `kp_default_repository_username: _json_key`.
- `KP-DEFAULT-REPO-PASSWORD` is the password for the user that can write to `KP-DEFAULT-REPO`. You can `docker push` to this location with this credential. This credential can also be configured by using a Secret reference. For more information, see [Install Tanzu Build Service](#) for details.
  - For Google Cloud Registry, use the contents of the service account JSON file.
- `SERVER-NAME` is the host name of the registry server. Examples:
  - Harbor has the form `server: "my-harbor.io"`.
  - Docker Hub has the form `server: "index.docker.io"`.
  - Google Cloud Registry has the form `server: "gcr.io"`.
- `REPO-NAME` is where workload images are stored in the registry. Images are written to `SERVER-NAME/REPO-NAME/workload-name`. Examples:
  - Harbor has the form `repository: "my-project/supply-chain"`.
  - Docker Hub has the form `repository: "my-dockerhub-user"`.
  - Google Cloud Registry has the form `repository: "my-project/supply-chain"`.
- `SSH-SECRET-KEY` is the SSH secret key in the developer namespace for the supply chain to fetch source code from and push configuration to. See [Git authentication](#) for more information.
- `TAP-ITERATE-CNRS-DOMAIN` is the iterate cluster CNRS domain.
- `VIEW-CLUSTER-INGRESS-DOMAIN` is the subdomain you setup on the View profile cluster. This matches the value key `appliveview.ingressDomain` or `shared.ingress_domain` on the view cluster. Include the default host name `appliveview.` ahead of the domain.
- `CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER` (optional) refers to the Entitlement Account Number (EAN), which is a unique identifier VMware assigns to its customers. Tanzu Application Platform telemetry uses this number to identify data that belongs to a particular

customers and prepare usage reports. See the [Tanzu Kubernetes Grid documentation](#) for more information about identifying the Entitlement Account Number.

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

If you set `shared.ingress_domain` in the iterate profile, the `appliveview_connector.backend.host` is automatically configured as `host: appliveview.INGRESS-DOMAIN`. To override the shared ingress for Application Live View to connect to the view cluster, set the `appliveview_connector.backend.host` key to `appliveview.VIEW-CLUSTER-INGRESS-DOMAIN`.

# Get started with Tanzu Application Platform

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

## Prerequisites

Before you start, verify you have successfully:

- **Installed Tanzu Application Platform**  
See [Installing Tanzu Application Platform](#).
- **Installed Tanzu Application Platform on the target Kubernetes cluster**  
See [Installing the Tanzu CLI](#) and [Installing the Tanzu Application Platform Package and Profiles](#).
- **Set the default kubeconfig context to the target Kubernetes cluster**  
See [Changing clusters](#).
- **Installed Out of The Box (OOTB) Supply Chain Basic**  
See [Install Out of The Box Supply Chain Basic](#). If you used the default profiles provided in [Installing the Tanzu Application Platform Package and Profiles](#), you have already installed the Out of The Box (OOTB) Supply Chain Basic.
- **Installed Tekton Pipelines**  
See [Install Tekton Pipelines](#). If you used the default profiles provided in [Installing the Tanzu Application Platform Package and Profiles](#), you have already installed Tekton Pipelines.
- **Set up a developer namespace to accommodate the developer workload**  
See [Set up developer namespaces to use your installed packages](#).
- **Installed Tanzu Application Platform GUI**  
See [Install Tanzu Application Platform GUI](#). If you used the Full or View profiles provided in [Installing the Tanzu Application Platform Package and Profiles](#), you have already installed Tanzu Application Platform GUI.
- **Installed the VS Code Tanzu Extension**  
See [Install the Visual Studio Code Tanzu Extension](#) for instructions.

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

## Next steps

For developers:

- [Deploy an app on Tanzu Application Platform](#)

For operators:

- [Create an application accelerator](#)

## Get started with Tanzu Application Platform

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

## Prerequisites

Before you start, verify you have successfully:

- **Installed Tanzu Application Platform**  
See [Installing Tanzu Application Platform](#).
- **Installed Tanzu Application Platform on the target Kubernetes cluster**  
See [Installing the Tanzu CLI](#) and [Installing the Tanzu Application Platform Package and Profiles](#).
- **Set the default kubeconfig context to the target Kubernetes cluster**  
See [Changing clusters](#).
- **Installed Out of The Box (OOTB) Supply Chain Basic**  
See [Install Out of The Box Supply Chain Basic](#). If you used the default profiles provided in [Installing the Tanzu Application Platform Package and Profiles](#), you have already installed the Out of The Box (OOTB) Supply Chain Basic.
- **Installed Tekton Pipelines**  
See [Install Tekton Pipelines](#). If you used the default profiles provided in [Installing the Tanzu Application Platform Package and Profiles](#), you have already installed Tekton Pipelines.
- **Set up a developer namespace to accommodate the developer workload**  
See [Set up developer namespaces to use your installed packages](#).
- **Installed Tanzu Application Platform GUI**  
See [Install Tanzu Application Platform GUI](#). If you used the Full or View profiles provided in [Installing the Tanzu Application Platform Package and Profiles](#), you have already installed Tanzu Application Platform GUI.
- **Installed the VS Code Tanzu Extension**  
See [Install the Visual Studio Code Tanzu Extension](#) for instructions.

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

## Next steps

For developers:

- [Deploy an app on Tanzu Application Platform](#)

For operators:

- [Create an application accelerator](#)

## Create an accelerator

This topic guides you through creating an accelerator and registering it in a Tanzu Application Platform instance.



### Note

This guide follows a quick start format. See the [Application Accelerator documentation](#) for advanced features.



## What you will do

- Create a new accelerator project that contains an `accelerator.yaml` file and `README.md` file.
- Configure the `accelerator.yaml` file to alter the project's `README.md`.
- Test your accelerator locally using the Tanzu CLI `generate-from-local` command.
- Create a new Git repository for the project and push the project to it.
- Register the accelerator in a Tanzu Application Platform instance.
- Verify project generation with the new accelerator by using Tanzu Application Platform GUI.

## Set up Visual Studio Code

1. To simplify accelerator authoring, code assist capabilities are available. To install the extension, navigate to the [Marketplace page for the YAML plug-in](#) and click **Install**.



### Note

Code assist for authoring accelerators is also available in the IntelliJ IDE. You can enable this by selecting **Application Accelerator** in the schema mapping drop-down menu. For more information about how to enable this, see the IntelliJ [Using schemas from JSON Schema Store](#) documentation.

2. After you install the plug-in, editing files entitled `accelerator.yaml` automatically uses the code assist capabilities.

## Create a simple project

To create your project, follow these instructions to set up the project directory, prepare the `README.md` and `accelerator.yaml`, and test your accelerator.

### Set up the project directory

1. Create a new directory for the project named `myProject` and change to the newly created directory.

```
mkdir myProject
cd myProject
```

2. Create two new files in the `myProject` directory named `README.md` and `accelerator.yaml`.

```
touch README.MD accelerator.yaml
```

### Prepare the `README.md` and `accelerator.yaml`

The following instructions require using Visual Studio Code to edit the files.

1. Using Visual Studio Code, open the `README.md`, copy and paste the following code block into it, and save the file. `CONFIGURABLE_PARAMETER_#` is targeted to be transformed during project generation in the upcoming `accelerator.yaml` definition.

```
## Tanzu Application Accelerator Sample Project
```

```
This is some very important placeholder text that should describe what this project can do and how to use it.

Here are some configurable parameters:

* CONFIGURABLE_PARAMETER_1
* CONFIGURABLE_PARAMETER_2
```

2. Open `accelerator.yaml` and begin populating the file section using the snippet below. This section contains important information, such as the accelerator's display name, description, tags, and more.

For all possible parameters available in this section, see [Creating accelerator.yaml](#).

```
accelerator:
  displayName: Simple Accelerator
  description: Contains just a README
  imageUrl: https://blogs.vmware.com/wp-content/uploads/2022/02/tap.png
  tags:
    - simple
    - getting-started
```

3. Add the configuration parameters using the following code snippet. This configures what parameters are displayed in the accelerator form during project creation.

In this example snippet, the field `firstConfigurableParameter` takes in text the user provides. The `secondConfigurableParameter` does the same, except it is only displayed if the user checks `secondConfigurableParameterCheckbox` because of the `dependsOn` parameter.

For more information about possible options, see [Creating accelerator.yaml](#).

```
# Place this after the 'tags' section from the previous step
options:
  - name: firstConfigurableParameter
    inputType: text
    label: The text used to replace the first placeholder text in the README.
    md. Converted to lowercase.
    defaultValue: Configurable Parameter 1
    required: true
  - name: secondConfigurableParameterCheckbox
    inputType: checkbox
    dataType: boolean
    label: Enable to configure the second configurable parameter, otherwise use the default value.
  - name: secondConfigurableParameter
    inputType: text
    label: The text used to replace the second placeholder text in the README.
    md. Converted to lowercase.
    defaultValue: Configurable Parameter 2
    dependsOn:
      name: secondConfigurableParameterCheckbox
```

4. Add the `engine` configuration by using the following code snippet and save the file.

The `engine` configuration tells the `accelerator engine` behind the scenes what must be done to the project files during project creation. In this example, this instructs the engine to replace `CONFIGURABLE_PARAMETER_1` and, if the check box is checked, `CONFIGURABLE_PARAMETER_2` with the parameters that the user passes in during project creation.

This also leverages [Spring Expression Language \(SpEL\)](#) syntax to convert the text input to all lowercase.

For more information about the possible parameters for use within the `engine` section, see [Creating accelerator.yaml](#).

```
# Place this after the `options` section from the previous step
engine:
  merge:
    - include: [ "README.md" ]
      chain:
        - type: ReplaceText
          substitutions:
            - text: "CONFIGURABLE_PARAMETER_1"
              with: "#firstConfigurableParameter.toLowerCase()"
        - condition: "#secondConfigurableParameterCheckbox"
          chain:
            - type: ReplaceText
              substitutions:
                - text: "CONFIGURABLE_PARAMETER_2"
                  with: "#secondConfigurableParameter.toLowerCase()"
```

## Test the accelerator

It is important to quickly test and iterate on accelerators as they are being developed to ensure that the resulting project is generated as expected.

1. Using the terminal of your choice with access to the `tanzu` command, run the following command to test the accelerator created earlier.

This step takes the local `accelerator.yaml` and project files, configures the project using the parameters passed in through the `--options` field, and outputs the project to a specified directory.



### Important

This step requires that the `TANZU-APPLICATION-ACCELERATOR-URL` endpoint is exposed and accessible. For more information, see [Server API connections for operators and developers](#).

```
tanzu accelerator generate-from-local \
  --accelerator-path simple-accelerator="$(pwd)" `# The path to new accelerator` \
  --server-url TANZU-APPLICATION-ACCELERATOR-URL `# Example: https://accelerator.mytapcluster.myorg.com` \
  --options '{"firstConfigurableParameter": "Parameter 1", "secondConfigurableParameterCheckbox": true, "secondConfigurableParameter": "Parameter 2"}' \
  -o "${HOME}/simple-accelerator/" `# Change this path to change where the project folder gets generated`
```

2. After the project is generated, a status message is displayed.

```
generated project simple-accelerator
```

3. Navigate to the output directory and verify that the `README.md` is updated based on the `--options` specified in the preceding `generate-from-local` command.

```
## Tanzu Application Accelerator Sample Project

This is some very important placeholder text that should describe what this project can do and how to use it.

Here are some configurable parameters:
```

```
- parameter 1
- parameter 2
```

## Upload the project to a Git repository

The Application Accelerator system and Tanzu Application Platform GUI depend on an accelerator project residing inside a Git repository. For this example, [GitHub](#) is used.

1. [Create a new repository in GitHub](#) and ensure that **Visibility** is set to **Public**. Click **Create Repository**.
2. To push your accelerator project (**not** the generated project from `generate-from-local`) to GitHub, follow the instructions that GitHub provides for the *...or create a new repository on the command line* that is shown after clicking **Create Repository**. Instructions can also be found in the [GitHub documentation](#).
3. Verify that the project is pushed to the target repository.

## Register the accelerator to the Tanzu Application Platform and verify project generation output

Now that the accelerator is committed to its own repository, you can register the accelerator to Tanzu Application Platform GUI for developers to generate projects from the newly created accelerator.

To do so, use the URL of the Git repository and branch name created earlier and run the following command using the Tanzu CLI to register the accelerator to Tanzu Application Platform GUI.



### Note

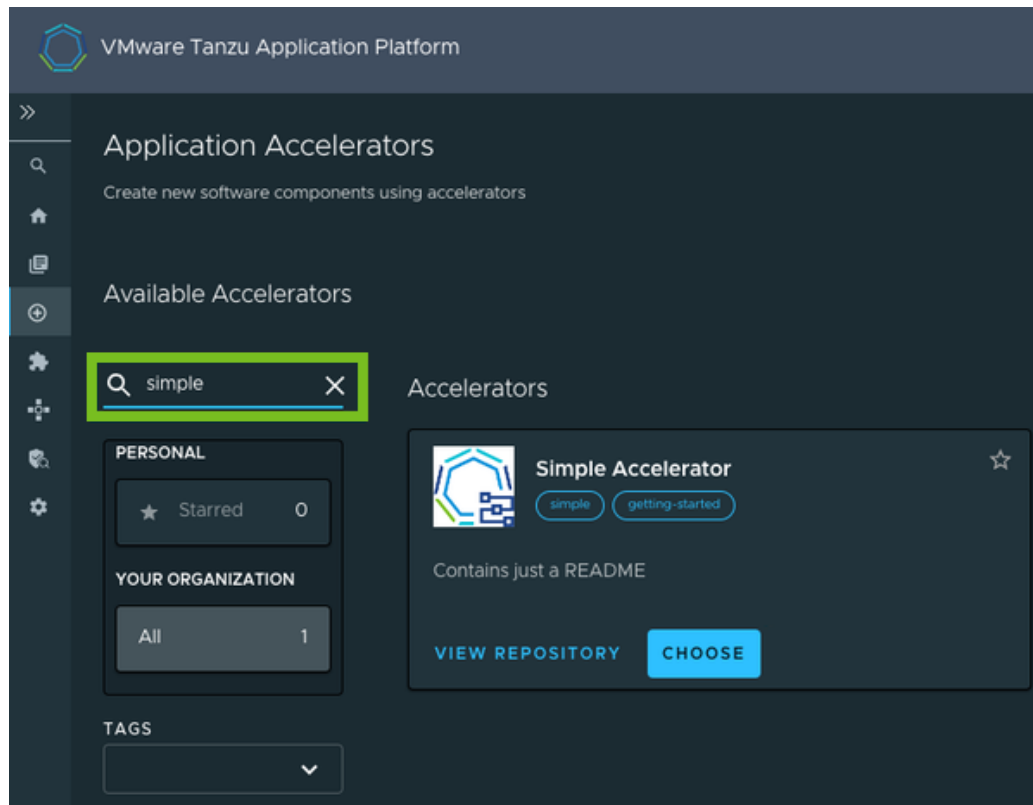
`tanzu accelerator create` works with monorepos as well. Add the `--git-sub-path` parameter with the desired subpath to fetch the accelerator project in that directory. For more information, see [tanzu accelerator create](#).

```
tanzu accelerator create simple-accelerator --git-repository https://github.com/myuser
name/myprojectrepository --git-branch main
```

The accelerator can take time to reconcile. After it has reconciled, it is available for use in Tanzu Application Platform GUI and the Application Accelerator extension for Visual Studio Code.

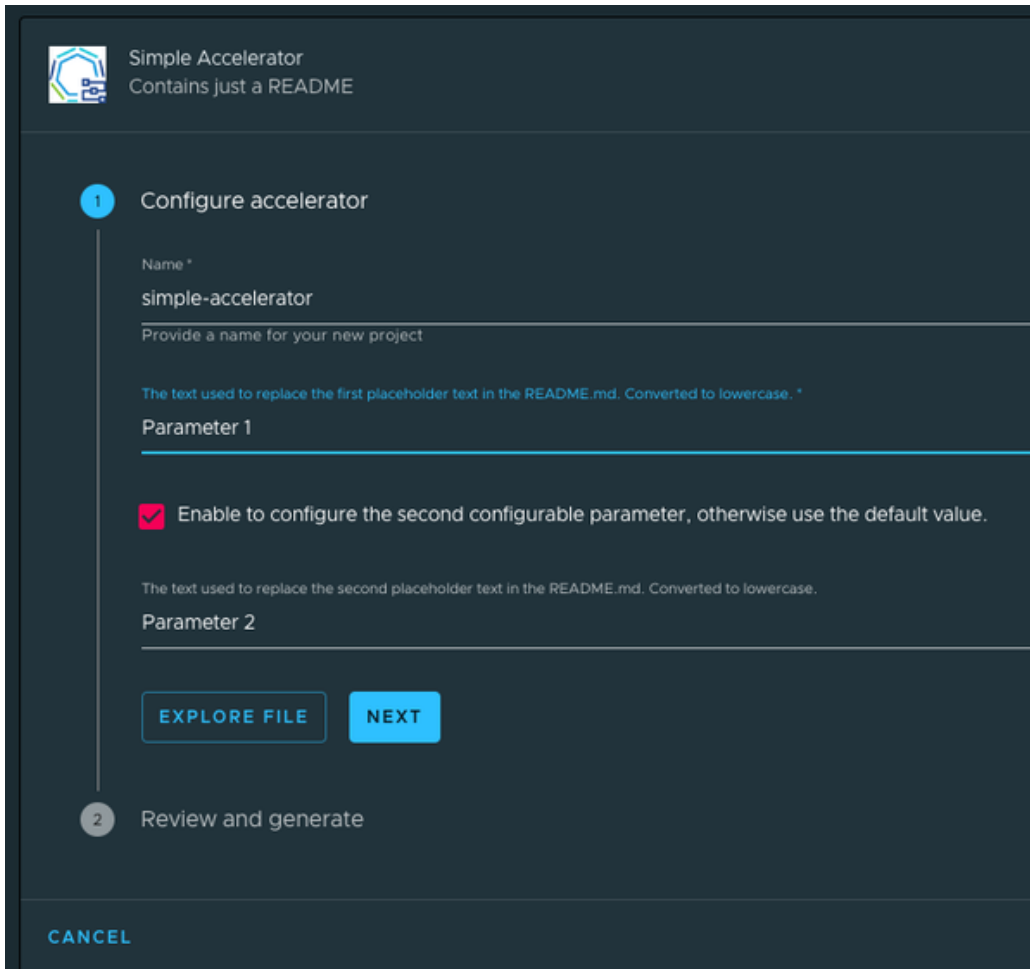
## Verify project generation output by using Tanzu Application Platform GUI

1. Navigate to your organization's instance of Tanzu Application Platform GUI.
2. On the left navigation pane, click **Create**.
3. Using the search bar near the left side of the page, search for `simple accelerator`. After you've found it, click **Choose** on the accelerator card.



4. Configure the project by filling in the parameters in the form.

The options you defined in `accelerator.yaml` are now displayed for you to configure. The `secondConfigurableParameter dependsOn secondConfigurableParameterCheckbox` might be hidden depending on whether the check box is selected.

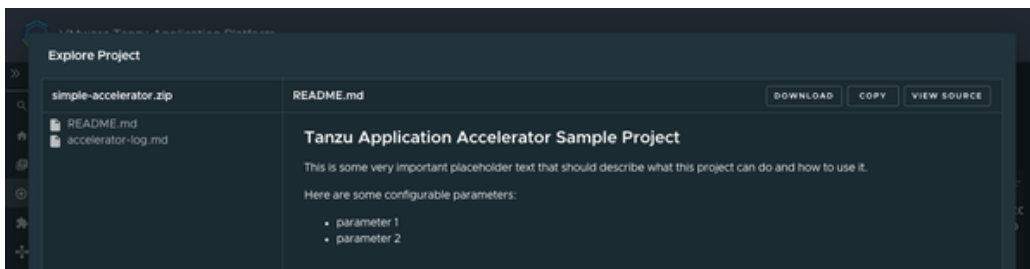


5. After configuration is complete, click **Next**.

**Note**

Depending on your organization’s Tanzu Application Platform configuration, you might be presented with an option to create a Git repository. In this guide, this is skipped and is covered in [Deploy an app on Tanzu Application Platform](#).

6. On the **Review and generate** step, review the parameters and click **Generate Accelerator**.
7. Explore the ZIP file of the configured project and verify that the project is generated with the parameters you provided during configuration.



## Learn more about Application Accelerator

- For advanced functionality when creating accelerators, such as accelerator best practices, accelerator fragments, engine transforms, and more, see the [Application Accelerator documentation](#).
- For more information about Application Accelerator configurations, see the [Configure Application Accelerator documentation](#).
- For information about installing the Application Accelerator extension for Visual Studio Code, see the [Application Accelerator Visual Studio Code extension documentation](#).
- For general accelerator troubleshooting, see [Troubleshooting Application Accelerator for VMware Tanzu](#).

## Create an accelerator

This topic guides you through creating an accelerator and registering it in a Tanzu Application Platform instance.



### Note

This guide follows a quick start format. See the [Application Accelerator documentation](#) for advanced features.

## What you will do

- Create a new accelerator project that contains an `accelerator.yaml` file and `README.md` file.
- Configure the `accelerator.yaml` file to alter the project's `README.md`.
- Test your accelerator locally using the Tanzu CLI `generate-from-local` command.
- Create a new Git repository for the project and push the project to it.
- Register the accelerator in a Tanzu Application Platform instance.
- Verify project generation with the new accelerator by using Tanzu Application Platform GUI.

## Set up Visual Studio Code

1. To simplify accelerator authoring, code assist capabilities are available. To install the extension, navigate to the [Marketplace page for the YAML plug-in](#) and click **Install**.



### Note

Code assist for authoring accelerators is also available in the IntelliJ IDE. You can enable this by selecting **Application Accelerator** in the schema mapping drop-down menu. For more information about how to enable this, see the [IntelliJ Using schemas from JSON Schema Store](#) documentation.

2. After you install the plug-in, editing files entitled `accelerator.yaml` automatically uses the code assist capabilities.

## Create a simple project

To create your project, follow these instructions to set up the project directory, prepare the `README.md` and `accelerator.yaml`, and test your accelerator.

## Set up the project directory

1. Create a new directory for the project named `myProject` and change to the newly created directory.

```
mkdir myProject
cd myProject
```

2. Create two new files in the `myProject` directory named `README.md` and `accelerator.yaml`.

```
touch README.MD accelerator.yaml
```

## Prepare the `README.md` and `accelerator.yaml`

The following instructions require using Visual Studio Code to edit the files.

1. Using Visual Studio Code, open the `README.md`, copy and paste the following code block into it, and save the file. `CONFIGURABLE_PARAMETER_#` is targeted to be transformed during project generation in the upcoming `accelerator.yaml` definition.

```
## Tanzu Application Accelerator Sample Project

This is some very important placeholder text that should describe what this project can do and how to use it.

Here are some configurable parameters:

* CONFIGURABLE_PARAMETER_1
* CONFIGURABLE_PARAMETER_2
```

2. Open `accelerator.yaml` and begin populating the file section using the snippet below. This section contains important information, such as the accelerator's display name, description, tags, and more.

For all possible parameters available in this section, see [Creating accelerator.yaml](#).

```
accelerator:
  displayName: Simple Accelerator
  description: Contains just a README
  iconUrl: https://blogs.vmware.com/wp-content/uploads/2022/02/tap.png
  tags:
    - simple
    - getting-started
```

3. Add the configuration parameters using the following code snippet. This configures what parameters are displayed in the accelerator form during project creation.

In this example snippet, the field `firstConfigurableParameter` takes in text the user provides. The `secondConfigurableParameter` does the same, except it is only displayed if the user checks `secondConfigurableParameterCheckbox` because of the `dependsOn` parameter.

For more information about possible options, see [Creating accelerator.yaml](#).

```
# Place this after the 'tags' section from the previous step
options:
  - name: firstConfigurableParameter
```



```

    inputType: text
    label: The text used to replace the first placeholder text in the README.
    md. Converted to lowercase.
    defaultValue: Configurable Parameter 1
    required: true
  - name: secondConfigurableParameterCheckbox
    inputType: checkbox
    dataType: boolean
    label: Enable to configure the second configurable parameter, otherwise use
    the default value.
  - name: secondConfigurableParameter
    inputType: text
    label: The text used to replace the second placeholder text in the README.
    md. Converted to lowercase.
    defaultValue: Configurable Parameter 2
    dependsOn:
      name: secondConfigurableParameterCheckbox

```

4. Add the `engine` configuration by using the following code snippet and save the file.

The `engine` configuration tells the `accelerator engine` behind the scenes what must be done to the project files during project creation. In this example, this instructs the engine to replace `CONFIGURABLE_PARAMETER_1` and, if the check box is checked, `CONFIGURABLE_PARAMETER_2` with the parameters that the user passes in during project creation.

This also leverages [Spring Expression Language \(SpEL\)](#) syntax to convert the text input to all lowercase.

For more information about the possible parameters for use within the `engine` section, see [Creating accelerator.yaml](#).

```

# Place this after the `options` section from the previous step
engine:
  merge:
    - include: [ "README.md" ]
    chain:
      - type: ReplaceText
        substitutions:
          - text: "CONFIGURABLE_PARAMETER_1"
            with: "#firstConfigurableParameter.toLowerCase()"
      - condition: "#secondConfigurableParameterCheckbox"
        chain:
          - type: ReplaceText
            substitutions:
              - text: "CONFIGURABLE_PARAMETER_2"
                with: "#secondConfigurableParameter.toLowerCase()"

```

## Test the accelerator

It is important to quickly test and iterate on accelerators as they are being developed to ensure that the resulting project is generated as expected.

1. Using the terminal of your choice with access to the `tanzu` command, run the following command to test the accelerator created earlier.

This step takes the local `accelerator.yaml` and project files, configures the project using the parameters passed in through the `--options` field, and outputs the project to a specified directory.



### Important

This step requires that the `TANZU-APPLICATION-ACCELERATOR-URL` endpoint is exposed and accessible. For more information, see [Server API connections for operators and developers](#).

```
tanzu accelerator generate-from-local \
  --accelerator-path simple-accelerator="$(pwd)" `# The path to new accelerator` \
  --server-url TANZU-APPLICATION-ACCELERATOR-URL `# Example: https://accelerator.mytapcluster.myorg.com` \
  --options '{"firstConfigurableParameter": "Parameter 1", "secondConfigurableParameterCheckbox": true, "secondConfigurableParameter": "Parameter 2"}' \
  -o "${HOME}/simple-accelerator/" `# Change this path to change where the project folder gets generated`
```

2. After the project is generated, a status message is displayed.

```
generated project simple-accelerator
```

3. Navigate to the output directory and verify that the `README.md` is updated based on the `--options` specified in the preceding `generate-from-local` command.

```
## Tanzu Application Accelerator Sample Project

This is some very important placeholder text that should describe what this project can do and how to use it.

Here are some configurable parameters:

- parameter 1
- parameter 2
```

## Upload the project to a Git repository

The Application Accelerator system and Tanzu Application Platform GUI depend on an accelerator project residing inside a Git repository. For this example, [GitHub](#) is used.

1. [Create a new repository in GitHub](#) and ensure that **Visibility** is set to **Public**. Click **Create Repository**.
2. To push your accelerator project (**not** the generated project from `generate-from-local`) to GitHub, follow the instructions that GitHub provides for the *...or create a new repository on the command line* that is shown after clicking **Create Repository**. Instructions can also be found in the [GitHub documentation](#).
3. Verify that the project is pushed to the target repository.

## Register the accelerator to the Tanzu Application Platform and verify project generation output

Now that the accelerator is committed to its own repository, you can register the accelerator to Tanzu Application Platform GUI for developers to generate projects from the newly created accelerator.

To do so, use the URL of the Git repository and branch name created earlier and run the following command using the Tanzu CLI to register the accelerator to Tanzu Application Platform GUI.



### Note

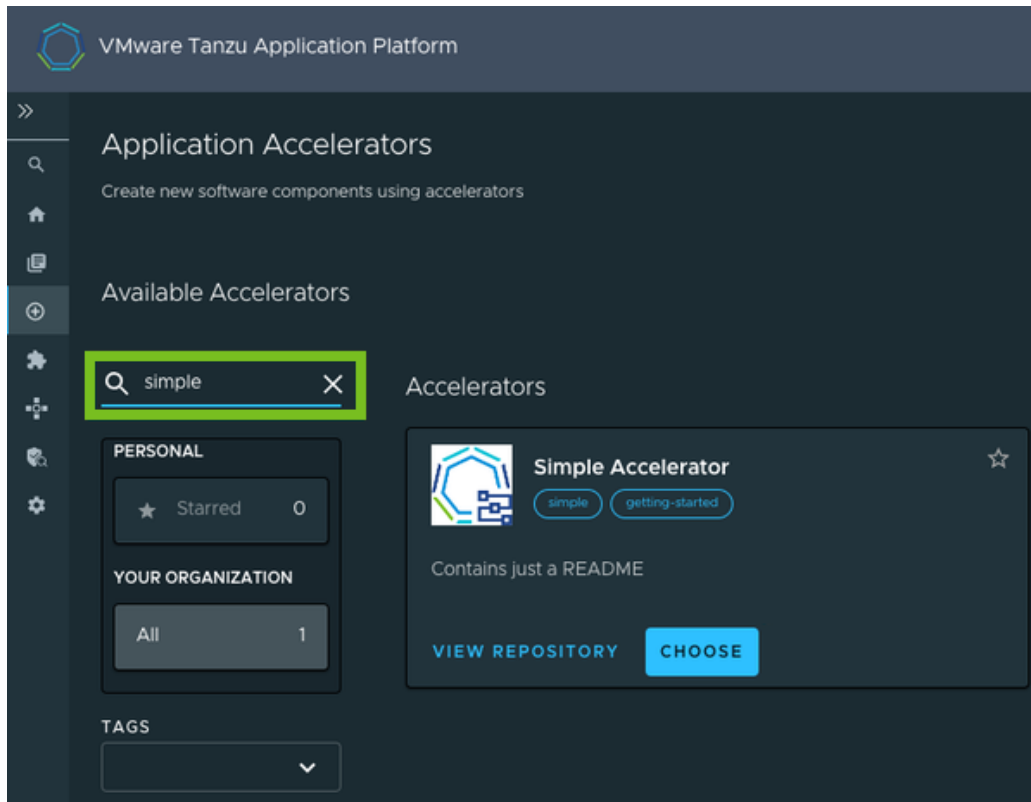
`tanzu accelerator create` works with monorepos as well. Add the `--git-sub-path` parameter with the desired subpath to fetch the accelerator project in that directory. For more information, see [tanzu accelerator create](#).

```
tanzu accelerator create simple-accelerator --git-repository https://github.com/myusername/myprojectrepository --git-branch main
```

The accelerator can take time to reconcile. After it has reconciled, it is available for use in Tanzu Application Platform GUI and the Application Accelerator extension for Visual Studio Code.

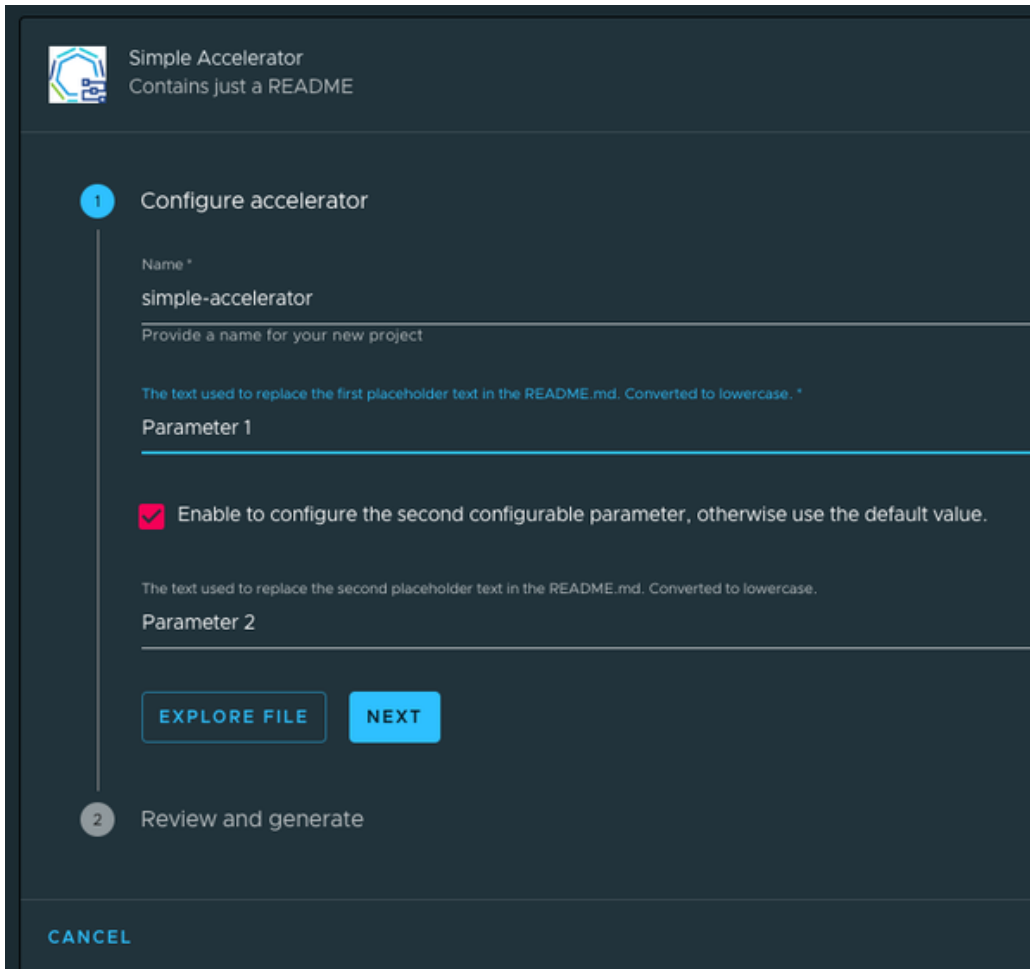
## Verify project generation output by using Tanzu Application Platform GUI

1. Navigate to your organization's instance of Tanzu Application Platform GUI.
2. On the left navigation pane, click **Create**.
3. Using the search bar near the left side of the page, search for `simple accelerator`. After you've found it, click **Choose** on the accelerator card.




4. Configure the project by filling in the parameters in the form.

The options you defined in `accelerator.yaml` are now displayed for you to configure. The `secondConfigurableParameter dependsOn secondConfigurableParameterCheckbox` might be hidden depending on whether the check box is selected.

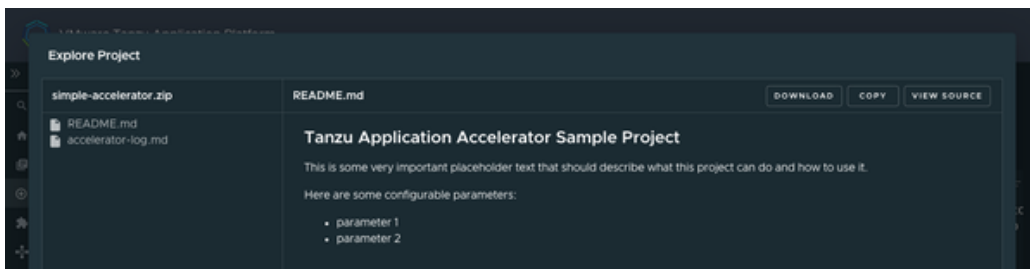


5. After configuration is complete, click **Next**.

 **Note**

Depending on your organization’s Tanzu Application Platform configuration, you might be presented with an option to create a Git repository. In this guide, this is skipped and is covered in [Deploy an app on Tanzu Application Platform](#).

6. On the **Review and generate** step, review the parameters and click **Generate Accelerator**.
7. Explore the ZIP file of the configured project and verify that the project is generated with the parameters you provided during configuration.



## Learn more about Application Accelerator

- For advanced functionality when creating accelerators, such as accelerator best practices, accelerator fragments, engine transforms, and more, see the [Application Accelerator documentation](#).
- For more information about Application Accelerator configurations, see the [Configure Application Accelerator documentation](#).
- For information about installing the Application Accelerator extension for Visual Studio Code, see the [Application Accelerator Visual Studio Code extension documentation](#).
- For general accelerator troubleshooting, see [Troubleshooting Application Accelerator for VMware Tanzu](#).

## Add testing and scanning to your application

This topic guides you through installing the optional OOTB Supply Chain with Testing and the optional OOTB Supply Chain with Testing and Scanning.

For more information about available supply chains, see [Supply chains on Tanzu Application Platform](#).

### What you will do

- Install OOTB Supply Chain with Testing.
- Add a Tekton pipeline to the cluster and update the workload to point to the pipeline and resolve errors.
- Install OOTB Supply Chain with Testing and Scanning.
- Update the workload to point to the Tekton pipeline and resolve errors.
- Query for vulnerabilities and dependencies.

### Overview

The default Out of the Box (OOTB) Supply Chain Basic and its dependencies were installed on your cluster during the Tanzu Application Platform install. As demonstrated in this guide, you can add testing and security scanning to your application. When you activate OOTB Supply Chain with Testing, it deactivates OOTB Supply Chain Basic.

The following installations also provide a sample Tekton pipeline that tests your sample application. The pipeline is configurable. Therefore, you can customize the steps to perform either additional testing or other tasks with Tekton Pipelines.

## Install OOTB Supply Chain with Testing

To install OOTB Supply Chain with Testing:

1. You can activate the OOTB Supply Chain with Testing by updating your profile to use `testing` rather than `basic` as the selected supply chain for workloads in this cluster. The `tap-values.yaml` is the file used to customize the profile in `Tanzu package install tap --values-file=...`. Update `tap-values.yaml` with the following changes:

```
- supply_chain: basic
+ supply_chain: testing

- ootb_supply_chain_basic:
+ ootb_supply_chain_testing:
  registry:
```

```
server: "SERVER-NAME"
repository: "REPO-NAME"
```

Where:

- `SERVER-NAME` is the name of your server.
- `REPO-NAME` is the name of the image repository that hosts the container images.

2. Update the installed profile by running:

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

Where `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.4.13`.

## Tekton pipeline config example

In this section, a Tekton pipeline is added to the cluster. In the next section, the workload is updated to point to the pipeline and resolve any current errors.



### Note

Developers can perform this step because they know how their application must be tested. The operator can also add the Tekton pipeline to a cluster before the developer gets access.

To add the Tekton pipeline to the cluster, apply the following YAML to the cluster:

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: developer-defined-tekton-pipeline
  namespace: DEVELOPER-NAMESPACE
  labels:
    apps.tanzu.vmware.com/pipeline: test      # (!) required
spec:
  params:
    - name: source-url                        # (!) required
    - name: source-revision                  # (!) required
  tasks:
    - name: test
      params:
        - name: source-url
          value: $(params.source-url)
        - name: source-revision
          value: $(params.source-revision)
      taskSpec:
        params:
          - name: source-url
          - name: source-revision
        steps:
          - name: test
            image: gradle
            script: |-
              cd `mktemp -d`

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

Where `DEVELOPER-NAMESPACE` is the name of your developer namespace.

The preceding YAML puts a Tekton pipeline in the developer namespace you specify. It defines the Tekton pipeline with a single step. The step contained in the `steps` pulls the code from the repository indicated in the developers `workload` and runs the tests within the repository. The steps of the Tekton pipeline are configurable and allow the developer to add additional items needed to test their code.

There are many steps in the supply chain. In this case, the next step is an image build. Any additional steps the developer adds to the Tekton pipeline to test their code are independent of the image being built and of any subsequent steps executed in the supply chain. This independence gives the developer freedom to focus on testing their code.

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

## Workload update

To connect the new supply chain to the workload, the workload must be updated to point at your Tekton pipeline.

1. Update the workload by running the following with the Tanzu CLI:

```
tanzu apps workload apply tanzu-java-web-app \
  --git-repo https://github.com/vmware-tanzu/application-accelerator-samples \
  --sub-path tanzu-java-web-app \
  --git-branch main \
  --type web \
  --label apps.tanzu.vmware.com/has-tests=true \
  --label app.kubernetes.io/part-of=tanzu-java-web-app \
  --yes \
  --namespace ${DEVELOPER_NAMESPACE}
```

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

```
kubectl get workload,gitrepository,pipelinerun,images.kpack,podintent,app,services.serving
```

The result is output similar to the following example that shows the objects created by Supply Chain Choreographer:

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

NAME                                         URL                                         AGE
READY   STATUS
gitrepository.source.toolkit.fluxcd.io/tanzu-java-web-app  https://github.com/vmware-tanzu/application-accelerator-samples  True   Fetched revision: main/872ff44c8866b7805fb2425130edb69a9853bdfd  109s

NAME                                         SUCCEEDED REASON   START
TIME   COMPLETIONTIME
pipelinerun.tekton.dev/tanzu-java-web-app-4ft1b  True       Succeeded  104s77s

NAME                                         LATESTIMAGE
READY
image.kpack.io/tanzu-java-web-app  10.188.0.3:5000/foo/tanzu-java-web-app@sha256:1d5bc4d3d1ffeb8629fbb721fcd1c4d28b896546e005f1efd98fbc4e79b7552c  True

NAME                                         READY   REASON
AGE
```

```

podintent.conventions.carto.run/tanzu-java-web-app  True          7s

NAME                                DESCRIPTION          SINCE-DEPLOY
AGE
app.kappctrl.k14s.io/tanzu-java-web-app  Reconcile succeeded  1s
2s

NAME                                URL
LATESTCREATED                      LATESTREADY          READY          REASON
service.serving.knative.dev/tanzu-java-web-app  http://tanzu-java-web-app.deve
loper.example.com  tanzu-java-web-app-00001  tanzu-java-web-app-00001  Unkno
wn  IngressNotConfigured

```

## Install OOTB Supply Chain with Testing and Scanning

### Prerequisites

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



#### Note

When leveraging both Tanzu Build Service and Grype in your Tanzu Application Platform supply chain, you can receive enhanced scanning coverage for the languages and frameworks with check marks in the column “Extended Scanning Coverage using Anchore Grype” on the [Language and Framework Support Table](#).

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

To install OOTB Supply Chain with Testing and Scanning:

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

```

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

```

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

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

A ScanPolicy is required and must be in the required namespace. A sample ScanPolicy is provided as follows to block a supply chain when CVEs with critical, high, and unknown ratings are found using `notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]`. You can also configure the supply chain to use your own custom policies and apply exceptions when you want to ignore certain CVEs. See [Out of the Box Supply Chain with Testing and Scanning](#). To apply the sample ScanPolicy, you can either add the namespace flag to the kubectl command or add the namespace text box to the template by running:



```
kubectl apply -f - -o yaml << EOF
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: scan-policy
  labels:
    'app.kubernetes.io/part-of': 'enable-in-gui'
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "UnknownSeverity"
    notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]
    ignoreCves := []

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

    isSafe(match) {
      severities := { e | e := match.ratings.rating.severity } | { e | e := match.ratings.rating[_].severity }
      some i
      fails := contains(notAllowedSeverities, severities[i])
      not fails
    }

    isSafe(match) {
      ignore := contains(ignoreCves, match.id)
      ignore
    }

    deny[msg] {
      comps := { e | e := input.bom.components.component } | { e | e := input.bom.components.component[_] }
      some i
      comp := comps[i]
      vulns := { e | e := comp.vulnerabilities.vulnerability } | { e | e := comp.vulnerabilities.vulnerability[_] }
      some j
      vuln := vulns[j]
      ratings := { e | e := vuln.ratings.rating.severity } | { e | e := vuln.ratings.rating[_].severity }
      not isSafe(vuln)
      msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
    }
EOF
```

- (optional) The Tanzu Application Platform profiles install the [Supply Chain Security Tools - Store](#) package by default. To persist and query the vulnerability results post-scan, confirm it is installed by running:

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

If the package is not installed, follow the installation instructions at [Install Supply Chain Security Tools - Store independent from Tanzu Application Platform profiles](#).

- 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, `1.4.13`.

## Workload update

To connect the new supply chain to the workload, update the workload to point to your Tekton pipeline:

#### 1. Update the workload by running the following using the Tanzu CLI:

```

tanzu apps workload apply tanzu-java-web-app \
--git-repo https://github.com/vmware-tanzu/application-accelerator-samples \
--sub-path tanzu-java-web-app \
--git-branch main \
--type web \
--label apps.tanzu.vmware.com/has-tests=true \
--label app.kubernetes.io/part-of=tanzu-java-web-app \
--yes \
--namespace ${DEVELOPER_NAMESPACE}

```

#### 2. After accepting the workload creation, view the new resources that the workload created by running:

```

kubectl get workload,gitrepository,sourcecan,pipelinerun,images.kpack,imagesca
n,podintert,app,services.serving

```

The following is an example output, which shows the objects that Supply Chain Choreographer created:

```

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

NAME                                URL
READY STATUS
gitrepository.source.toolkit.fluxcd.io/tanzu-java-web-app https://github.com/
vmware-tanzu/application-accelerator-samples True   Fetched revision: main/8
72ff44c8866b7805fb2425130edb69a9853bdfd 109s

NAME                                PHASE    SCAN
NEDREVISION                         SCANNEDREPOSITORY
AGE CRITICAL HIGH MEDIUM LOW UNKNOWN CVETOTAL
sourcecan.scanning.apps.tanzu.vmware.com/tanzu-java-web-app Completed 1878
50b39b754e425621340787932759a0838795 https://github.com/vmware-tanzu/applicat
ion-accelerator-samples 90s

NAME                                SUCCEEDED REASON    START
TIME COMPLETIONTIME
pipelinerun.tekton.dev/tanzu-java-web-app-4ftlb True       Succeeded 104s
77s

```

```

NAME                                LATESTIMAGE
READY
image.kpack.io/tanzu-java-web-app  10.188.0.3:5000/foo/tanzu-java-web-app@sha2
56:1d5bc4d3d1ffeb8629fbb721fcd1c4d28b896546e005f1efd98fbc4e79b7552c  True

NAME                                PHASE      SCANN
EDIMAGE
AGE  CRITICAL  HIGH  MEDIUM  LOW  UNKNOWN  CVETOTAL
imagescan.scanning.apps.tanzu.vmware.com/tanzu-java-web-app  Completed  10.18
8.0.3:5000/foo/tanzu-java-web-app@sha256:1d5bc4d3d1ffeb8629fbb721fcd1c4d28b8965
46e005f1efd98fbc4e79b7552c  14s

NAME                                READY      REASON
AGE
podintent.conventions.carto.run/tanzu-java-web-app  True      7s

NAME                                DESCRIPTION  SINCE-DEPLOY
AGE
app.kappctrl.k14s.io/tanzu-java-web-app  Reconcile succeeded  1s
2s

NAME                                URL
LATESTCREATED                      LATESTREADY  READY  REASON
service.serving.knative.dev/tanzu-java-web-app  http://tanzu-java-web-app.deve
loper.example.com  tanzu-java-web-app-00001  tanzu-java-web-app-00001  Unkno
wn  IngressNotConfigured

```



### Important

If the source or image scan has a “Failed” phase this means that the scan failed due to a scan policy violation and the supply chain stops. For information about the CVE triage workflow, see [Out of the Box Supply Chain with Testing and Scanning](#).

## Query for vulnerabilities

Scan reports are automatically saved to the [Supply Chain Security Tools - Store](#), and you can query them for vulnerabilities and dependencies. For example, related to open-source software (OSS) or third-party packages.

Query the tanzu-java-web-app image dependencies and vulnerabilities by running:

```

tanzu insight image get --digest DIGEST
tanzu insight image vulnerabilities --digest DIGEST

```

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

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

Congratulations! You have successfully added testing and security scanning to your application on the Tanzu Application Platform.

Take the next steps to learn about recommended supply chain security best practices and gain a powerful services journey experience on the Tanzu Application Platform by enabling several advanced use cases.

## Next steps

- [Configure image signing and verification in your supply chain](#)

## Configure image signing and verification in your supply chain

This topic guides you through configuring your Tanzu Application Platform supply chain to sign and verify your image builds.

### What you will do

- Configure your supply chain to sign your image builds.
- Configure an admission control policy to verify image signatures before admitting pods to the cluster.

### Configure your supply chain to sign and verify your image builds

1. Use Cosign to configure Tanzu Build Service to sign your container image builds. For instructions, see [Configure Tanzu Build Service to sign your image builds](#).
2. Create a `values.yaml` file, and install the Supply Chain Security Tools - Policy Controller. For instructions, see [Install Supply Chain Security Tools - Policy Controller](#).
3. Create a `ClusterImagePolicy` that passes Tanzu Application Platform images. It is planned for a future release for these to be signed and verifiable, but currently we recommend creating a policy to pass them:

For example:

```
kubectl apply -f - -o yaml << EOF
---
apiVersion: policy.sigstore.dev/v1beta1
kind: ClusterImagePolicy
metadata:
  name: image-policy-exceptions
spec:
  images:
  - glob: registry.example.org/myproject/*
  - glob: REPO-NAME*
  authorities:
  - static:
    action: pass
EOF
```

Where:

- `REPO-NAME` is the repository in your registry where Tanzu Build Service dependencies are stored. This is the exact same value configured in the `kp_default_repository` inside your `tap-values.yaml` or `tbs-values.yaml` files. Examples:
  - Harbor has the form `"my-harbor.io/my-project/build-service"`.
  - Docker Hub has the form `"my-dockerhub-user/build-service"` or `"index.docker.io/my-user/build-service"`.
  - Google Cloud Registry has the form `"gcr.io/my-project/build-service"`.
- Add any unsigned image that must run in your namespace to the previous policy. For example, if you add a Tekton pipeline that runs a Gradle image for testing, you need to add `glob: index.docker.io/library/gradle*` to `spec.images.glob` in the preceding code.

- o Replace `registry.example.org/myproject/*` with your target registry for your Tanzu Application Platform images. If you did not relocate the Tanzu Application Platform images to your own registry during installation, use `registry.tanzu.vmware.com/tanzu-application-platform/tap-packages*`.
4. Configure and apply a `ClusterImagePolicy` resource to the cluster to verify image signatures when deploying resources. For instructions, see [Create a ClusterImagePolicy resource](#).

For example:

```
kubectl apply -f - -o yaml << EOF
---
apiVersion: policy.sigstore.dev/v1beta1
kind: ClusterImagePolicy
metadata:
  name: example-policy
spec:
  images:
  - glob: registry.example.org/myproject/*
  authorities:
  - key:
    data: |
      -----BEGIN PUBLIC KEY-----
      <content ...>
      -----END PUBLIC KEY-----
EOF
```

5. Enable the policy controller verification in your namespace by adding the label `policy.sigstore.dev/include: "true"` to the namespace resource.

For example:

```
kubectl label namespace YOUR-NAMESPACE policy.sigstore.dev/include=true
```

Where `YOUR-NAMESPACE` is the name of your secure namespace.



#### Note

Supply Chain Security Tools - Policy Controller only validates resources in namespaces that have chosen to opt in.

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

## Next steps

- [Consume services on Tanzu Application Platform](#)

Or learn more about Supply Chain Security Tools:

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

## Set up services for consumption by developers

This topic for service operators and application operators guides you through setting up services for consumption by application developers. In this example, you set up the RabbitMQ Cluster Kubernetes operator, but the process is the same for any other services you want to set up.

You will learn about the `tanzu services` CLI plug-in and the most important APIs for working with services on Tanzu Application Platform.

## What you will do

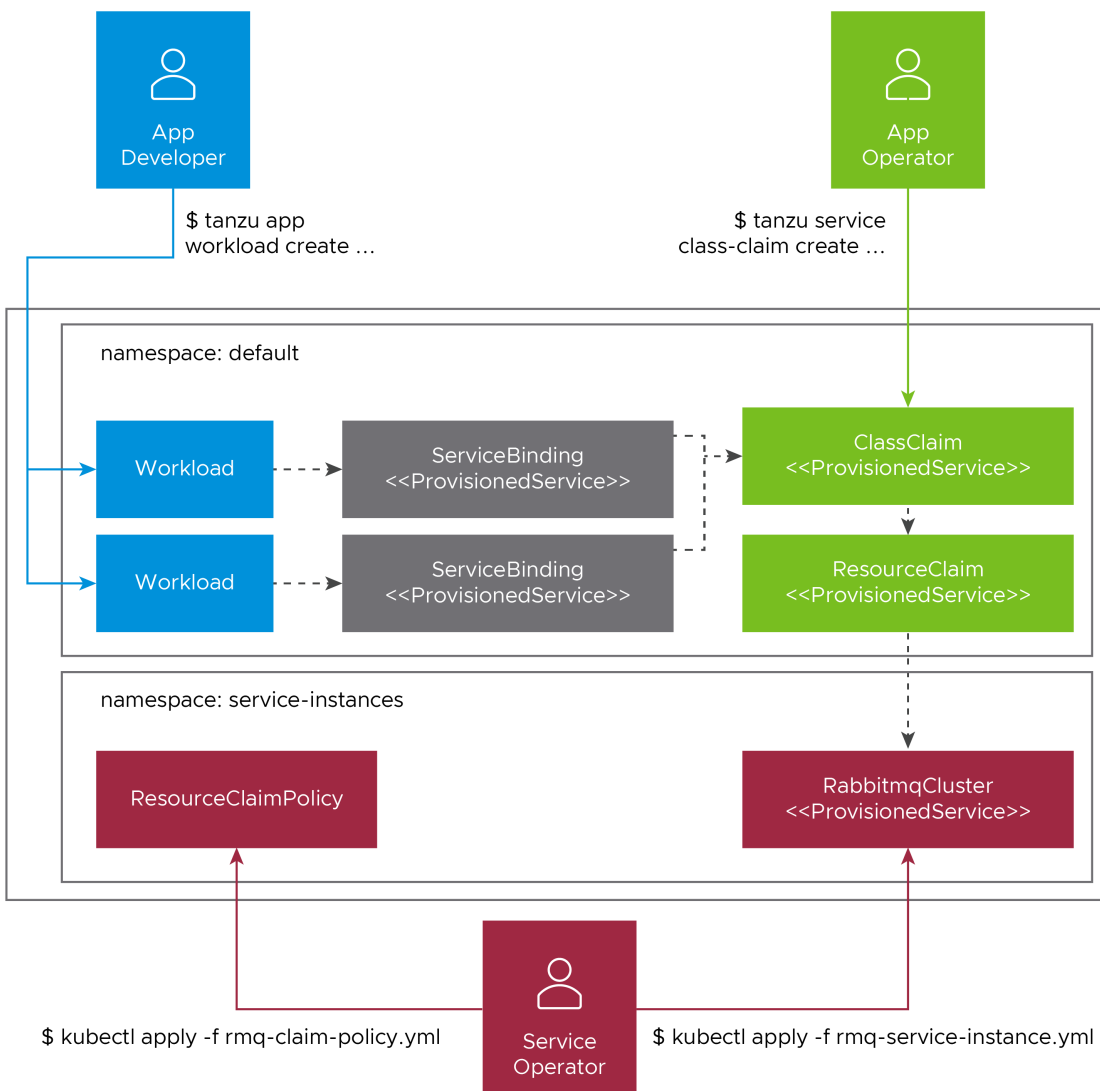
- Set up a service.
- Create a service instance.
- Claim a service instance.

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

Before you begin, for important background, see [Consume services on Tanzu Application Platform](#).

## Overview

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



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

1. There is a clear separation of concerns across the various user roles.
  - Application developers set the life cycle of workloads.
  - Application operators set the life cycle of claims.
  - Service operators set the life cycle of service instances.
  - The life cycle of service bindings is implicitly tied to the life cycle of workloads.
2. 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 class claim, to the resource claim, to the service binding, and ultimately to the application workload. For more information, see [ProvisionedService](#) on GitHub.
4. Exclusivity of claims: claims are considered to be mutually exclusive, meaning that claims for a service instance can only be fulfilled by at most one claim at a time.

## Prerequisites

Before following this walkthrough, you must:

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

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

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

- [VMware RabbitMQ for Kubernetes](#).
- [VMware SQL with Postgres for Kubernetes](#).
- [VMware SQL with MySQL for Kubernetes](#).

## Set up a service

This section covers the following:

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

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

To set up a service:

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

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

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

PostgreSQL: [Installing a Tanzu Postgres Operator](#)

MySQL: [Installing the Tanzu SQL for Kubernetes Operator](#)

2. Apply RBAC rules to grant Tanzu Application Platform permission to interact with the new API.
  1. In a file named `rmq-reader-for-binding-and-claims.yaml`, create a `ClusterRole` that defines the rules and label it so that the rules are aggregated to the appropriate controller:

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

2. Apply `rmq-reader-for-binding-and-claims.yaml` by running:

```
kubectl apply -f rmq-reader-for-binding-and-claims.yaml
```

PostgreSQL: [Creating Service Bindings](#)

MySQL: [Connecting an Application to a MySQL Instance](#)

3. Make the new API discoverable to application operators.
  1. In a file named `rmq-class.yaml`, create a `ClusterInstanceClass` that refers to the new service, and set any additional metadata. For example:

```
# rmq-class.yaml
---
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ClusterInstanceClass
metadata:
  name: rabbitmq
spec:
  description:
    short: It's a RabbitMQ cluster!
  pool:
    group: rabbitmq.com
    kind: RabbitmqCluster
# for Postgres
#   group: sql.tanzu.vmware.com
#   kind: Postgres
# for MySQL
#   group: with.sql.tanzu.vmware.com
#   kind: MySQL
```

2. Apply `rmq-class.yaml` by running:

```
kubectl apply -f rmq-class.yaml
```



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

## Create a service instance

This section covers:

- 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. There is no need to create a resource claim policy if the service instance resides in the same namespace as the application workload.

2. Create a `RabbitmqCluster` service instance.

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

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



### Note

If using Openshift, you might have to provide additional configuration for the `RabbitmqCluster`. For more details, see [Using the RabbitMQ Kubernetes Operators on Openshift in the RabbitMQ documentation](#).

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

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

PostgreSQL: [Deploying a Postgres Instance](#)

MySQL: [Creating a MySQL Instance](#)

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

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

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

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

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:

- Using `tanzu service class list` to view details about available classes from which instances can be claimed.
- Using `tanzu service class-claim create` to create a claim for the service instance.

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

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 to use with their application workloads without having to create and manage the services themselves. For more information, see [Resource Claims](#).

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

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

- `--class`

To claim a service instance:

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

```
tanzu service class list
```

Expected output:

```
tanzu service class list

NAME          DESCRIPTION
rabbitmq      It's a RabbitMQ cluster!
```

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

```
tanzu service class-claim create rmq-1 \
  --class rabbitmq
```

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

## Further use cases and reading

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

Use Case	Short Description
<a href="#">Introducing Different Service Implementations in Different Environments</a>	Using classes to have a claim resolve to a different backing service resource depending on which environment it is in. This removes the need for application operators to change `ClassClaim`s and `Workload`s as they are promoted through environments.
<a href="#">Consuming AWS RDS on Tanzu Application Platform</a>	Using the Controllers for Kubernetes (ACK) to provision an RDS instance and consume it from a Tanzu Application Platform workload. Involves making a third-party API consumable from Tanzu Application Platform.
<a href="#">Consuming AWS RDS on Tanzu Application Platform with Crossplane</a>	Using <a href="#">Crossplane</a> to provision an RDS instance and consume it from a Tanzu Application Platform workload. Involves making a third-party API consumable from Tanzu Application Platform.
<a href="#">Consuming Google Cloud SQL on Tanzu Application Platform with Config Connector</a>	Using GCP Config Connector to provision a Cloud SQL instance and consume it from a Tanzu Application Platform workload. Involves making a third-party API consumable from Tanzu Application Platform.
<a href="#">Consuming Google Cloud SQL on Tanzu Application Platform with Crossplane</a>	Using <a href="#">Crossplane</a> to provision a Cloud SQL instance and consume it from a Tanzu Application Platform workload. Involves making a third-party API consumable from Tanzu Application Platform.
<a href="#">Direct Secret References</a>	Binding to services running external to the cluster, for example, an in-house oracle database. Binding to services that do not conform with the binding specification.
<a href="#">Dedicated Service Clusters (Experimental)</a>	Separates application workloads from service instances across dedicated clusters.

For more information about the APIs and concepts underpinning Services on Tanzu Application Platform, see the [Services Toolkit Component documentation](#)

## Next steps

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

- [Multicluster Tanzu Application Platform](#)

## Deploy an app on Tanzu Application Platform

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



### Note

This walkthrough uses Tanzu Application Platform GUI. Alternatively, you can deploy your first application on Tanzu Application Platform using the [Application Accelerator Extension for VS Code](#).

## What you will do

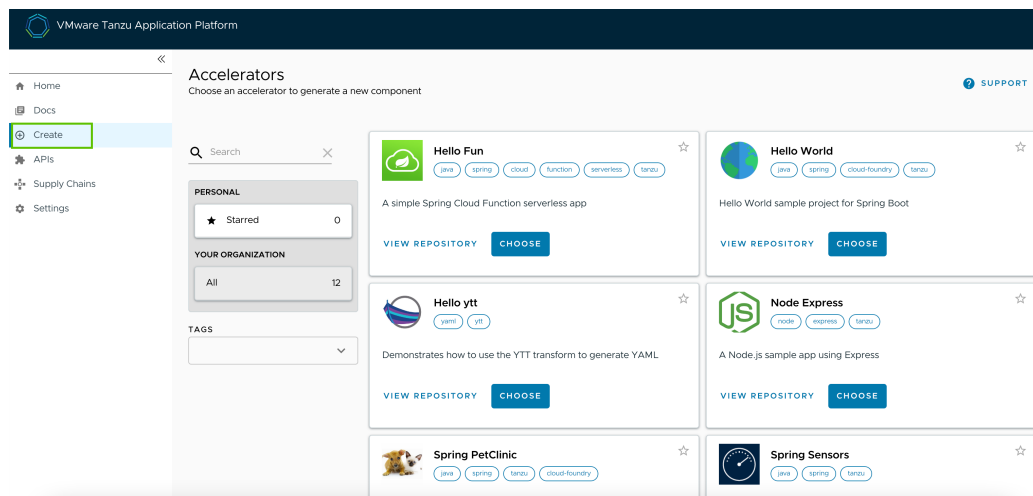
- Generate a project from an application accelerator.
- (Optional) Provision a new Git repository for the project.
- Upload it to your Git repository of choice.
- Deploy the app using the Tanzu CLI.
- View the build and runtime logs for your app.
- View the web app in your browser.
- (Optional) Add your application to Tanzu Application Platform GUI software catalog.

Before you start, complete all [Getting started prerequisites](#). For background on application accelerators, see [Application Accelerator](#).

## Generate a new project using an application accelerator

In this example, you use the [Tanzu-Java-Web-App](#) accelerator. You also use Tanzu Application Platform GUI. For information about connecting to Tanzu Application Platform GUI, see [Access Tanzu Application Platform GUI](#).

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



2. Locate the Tanzu Java Web App accelerator and click **CHOOSE**.
3. In the **Generate Accelerators** dialog box, replace the default value `dev.local` in the **prefix for container image registry** text box with the registry in the form of `SERVER-NAME/REPO-`

`NAME`. The `SERVER-NAME/REPO-NAME` must match what was specified for `registry` as part of the installation values for `ootb_supply_chain_basic`. See the Full Profile section on [Installing Tanzu Application Platform package and profiles](#). Click **NEXT**.

**Tanzu Java Web App**  
A sample Spring Boot web application built with Tanzu supply-chain

**1** Configure accelerator

Name\*  
tanzu-java-web-app  
Provide a name for your new project

Prefix for the container image repository\*  
dev.local

EXPLORE FILE    NEXT

**2** Git repository

**3** Review and generate

4. If your instance has optional Git repository support enabled, continue with the following sub-steps. If your instance *does not* support this, skip to step 5, “Verify the provided information.”
  1. Select `Create Git repo?`
  2. Select the `Host` Git repository provider from the drop-down menu. In this example, select `github.com`.
  3. Populate the `Owner` and `Repository` properties.

**2** Git repository

Create Git repo?

**Host**

github.com ▼

The host where the repository will be created

**Owner \***

my-org

The organization, user or project that this repo will belong to

**Repository \***

my-new-repo

The name of the repository

**Default Branch**

main

[BACK](#) [NEXT](#)

4. As you are populating the form, a dialog box appears asking for permission to provision Git repositories. Follow the prompts and continue.
5. Click **NEXT**.
5. Verify the provided information, and click **GENERATE ACCELERATOR**.
6. After the Task Activity processes complete, click **DOWNLOAD ZIP FILE**.
7. After downloading the ZIP file, expand it in a workspace directory. If you did not create a Git repository in the preceding steps, follow your preferred procedure for uploading the generated project files to a Git repository for your new project.

## Deploy your application through Tanzu Application Platform GUI

1. Set up environment variables by running:

```
export GIT_PROJECT_URL=GIT-URL-TO-PROJECT-REPO
export GIT_BRANCH=GIT-PROJECT-BRANCH
export DEVELOPER_NAMESPACE=YOUR-DEVELOPER-NAMESPACE
```

Where:

- o `GIT-URL-TO-PROJECT-REPO` is the git URL of the repository you uploaded your source code to earlier. e.g. `https://github.com/my-org/repository.git`
- o `GIT-PROJECT-BRANCH` is the git branch of the project. e.g. `main`

- o `YOUR-DEVELOPER-NAMESPACE` is the namespace configured earlier.
2. 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_PROJECT_URL} \
  --git-branch ${GIT_BRANCH} \
  --type web \
  --label apps.tanzu.vmware.com/has-tests=true \
  --label app.kubernetes.io/part-of=tanzu-java-web-app \
  --yes \
  --namespace ${DEVELOPER_NAMESPACE}
```

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



#### Note

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

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

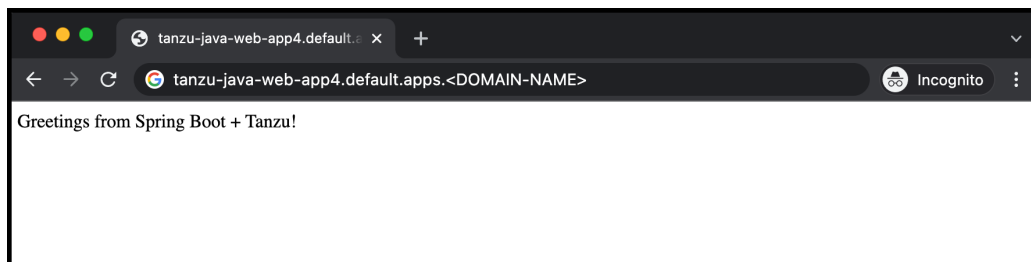
```
tanzu apps workload tail tanzu-java-web-app --since 10m --timestamp --namespace YOUR-DEVELOPER-NAMESPACE
```

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

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

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

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



## Add your application to Tanzu Application Platform GUI software catalog

1. Navigate to the home page of Tanzu Application Platform GUI and click **Home**, located on the left navigation pane. 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 text box. For example, `https://github.com/USERNAME/PROJECTNAME/blob/main/catalog-info.yaml`.

## Register an existing component

Start tracking your component in Tanzu Application Platform

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.

## Next steps

- [Iterate on your new app](#)

## Deploy an app on Tanzu Application Platform

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



### Note

This walkthrough uses Tanzu Application Platform GUI. Alternatively, you can deploy your first application on Tanzu Application Platform using the [Application](#)



## Accelerator Extension for VS Code.

## What you will do

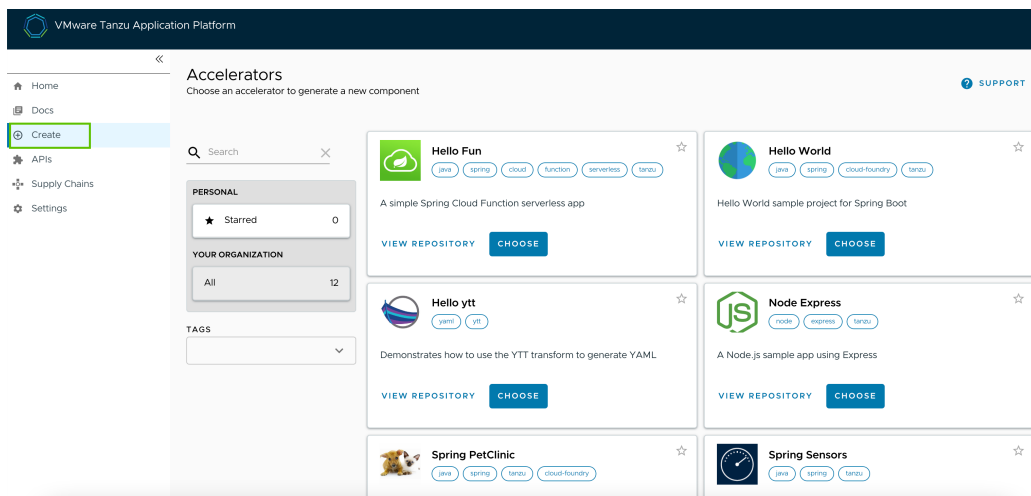
- Generate a project from an application accelerator.
- (Optional) Provision a new Git repository for the project.
- Upload it to your Git repository of choice.
- Deploy the app using the Tanzu CLI.
- View the build and runtime logs for your app.
- View the web app in your browser.
- (Optional) Add your application to Tanzu Application Platform GUI software catalog.

Before you start, complete all [Getting started prerequisites](#). For background on application accelerators, see [Application Accelerator](#).

## Generate a new project using an application accelerator

In this example, you use the [Tanzu-Java-Web-App](#) accelerator. You also use Tanzu Application Platform GUI. For information about connecting to Tanzu Application Platform GUI, see [Access Tanzu Application Platform GUI](#).

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



2. Locate the Tanzu Java Web App accelerator and click **CHOOSE**.
3. In the **Generate Accelerators** dialog box, replace the default value `dev.local` in the **prefix for container image registry** text box with the registry in the form of `SERVER-NAME/REPO-NAME`. The `SERVER-NAME/REPO-NAME` must match what was specified for `registry` as part of the installation values for `ootb_supply_chain_basic`. See the Full Profile section on [Installing Tanzu Application Platform package and profiles](#). Click **NEXT**.

**Tanzu Java Web App**  
A sample Spring Boot web application built with Tanzu supply-chain

**1** Configure accelerator

Name\*  
tanzu-java-web-app  
Provide a name for your new project

Prefix for the container image repository\*  
dev.local

EXPLORE FILE NEXT

**2** Git repository

**3** Review and generate

4. If your instance has optional Git repository support enabled, continue with the following sub-steps. If your instance *does not* support this, skip to step 5, “Verify the provided information.”
  1. Select `Create Git repo?`
  2. Select the `Host` Git repository provider from the drop-down menu. In this example, select `github.com`.
  3. Populate the `Owner` and `Repository` properties.

**2** Git repository

Create Git repo?

**Host**

github.com ▼

The host where the repository will be created

**Owner \***

my-org

The organization, user or project that this repo will belong to

**Repository \***

my-new-repo

The name of the repository

**Default Branch**

main

[BACK](#) [NEXT](#)

4. As you are populating the form, a dialog box appears asking for permission to provision Git repositories. Follow the prompts and continue.
5. Click **NEXT**.
5. Verify the provided information, and click **GENERATE ACCELERATOR**.
6. After the Task Activity processes complete, click **DOWNLOAD ZIP FILE**.
7. After downloading the ZIP file, expand it in a workspace directory. If you did not create a Git repository in the preceding steps, follow your preferred procedure for uploading the generated project files to a Git repository for your new project.

## Deploy your application through Tanzu Application Platform GUI

1. Set up environment variables by running:

```
export GIT_PROJECT_URL=GIT-URL-TO-PROJECT-REPO
export GIT_BRANCH=GIT-PROJECT-BRANCH
export DEVELOPER_NAMESPACE=YOUR-DEVELOPER-NAMESPACE
```

Where:

- o `GIT-URL-TO-PROJECT-REPO` is the git URL of the repository you uploaded your source code to earlier. e.g. `https://github.com/my-org/repository.git`
- o `GIT-PROJECT-BRANCH` is the git branch of the project. e.g. `main`

- o `YOUR-DEVELOPER-NAMESPACE` is the namespace configured earlier.
2. 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_PROJECT_URL} \
  --git-branch ${GIT_BRANCH} \
  --type web \
  --label apps.tanzu.vmware.com/has-tests=true \
  --label app.kubernetes.io/part-of=tanzu-java-web-app \
  --yes \
  --namespace ${DEVELOPER_NAMESPACE}
```

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



#### Note

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

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

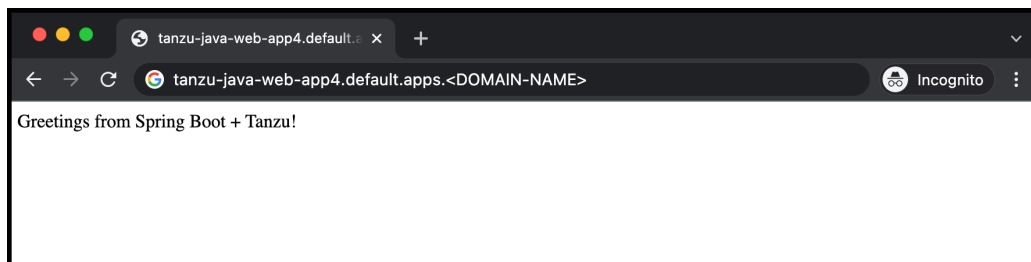
```
tanzu apps workload tail tanzu-java-web-app --since 10m --timestamp --namespace YOUR-DEVELOPER-NAMESPACE
```

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

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

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

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



## Add your application to Tanzu Application Platform GUI software catalog

1. Navigate to the home page of Tanzu Application Platform GUI and click **Home**, located on the left navigation pane. 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 text box. For example, `https://github.com/USERNAME/PROJECTNAME/blob/main/catalog-info.yaml`.

## Register an existing component

Start tracking your component in Tanzu Application Platform

1 Select URL

Repository URL \*

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

ANALYZE

2 Import Actions  
Optional

3 Review

4 Finish

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.

## Next steps

- [Iterate on your new app](#)

## Iterate on your new app

This topic guides you through starting to iterate on your first application on Tanzu Application Platform, which you deployed in the previous how-to, [Deploy your first application](#).

## What you will do

- Prepare your IDE to iterate on your application.
- Live update your application to view code changes updating live on the cluster.

- Debug your application.
- Monitor your running application on the Application Live View UI.

## Prepare your IDE to iterate on your application

In the previous Getting started how-to topic, [Deploy your first application](#), you deployed your first application on Tanzu Application Platform. Now that you have a skeleton workload developed, you are ready to begin to iterate on your new application and test code changes on the cluster.

Tanzu Developer Tools for Visual Studio Code is VMware Tanzu's official IDE extension for VS Code. It helps you develop and receive fast feedback on your workloads running on the Tanzu Application Platform.

The VS Code extension enables live updates of your application while running on the cluster and allows you to debug your application directly on the cluster. For information about installing the prerequisites and the Tanzu Developer Tools for VS Code extension, see [Install Tanzu Developer Tools for Visual Studio Code](#).



### Important

Use Tilt v0.30.12 or a later version for the sample application.

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



### Note

See the documentation for the registry you're using to find out which steps are necessary to authenticate and gain push access.

For example, if you use Docker, see the [Docker documentation](#), or if you use Harbor, see the [Harbor documentation](#).

You are now ready to iterate on your application.

## Live update your application

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

Follow the following steps to live update your application:

1. From the Command Palette (⇧⌘P), type in and select `Tanzu: Live Update Start`. You can view output from Tanzu Application Platform and from Tilt indicating that the container is being built and deployed.

- You see “Live Update starting...” in the status bar at the bottom right.
- Live update can take 1 to 3 minutes while the workload deploys and the Knative service becomes available.



#### Note

Depending on the type of cluster you use, you might see an error similar to the following:

```
ERROR: Stop! cluster-name might be production. If you're sure you
want to deploy there, add: allow_k8s_contexts('cluster-name') to
your Tiltfile. Otherwise, switch k8scontexts and restart Tilt. Follow
the instructions and add the line, allow_k8s_contexts('cluster-name') to
your Tiltfile.
```

2. When the Live Update status in the status bar is visible, resolve to “Live Update Started,” navigate to `http://localhost:8080` in your browser, and view your running application.
3. In the IDE, make a change to the source code. For example, in `HelloController.java`, edit the string returned to say `Hello!` and save.
4. The container is updated when the logs stop streaming. Navigate to your browser and refresh the page.
5. View the changes to your workload running on the cluster.
6. Either continue making changes, or stop and deactivate the live update when finished. Open the command palette (⌘⇧P), type **Tanzu**, and choose an option.

## Debug your application

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

Use the following steps to debug your cluster:

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

## Monitor your running application

Inspect the runtime characteristics of your running application using the Application Live View UI to monitor:

- Resource consumption
- Java Virtual Machine (JVM) status
- Incoming traffic
- Change log level

You can also troubleshoot environment variables and fine-tune the running application.

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

1. Confirm that the Application Live View components installed successfully. For instructions, see [Install Application Live View](#).
2. Access the Application Live View Tanzu Application Platform GUI. For instructions, see [Entry point to Application Live View plug-in](#).
3. Select your running application to view the diagnostic options and inside the application. For more information, see [Application Live View features](#).

## Next steps

- [Consume services on Tanzu Application Platform](#)

## Consume services on Tanzu Application Platform

This topic for application developers guides you through deploying two application workloads and configuring them to communicate using a service instance. The topic uses RabbitMQ as an example, but the process is the same regardless of the service you want to consume.

You will use the Tanzu Service CLI plug-in and will learn about classes, claims, and bindings.

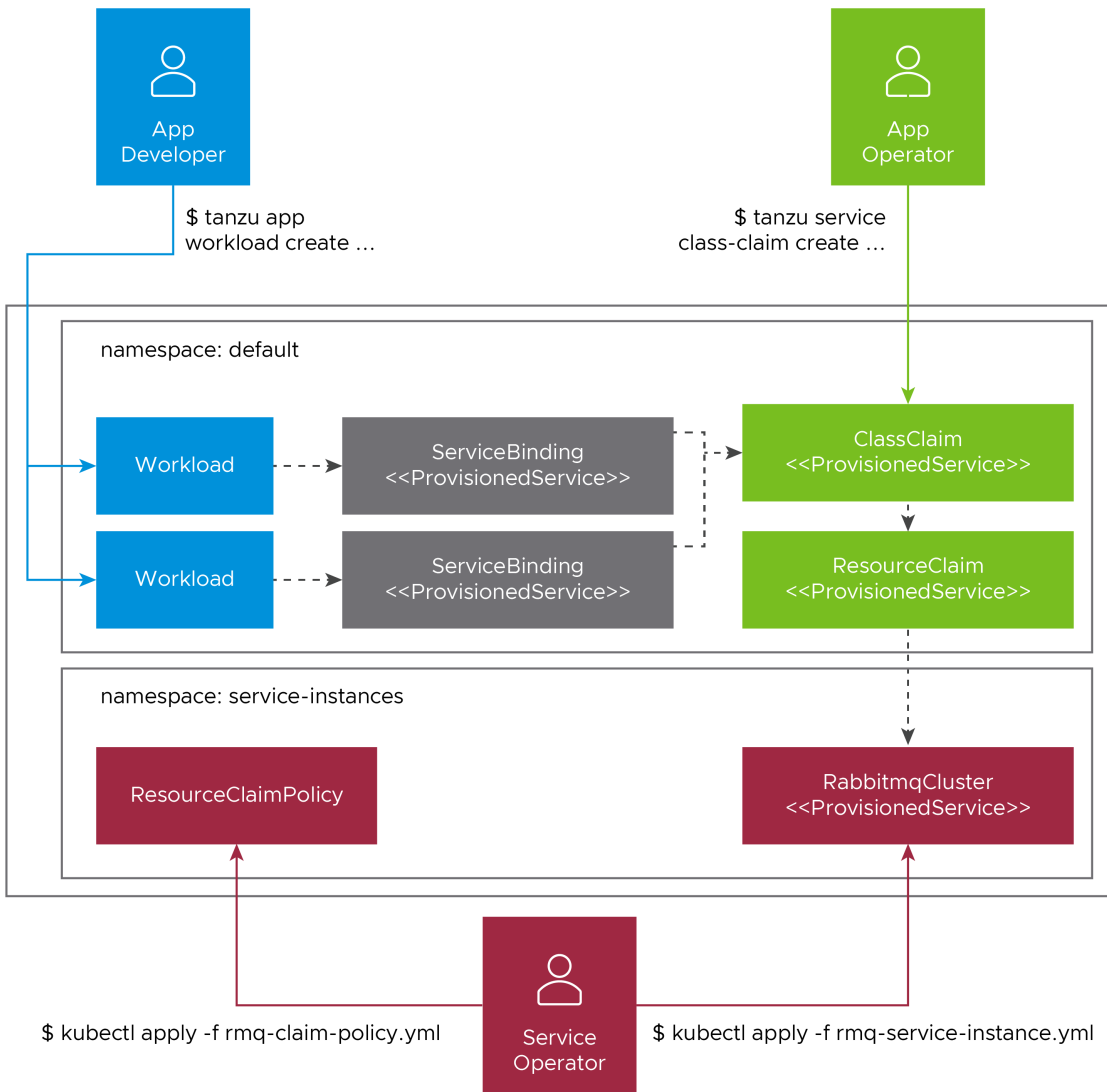
## What you will do

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

## Overview

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





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

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

## Prerequisites

Before following this walkthrough, as app developer you must:

1. Have access to a cluster with Tanzu Application Platform installed.
2. Have downloaded and installed the Tanzu CLI and the corresponding plug-ins.
3. Have set up the `default` namespace to use installed packages and use it as your developer namespace. For more information, see [Set up developer namespaces to use your installed](#)

[packages](#).

4. Ensure that your Tanzu Application Platform cluster can pull source code from GitHub.
5. Ensure that the service operator and application operator have completed:
  - o Setting up the service.
  - o Creating the service instance.
  - o Creating a claim for the service instance.

After you've completed these prerequisites, you are ready to inspect the claim created for the service instance by the application operator in [Set up services for consumption by developers](#) and use it to bind to application workloads.

## Bind an application workload to the service instance

This section covers:

- Using `tanzu service class-claim list` and `tanzu service class-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.

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

To create an application workload:

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

```
tanzu services class-claims list
```

Expected output:

NAME	CLASS	READY	REASON
rmq-1	rabbitmq	True	Ready

2. Retrieve detailed information about the claim by running:

```
tanzu services class-claims get rmq-1
```

Expected output:

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

- Record the value of [Claim Reference](#) from the previous command. This is the value to pass to `--service-ref` to create the application workload.
- Create the application workload by running:

```
tanzu apps workload create spring-sensors-consumer-web \
  --git-repo https://github.com/tanzu-end-to-end/spring-sensors \
  --git-branch rabbit \
  --type web \
  --label app.kubernetes.io/part-of=spring-sensors \
  --annotation autoscaling.knative.dev/minScale=1 \
  --service-ref="rmq=services.apps.tanzu.vmware.com/v1alpha1:ClassClaim: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:ClassClaim:rmq-1"
```

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

You are not passing a service ref to the [RabbitmqCluster](#) service instance directly, but rather to the claim that has claimed the [RabbitmqCluster](#) service instance. See the [consuming services diagram](#) at the beginning of this walkthrough.



#### Note

The deliverable produced eventually fails if the referenced resource in `--service-ref` consistently does not exist. This behavior is encoded in the OOTB supply chains through the use of the [OOTB templates](#). The [service-bindings](#) OOTB template can be used to replicate the same behavior in bespoke supply chains.

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

## Further use cases and reading

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

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

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

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

## Next steps

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

- [Multicluster Tanzu Application Platform](#)

## Deploy an air-gapped workload on Tanzu Application Platform

This topic for developers guides you through deploying your first workload on Tanzu Application Platform (commonly known as TAP) in an air-gapped environment.

For information about installing Tanzu Application Platform in an air-gapped environment, see [Install Tanzu Application Platform in an air-gapped environment](#).

## What you will do

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

## Create a workload from Git

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

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

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

2. (Optional) To pass in login credentials for a Git repository with the certificate authority (CA) certificate, create a file called `git-credentials.yaml`. For example:

```
apiVersion: v1
kind: Secret
```

```

metadata:
  name: git-ca
  # namespace: default
type: Opaque
stringData:
  username: USERNAME
  password: PASSWORD
  caFile: |
    CADATA

```

Where:

- o `USERNAME` is the user name.
- o `PASSWORD` is the password.
- o `CADATA` is the PEM-encoded CA certificate for the Git repository.

3. To pass in a custom `settings.xml` for Java, create a file called `settings-xml.yaml`. For example:

```

apiVersion: v1
kind: Secret
metadata:
  name: settings-xml
type: service.binding/maven
stringData:
  type: maven
  provider: sample
  settings.xml: |
    <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 https://maven.
apache.org/xsd/settings-1.0.0.xsd">
      <mirrors>
        <mirror>
          <id>reposilite</id>
          <name>Tanzu seal Internal Repo</name>
          <url>https://reposilite.tap-trust.cf-app.com/releases</url>
          <mirrorOf>*</mirrorOf>
        </mirror>
      </mirrors>
      <servers>
        <server>
          <id>reposilite</id>
          <username>USERNAME</username>
          <password>PASSWORD</password>
        </server>
      </servers>
    </settings>

```

4. Apply the file:

```
kubectl create -f settings-xml.yaml -n DEVELOPER-NAMESPACE
```

## Create a basic supply chain workload

Next, create your basic supply chain workload.

To pass the CA certificate in when you create the workload, run:

```
tanzu apps workload create APP-NAME --git-repo https://GITREPO --git-branch BRANCH --
type web --label app.kubernetes.io/part-of=CATALOGNAME --yes --param-yaml buildService
```

```
Bindings='[{"name": "settings-xml", "kind": "Secret"}]' --param "gitops_ssh_secret=git-ca"
```

## Create a testing supply chain workload

For instructions about creating a workload with the testing supply chain, see [Install OOTB Supply Chain with Testing](#).

To add the Tekton supply chain to the cluster, apply the following YAML to the cluster:

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: developer-defined-tekton-pipeline
  labels:
    apps.tanzu.vmware.com/pipeline: test      # (!) required
spec:
  params:
    - name: source-url                        # (!) required
    - name: source-revision                  # (!) required
  tasks:
    - name: test
      params:
        - name: source-url
          value: $(params.source-url)
        - name: source-revision
          value: $(params.source-revision)
      taskSpec:
        params:
          - name: source-url
          - name: source-revision
        steps:
          - name: test
            image: MY-REGISTRY/gradle
            script: |-
              cd `mktemp -d`
```

Where **MY-REGISTRY** is your container image registry. Relocate all the images given in the pipeline YAML to your private container registry.

Create the workload by running:

```
tanzu apps workload create APP-NAME --git-repo https://GITURL --git-branch BRANCH --t
ype web --label app.kubernetes.io/part-of=CATALOGNAME --yes --param-yaml --label apps.
tanzu.vmware.com/has-tests=true buildServiceBindings='[{"name": "settings-xml", "kin
d": "Secret"}]'
```

To instead pass the CA certificate when you create the workload, run:

```
tanzu apps workload create APP-NAME --git-repo https://GITREPO --git-branch BRANCH --
type web --label app.kubernetes.io/part-of=CATALOGNAME --yes --param-yaml --label app
s.tanzu.vmware.com/has-tests=true buildServiceBindings='[{"name": "settings-xml", "kin
d": "Secret"}]' --param "gitops_ssh_secret=git-ca"
```

## Create a testing scanning supply chain workload

For instructions about creating a workload with the testing and scanning supply chain, see [Install OOTB Supply Chain with Testing and Scanning](#).

In addition to the prerequisites given at [Prerequisites](#), follow [Using Grype in offline and air-gapped environments](#) before workload creation.

Create workload by running:

```
tanzu apps workload create APP-NAME --git-repo https://GITURL --git-branch BRANCH --type web --label app.kubernetes.io/part-of=CATALOGNAME --yes --param-yaml --label apps.tanzu.vmware.com/has-tests=true buildServiceBindings='[{"name": "settings-xml", "kind": "Secret"}]'
```

To instead pass the CA certificate when you create the workload, run:

```
tanzu apps workload create APP-NAME --git-repo https://GITREPO --git-branch BRANCH --type web --label app.kubernetes.io/part-of=CATALOGNAME --yes --param-yaml --label apps.tanzu.vmware.com/has-tests=true buildServiceBindings='[{"name": "settings-xml", "kind": "Secret"}]'
```

## Learn about Tanzu Application Platform

The topics in this section explain concepts important to getting started with Tanzu Application Platform.

In this section:

- [Application Accelerator](#)
- [Supply chains on Tanzu Application Platform](#)
- [Vulnerability scanning and metadata storage for your supply chain](#)
- [Consume services on Tanzu Application Platform](#)

## Application accelerators on Tanzu Application Platform

This topic describes the key concepts you need to know about application accelerators on Tanzu Application Platform (commonly known as TAP).

### What are application accelerators

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

Enterprise Architects use Application Accelerator to create application accelerators, which provide developers and admins in their organization with ready-made, enterprise-conforming code and configurations. Accelerators contain complete and runnable application code and deployment configurations. They also contain metadata for altering the code and deployment configurations based on input values provided for specific options defined in the accelerator metadata.

### Working with accelerators

The Application Accelerator plug-in for Tanzu Application Platform GUI helps you to discover accelerators and to enter extra information used for processing the files before downloading. As of Tanzu Application Platform v1.2, developers can also discover and work on accelerators right in Visual Studio Code with the Tanzu Application Accelerator for VS Code extension. Developers can use the `list`, `get`, and `generate` commands to use accelerators available in an Application Accelerator server.

Admins use the `create`, `update`, and `delete` commands for managing accelerators in a Kubernetes context. When admins want to use the `get` and `list` commands, they can specify the `--from-context` flag to access accelerators in a Kubernetes context.

## Next steps

Apply what you have learned:

Developers:

- [Deploy an app on Tanzu Application Platform](#)

Operators:

- [Create an application accelerator](#)

## Supply chains on Tanzu Application Platform

This topic describes the key concepts you need to know about supply chains and Continuous Integration/Continuous Delivery (CI/CD) on Tanzu Application Platform (commonly known as TAP).

### What are supply chains

Supply chains provide a way of codifying all of the steps of your path to production, more commonly known as CI/CD. CI/CD is a method to frequently deliver applications by introducing automation into the stages of application development. The main concepts attributed to CI/CD are continuous integration, continuous delivery, and continuous deployment.

CI/CD is the method used by supply chains to deliver applications through automation. Tanzu Application Platform supply chains allow you to use CI/CD and add any other steps necessary for an application to reach production or a different environment, such as staging.



### A path to production

A path to production allows you to create a unified access point for all of the tools required for your applications to reach a customer-facing environment. Instead of having four tools that are loosely coupled to each other, a path to production defines all four tools in a single, unified layer of abstraction. The path to production can be automated and repeatable between teams for applications at scale.

Typically tools cannot integrate with one another without scripting or webhooks. Whereas with a path to production, there is a unified automation tool to codify all the interactions between each of the tools. Supply chains that are used to codify the path to production for an organization are configurable. This allows their authors to add all of the steps of the path to production for their applications.

### Available supply chains

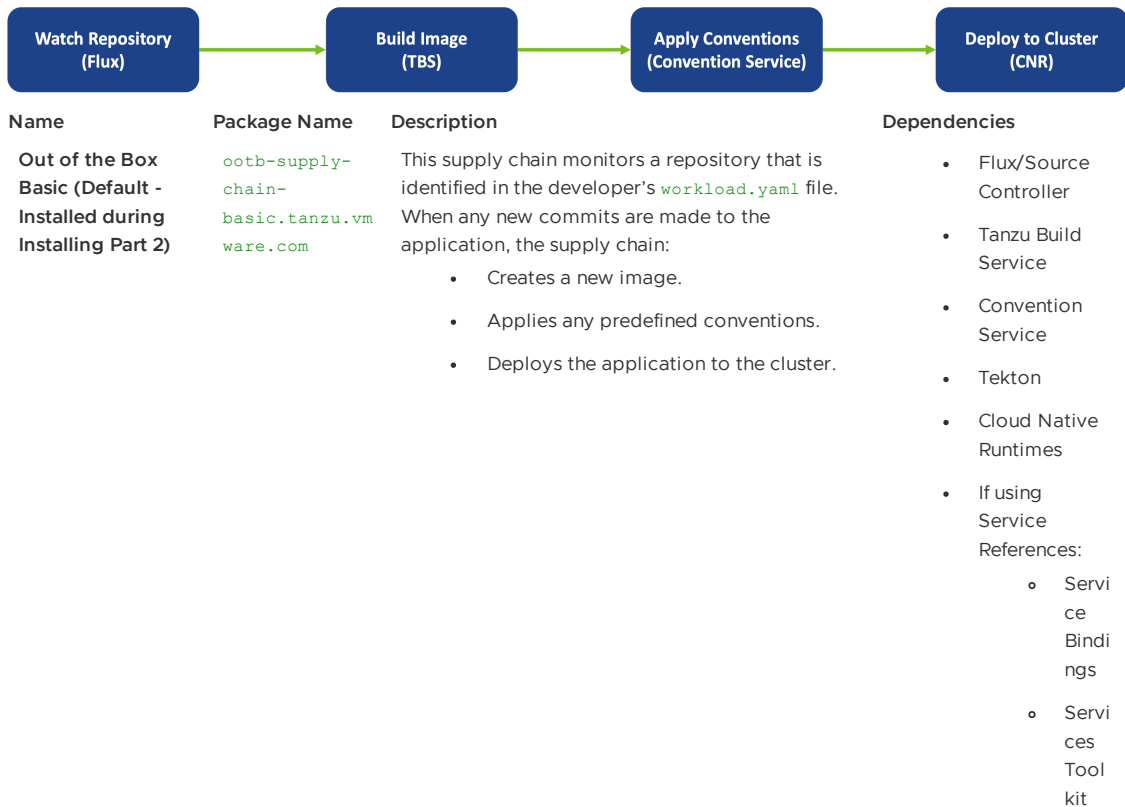
Tanzu Application Platform provides three out of the box (OOTB) supply chains to work with the Tanzu Application Platform components. They include:

- [OOTB Supply Chain Basic](#) (default)
- [OOTB Supply Chain with Testing](#) (optional)
- [OOTB Supply Chain with Testing+Scanning](#) (optional)

#### 1: OOTB Basic (default)

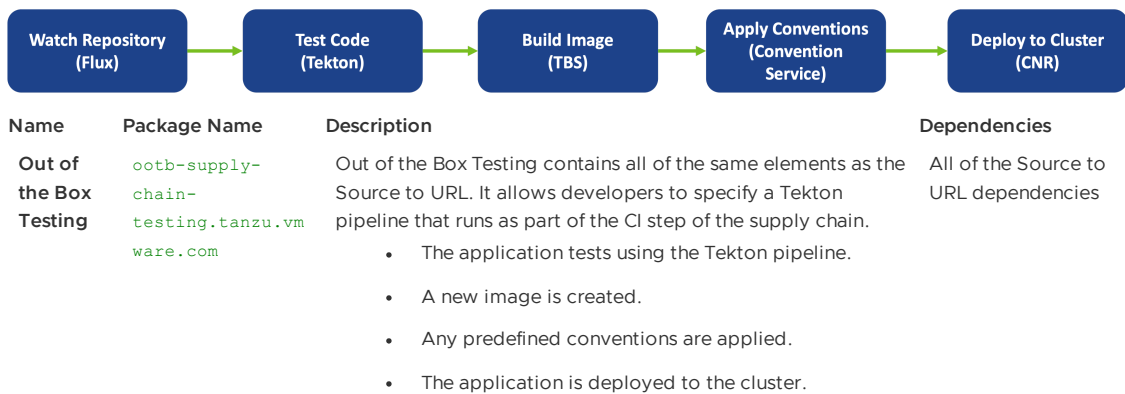


The default **OOTB Basic** supply chain and its dependencies were installed on your cluster during the Tanzu Application Platform install. The following diagram and table provide a description of the supply chain and dependencies provided with Tanzu Application Platform.



## 2: OOTB Testing

**OOTB Testing** supply chain runs a Tekton pipeline within the supply chain. The following diagram and table provide a description of the supply chain and dependencies provided with Tanzu Application Platform.



## 3: OOTB Testing+Scanning

**OOTB Testing+Scanning** supply chain includes integrations for secure scanning tools. The following diagram and table provide a description of the supply chain and dependencies provided with Tanzu Application Platform.



Name	Package Name	Description	Dependencies
<b>Out of the Box Testing and Scanning</b>	<code>ootb-supply-chain-testing-scanning.tanzu.vmware.com</code>	<p>Out of the Box Testing and Scanning contains all of the same elements as the Out of the Box Testing supply chain, and it also includes integrations with the secure scanning components of Tanzu Application Platform.</p> <ul style="list-style-type: none"> <li>The application is tested using the provided Tekton pipeline.</li> <li>The application source code is scanned for vulnerabilities.</li> <li>A new image is created.</li> <li>The image is scanned for vulnerabilities.</li> <li>Any predefined conventions are applied.</li> <li>The application deploys to the cluster.</li> </ul>	<p>All of the Source to URL dependencies, and:</p> <ul style="list-style-type: none"> <li>The secure scanning components included with Tanzu Application Platform</li> </ul>

## Next steps

Apply what you have learned:

- [Add testing and scanning to your application](#)

Or learn about:

- [Vulnerability scanning and metadata storage for your supply chain](#)

## Vulnerability scanning, storing, and viewing for your supply chain

This topic describes the vulnerability scanning features you can use with Tanzu Application Platform (commonly known as TAP).

This feature set allows an application operator to introduce source code and image vulnerability scanning, storing, and viewing to their Tanzu Application Platform supply chain. It also allows for the creation of scan-time rules that prevent critical vulnerabilities from flowing to the supply chain unresolved.

## Features

Features include:

- Scan source code repositories and images for known common vulnerabilities and exposures (CVEs) before deploying to a cluster.
- Identify CVEs by scanning continuously on each new code commit or each new image built.
- Analyze scan results against user-defined policies by using Open Policy Agent. Create scan policy to prevent vulnerable components from going into production.
- Produce vulnerability scan results and post them to the SCST - Store where they can be queried.
- Query the store for such use cases as:
  - What images and packages are affected by a specific vulnerability?
  - What source code repositories are affected by a specific vulnerability?
  - What packages and vulnerabilities does a particular image have?
- Visualize the supply chain and its packages and vulnerabilities of your supply chain.

## Components

- [Supply Chain Security Tools \(SCST\) - Scan](#) scans source code and images for their packages and vulnerabilities.
- [SCST - Store](#) takes the vulnerability scanning results and stores them.
- [Tanzu Insight plug-in](#) provides a CLI to query for packages and vulnerabilities.
- [Supply Chain Choreographer in Tanzu Application Platform GUI](#) visualizes the supply chain, including scans, packages, and vulnerabilities.

## Next steps

Apply what you have learned:

- [Add testing and scanning to your application](#)
- [Enable CVE scan results in Supply Chain Choreographer in Tanzu Application Platform GUI](#)

Or learn about:

- [Supply chains on Tanzu Application Platform](#)

Or go deeper into scanning on Tanzu Application Platform:

- [Scan samples](#) to try the scan and store features as individual one-off scans
- [Configure Code Repositories and Image Artifacts to be Scanned](#)
- [Code and Image Compliance Policy Enforcement Using Open Policy Agent \(OPA\)](#)
- [How to Create a ScanTemplate](#)
- [Viewing and Understanding Scan Status Conditions](#)
- [Observing and Troubleshooting](#)
- [Tanzu Insight plug-in overview](#)

## Troubleshooting

- [SCST Scan - Observing and Troubleshooting](#)
- [SCST Store - Troubleshooting](#)
- [TAP GUI - Troubleshooting](#)

## About consuming services on Tanzu Application Platform

This topic describes the key concepts and terms you need to know about consuming services on Tanzu Application Platform (commonly known as TAP).

As part of Tanzu Application Platform, you can work with backing services such as RabbitMQ, PostgreSQL, and MySQL among others. The most common use of services is binding an application workload to a service instance.

## Key concepts

When working with services on Tanzu Application Platform, you must be familiar with service instances, service bindings, and resource claims. This section provides a brief overview of each of these key concepts.

## Service instances

A **service instance** is a logical grouping of one or more Kubernetes resources that together expose a known capability through a well-defined interface. For example, a theoretical “MySQL” service instance might consist of a [MySQLDatabase](#) and a [MySQLUser](#) resource. When considering compatibility of service instances for Tanzu Application Platform, one of the resources of a service instance must adhere to the [Service Binding for Kubernetes](#) specification.

## Service bindings

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

## Resource claims

**Resource claims** are inspired in part by Persistent Volume Claims. For more information, see the [Kubernetes documentation](#). Resource claims provide a mechanism for users to claim service instances on a cluster, while also decoupling the life cycle of application workloads and service instances.

## Services you can use with Tanzu Application Platform

The following list of Kubernetes operators expose APIs that integrate well with Tanzu Application Platform:

1. [VMware RabbitMQ for Kubernetes](#).
2. [VMware SQL with Postgres for Kubernetes](#).
3. [VMware SQL with MySQL for Kubernetes](#).

Compatibility of a service with Tanzu Application Platform ranges on a scale between fully compatible and incompatible. The minimum requirement for compatibility is that there must be a declarative, Kubernetes-based API on which at least one API resource type adheres to the [Provisioned Service](#) duck type defined by the [Service Binding Specification for Kubernetes](#) in GitHub. This duck type includes any resource type with the following schema:

```
status:
  binding:
    name: # string
```

The value of `.status.binding.name` must point to a [Secret](#) in the same namespace. The [Secret](#) contains required credentials and connectivity information for the resource.

Typically, APIs that include these resource types are installed onto the Tanzu Application Platform cluster as Kubernetes operators. These Kubernetes operators provide custom resource definitions (CRDs) and corresponding controllers to reconcile the resources of the CRDs, as is the case with the three Kubernetes operators listed earlier.

For services that do not provide a resource adhering to the Service Binding Specification for Kubernetes, it may still be possible to provide configurations allowing such services to integrate with Tanzu Application Platform. See the following for examples of how to do this for Amazon AWS RDS.

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

## User roles and responsibilities

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

User role	Exists as a default role in Tanzu Application Platform?	Responsibilities
Service operator	Yes - <a href="#">service-operator</a>	<ul style="list-style-type: none"> <li>Life cycle management (CRUD) of service instances</li> <li>Life cycle management (CRUD) of service instance classes</li> <li>Life cycle management (CRUD) of resource claim policies</li> <li>View and query for resource claims across namespaces</li> </ul>
Application operator	Yes - <a href="#">app-operator</a>	Life cycle management (CRUD) of resource claims
Application developer	Yes - <a href="#">app-editor</a> and <a href="#">app-viewer</a>	Binding service instances to application workloads

## Next steps

Apply what you've learned:

- [Consume services on Tanzu Application Platform](#)

# Set up Tanzu Service Mesh

This topic tells you how to set up a Tanzu Application Platform application deployed on Kubernetes with Tanzu Service Mesh (commonly called TSM).

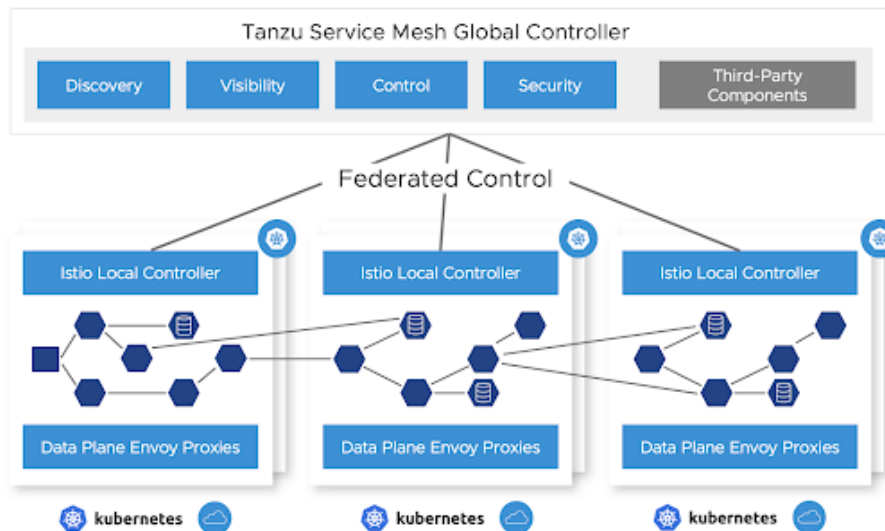
Sample applications are used to demonstrate how a global namespace can provide a network for Kubernetes workloads that are connected and secured within and across clusters, and across clouds.

## Prerequisites

Meet the [prerequisites](#), which includes having

- A supported Kubernetes platform
- The correct resource configuration (number of nodes, CPUs, RAM, and so on)
- The required connectivity requirements

Connectivity is only required from your local clusters out to Tanzu Service Mesh and not inwards. This can traverse a corporate proxy as well. In addition, connectivity in the data plane is required between the clusters that must communicate, specifically egress to ingress gateways. No data plane traffic needs to reach the Tanzu Service Mesh software as a service (SaaS) management plane.



## Activate your Tanzu Service Mesh subscription

Activate your Tanzu Service Mesh subscription at [cloud.vmware.com](https://cloud.vmware.com). After purchasing your Tanzu Service Mesh subscription, the VMware Cloud team sends you instructions. If you don't receive them, you can follow [these instructions](#).

## Onboard clusters

Onboard your clusters to Tanzu Service Mesh as described later in this topic. This deploys the Tanzu Service Mesh local control plane and OSS Istio on your Kubernetes cluster and connects the local control plane to your Tanzu Service Mesh tenant.

As part of the onboarding of the cluster and Tanzu Application Platform integration as well as upgrades to the clusters, these namespaces must remain excluded while getting the Envoy proxy sidecars injected for Run profiles.

Including them might cause the components to stop working at some point in the future when a pod within them is rescheduled or updated.

The following namespaces must be specified as part of the onboarding process and excluded:

- api-auto-registration
- app-live-view-connector
- appssso
- cartographer-system
- cert-manager
- cosign-system
- default
- flux-system
- image-policy-system
- kapp-controller
- knative-eventing
- knative-serving
- knative-sources
- kube-node-lease
- kube-public
- kube-system
- secretgen-controller
- service-bindings
- services-toolkit
- source-system
- tanzu-cluster-essentials
- tanzu-package-repo-global
- tanzu-system-ingress
- tap-install
- tap-telemetry
- triggermesh
- Vmware-sources

You must also exclude these namespaces in case of an upgrade to Tanzu Application Platform. For more information, see [Onboard a Cluster to Tanzu Service Mesh](#).

## Set up Tanzu Application Platform

**Important**

Tanzu Application Platform Build cluster support for Tanzu Service Mesh is limited to basic and testing supply chains. Supply Chains with scanning are not currently supported.

To enable Tanzu Service Mesh support in Tanzu Application Platform Build clusters:

1. Add the following key to `tap-values.yaml` under the `buildservice` top-level key:

```
buildservice:
  injected_sidecar_support: true
```

2. [Install Tanzu Application Platform](#) on the run cluster.

## End-to-end workload build and deployment scenario

The following sections describe how to build and deploy a workload.

### Apply a workload resource to a build cluster

Workloads can be built by using a Tanzu Application Platform supply chain by applying a workload resource to a build cluster. At this time, Tanzu Service Mesh and Tanzu Application Platform cannot use the Knative resources that are the default runtime target when using the `web` resource type.

In Tanzu Application Platform v1.4, two workload types support a Tanzu Service Mesh and Tanzu Application Platform integration: **server** and **worker**.

To work with Tanzu Service Mesh, web workloads must be converted to the `server` or `worker` workload type. Server workloads cause a Kubernetes `Deployment` resource to be created with a `Service` resource that uses port 8080 by default.

1. If the service port that you want is 80 or some other port, add port information to `workload.yaml`. The following example YAML snippets show the changes to make from the `web` to `server` workload type. This is an example before applying the changes:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: hungryman
  labels:
    apps.tanzu.vmware.com/workload-type: web
    app.kubernetes.io/part-of: hungryman-api-gateway
spec:
  params:
    - name: annotations
  value:
    autoscaling.knative.dev/minScale: "1"
  source:
    git:
      url: https://github.com/gm2552/hungryman.git
      ref:
        branch: main
      subPath: hungryman-api-gateway
```

This is an example modified for Tanzu Service Mesh, which includes the removal of the autoscaling annotation:

```
apiVersion: carto.run/v1alpha1
kind: Workload
```



```

metadata:
  name: hungryman
  labels:
    apps.tanzu.vmware.com/workload-type: server # modification
    app.kubernetes.io/part-of: hungryman-api-gateway
spec:
  params:
  - name: ports # modification
    value:
  - port: 80 # modification
    containerPort: 8080 # modification
    name: http # modification
  source:
    git:
      url: https://github.com/gm2552/hungryman.git
      ref:
        branch: main
      subPath: hungryman-api-gateway

```

This results in a deployment and a service that listens on port 80 and forwards traffic to port 8080 on the pod's workload container.

2. Submit the modified YAML to your build cluster by running:

```
tanzu apps NAMESPACE apply --file WORKLOAD-YAML-FILE
```

Where:

- o `NAMESPACE` is the namespace that the build cluster uses for building.
- o `WORKLOAD-YAML-FILE` is the name of your workload YAML file, such as `workload.yaml`.

After your workload is built a `Deliverable` resource is created.

## Configure egress for Tanzu Build Service

For Tanzu Build Service to properly work, provide egress to access the registry where Tanzu Build Service writes application images, and define the registry in the `kp_default_repository` key and the Tanzu Application Platform install registry.

Additionally, configure egress for buildpack builds to download any required dependencies. This configuration varies with different buildpacks and language environments. For example, Java builds might need to download dependencies from Maven central.

## Create a global namespace

Using the Tanzu Service Mesh portal or API, create a global namespace (GNS) that includes the namespaces where your application components are deployed. For more information, see [Global Namespaces](#)

Whether in a single cluster or multiple clusters, or within the same site or across clouds, after you add a namespace selection to the GNS, the services that Tanzu Application Platform deploys are connected based on the GNS configuration for service discovery and connectivity policies.

If a service must be accessible through the ingress from the outside, it can be configured through the public service option in Tanzu Service Mesh or directly through Istio on the clusters where that service resides. It's best practice to configure the service's accessibility through the GNS.

## Run cluster deployment

Before deploying a workload to a run cluster, ensure that any prerequisite resources have already been created on the run cluster. This includes concepts such as data, messaging, routing, security

services, RBAC, ResourceClaims, and so on.

After a successful build in a build cluster, workloads can be deployed to the run cluster by applying resulting deliverable resources to the run cluster as described in [Getting Started with Multicler Tanzu Application Platform](#).

Another option is to create a kapp application that references a GitOps repository to include all deliverable resources for a given cluster. See the following example of a kapp definition that points to a GitOps repository:

```
apiVersion: kappctrl.k14s.io/v1alpha1
kind: App
metadata:
  name: deliverable-gitops
  namespace: hungryman
spec:
  serviceAccountName: default
  fetch:
  - git:
    url: https://github.com/gm2552/tap-play-gitops
    ref: origin/deliverables-tap-east01
    subPath: config
  template:
  - ytt: {}
  deploy:
  - kapp: {}
```

The advantage of this model is that applications can be deployed or uninstalled from a cluster by managing the contents of the deliverable resources from within the GitOps repository and enabling a GitOps workflow for application and service change control.

## Deployment use case: Hungryman

The following instructions describe an end-to-end process for configuring, building, and deploying the Hungryman application into a Tanzu Service Mesh global namespace.

These instructions use the default configuration of Hungryman, which consists of only needing a single-node RabbitMQ cluster, an in-memory database, and no security. The application is deployed across two Tanzu Application Platform run clusters. It requires the `ytt` command to execute the build and deployment commands.

The configuration resources referenced in this scenario are located in the [hungryman-tap-tsm](#) GitHub repository.

### Create an initial set of configuration files from the accelerator

This use case deployment includes a pre-built set of configuration files in a Git repository. However, they were created from a set of configuration files by using a bootstrapped process that uses the Hungryman accelerator, and were later modified.

For reference, you can create an initial set of configuration files from the Hungryman accelerator, which is available in Tanzu Application Platform v1.3.

This section does not include instructions for modifying the configuration files from the accelerator into configuration files used in a later section.

From the accelerator, accept all of the default options with the following exceptions:

- **Workload namespace:** Update this field with the name of the namespace you will use to build the application in your build cluster

- **Service namespace:** Update this field with the name of the namespace you will use to deploy a RabbitMQ cluster on your Tanzu Application Platform run cluster

## Apply the workload resources to your build cluster

To build the application services, run the following command to apply the workload resources to your build cluster. You can also clone or fork the repository in this command to either use the YAML files locally or point to your own Git repository.

```
ytt -f workloads.yaml -v workloadNamespace=WORKLOAD-NAMESPACE | kubectl apply -f-
```

Where `WORKLOAD-NAMESPACE` is the name of your build namespace

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/hungryman-tap-tsm/main/workloads.yaml \
-v workloadNamespace=workloads | kubectl apply -f-
```

If you are using a GitOps workflow with your build cluster, after the workloads are built the deployment information is pushed to your GitOps repository.

If you follow these instructions without pull requests in the GitOps workflow, the config-writer pods that commit deployment information to the GitOps repository might fail because of concurrency conflicts. A workaround for this is to delete the failed workloads from the build cluster and re-run the command provided in the instructions.

## Install service claim resources on the cluster

Hungryman requires a RabbitMQ cluster installed on your run cluster. You must install RabbitMQ on the same run cluster that is named `RunCluster01` in the following deployment section. Additionally, you must install service claim resources on this cluster.

1. If you haven't already done so, install the RabbitMQ Cluster Operator on the run cluster by running:

```
kubectl apply -f "https://github.com/rabbitmq/cluster-operator/releases/download/v1.13.1/cluster-operator.yml"
```

2. Spin up an instance of a RabbitMQ cluster by running:

```
kubectl create ns SERVICE-NAMESPACE

ytt -f rmqCluster.yaml -v serviceNamespace=SERVICE-NAMESPACE | kubectl apply -f-
```

Where `SERVICE-NAMESPACE` is the namespace of where you want to deploy your RabbitMQ cluster

For example:

```
kubectl create ns service-instances

ytt -f https://raw.githubusercontent.com/gm2552/hungryman-tap-tsm/main/rmqCluster.yaml -v \
serviceNamespace=service-instances | kubectl apply -f-
```

3. Create service toolkit resources for the RabbitMQ class and resource claim by running:

```
ytt -f rmqResourceClaim.yaml -v serviceNamespace=SERVICE-NAMESPACE -v \
workloadNamespace=WORKLOAD-NAMESPACE | kubectl apply -f-
```

Where `SERVICE-NAMESPACE` and `WORKLOAD-NAMESPACE` are the namespaces where you deployed your RabbitMQ cluster and the namespace where the application service will run.

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/hungryman-tap-tsm/main/rmqResourceClaim.yaml \
-v serviceNamespace=service-instances -v workloadNamespace=hungryman | kubectl apply -f-
```

## Run cluster deployment

Workloads are deployed to the run cluster using deliverable resources. This section applies the deliverable resources directly to the run clusters instead of using a kapp application.

This deployment assumes that two clusters are part of the Tanzu Service Mesh GNS Hungryman. These example clusters are named `RunCluster01` and `RunCluster02`. The majority of the workload is deployed to `RunCluster01` while the crawler workload is deployed to `RunCluster02`.

The deliverable objects reference the GitOps repository, where the build cluster has written deployment information, and needs to reference this repository in the following commands.

Deploy the workloads to the run clusters by running these commands against their respective clusters:

```
ytt -f cluster01Deliverables.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
gitOpsSecret=GIT-OPS-SECRET -v gitOpsRepo=GIT-OPS-REPO | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads are deployed
- `GIT-OPS-SECRET` is the GitOps secret used to access the GitOps repository
- `GIT-OPS-REPO` is the URL of the GitOps repository where the build cluster wrote out deployment configuration information

```
ytt -f cluster02Deliverables.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
gitOpsSecret=GIT-OPS-SECRET -v gitOpsRepo=GIT-OPS-REPO | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads are deployed
- `GIT-OPS-SECRET` is the GitOps secret used to access the GitOps repository
- `GIT-OPS-REPO` is the URL of the GitOps repository where the build cluster wrote out deployment configuration information

To run this deployment on cluster `RunCluster01`, for example, you run:

```
ytt -f https://raw.githubusercontent.com/gm2552/hungryman-tap-tsm/main/cluster01Deliverables.yaml -v \
workloadNamespace=hungryman -v gitOpsSecret=tap-play-gitops-secret -v \
gitOpsRepo=https://github.com/gm2552/tap-play-gitops.git | kubectl apply -f-
```

To run this deployment on cluster `RunCluster02`, for example, you run:

```
ytt -f https://raw.githubusercontent.com/gm2552/hungryman-tap-tsm/main/cluster02Deliverables.yaml -v \
```

```
workloadNamespace=hungryman -v gitOpsSecret=tap-play-gitops-secret -v \
gitOpsRepo=https://github.com/gm2552/tap-play-gitops.git | kubectl apply -f-
```

You can create an Istio ingress resource on `RunCluster01` if you do not plan on using the GNS capabilities to expose the application to external networks.

You must create a domain name system address (DNS A) record in your DNS provider's configuration tool to point to the Istio load-balanced IP address of `RunCluster01`. The DNS configuration is out of the scope of this topic.

Create the ingress by running:

```
ytt -f ingress.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v domainName=DOMAIN-NAME
| kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workload is deployed
- `DOMAIN-NAME` is the public domain that will host your application

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/hungryman-tap-tsm/main/ingress.yaml -v \
workloadNamespace=hungryman -v domainName=tsmdemo.perfect300rock.com | kubectl apply -f-
```

## Create a global namespace

The example clusters have the names `RunCluster01` and `RunCluster02`, and they assume the workload and service namespaces of `Hungryman` and `service-instances`, respectively.

1. Open the Tanzu Service Mesh console and create a new GNS.
2. Configure the following settings in each step:
  1. General details
    - **GNS Name:** hungryman
    - **Domain:** hungryman.lab
  2. Namespace mapping
    - Namespace mapping Rule 1
      - **Cluster name:** RunCluster01
      - **Namespace:** hungryman
    - Namespace Mapping Rule 2
      - **Cluster name:** RunCluster02
      - **Namespace:** hungryman
    - Namespace Mapping Rule 3
      - **Cluster name:** RunCluster01
      - **Namespace:** service-instances
3. Autodiscovery. Use the default settings.
4. Public services
  - **Service name:** hungryman
  - **Service port:** 80

- **Public URL:** http hungryman . Select a domain.

5. Global server load balancing and resiliency. Use the default settings.

You can now access the Hungryman application with the URL configured earlier.

## Deployment use case: ACME Fitness Store

The following instructions describe an end-to-end process for configuring, building, and deploying the ACME Fitness Store application into a Tanzu Service Mesh GNS. In this use case, the application is deployed across two Tanzu Application Platform run clusters. ytt is used to run the build and deployment commands.

The configuration resources referenced in this scenario are in the [acme-fitness-tap-tsm](#) Git repository.

### Deploy AppSSO

ACME requires the use of an AppSSO authorization server and client registration resource. Install these resources on the same run cluster that is named `RunCluster01` in the deployment section.

1. Deploy the authorization server instance by running:

```
ytt -f appSSOInstance.yaml -v workloadNamespace=WORKLOAD-NAMESPACE \
-v devDefaultAccountUsername=DEV-DEFAULT-ACCOUNT-USERNAME -v \
devDefaultAccountPassword=DEV-DEFAULT-ACCOUNT-PASSWORD | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads will be deployed
- `DEV-DEFAULT-ACCOUNT-USERNAME` is the user name for the ACME application authentication
- `DEV-DEFAULT-ACCOUNT-PASSWORD` is the password for the ACME application authentication

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/appSSOInstance.yaml -v \
workloadNamespace=acme -v devDefaultAccountUsername=acme -v \
devDefaultAccountPassword=fitness | kubectl apply -f-
```

2. Create a `ClientRegistration` resource by running:

```
ytt -f appSSOInstance.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
appSSORedirectURI=APP-SSO-REDIRECT-URI | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads will be deployed.
- `APP-SSO-REDIRECT-URI` is the public URI that the authorization server redirects to after a login

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/clientRegistrationResourceClaim.yaml \
-v workloadNamespace=acme -v \
appSSORedirectURI=http://acme-fitness.tsmdemo.perfect300rock.com/login/oauth2/code/sso | kubectl apply -f-
```

3. Obtain the appSSO Issuer URI by running:

```
kubectl get authserver -n WORKLOAD-NAMESPACE
```

Where `WORKLOAD-NAMESPACE` is the name of the namespace where the workloads will be deployed.

4. Record the Issuer URI because you need it for the next section.

## Apply the workload resources to your build cluster

To build the application services, run the following command to apply the workload resources to your build cluster. You can also clone or fork the repository in the following command to either use the YAML files locally or point to your own Git repository.

```
ytt -f workloads.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
appSSOIssuerURI=APP-SSO-ISSUER-URL | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the name of your build namespace
- `APP-SSO-ISSUER-URL` is the URL of the AppSSO authorization server that you deployed earlier

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/workloads.yaml -v \
workloadNamespace=workloads -v \
appSSOIssuerURI=http://appssso-acme-fitness.acme.tsmdemo.perfect300rock.com | kubectl apply -f-
```

If you are using a GitOps workflow with your build cluster then, after building the workloads, the deployment information is pushed to your GitOps repository.

If you follow these instructions without pull requests in the GitOps workflow, the `config-writer` pods that commit deployment information to the GitOps repository might fail because of concurrency conflicts. A workaround for this is to delete the failed workloads from the build cluster and re-run the command provided in these instructions.

## Create the Istio ingress resources

The authorization server requires a publicly accessible URL and must be available before the Spring Cloud Gateway can deploy properly. The authorization server is deployed at the URI `authserver` app domain.

You must create a domain name system address (DNS A) record in your DNS provider's configuration tool to point to the Istio load-balanced IP address of `RunCluster01`. The DNS configuration is out of the scope of this topic.

Create the Istio ingress resources by running:

```
ytt -f istioGateway.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
appDomainName=APP-DOMAIN | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the name of your build namespace
- `APP-DOMAIN` is the application's DNS domain

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/istioGateway.yaml -v \
workloadNamespace=acme -v appDomainName=tsmdemo.perfect300rock.com | kubectl apply -f-
```

## Deploy Redis

A Redis instance is needed for caching the ACME fitness store cart service. Deploy the Redis instance by running:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/redis.yaml -v \
workloadNamespace=WORKLOAD-NAMESPACE -v redisPassword=REDIS-PASSWORD | kb apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads will be deployed
- `REDIS-PASSWORD` is your password

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/redis.yaml -v \
workloadNamespace=acme -v redisPassword=fitness | kubectl apply -f-
```

## Run cluster deployment

Workloads are deployed to the run cluster by using deliverable resources. In this section you apply the deliverable resources directly to the run clusters, instead of using a kapp application. This deployment assumes that two clusters are part of the Tanzu Service Mesh GNS ACME. In this example these clusters are named `RunCluster01` and `RunCluster02`.

The deliverable objects reference the GitOps repository, where the build cluster has written deployment information, and need to reference this repository in the following commands.

To deploy the workloads to the run clusters, run these commands against their respective clusters:

```
ytt -f cluster01Deliverables.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
gitOpsSecret=GIT-OPS-SECRET -v gitOpsRepo=GIT-OPS-REPO | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads will be deployed
- `GIT-OPS-SECRET` is the GitOps secret used to access the GitOps repository
- `GIT-OPS-REPO` is the URL of the GitOps repository where the build cluster wrote out deployment configuration information

```
ytt -f cluster02Deliverables.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
gitOpsSecret=GIT-OPS-SECRET -v gitOpsRepo=GIT-OPS-REPO | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads will be deployed
- `GIT-OPS-SECRET` is the GitOps secret used to access the GitOps repository
- `GIT-OPS-REPO` is the URL of the GitOps repository where the build cluster wrote out deployment configuration information



For the `RunCluster01` example, run:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/cluster01Deliverables.yaml \
-v workloadNamespace=acme -v gitOpsSecret=tap-play-gitops-secret -v \
gitOpsRepo=https://github.com/gm2552/tap-play-gitops.git | kubectl apply -f-
```

For the `RunCluster02` example, run:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/cluster02Deliverables.yaml \
-v workloadNamespace=acme -v gitOpsSecret=tap-play-gitops-secret -v \
gitOpsRepo=https://github.com/gm2552/tap-play-gitops.git | kubectl apply -f-
```

## Deploy Spring Cloud Gateway

The following sections describe how to deploy Spring Cloud Gateway.

### Install the Spring Cloud Gateway package

The section requires the Spring Cloud Gateway for Kubernetes package to be installed on `RunCluster01`. For instructions, see [Installing Spring Cloud Gateway for Kubernetes using the Tanzu CLI](#).

The Spring Cloud Gateway `spec.service.name` configuration was not built with multi, cross-cluster support. The configuration for the gateway routes currently implements a workaround, which is brittle in terms of where certain services are deployed. Future releases of the gateway might have better support for this use case.

Deploy the gateway and applicable routes by running:

```
ytt -f scgInstance.yaml -v workloadNamespace=WORKLOAD-NAMESPACE
```

Where `WORKLOAD-NAMESPACE` is the namespace where the workload is deployed.

```
ytt -f scgRoutes.yaml -v workloadNamespace=WORKLOAD-NAMESPACE
```

Where `WORKLOAD-NAMESPACE` is the namespace where the workload is deployed.

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/scgInstance.yaml -v \
workloadNamespace=acme | kubectl apply -f-
```

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/scgRoutes.yaml -v \
workloadNamespace=acme | kubectl apply -f-
```

## Create a global namespace

The example clusters are named `RunCluster01` and `RunCluster02`, and they assume a workload namespace of ACME.

1. Open the Tanzu Service Mesh console and create a new global namespace.
2. Configure the following settings in each step:
  1. General details
    - **GNS name:** acme-tap

- **Domain:** acme-tap.lab
2. Namespace mapping
    - Namespace mapping Rule 1
      - **Cluster name:** RunCluster01
      - **Namespace:** acme
    - Namespace Mapping Rule 2
      - **Cluster name:** RunCluster02
      - **Namespace:** acme
  3. Autodiscovery. Use the default settings.
  4. Public Services
    - No Public service
  5. Global server load-balancing and resiliency. Use the default settings.

You can access the application by going to the URL <http://acme-fitness>.

## Set up Tanzu Service Mesh

This topic tells you how to set up a Tanzu Application Platform application deployed on Kubernetes with Tanzu Service Mesh (commonly called TSM).

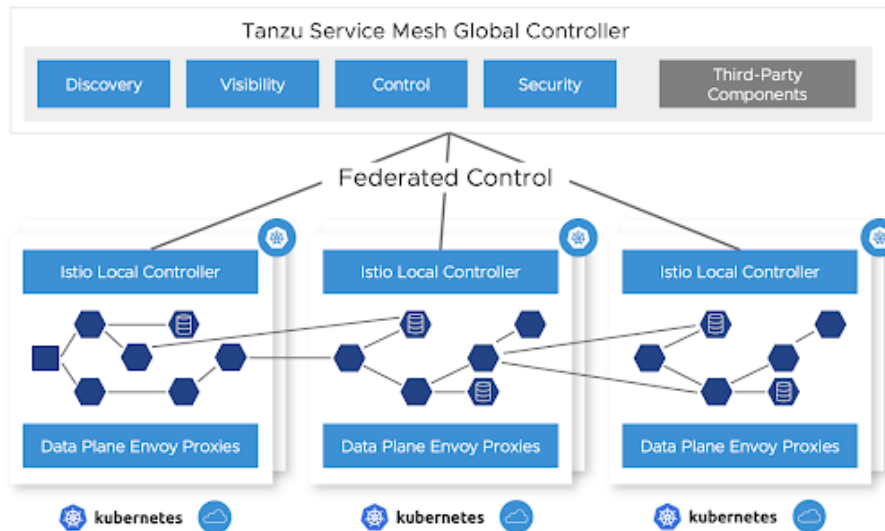
Sample applications are used to demonstrate how a global namespace can provide a network for Kubernetes workloads that are connected and secured within and across clusters, and across clouds.

## Prerequisites

Meet the [prerequisites](#), which includes having

- A supported Kubernetes platform
- The correct resource configuration (number of nodes, CPUs, RAM, and so on)
- The required connectivity requirements

Connectivity is only required from your local clusters out to Tanzu Service Mesh and not inwards. This can traverse a corporate proxy as well. In addition, connectivity in the data plane is required between the clusters that must communicate, specifically egress to ingress gateways. No data plane traffic needs to reach the Tanzu Service Mesh software as a service (SaaS) management plane.



## Activate your Tanzu Service Mesh subscription

Activate your Tanzu Service Mesh subscription at [cloud.vmware.com](https://cloud.vmware.com). After purchasing your Tanzu Service Mesh subscription, the VMware Cloud team sends you instructions. If you don't receive them, you can follow [these instructions](#).

## Onboard clusters

Onboard your clusters to Tanzu Service Mesh as described later in this topic. This deploys the Tanzu Service Mesh local control plane and OSS Istio on your Kubernetes cluster and connects the local control plane to your Tanzu Service Mesh tenant.

As part of the onboarding of the cluster and Tanzu Application Platform integration as well as upgrades to the clusters, these namespaces must remain excluded while getting the Envoy proxy sidecars injected for Run profiles.

Including them might cause the components to stop working at some point in the future when a pod within them is rescheduled or updated.

The following namespaces must be specified as part of the onboarding process and excluded:

- api-auto-registration
- app-live-view-connector
- appssso
- cartographer-system
- cert-manager
- cosign-system
- default
- flux-system
- image-policy-system
- kapp-controller
- knative-eventing
- knative-serving
- knative-sources

- kube-node-lease
- kube-public
- kube-system
- secretgen-controller
- service-bindings
- services-toolkit
- source-system
- tanzu-cluster-essentials
- tanzu-package-repo-global
- tanzu-system-ingress
- tap-install
- tap-telemetry
- triggermesh
- Vmware-sources

You must also exclude these namespaces in case of an upgrade to Tanzu Application Platform. For more information, see [Onboard a Cluster to Tanzu Service Mesh](#).

## Set up Tanzu Application Platform



### Important

Tanzu Application Platform Build cluster support for Tanzu Service Mesh is limited to basic and testing supply chains. Supply Chains with scanning are not currently supported.

To enable Tanzu Service Mesh support in Tanzu Application Platform Build clusters:

1. Add the following key to `tap-values.yaml` under the `buildservice` top-level key:

```
buildservice:
  injected_sidecar_support: true
```

2. [Install Tanzu Application Platform](#) on the run cluster.

## End-to-end workload build and deployment scenario

The following sections describe how to build and deploy a workload.

### Apply a workload resource to a build cluster

Workloads can be built by using a Tanzu Application Platform supply chain by applying a workload resource to a build cluster. At this time, Tanzu Service Mesh and Tanzu Application Platform cannot use the Knative resources that are the default runtime target when using the `web` resource type.

In Tanzu Application Platform v1.4, two workload types support a Tanzu Service Mesh and Tanzu Application Platform integration: **server** and **worker**.

To work with Tanzu Service Mesh, web workloads must be converted to the `server` or `worker` workload type. Server workloads cause a Kubernetes `Deployment` resource to be created with a

Service resource that uses port 8080 by default.

1. If the service port that you want is 80 or some other port, add port information to `workload.yaml`. The following example YAML snippets show the changes to make from the `web` to `server` workload type. This is an example before applying the changes:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: hungryman
  labels:
    apps.tanzu.vmware.com/workload-type: web
    app.kubernetes.io/part-of: hungryman-api-gateway
spec:
  params:
    - name: annotations
value:
  autoscaling.knative.dev/minScale: "1"
  source:
    git:
      url: https://github.com/gm2552/hungryman.git
      ref:
        branch: main
      subPath: hungryman-api-gateway
```

This is an example modified for Tanzu Service Mesh, which includes the removal of the autoscaling annotation:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: hungryman
  labels:
    apps.tanzu.vmware.com/workload-type: server # modification
    app.kubernetes.io/part-of: hungryman-api-gateway
spec:
  params:
    - name: ports # modification
value:
  - port: 80 # modification
    containerPort: 8080 # modification
    name: http # modification
  source:
    git:
      url: https://github.com/gm2552/hungryman.git
      ref:
        branch: main
      subPath: hungryman-api-gateway
```

This results in a deployment and a service that listens on port 80 and forwards traffic to port 8080 on the pod's workload container.

2. Submit the modified YAML to your build cluster by running:

```
tanzu apps NAMESPACE apply --file WORKLOAD-YAML-FILE
```

Where:

- o `NAMESPACE` is the namespace that the build cluster uses for building.
- o `WORKLOAD-YAML-FILE` is the name of your workload YAML file, such as `workload.yaml`.

After your workload is built a `Deliverable` resource is created.

## Configure egress for Tanzu Build Service

For Tanzu Build Service to properly work, provide egress to access the registry where Tanzu Build Service writes application images, and define the registry in the `kp_default_repository` key and the Tanzu Application Platform install registry.

Additionally, configure egress for buildpack builds to download any required dependencies. This configuration varies with different buildpacks and language environments. For example, Java builds might need to download dependencies from Maven central.

## Create a global namespace

Using the Tanzu Service Mesh portal or API, create a global namespace (GNS) that includes the namespaces where your application components are deployed. For more information, see [Global Namespaces](#)

Whether in a single cluster or multiple clusters, or within the same site or across clouds, after you add a namespace selection to the GNS, the services that Tanzu Application Platform deploys are connected based on the GNS configuration for service discovery and connectivity policies.

If a service must be accessible through the ingress from the outside, it can be configured through the public service option in Tanzu Service Mesh or directly through Istio on the clusters where that service resides. It's best practice to configure the service's accessibility through the GNS.

## Run cluster deployment

Before deploying a workload to a run cluster, ensure that any prerequisite resources have already been created on the run cluster. This includes concepts such as data, messaging, routing, security services, RBAC, ResourceClaims, and so on.

After a successful build in a build cluster, workloads can be deployed to the run cluster by applying resulting deliverable resources to the run cluster as described in [Getting Started with Multicluster Tanzu Application Platform](#).

Another option is to create a kapp application that references a GitOps repository to include all deliverable resources for a given cluster. See the following example of a kapp definition that points to a GitOps repository:

```
apiVersion: kappctrl.k14s.io/v1alpha1
kind: App
metadata:
  name: deliverable-gitops
  namespace: hungryman
spec:
  serviceAccountName: default
  fetch:
  - git:
    url: https://github.com/gm2552/tap-play-gitops
    ref: origin/deliverables-tap-east01
    subPath: config
  template:
  - ytt: {}
  deploy:
  - kapp: {}
```

The advantage of this model is that applications can be deployed or uninstalled from a cluster by managing the contents of the deliverable resources from within the GitOps repository and enabling a GitOps workflow for application and service change control.

## Deployment use case: Hungryman

The following instructions describe an end-to-end process for configuring, building, and deploying the Hungryman application into a Tanzu Service Mesh global namespace.

These instructions use the default configuration of Hungryman, which consists of only needing a single-node RabbitMQ cluster, an in-memory database, and no security. The application is deployed across two Tanzu Application Platform run clusters. It requires the `ytt` command to execute the build and deployment commands.

The configuration resources referenced in this scenario are located in the [hungryman-tap-tsm](#) GitHub repository.

## Create an initial set of configuration files from the accelerator

This use case deployment includes a pre-built set of configuration files in a Git repository. However, they were created from a set of configuration files by using a bootstrapped process that uses the Hungryman accelerator, and were later modified.

For reference, you can create an initial set of configuration files from the Hungryman accelerator, which is available in Tanzu Application Platform v1.3.

This section does not include instructions for modifying the configuration files from the accelerator into configuration files used in a later section.

From the accelerator, accept all of the default options with the following exceptions:

- **Workload namespace:** Update this field with the name of the namespace you will use to build the application in your build cluster
- **Service namespace:** Update this field with the name of the namespace you will use to deploy a RabbitMQ cluster on your Tanzu Application Platform run cluster

## Apply the workload resources to your build cluster

To build the application services, run the following command to apply the workload resources to your build cluster. You can also clone or fork the repository in this command to either use the YAML files locally or point to your own Git repository.

```
ytt -f workloads.yaml -v workloadNamespace=WORKLOAD-NAMESPACE | kubectl apply -f-
```

Where `WORKLOAD-NAMESPACE` is the name of your build namespace

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/hungryman-tap-tsm/main/workloads.yaml \
-v workloadNamespace=workloads | kubectl apply -f-
```

If you are using a GitOps workflow with your build cluster, after the workloads are built the deployment information is pushed to your GitOps repository.

If you follow these instructions without pull requests in the GitOps workflow, the config-writer pods that commit deployment information to the GitOps repository might fail because of concurrency conflicts. A workaround for this is to delete the failed workloads from the build cluster and re-run the command provided in the instructions.

## Install service claim resources on the cluster

Hungryman requires a RabbitMQ cluster installed on your run cluster. You must install RabbitMQ on the same run cluster that is named `RunCluster01` in the following deployment section. Additionally, you must install service claim resources on this cluster.

1. If you haven't already done so, install the RabbitMQ Cluster Operator on the run cluster by running:

```
kubectl apply -f "https://github.com/rabbitmq/cluster-operator/releases/download/v1.13.1/cluster-operator.yml"
```

2. Spin up an instance of a RabbitMQ cluster by running:

```
kubectl create ns SERVICE-NAMESPACE

ytt -f rmqCluster.yaml -v serviceNamespace=SERVICE-NAMESPACE | kubectl apply -f -
```

Where `SERVICE-NAMESPACE` is the namespace of where you want to deploy your RabbitMQ cluster

For example:

```
kubectl create ns service-instances

ytt -f https://raw.githubusercontent.com/gm2552/hungryman-tap-tsm/main/rmqCluster.yaml -v \
serviceNamespace=service-instances | kubectl apply -f-
```

3. Create service toolkit resources for the RabbitMQ class and resource claim by running:

```
ytt -f rmqResourceClaim.yaml -v serviceNamespace=SERVICE-NAMESPACE -v \
workloadNamespace=WORKLOAD-NAMESPACE | kubectl apply -f-
```

Where `SERVICE-NAMESPACE` and `WORKLOAD-NAMESPACE` are the namespaces where you deployed your RabbitMQ cluster and the namespace where the application service will run.

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/hungryman-tap-tsm/main/rmqResourceClaim.yaml \
-v serviceNamespace=service-instances -v workloadNamespace=hungryman | kubectl apply -f-
```

## Run cluster deployment

Workloads are deployed to the run cluster using deliverable resources. This section applies the deliverable resources directly to the run clusters instead of using a kapp application.

This deployment assumes that two clusters are part of the Tanzu Service Mesh GNS Hungryman. These example clusters are named `RunCluster01` and `RunCluster02`. The majority of the workload is deployed to `RunCluster01` while the crawler workload is deployed to `RunCluster02`.

The deliverable objects reference the GitOps repository, where the build cluster has written deployment information, and needs to reference this repository in the following commands.

Deploy the workloads to the run clusters by running these commands against their respective clusters:

```
ytt -f cluster01Deliverables.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
gitOpsSecret=GIT-OPS-SECRET -v gitOpsRepo=GIT-OPS-REPO | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads are deployed
- `GIT-OPS-SECRET` is the GitOps secret used to access the GitOps repository



- `GIT-OPS-REPO` is the URL of the GitOps repository where the build cluster wrote out deployment configuration information

```
ytt -f cluster02Deliverables.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
gitOpsSecret=GIT-OPS-SECRET -v gitOpsRepo=GIT-OPS-REPO | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads are deployed
- `GIT-OPS-SECRET` is the GitOps secret used to access the GitOps repository
- `GIT-OPS-REPO` is the URL of the GitOps repository where the build cluster wrote out deployment configuration information

To run this deployment on cluster `RunCluster01`, for example, you run:

```
ytt -f https://raw.githubusercontent.com/gm2552/hungryman-tap-tsm/main/cluster01Deliverables.yaml -v \
workloadNamespace=hungryman -v gitOpsSecret=tap-play-gitops-secret -v \
gitOpsRepo=https://github.com/gm2552/tap-play-gitops.git | kubectl apply -f-
```

To run this deployment on cluster `RunCluster02`, for example, you run:

```
ytt -f https://raw.githubusercontent.com/gm2552/hungryman-tap-tsm/main/cluster02Deliverables.yaml -v \
workloadNamespace=hungryman -v gitOpsSecret=tap-play-gitops-secret -v \
gitOpsRepo=https://github.com/gm2552/tap-play-gitops.git | kubectl apply -f-
```

You can create an Istio ingress resource on `RunCluster01` if you do not plan on using the GNS capabilities to expose the application to external networks.

You must create a domain name system address (DNS A) record in your DNS provider's configuration tool to point to the Istio load-balanced IP address of `RunCluster01`. The DNS configuration is out of the scope of this topic.

Create the ingress by running:

```
ytt -f ingress.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v domainName=DOMAIN-NAME
| kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workload is deployed
- `DOMAIN-NAME` is the public domain that will host your application

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/hungryman-tap-tsm/main/ingress.yaml -v \
workloadNamespace=hungryman -v domainName=tsmdemo.perfect300rock.com | kubectl apply -f-
```

## Create a global namespace

The example clusters have the names `RunCluster01` and `RunCluster02`, and they assume the workload and service namespaces of Hungryman and service-instances, respectively.

1. Open the Tanzu Service Mesh console and create a new GNS.
2. Configure the following settings in each step:
  1. General details

- **GNS Name:** hungryman
  - **Domain:** hungryman.lab
2. Namespace mapping
    - Namespace mapping Rule 1
      - **Cluster name:** RunCluster01
      - **Namespace:** hungryman
    - Namespace Mapping Rule 2
      - **Cluster name:** RunCluster02
      - **Namespace:** hungryman
    - Namespace Mapping Rule 3
      - **Cluster name:** RunCluster01
      - **Namespace:** service-instances
  3. Autodiscovery. Use the default settings.
  4. Public services
    - **Service name:** hungryman
    - **Service port:** 80
    - **Public URL:** http hungryman . Select a domain.
  5. Global server load balancing and resiliency. Use the default settings.

You can now access the Hungryman application with the URL configured earlier.

## Deployment use case: ACME Fitness Store

The following instructions describe an end-to-end process for configuring, building, and deploying the ACME Fitness Store application into a Tanzu Service Mesh GNS. In this use case, the application is deployed across two Tanzu Application Platform run clusters. `ytt` is used to run the build and deployment commands.

The configuration resources referenced in this scenario are in the [acme-fitness-tap-tsm](#) Git repository.

### Deploy AppSSO

ACME requires the use of an AppSSO authorization server and client registration resource. Install these resources on the same run cluster that is named `RunCluster01` in the deployment section.

1. Deploy the authorization server instance by running:

```
ytt -f appSSOInstance.yaml -v workloadNamespace=WORKLOAD-NAMESPACE \
-v devDefaultAccountUsername=DEV-DEFAULT-ACCOUNT-USERNAME -v \
devDefaultAccountPassword=DEV-DEFAULT-ACCOUNT-PASSWORD | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads will be deployed
- `DEV-DEFAULT-ACCOUNT-USERNAME` is the user name for the ACME application authentication
- `DEV-DEFAULT-ACCOUNT-PASSWORD` is the password for the ACME application authentication

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/appSSOInstance.yaml -v \
workloadNamespace=acme -v devDefaultAccountUsername=acme -v \
devDefaultAccountPassword=fitness | kubectl apply -f-
```

2. Create a `ClientRegistration` resource by running:

```
ytt -f appSSOInstance.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
appSSORedirectURI=APP-SSO-REDIRECT-URI | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads will be deployed.
- `APP-SSO-REDIRECT-URI` is the public URI that the authorization server redirects to after a login

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/clientRegistrationResourceClaim.yaml \
-v workloadNamespace=acme -v \
appSSORedirectURI=http://acme-fitness.tsmdemo.perfect300rock.com/login/oauth2/code/sso | kubectl apply -f-
```

3. Obtain the appSSO Issuer URI by running:

```
kubectl get authserver -n WORKLOAD-NAMESPACE
```

Where `WORKLOAD-NAMESPACE` is the name of the namespace where the workloads will be deployed.

4. Record the Issuer URI because you need it for the next section.

## Apply the workload resources to your build cluster

To build the application services, run the following command to apply the workload resources to your build cluster. You can also clone or fork the repository in the following command to either use the YAML files locally or point to your own Git repository.

```
ytt -f workloads.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
appSSOIssuerURI=APP-SSO-ISSUER-URL | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the name of your build namespace
- `APP-SSO-ISSUER-URL` is the URL of the AppSSO authorization server that you deployed earlier

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/workloads.yaml -v \
workloadNamespace=workloads -v \
appSSOIssuerURI=http://appssso-acme-fitness.acme.tsmdemo.perfect300rock.com | kubectl apply -f-
```

If you are using a GitOps workflow with your build cluster then, after building the workloads, the deployment information is pushed to your GitOps repository.

If you follow these instructions without pull requests in the GitOps workflow, the `config-writer` pods that commit deployment information to the GitOps repository might fail because of concurrency conflicts. A workaround for this is to delete the failed workloads from the build cluster and re-run the command provided in these instructions.

## Create the Istio ingress resources

The authorization server requires a publicly accessible URL and must be available before the Spring Cloud Gateway can deploy properly. The authorization server is deployed at the URI authserver app domain.

You must create a domain name system address (DNS A) record in your DNS provider's configuration tool to point to the Istio load-balanced IP address of `RunCluster01`. The DNS configuration is out of the scope of this topic.

Create the Istio ingress resources by running:

```
ytt -f istioGateway.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
appDomainName=APP-DOMAIN | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the name of your build namespace
- `APP-DOMAIN` is the application's DNS domain

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/istioGateway.yaml -v \
workloadNamespace=acme -v appDomainName=tsmdemo.perfect300rock.com | kubectl apply -f-
```

## Deploy Redis

A Redis instance is needed for caching the ACME fitness store cart service. Deploy the Redis instance by running:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/redis.yaml -v \
workloadNamespace=WORKLOAD-NAMESPACE -v redisPassword=REDIS-PASSWORD | kb apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads will be deployed
- `REDIS-PASSWORD` is your password

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/redis.yaml -v \
workloadNamespace=acme -v redisPassword=fitness | kubectl apply -f-
```

## Run cluster deployment

Workloads are deployed to the run cluster by using deliverable resources. In this section you apply the deliverable resources directly to the run clusters, instead of using a kapp application. This deployment assumes that two clusters are part of the Tanzu Service Mesh GNS ACME. In this example these clusters are named `RunCluster01` and `RunCluster02`.

The deliverable objects reference the GitOps repository, where the build cluster has written deployment information, and need to reference this repository in the following commands.

To deploy the workloads to the run clusters, run these commands against their respective clusters:

```
ytt -f cluster01Deliverables.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
gitOpsSecret=GIT-OPS-SECRET -v gitOpsRepo=GIT-OPS-REPO | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads will be deployed
- `GIT-OPS-SECRET` is the GitOps secret used to access the GitOps repository
- `GIT-OPS-REPO` is the URL of the GitOps repository where the build cluster wrote out deployment configuration information

```
ytt -f cluster02Deliverables.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
gitOpsSecret=GIT-OPS-SECRET -v gitOpsRepo=GIT-OPS-REPO | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads will be deployed
- `GIT-OPS-SECRET` is the GitOps secret used to access the GitOps repository
- `GIT-OPS-REPO` is the URL of the GitOps repository where the build cluster wrote out deployment configuration information

For the `RunCluster01` example, run:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/cluster01Deliverables.yaml \
-v workloadNamespace=acme -v gitOpsSecret=tap-play-gitops-secret -v \
gitOpsRepo=https://github.com/gm2552/tap-play-gitops.git | kubectl apply -f-
```

For the `RunCluster02` example, run:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/cluster02Deliverables.yaml \
-v workloadNamespace=acme -v gitOpsSecret=tap-play-gitops-secret -v \
gitOpsRepo=https://github.com/gm2552/tap-play-gitops.git | kubectl apply -f-
```

## Deploy Spring Cloud Gateway

The following sections describe how to deploy Spring Cloud Gateway.

### Install the Spring Cloud Gateway package

The section requires the Spring Cloud Gateway for Kubernetes package to be installed on `RunCluster01`. For instructions, see [Installing Spring Cloud Gateway for Kubernetes using the Tanzu CLI](#).

The Spring Cloud Gateway `spec.service.name` configuration was not built with multi, cross-cluster support. The configuration for the gateway routes currently implements a workaround, which is brittle in terms of where certain services are deployed. Future releases of the gateway might have better support for this use case.

Deploy the gateway and applicable routes by running:

```
ytt -f scgInstance.yaml -v workloadNamespace=WORKLOAD-NAMESPACE
```

Where `WORKLOAD-NAMESPACE` is the namespace where the workload is deployed.

```
ytt -f scgRoutes.yaml -v workloadNamespace=WORKLOAD-NAMESPACE
```

Where `WORKLOAD-NAMESPACE` is the namespace where the workload is deployed.

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/scgInstance.yaml -v \
workloadNamespace=acme | kubectl apply -f-
```

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/scgRoutes.yaml -v \
workloadNamespace=acme | kubectl apply -f-
```

## Create a global namespace

The example clusters are named `RunCluster01` and `RunCluster02`, and they assume a workload namespace of `ACME`.

1. Open the Tanzu Service Mesh console and create a new global namespace.
2. Configure the following settings in each step:
  1. General details
    - **GNS name:** acme-tap
    - **Domain:** acme-tap.lab
  2. Namespace mapping
    - Namespace mapping Rule 1
      - **Cluster name:** RunCluster01
      - **Namespace:** acme
    - Namespace Mapping Rule 2
      - **Cluster name:** RunCluster02
      - **Namespace:** acme
  3. Autodiscovery. Use the default settings.
  4. Public Services
    - No Public service
  5. Global server load-balancing and resiliency. Use the default settings.

You can access the application by going to the URL <http://acme-fitness>.

# Overview of workloads

This topic provides you with an overview of workload types in Tanzu Application Platform (commonly known as TAP).

## Workload features

Tanzu Application Platform allows you to quickly build and test applications regardless of your familiarity with Kubernetes.

You can turn source code into a workload that runs in a container with a URL. You can also use supply chains to build applications that process work from a message queue, or provide arbitrary network services.

A workload allows you to choose application specifications, such as repository location, environment variables, service binding, and so on. For more information about workload creation and management, see [Commands Details](#).

The Out of the Box supply chains support a range of workload types, including

- Scalable web applications ([web](#))
- Traditional application servers ([server](#))
- Background applications ([worker](#))
- Serverless functions

You can use a collection of workloads of different types to deploy microservices that function as a logical application. Alternatively, you can deploy your entire application as a single monolith.

If you build your own supply chains, you can define additional deployment methods beyond those in the Out of the Box supply-chain templates.

## Available workload types

When using the Out of the Box supply chain, the [apps.tanzu.vmware.com/workload-type](https://apps.tanzu.vmware.com/workload-type) annotation selects which style of deployment is suitable for your application. The valid values are:

Type	Description	Indicators
<a href="#">web</a>	Scalable web applications	<ul style="list-style-type: none"> <li>• Scales based on request load</li> <li>• Automatically exposed by HTTP Ingress</li> <li>• Does not perform background work</li> <li>• Works with Service Bindings</li> <li>• Stateless</li> <li>• Quick startup time</li> </ul>

Type	Description	Indicators
<code>server</code>	Traditional applications	<ul style="list-style-type: none"> <li>Provides HTTP or TCP services on the network</li> <li>Exposed by external Ingress or LoadBalancer settings</li> <li>Might perform background work from a queue</li> <li>Works with Service Bindings</li> <li>Fixed scaling, no disk persistence</li> <li>Startup time not an issue</li> </ul>
<code>worker</code>	Background applications	<ul style="list-style-type: none"> <li>Does not provide network services</li> <li>Not exposed externally as a network service</li> <li>Performs background work from a queue</li> <li>Works with Service Bindings</li> <li>Fixed scaling, no disk persistence</li> <li>Startup time not an issue</li> </ul>

## Overview of workloads

This topic provides you with an overview of workload types in Tanzu Application Platform (commonly known as TAP).

## Workload features

Tanzu Application Platform allows you to quickly build and test applications regardless of your familiarity with Kubernetes.

You can turn source code into a workload that runs in a container with a URL. You can also use supply chains to build applications that process work from a message queue, or provide arbitrary network services.

A workload allows you to choose application specifications, such as repository location, environment variables, service binding, and so on. For more information about workload creation and management, see [Commands Details](#).

The Out of the Box supply chains support a range of workload types, including

- Scalable web applications (`web`)
- Traditional application servers (`server`)
- Background applications (`worker`)
- Serverless functions

You can use a collection of workloads of different types to deploy microservices that function as a logical application. Alternatively, you can deploy your entire application as a single monolith.

If you build your own supply chains, you can define additional deployment methods beyond those in the Out of the Box supply-chain templates.

## Available workload types

When using the Out of the Box supply chain, the `apps.tanzu.vmware.com/workload-type` annotation selects which style of deployment is suitable for your application. The valid values are:



Type	Description	Indicators
<code>web</code>	Scalable web applications	<ul style="list-style-type: none"> <li>• Scales based on request load</li> <li>• Automatically exposed by HTTP Ingress</li> <li>• Does not perform background work</li> <li>• Works with Service Bindings</li> <li>• Stateless</li> <li>• Quick startup time</li> </ul>
<code>server</code>	Traditional applications	<ul style="list-style-type: none"> <li>• Provides HTTP or TCP services on the network</li> <li>• Exposed by external Ingress or LoadBalancer settings</li> <li>• Might perform background work from a queue</li> <li>• Works with Service Bindings</li> <li>• Fixed scaling, no disk persistence</li> <li>• Startup time not an issue</li> </ul>
<code>worker</code>	Background applications	<ul style="list-style-type: none"> <li>• Does not provide network services</li> <li>• Not exposed externally as a network service</li> <li>• Performs background work from a queue</li> <li>• Works with Service Bindings</li> <li>• Fixed scaling, no disk persistence</li> <li>• Startup time not an issue</li> </ul>

## Use web workloads

This topic tells you how to use the `web` workload type in Tanzu Application Platform (commonly known as TAP).

### Overview

The `web` workload type allows you to deploy web applications on Tanzu Application Platform. Using an application workload specification, you can turn source code into a scalable, stateless application that runs in a container with an automatically-assigned URL. This type of application is often called serverless, and is deployed using Knative.

The `web` workload type is suitable for modern stateless web applications that follow [the twelve-factor app](#) methodology and have the following characteristics:

- Perform all work through HTTP requests, including gRPC and WebSocket
- Do not perform work except when processing a request
- Start up quickly
- Store state in external databases instead of storing state locally

Applications using the `web` workload type have the following features:

- Automatic request-based scaling, including scale-to-zero
- Automatic URL provisioning and optional certificate provisioning
- Automatic health-check definitions, if not provided by a convention
- Blue-green application rollouts

When creating a workload with the `tanzu apps workload create` command, you can use the `--type=web` argument to select the `web` workload type. For more information, see [Use the web Workload Type](#) later in this topic.

You can also use the `apps.tanzu.vmware.com/workload-type:web` label in the YAML workload description to support this deployment type.

## Use the `web` workload type

The `tanzu-java-web-app` workload mentioned in [Deploy an app on Tanzu Application Platform](#) is a good match for the `web` workload type. It is a good match because it serves HTTP requests and does not perform any background processing.

You can experiment with the differences between the `web` and `server` workload types by changing the workload type. To change the workload type run:

```
tanzu apps workload apply tanzu-java-web-app --type=server
```

After changing the workload type to `server`, the application does not auto-scale or expose an external URL. For more information about the server workload type, see [Use Server workloads](#).

Switch back to the `web` workload by running:

```
tanzu apps workload apply tanzu-java-web-app --type=web
```

Use this to test which applications can function well as serverless web applications, and which are more suited to the `server` application style.

## Calling `web` workloads within a cluster

When a `web` workload type is created, a Knative service is deployed to the cluster. To access your application, you need the URL for the route created by the Knative Service. Obtain it by running one of these commands:

```
tanzu apps workload get WORKLOAD-NAME --namespace DEVELOPER-NAMESPACE
kubectl get ksvc WORKLOAD-NAME -n YOUR-DEVELOPER-NAMESPACE -ojsonpath="{status.addresses.url}"
```

When calling a Knative service, both the Service name and namespace are required. This behavior is distinct from `server` type workloads, which do not rely on the namespace name to establish service to service communication between applications within the same namespace.

## Example of service to service communication for `web` and `server` workloads

You have three applications deployed to the namespace called `dev-namespace`:

1. A `server` type workload named `server-workload`
2. A `web` type workload named `web-workload`
3. A pod running the `busybox` image with `curl`, named `busybox`

Open a shell to the running container of the `busybox` pod and send requests to the `server` and `web` workloads using `curl`. Specify the namespace for both, as follows:

```
kubectl exec busybox -n dev-namespace -- curl server-workload.dev-namespace.svc.cluster.local -v
```

```
kubectl exec busybox -n dev-namespace -- curl web-workload.dev-namespace.svc.cluster.local -v
```

## Use server workloads

This topic tells you how to use the `server` workload type in Tanzu Application Platform (commonly known as TAP).

### Overview

The `server` workload type allows you to deploy traditional network applications on Tanzu Application Platform.

Using an application workload specification, you can build and deploy application source code to a manually-scaled Kubernetes deployment which exposes an in-cluster Service endpoint. If required, you can use environment-specific LoadBalancer Services or Ingress resources to expose these applications outside the cluster.

The `server` workload is suitable for traditional applications, including HTTP applications, which have the following characteristics:

- Store state locally
- Run background tasks outside of requests
- Provide multiple network ports or non-HTTP protocols
- Are not a good match for the `web` workload type

An application using the `server` workload type has the following features:

- Does not natively autoscale, but you can use these applications with the Kubernetes Horizontal Pod Autoscaler.
- By default, is exposed only within the cluster using a `ClusterIP` service.
- Uses health checks if defined by a convention.
- Uses a rolling update pattern by default.

When creating a workload with the `tanzu apps workload create` command, you can use the `--type=server` argument to select the `server` workload type. For more information, see [Use the server Workload Type](#) later in this topic. You can also use the `apps.tanzu.vmware.com/workload-type:server` annotation in the YAML workload description to support this deployment type.

### Use the `server` workload type

The `spring-sensors-consumer-web` workload in [Bind an application workload to the service instance](#) in the Get started guide is a good match for the `server` workload type.

This is because it runs continuously to extract information from a RabbitMQ queue, and stores the resulting data locally in memory and presents it through a web UI.

In the Services Toolkit example in [Bind an application workload to the service instance](#), you can update the `spring-sensors-consumer-web` workload to use the `server` supply chain by changing the workload:

```
tanzu apps workload apply spring-sensors-consumer-web --type=server
```

This shows the change in the workload label and prompts you to accept the change. After the workload finishes the new deployment, there are a few differences:

- The workload no longer exposes a URL. It's available within the cluster as `spring-sensors-consumer-web` within the namespace, but you must use `kubectl port-forward service/spring-sensors-consumer-web 8080` to access the web service on port 8080.

You can set up a Kubernetes Ingress rule to direct traffic from outside the cluster to the workload. Use an Ingress rule to specify that specific host names or paths must be routed to the application. For more information about Ingress rules, see the [Kubernetes documentation](#)

- The workload no longer autoscales based on request traffic. For the `spring-sensors-consumer-web` workload, this means that it never spawns a second instance that consumes part of the request queue. Also, it does not scale down to zero instances.

## server-specific workload parameters

In addition to the common supply chain parameters, `server` workloads can expose one or more network ports from the application to the Kubernetes cluster by using the `ports` parameter. This parameter is a list of port objects, similar to a Kubernetes service specification.

If you do not configure the `ports` parameter, the applied container conventions in the cluster establishes the set of exposed ports.

The following configuration exposes two ports on the Kubernetes cluster under the `my-app` host name:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: my-app
  labels:
    apps.tanzu.vmware.com/workload-type: server
spec:
  params:
  - name: ports
    value:
  - containerPort: 2025
    name: smtp
    port: 25
  - port: 8080
  ...
```

This snippet configures:

- One service on port 25, which is redirected to port 2025 on the application
- One service on port 8080, which is routed to port 8080 on the application

You can set the `ports` parameter from the `tanzu apps workload create` command as `--param-yaml 'ports=[{"port": 8080}]'`.

The following values are valid within the `ports` argument:

Field	Value
<code>port</code>	The port on which the application is exposed to the rest of the cluster
<code>containerPort</code>	The port on which the application listens for requests. Defaults to <code>port</code> if not set.
<code>name</code>	A human-readable name for the port. Defaults to <code>port</code> if not set.

## Expose `server` workloads outside the cluster

This section tells you how to expose `server` workloads outside the cluster.

## Manual configuration for HTTP workloads

Expose HTTP `server` workloads by creating an Ingress resource and using cert-manager to provision TLS-signed certificates.

1. Use the `spring-sensors-consumer-web` workload as an example from [Bind an application workload to the service instance](#). Create the following Ingress:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: spring-sensors-consumer-web
  namespace: DEVELOPER-NAMESPACE
  annotations:
    cert-manager.io/cluster-issuer: tap-ingress-selfsigned
    ingress.kubernetes.io/force-ssl-redirect: "true"
    kubernetes.io/ingress.class: contour
    kubernetes.io/tls-acme: "true"
spec:
  tls:
  - secretName: spring-sensors-consumer-web
    hosts:
    - "spring-sensors-consumer-web.INGRESS-DOMAIN"
  rules:
  - host: "spring-sensors-consumer-web.INGRESS-DOMAIN"
    http:
      paths:
      - pathType: Prefix
        path: /
        backend:
          service:
            name: spring-sensors-consumer-web
            port:
              number: 8080
```

- o Replace `DEVELOPER-NAMESPACE` with your developer namespace
- o Replace `INGRESS-DOMAIN` with the domain name defined in `tap-values.yaml` during the installation
- o Set the annotation `cert-manager.io/cluster-issuer` to the `shared.ingress_issuer` value configured during installation or leave it as `tap-ingress-selfsigned` to use the default one
- o Update the port exposed by your `Service` resource, in the previous snippet it is set to `8080`

2. Access the `server` workload with https:

```
curl -k https://spring-sensors-consumer-web.INGRESS-DOMAIN
```

## Define a workload type that exposes `server` workloads outside the cluster

Tanzu Application Platform allows you to create new workload types. You start by adding an `Ingress` resource to the `server-template ClusterConfigTemplate` when this new type of workload is created.

1. Delete the `Ingress` resource previously created.
2. Install the `yq` CLI on your local machine.

3. Save the existing `server-template` in a local file by running:

```
kubectl get ClusterConfigTemplate server-template -o yaml > secure-server-template.yaml
```

4. Extract the `.spec.ytt` field from this file and create another file by running:

```
yq eval '.spec.ytt' secure-server-template.yaml > spec-ytt.yaml
```

5. In the next step, you add the `Ingress` resource snippet to `spec-ytt.yaml`. This step provides a sample `Ingress` resource snippet. Make the following edits before adding the `Ingress` resource snippet to `spec-ytt.yaml`:

- o Replace `INGRESS-DOMAIN` with the Ingress domain you set during the installation.
- o Set the annotation `cert-manager.io/cluster-issuer` to the `shared.ingress_issuer` value configured during installation or leave it as `tap-ingress-selfsigned` to use the default one.
- o This configuration is based on your workload service running on port `8080`.

The `Ingress` resource snippet looks like this:

```
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: #@ data.values.workload.metadata.name
  annotations:
    cert-manager.io/cluster-issuer: tap-ingress-selfsigned
    ingress.kubernetes.io/force-ssl-redirect: "true"
    kubernetes.io/ingress.class: contour
    kubernetes.io/tls-acme: "true"
    kapp.kl4s.io/change-rule: "upsert after upserting Services"
  labels: #@ merge_labels({ "app.kubernetes.io/component": "run", "carto.run/workload-name": data.values.workload.metadata.name })
spec:
  tls:
    - secretName: #@ data.values.workload.metadata.name
      hosts:
        - #@ data.values.workload.metadata.name + ".INGRESS-DOMAIN"
  rules:
    - host: #@ data.values.workload.metadata.name + ".INGRESS-DOMAIN"
      http:
        paths:
          - pathType: Prefix
            path: /
            backend:
              service:
                name: #@ data.values.workload.metadata.name
                port:
                  number: 8080
```

6. Add the `Ingress` resource snippet to the `spec-ytt.yaml` file and save. Look for the `Service` resource, and insert the snippet before the last `#@ end`. For example:

```
# THE TOP OF THE FILE IS NOT SHOWN

---
apiVersion: v1
kind: Service
metadata:
  name: #@ data.values.workload.metadata.name
```

```

    labels: #@ merge_labels({ "app.kubernetes.io/component": "run", "carto.run/workload-name": data.values.workload.metadata.name })
spec:
  selector: #@ data.values.config.metadata.labels
  ports:
    #@ hasattr(data.values.params, "ports") and len(data.values.params.ports) or
    assert.fail("one or more ports param must be provided.")
    #@ declared_ports = {}
    #@ if "ports" in data.values.params:
    #@   declared_ports = data.values.params.ports
    #@ else:
    #@   declared_ports = struct.encode([{"containerPort": 8080, "port": 8080, "name": "http"}])
    #@ end
    #@ for p in merge_ports(declared_ports, data.values.config.spec.containers):
    - #@ p
    #@ end

# NEW INGRESS RESOURCE
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: #@ data.values.workload.metadata.name
  annotations:
    cert-manager.io/cluster-issuer: tap-ingress-selfsigned
    ingress.kubernetes.io/force-ssl-redirect: "true"
    kubernetes.io/ingress.class: contour
    kubernetes.io/tls-acme: "true"
    kapp.k14s.io/change-rule: "upsert after upserting Services"
  labels: #@ merge_labels({ "app.kubernetes.io/component": "run", "carto.run/workload-name": data.values.workload.metadata.name })
spec:
  tls:
    - secretName: #@ data.values.workload.metadata.name
      hosts:
        - #@ data.values.workload.metadata.name + ".INGRESS-DOMAIN"
  rules:
    - host: #@ data.values.workload.metadata.name + ".INGRESS-DOMAIN"
      http:
        paths:
          - pathType: Prefix
            path: /
            backend:
              service:
                name: #@ data.values.workload.metadata.name
                port:
                  number: 8080
# END NEW INGRESS RESOURCE

#@ end

---
apiVersion: v1
kind: ConfigMap
metadata:
  name: #@ data.values.workload.metadata.name + "-server"
  labels: #@ merge_labels({ "app.kubernetes.io/component": "config" })
data:
  delivery.yml: #@ yaml.encode(delivery())

```

7. Add the snippet to the `.spec.ytt` property in `secure-server-template.yaml`:

```

SPEC_YTT=$(cat spec-ytt.yaml) yq eval -i '.spec.ytt |= strenv(SPEC_YTT)' secure-server-template.yaml

```

8. Change the name of the `ClusterConfigTemplate` to `secure-server-template` by running:

```
yq eval -i '.metadata.name = "secure-server-template"' secure-server-template.yaml
```

9. Create the new `ClusterConfigTemplate` by running:

```
kubectl apply -f secure-server-template.yaml
```

10. Verify the new `ClusterConfigTemplate` is in the cluster by running:

```
kubectl get ClusterConfigTemplate
```

Expected output:

```
kubectl get ClusterConfigTemplate
NAME                AGE
api-descriptors     82m
config-template     82m
convention-template 82m
secure-server-template 22s
server-template     82m
service-bindings    82m
worker-template     82m
```

11. Add the new workload type to the `tap-values.yaml`. The new workload type is named `secure-server` and the `cluster_config_template_name` is `secure-server-template`.

```
ootb_supply_chain_basic:
  supported_workloads:
    - type: web
      cluster_config_template_name: config-template
    - type: server
      cluster_config_template_name: server-template
    - type: worker
      cluster_config_template_name: worker-template
    - type: secure-server
      cluster_config_template_name: secure-server-template
```

12. Update your Tanzu Application Platform installation as follows:

```
tanzu package installed update tap -p tap.tanzu.vmware.com --values-file "/path/to/your/config/tap-values.yaml" -n tap-install
```

13. Give privileges to the `deliverable` role to manage `Ingress` resources:

```
cat <<EOF | kubectl apply -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: deliverable-with-ingress
  labels:
    apps.tanzu.vmware.com/aggregate-to-deliverable: "true"
rules:
- apiGroups:
  - networking.k8s.io
  resources:
  - ingresses
  verbs:
  - get
  - list
  - watch
```



```
- create
- patch
- update
- delete
- deletecollection
EOF
```

14. Update the workload type to `secure-server`:



#### Note

If you created the `Ingress` resource manually in the previous section, delete it before this.

```
tanzu apps workload apply spring-sensors-consumer-web --type=secure-server
```

15. After the process finishes, you see the resources Deployment, Service, and Ingress by running:

```
kubectl get ingress,svc,deploy -l carto.run/workload-name=spring-sensors-consumer-web
```

Expected output:

```
kubectl get ingress,svc,deploy -l carto.run/workload-name=tanzu-java-web-app-js
NAME                                     CLASS      HOSTS
ADDRESS          PORTS      AGE
ingress.networking.k8s.io/spring-sensors-consumer-web  <none>    spring-sensors
-consumer-web.INGRESS-DOMAIN  34.111.111.111  80, 443  37s

NAME          TYPE          CLUSTER-IP      EXTERNAL-IP
PORT(S)      AGE
service/spring-sensors-consumer-web  ClusterIP     10.32.15.194    <none>
8080/TCP     36m

NAME          READY    UP-TO-DATE    AVAILABLE
AGE
deployment.apps/spring-sensors-consumer-web  1/1      1              1
37s
```

16. Access your `secure-server` workload with HTTPS by running:

```
curl -k https://spring-sensors-consumer-web.INGRESS-DOMAIN
```

## Use worker workloads

This topic tells you how to create and install a supply chain for the `worker` workload type in Tanzu Application Platform (commonly known as TAP).

### Overview

The `worker` workload type allows you to deploy applications that run continuously without network input on Tanzu Application Platform. Using an application workload specification, you can build and deploy application source code to a manually scaled Kubernetes deployment with no network exposure.

The `worker` workload is a good match for applications that manage their own work by reading from a worker or a background scheduled time source, and don't expose any network interfaces.

An application using the `worker` workload type has the following features:

- Does not natively auto-scale but you can use it with the Kubernetes Horizontal Pod Autoscaler
- Does not expose any network services
- Uses health checks if defined by a convention
- Uses a rolling update pattern by default

When creating a workload with `tanzu apps workload create`, you can use the `--type=worker` argument to select the `worker` workload type. For more information, see the [Use the worker Workload Type](#) section. You can also use the `apps.tanzu.vmware.com/workload-type:worker` annotation in the YAML workload description to support this deployment type.

## Use the `worker` workload type

The `spring-sensors-producer` workload in the example in [Consume services on Tanzu Application Platform](#) is a good match for the `worker` workload type. This is because it runs continuously without a UI to report sensor information to a RabbitMQ topic.

If you followed the Services Toolkit example, you can update the `spring-sensors-producer` to use the `worker` supply chain by changing the workload type. To do so, run:

```
tanzu apps workload apply spring-sensors-producer --type=worker
```

This shows a difference in the workload label, and prompts you to accept the change. After the workload finishes the new deployment, there are a few differences:

- The workload no longer has a URL. Because the workload does not present a web UI, this more closely matches the original application intent.
- The workload no longer auto-scales based on request traffic. For the `spring-sensors-producer` workload, this means that it does not scale down to zero instances when there is no request traffic.

## Parameter reference

This topic tells you about the default supply chains and templates provided by Tanzu Application Platform (commonly known as TAP). This topic describes the `workload.spec.params` parameters that are configured in workload objects, and the `deliverable.spec.params` parameters that are configured in the deliverable object.

## Workload Parameter Reference

The supply chains and templates provided by the Out of the Box packages contain a series of parameters that customize supply chain behavior. This section describes the `workload.spec.params` parameters that can be configured in workload objects.

The following table provides a list of supply chain resources organized by the resource in the supply chain where they are used. Some of these resources might not be applicable depending on the supply chain in use.

## List of Supply Chain Resources for Workload Object

Supply Chain Resource	Output Type	Purpose	Basic	Testing	Scanning
<a href="#">source-provider</a>	Source	Fetches source code	Yes	Yes	Yes
<a href="#">source-tester</a>	Source	Tests source code	No	Yes	Yes
<a href="#">source-scanner</a>	Source	Scans source code	No	No	Yes
<a href="#">image-provider</a>	Image	Builds application container image	Yes	Yes	Yes
<a href="#">image-scanner</a>	Image	Scans application container image	No	No	Yes
<a href="#">config-provider</a>	Podtemplate spec	Tailors a pod spec based on the application image and conventions set up in the cluster	Yes	Yes	Yes
<a href="#">app-config</a>	Kubernetes configuration	Creates Kubernetes config files (knative service/deployment - depending on workload type)	Yes	Yes	Yes
<a href="#">service-bindings</a>	Kubernetes configuration	Adds service bindings to the set of config files	Yes	Yes	Yes
<a href="#">api-descriptors</a>	Kubernetes configuration	Adds api descriptors to the set of config files	Yes	Yes	Yes
<a href="#">config-writer</a>	Kubernetes configuration	Writes configuration to a destination (git or registry) for further deployment to a run cluster	Yes	Yes	Yes
<a href="#">deliverable</a>	Kubernetes configuration	Writes deliverable content to be extracted for use in a run cluster	Yes	Yes	Yes

For information about supply chains, see:

- [Out of the Box Supply Chain Basic](#)
- [Out of the Box Supply Chain Testing](#)
- [Out of the Box Supply Chain Testing Scanning](#)

## source-provider

The `source-provider` resource in the supply chain creates objects that fetch either source code or pre-compiled Java applications depending on how the workload is configured. For more information, see [Building from Source](#).

### GitRepository

Use `gitrepository` when fetching source code from Git repositories. This resource makes further resources available in the supply chain, such as the contents of the Git repository as a tarball available in the cluster.

Parameters:

Parameter name	Meaning	Example
<code>gitImplementation</code>	VMware recommends that you use the underlying library for fetching the source code. Either <code>libgit2</code> , required for Azure DevOps, or <code>go-git</code> .	<pre> - name:   gitImplementation       value:   libgit2 </pre>

Parameter name	Meaning	Example
<code>gitops_ssh_secret</code>	The name of the secret in the same namespace as the `Workload` used for providing credentials for fetching source code from the Git repository. For more information, see <a href="#">Git authentication</a> .	<pre>- name:   gitops_ssh_secret   value:   git-credentials</pre>

It might not be necessary to change the default Git implementation, but some providers such as Azure DevOps, require you to use `libgit2` as the server-side implementation provides support only for `git's v2 protocol`.

For information about the features supported by each implementation, see [Git implementation](#) in the Flux documentation.

For information about how to create a workload that uses a GitHub repository as the provider of source code, see [Create a workload from GitHub repository](#).

For more information about `GitRepository` objects, see [Git Repository](#) in the Flux documentation.

## ImageRepository

Use the `ImageRepository` when fetching source code from container images. It makes the contents of the container image available as a tarball to further resources in the supply chain. The contents of the container image are fetched by using Git or Maven.

For more information, see [Create a workload from local source code](#).

Parameters:

Parameter Name	Meaning	Example
<code>serviceAccount</code>	The name of the service account (in the same namespace as the workload) to use to provide the credentials to `ImageRepository` for fetching the container images.	<pre>- name:   serviceAccount   value:   default</pre>

The `--service-account` flag sets the `spec.serviceAccountName` key in the workload object. To configure the `serviceAccount` parameter, use `--param serviceAccount=SERVICE-ACCOUNT`.

For information about custom resource details, see the [ImageRepository](#) reference topic.

For information about how to use `ImageRepository` with the Tanzu CLI, see [Create a workload](#).

## MavenArtifact

When carrying pre-built Java artifacts, `MavenArtifact` makes the artifact available to further resources in the supply chain as a tarball. You can wrap the tarball as a container image for further deployment. Differently from `git` and `image`, its configuration is solely driven by parameters in the workload.

Parameters:

Parameter Name	Meaning	Example
<code>maven</code>	Points to the maven artifact to fetch and the polling interval.	<pre> - name: maven   value:     artifactId: springboot-initial     groupId: com.example     version: RELEASE     classifier: sources # optional   type: # optional   artifactRetryTimeout: 1m0s # optional </pre>

For information about the custom resource, see the [MavenArtifact reference documentation](#).

For information about how to use the custom resource with the `tanzu apps workload` CLI plug-in, see [Create a workload from Maven repository artifact](#).

## source-tester

The `source-tester` resource is in `ootb-supply-chain-testing` and `ootb-supply-chain-testing-scanning`. This resource is responsible for instantiating a Tekton `PipelineRun` object that calls the execution of a Tekton Pipeline, in the same namespace as the workload, whenever its inputs change. For example, the source code revision that you want to test changes.

A `Runnable` object is instantiated to ensure that there's always a run for a particular set of inputs. The parameters are passed from the workload down to `Runnable`'s Pipeline selection mechanism through `testing_pipeline_matching_labels` and the execution of the `PipelineRuns` through `testing_pipeline_params`.

Parameters:

Parameter name	Meaning	Example
<code>testing_pipeline_matching_labels</code>	The set of labels to use when searching for Tekton Pipeline objects in the same namespace as the workload. By default, a Pipeline labeled as <code>'apps.tanzu.vmware.com/pipeline: test'</code> is selected, but when using this parameter, it's possible to override the behavior.	<pre> - name: testing_pipeline_matching_labels   value:     app s.tanzu.com/pipeline: test     my.company/language: golang </pre>
<code>testing_pipeline_params</code>	The set of extra parameters, aside from <code>'source-url'</code> and <code>'source-revision'</code> , to pass to the Tekton Pipeline. The Tekton Pipeline <b>must</b> declare both the required parameters <code>'source-url'</code> and <code>'source-revision'</code> and the extra ones declared in this table.	<pre> - name: testing_pipeline_params   value:     - name: e: verbose       value: e: true </pre>

For information about how to set up the Workload namespace for testing with Tekton, see [Out of the Box Supply Chain with Testing](#).

For information about how to use the parameters to customize this resource to test using a Jenkins cluster, see [Out of the Box Supply Chain with Testing on Jenkins](#).

## source-scanner

The `source-scanner` resource is available in `ootb-supply-chain-testing-scanning`. It scans the source code that is tested by pointing a `SourceScan` object at the same source code as the tests.

You can customize behavior for both [CVEs evaluation](#) with parameters.

Parameters:

Parameter name	Meaning	Example
<code>scanning_source_template</code>	The name of the <code>ScanTemplate</code> object (in the same namespace as the workload) to use for running the scans against the source code.	<pre>- name: scanning_source_template   value: private-source-scan-template</pre>
<code>scanning_source_policy</code>	The name of the <code>ScanPolicy</code> object (in the same namespace as the workload) to use when evaluating the scan results of a source scan.	<pre>- name: scanning_source_policy   value: allowlist-policy</pre>

For more information, see [Out of the Box Supply Chain with Testing and Scanning](#) for details about how to set up the workload namespace with the `ScanPolicy` and `ScanTemplate` required for this resource, and [SourceScan reference](#) for details about the `SourceScan` custom resource.

For information about how the artifacts found during scanning are catalogued, see [Supply Chain Security Tools for Tanzu – Store](#).

## image-provider

The `image-provider` in the supply chains provides a container image carrying the application already built to further resources.

Different semantics apply, depending on how the workload is configured, for example, if using [pre-built images](#) or [building from source](#):

- pre-built: an `ImageRepository` object is created aiming at providing a reference to the latest image found matching the name as specified in `workload.spec.image`
- building from source: an image builder object is created (either Kpack's `Image` or a `Runnable` for creating Tekton TaskRuns for building images from Dockerfiles)

### Kpack Image

Use the Kpack Image object to build a container image out of source code or pre-built Java artifact.

This makes the container image available to further resources in the supply chain through a content addressable image reference that's carried to the final deployment objects unchanged. For more information, see [Tanzu Build Service](#).

Parameters:

Parameter name	Meaning	Example
<code>serviceAccount</code>	The name of the serviceaccount (in the same namespace as the workload) to use for providing credentials to `Image` for pushing the container images it builds to the configured registry.	<pre>- name: serviceAccount   value: default</pre>
<code>clusterBuilder</code>	The name of the Kpack cluster builder to use in the Kpack Image object created.	<pre>- name: clusterBuilder   value: nodejs-cluster-builder</pre>
<code>buildServiceBindings</code>	The definition of a list of service bindings to use at build time. For example, providing credentials for fetching dependencies from repositories that require credentials.	<pre>- name: buildServiceBindings   value:     - name: settings-xml       kind: Secret       apiVersion: v1</pre>
<code>live-update</code>	Enables the use of Tilt's live-update function.	<pre>- name: live-update   value: "true"</pre>

The `--service-account` flag sets the `spec.serviceAccountName` key in the workload object. To configure the `serviceAccount` parameter, use `--param serviceAccount=SERVICE-ACCOUNT`.

For information about the integration with Tanzu Build Service, see [Tanzu Build Service Integration](#).

For information about `live-update`, see [Developer Conventions](#) and [Overview of Tanzu Developer Tools for IntelliJ](#).

For information about using Kpack builders with `clusterBuilder`, see [Builders](#).

For information about `buildServiceBindings`, see [Service Bindings](#).

### Runnable (TaskRuns for Dockerfile-based builds)

To perform Dockerfile-based builds, all the supply chains instantiate a Runnable object that instantiates Tekton TaskRun objects to call the execution of `kaniko` builds.

Parameters:

Parameter name	Meaning	Example
<code>dockerfile</code>	The relative path to the Dockerfile file in the build context.	<code>./Dockerfile</code>

Parameter name	Meaning	Example
<code>docker_build_context</code>	The relative path to the directory where the build context is.	<pre>.</pre>
<code>docker_build_extra_args</code>	List of flags to pass directly to Kaniko, such as providing arguments to a build.	<pre>- --build-arg=FOO =BAR</pre>

For information about how to use Dockerfile-based builds and limitations associated with the function, see [Dockerfile-based builds](#).

### Pre-built image (ImageRepository)

For applications that already have their container images built outside the supply chains, such as providing an image reference under `workload.spec.image`, an `ImageRepository` object is created to keep track of any images pushed under that name.

This makes the content-addressable name, such as the image name containing the digest, available for further resources in the supply chain.

Parameters:

Parameter name	Meaning	Example
<code>serviceAccount</code>	The name of the serviceaccount (in the same namespace as the workload) to use for providing the credentials to <code>ImageRepository</code> for fetching the container images.	<pre>- name: serviceAccount value: default</pre>

The `--service-account` flag sets the `spec.serviceAccountName` key in the workload object. To configure the `serviceAccount` parameter, use `--param serviceAccount=...`

For information about the `ImageRepository` resource, see the [ImageRepository reference documentation](#). For information about the prebuild image function, see [Using a prebuilt image](#).

## image-scanner

The `image-scanner` resource is included only in `ootb-supply-chain-testing-scanning`.

This resource scans a container image (either built by using the supply chain or prebuilt), persisting the results in the store, and gating the image from moving forward in case the CVEs found are not compliant with the `ScanPolicy` referenced by the `ImageScan` object create for doing so.

Parameters:

Parameter name	Meaning	Example
<code>scanning_image_template</code>	The name of the <code>ScanTemplate</code> object (in the same namespace as the workload) to use for running the scans against a container image.	<pre>- name: scanni ng_image_template value: priva te-image-scan-templa te</pre>



Parameter name	Meaning	Example
<code>scanning_image_policy</code>	The name of the ScanPolicy object (in the same namespace as the workload) to use when evaluating the scan results of an image scan.	<pre> - name: scanning_image_policy   value: allow list-policy                     </pre>

For information about the ImageScan custom resource, see [ImageScan reference](#).

For information about how the artifacts found during scanning are catalogued, see [Supply Chain Security Tools for Tanzu – Store](#).

## config-provider

The `config-provider` resource in the supply chains generates a PodTemplateSpec to use in application configs, such as Knative services and deployments, to represent the desired pod configuration to instantiate to run the application in containers. For more information, see [PodTemplateSpec](#) in the Kubernetes documentation.

The `config-provider` resource manages a [PodIntent](#) object that represents the intention of having PodTemplateSpec enhanced with conventions installed in the cluster whose final representation is then passed forward to other resources to form the final deployment configuration.

Parameters:

Parameter name	Meaning	Example
<code>serviceAccount</code>	The name of the serviceaccount (in the same namespace as the workload) to use for providing the necessary credentials to 'PodIntent' for fetching the container image to inspect the metadata to pass to convention servers and the serviceAccountName set in the podtemplatespec.	<pre> - name: serviceAccount   value: default                     </pre>
<code>annotations</code>	An extra set of annotations to pass down to the PodTemplateSpec.	<pre> - name: annotations   value:     name: my-application     version: v1.2.3     team: store                     </pre>
<code>debug</code>	Put the workload in debug mode.	<pre> - name: debug   value: "true"                     </pre>

Parameter name	Meaning	Example
<code>live-update</code>	Enable live-updating of the code (for innerloop development).	<pre>- name: live-update   value: "true"</pre>

The `--service-account` flag sets the `spec.serviceAccountName` key in the workload object. To configure the `serviceAccount` parameter, use `--param serviceAccount=SERVICE-ACCOUNT`.

For more information about the controller behind `PodIntent`, see [Cartographer Conventions](#).

For more details about the two convention servers enabled by default in Tanzu Application Platform installations, see [Developer Conventions](#) and [Spring Boot conventions](#).

## app-config

The `app-config` resource prepares a ConfigMap with the Kubernetes configuration that is used for instantiating an application in the form of a particular workload type in a cluster.

The resource is configured in the supply chain to allow, by default, three types of workloads with the selection of which workload type to apply based on the labels set in the workload object created by the developer:

- `apps.tanzu.vmware.com/workload-type: web`
- `apps.tanzu.vmware.com/workload-type: worker`
- `apps.tanzu.vmware.com/workload-type: server`

Only the `server` workload type has the following configurable parameters:

Parameter name	Meaning	example
<code>ports</code>	The set of network ports to expose from the application to the Kubernetes cluster.	<pre>- name: ports   value:     - containerPort: 2025       name: smtp       port: 25</pre>

For more information about the three different types of workloads, see [workload types](#). For a more detailed overview of the `ports` parameter, see [server-specific Workload parameters](#).

## service-bindings

The `service-bindings` resource adds [ServiceBindings](#) to the set of Kubernetes configuration files to promote for deployment.

Parameters:

Parameter name	Meaning	Example
<code>annotations</code>	The extra set of annotations to pass down to the ServiceBinding and ResourceClaim objects.	<pre> - name: annotations   value:     name: my-application     version: v1.2.3     team: store </pre>

For an example, see [--service-ref](#) in Tanzu CLI documentation.

For an overview of the function, see [Consume services on Tanzu Application Platform](#).

## api-descriptors

The `api-descriptor` resource adds an `APIDescriptor` to the set of Kubernetes objects to deploy. This enables API auto registration.

Parameters:

Parameter name	Meaning	Example
<code>annotations</code>	An extra set of annotations to pass down to the APIDescriptor object.	<pre> - name: annotations   value:     name: my-application     version: v1.2.3     team: store </pre>
<code>api_descriptor</code>	Information used to fill the state that you want of the APIDescriptor object (its spec).	<pre> - name: api_descriptor   value:     type: openapi     location:       baseURL: http://petclinic-hard-coded.my-apps.tapdemo.vmware.com/       path: "/v3/api"     owner: team-petclinic     system: pet-clinics     description: "example" </pre>

The workload must include the `apis.apps.tanzu.vmware.com/register-api: "true"` label to activate this function.

For more details about API auto registration, see [Use API Auto Registration](#).

## config-writer (git or registry)

The `config-writer` resource is responsible for performing the last mile of the supply chain: persisting in an external system (registry or Git repository) the Kubernetes configuration generated throughout the supply chain.

There are three methods:

- Publishing the configuration to a container image registry

- Publishing the configuration to a Git repository by using the push of a commit
- Publishing the configuration to a Git repository by pushing a commit and opening a pull request

For more information about the different modes of operation, see [Gitops vs RegistryOps](#).

## deliverable

The `deliverable` resource creates a `deliverable` object that represents the intention of delivering to the cluster the configurations that are produced by the supply chain.

Parameters:

Parameter name	Meaning	Example
<code>serviceAccount</code>	The name of the serviceaccount (in the same namespace as the deliverable) to use for providing the necessary permissions to create the children objects for deploying the objects created by the supply chain to the cluster.	<pre>- name: serviceAccount   value: default</pre>

The `--service-account` flag sets the `spec.serviceAccountName` key in the workload object. To configure the `serviceAccount` parameter, use `--param serviceAccount=SERVICE-ACCOUNT`.

On build clusters where a corresponding `ClusterDelivery` doesn't exist, the deliverable takes no effect (similarly to a workload without a `SupplyChain`, no action is taken).

## Deliverable Parameters Reference

The deliverable object applies the configuration produced by the resources defined by a `ClusterSupplyChain` to a Kubernetes cluster.

This section describes the `deliverable.spec.params` parameters that can be configured in the deliverable object. The following section describes the two resources defined in the `ClusterDelivery` resources section. These are part of the `ootb-delivery-basic` package:

### List of Cluster Delivery Resources for Deliverable Object

Cluster Delivery Resource	Output Type	Purpose
<a href="#">source provider</a>	Source	Fetches the Kubernetes configuration file from Git repository or image registry
<a href="#">app deployer</a>	Source	Applies configuration produced by a supply chain to the cluster

For information about the `ClusterDelivery` shipped with `ootb-delivery-basic`, and the templates used by it, see:

- [Out of the Box Delivery Basic](#)
- [Out of the Templates](#)

For information about the use of the deliverable object in a multicluster environment, see [Getting started with multicluster Tanzu Application Platform](#).

For reference information about deliverable, see [Deliverable and Delivery custom resources](#) in the Cartographer documentation.

## source-provider

The `source-provider` resource in the basic ClusterDelivery creates objects that continuously fetch Kubernetes configuration files from a Git repository or container image registry so that it can apply those to the cluster.

Regardless of where it fetches that Kubernetes configuration from (Git repository or image registry), it exposes those files to further resources along the ClusterDelivery as a tarball.

### GitRepository

A GitRepository object is instantiated when `deliverable.spec.source.git` is configured to continuously look for a Kubernetes configuration pushed to a Git repository, making it available for resources in the ClusterDelivery.

Parameters:

Parameter name	Meaning	Example
<code>gitImplementation</code>	VMware recommends that you use the underlying library for fetching the source code. Either <code>libgit2</code> , required for Azure DevOps, or <code>go-git</code> .	<pre> - name:   gitImplementation     value:   e: libgit2 </pre>
<code>gitops_ssh_secret</code>	The name of the secret in the same namespace as the `deliverable` used for providing credentials for fetching Kubernetes configuration files from the Git repository pointed at. See [Git authentication](../scc/git-auth.md).	<pre> - name:   gitops_ssh_secret     value:   e: git-credentials </pre>

It might not be necessary to change the default Git implementation but some providers, such as Azure DevOps, require you to use `libgit2` as the server-side implementation provides support only for `git's v2 protocol`.

For information about the features supported by each implementation, see [git implementation](#) in the Flux documentation.

For information about how to create a workload that uses a GitHub repository as the provider of source code, see [Create a workload from GitHub repository](#).

For information about GitRepository objects, see [GitRepository](#).

### ImageRepository

An ImageRepository object is instantiated when `deliverable.spec.source.image` is configured to continuously look for Kubernetes configuration files pushed to a container image registry as opposed to a Git repository.

Parameters:

Parameter name	Meaning	Example
<code>serviceAccount</code>	The name of the service account, in the same namespace as the deliverable, you want to use to provide the necessary permissions for `kapp-controller` to deploy the objects to the cluster.	<pre>- name:   serviceAccount   value:   default</pre>

The `--service-account` flag sets the `spec.serviceAccountName` key in the deliverable object. To configure the `serviceAccount` parameter, use `--param serviceAccount=SERVICE-ACCOUNT`.

For information about custom resource details, see the [ImageRepository reference documentation](#).

## app deployer

The `app-deploy` resource in the ClusterDelivery applies the Kubernetes configuration that is built by the supply chain, pushed to either a Git repository or image repository, and applied to the cluster.

### App

Regardless of where the configuration comes from, an `App` object is instantiated to deploy the set of Kubernetes configuration files to the cluster.

Parameters:

Parameter name	Meaning	Example
<code>serviceAccount</code>	The name of the service account, in the same namespace as the deliverable, you want to use to provide the necessary privileges for `App` to apply the Kubernetes objects to the cluster.	<pre>- name:   serviceAccount   value:   default</pre>
<code>gitops_sub_path</code> (deprecated)	The subdirectory within the configuration bundle used for looking up the files to apply to the Kubernetes cluster.	<pre>- name:   gitops_sub_path   value:   ./config</pre>

The `gitops_sub_path` parameter is deprecated. Use `deliverable.spec.source.subPath` instead.

The `--service-account` flag sets the `spec.serviceAccountName` key in the deliverable object.

To configure the `serviceAccount` parameter, use `--param serviceAccount=SERVICE-ACCOUNT`.

For details about RBAC and how `kapp-controller` uses the ServiceAccount provided to it using the `serviceAccount` parameter in the `deliverable` object, see [the Carvel documentation](#).

## Use functions (Beta)

This topic tells you how to create and deploy a HTTP or CloudEvent function from an Application Accelerator starter template in an online or air-gapped environment on Tanzu Application Platform (commonly known as TAP).

**Important**

Function Buildpacks for Knative and the corresponding Application Accelerator starter templates for Python and Java are deprecated and will be removed in Tanzu Application Platform v1.7. This beta product will not receive any future updates or patches.

## Overview

The function experience on Tanzu Application Platform enables you to deploy functions, use starter templates to bootstrap your function, and write only the code that matters to your business. You can run a single CLI command to deploy your 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 function buildpack manages the webserver. This means that you can focus on your business logic.
- 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.

## Supported languages and frameworks

For HTTP and CloudEvents:

Language/framework	HTTP	CloudEvents
Java	✓	✓
Python	✓	✓
NodeJS	✓	N/A

For REST API:

Language/framework	GET	POST
Java	N/A	✓
Python	✓	✓
NodeJS	✓	✓

## Prerequisites

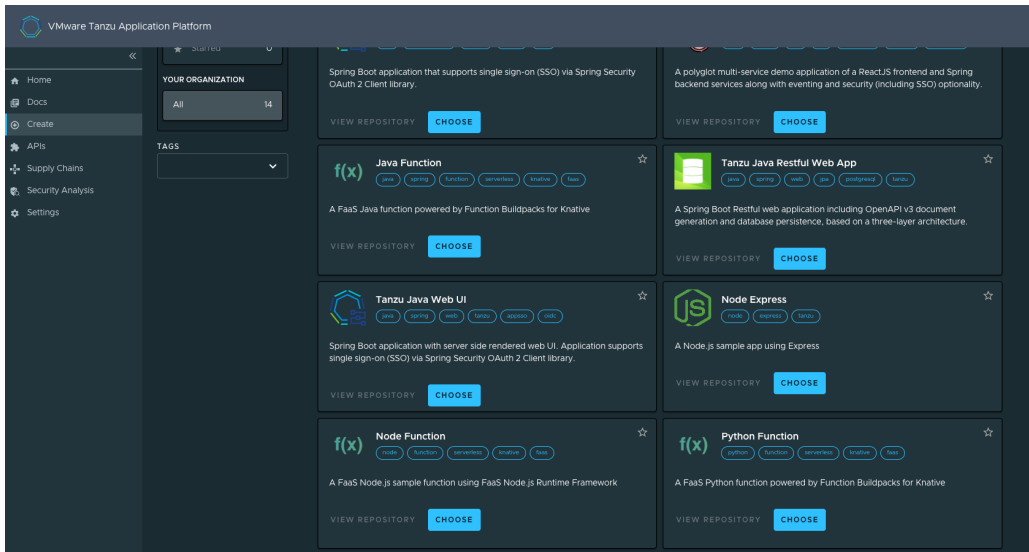
Before using function workloads, follow all instructions to install Tanzu Application Platform for your environment:

- [Installing Tanzu Application Platform online](#)
- [Installing Tanzu Application Platform in an air-gapped environment](#)

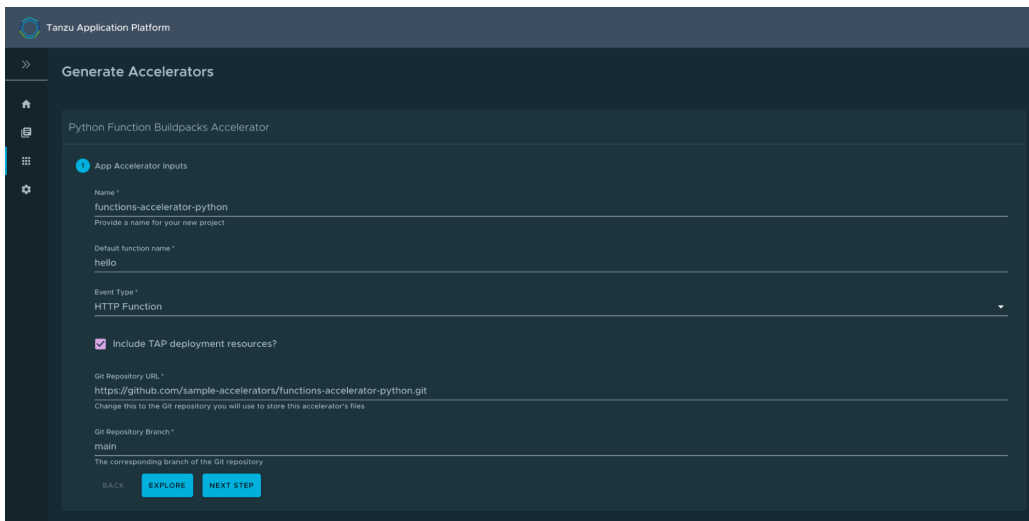
## Create a function project from an accelerator

To 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 accelerator in the language or framework of your choice and click **CHOOSE**.
3. Provide a name for your function project and your function.
4. If you are creating a Java function, select a project type.
5. Provide a Git repository to store the files for the accelerator.
6. Click **NEXT STEP**, verify the provided information, and then click **CREATE**.



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



- 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, to generate a function project using an accelerator template and then download the project artifacts as a ZIP file:

- Verify that you have added the function accelerator template to the application accelerator server by running:

```
tanzu accelerator list
```

- 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.

- 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
```

- 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

To deploy and verify 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 SOURCE-IMAGE \
--type web \
--yes
--namespace YOUR-DEVELOPER-NAMESPACE
--build-env 'BP_FUNCTION=func.hello'
```

Where:

- `SOURCE-IMAGE` is a writable repository in your registry in the form `REGISTRY/IMAGE:TAG`.
    - 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”.
  - `YOUR-DEVELOPER-NAMESPACE` is the namespace you configured earlier.
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 -
--namespace YOUR-DEVELOPER-NAMESPACE
```

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

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 --namespace YOUR-DEVELOPER
-NAMESPACE
```

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

4. (Optional) You can test your function using a `curl` command. To do so, you must have `curl` installed on your computer. Java function POST example:

```
curl -w'\n' URL-FROM-YOUR-WORKLOAD-KNATIVE-SERVICES-SECTION \
-H "Content-Type: application/json" \
-d '{"firstName":"John", "lastName":"Doe"}'
```

For language support for the REST API, see [Supported languages and frameworks](#) earlier in this topic.

## Use functions (Beta)

This topic tells you how to create and deploy a HTTP or CloudEvent function from an Application Accelerator starter template in an online or air-gapped environment on Tanzu Application Platform (commonly known as TAP).



### Important

Function Buildpacks for Knative and the corresponding Application Accelerator starter templates for Python and Java are deprecated and will be removed in Tanzu Application Platform v1.7. This beta product will not receive any future updates or patches.

## Overview

The function experience on Tanzu Application Platform enables you to deploy functions, use starter templates to bootstrap your function, and write only the code that matters to your business. You can run a single CLI command to deploy your 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 function buildpack manages the webserver. This means that you can focus on your business logic.
- 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.

## Supported languages and frameworks

For HTTP and CloudEvents:

Language/framework	HTTP	CloudEvents
Java	✓	✓
Python	✓	✓
NodeJS	✓	N/A

For REST API:

Language/framework	GET	POST
Java	N/A	✓
Python	✓	✓
NodeJS	✓	✓

## Prerequisites

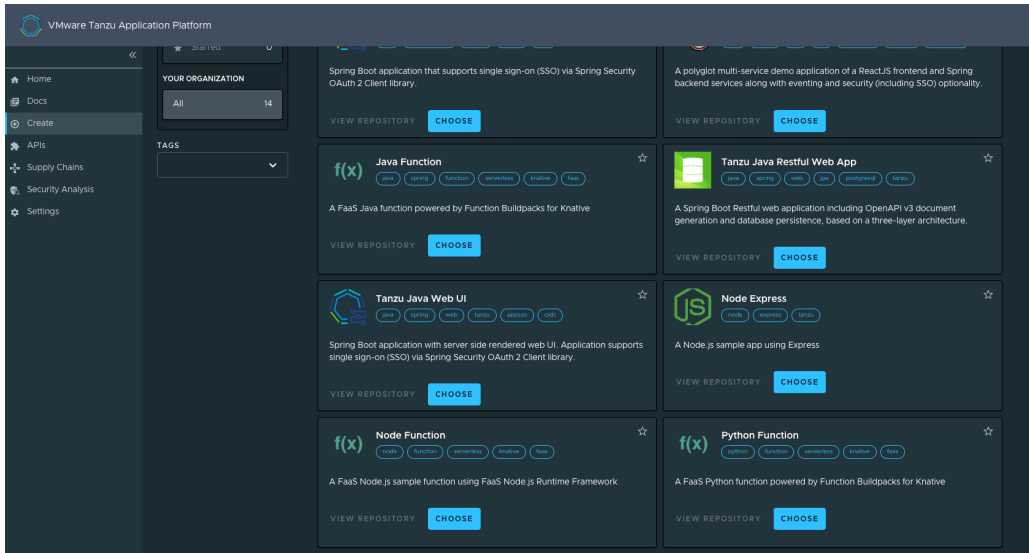
Before using function workloads, follow all instructions to install Tanzu Application Platform for your environment:

- Installing Tanzu Application Platform online
- Installing Tanzu Application Platform in an air-gapped environment

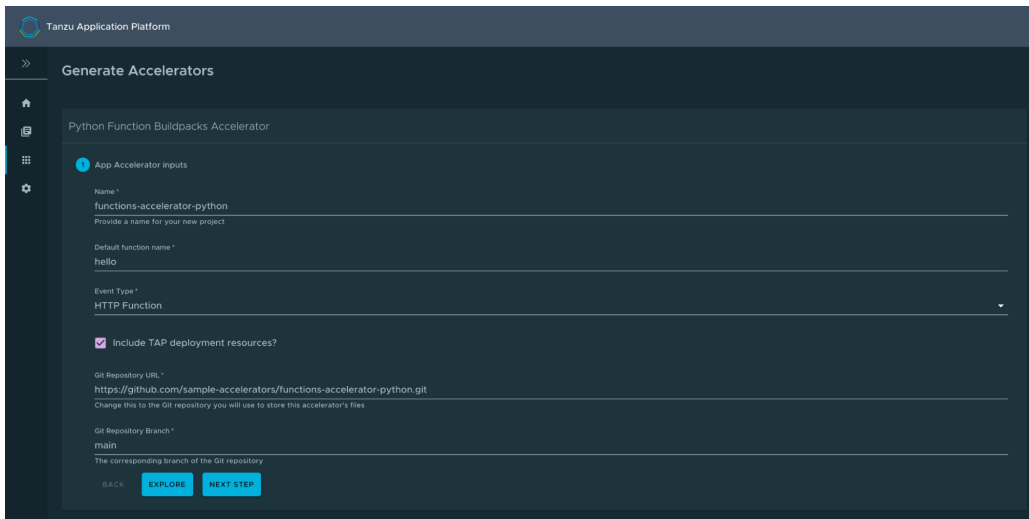
## Create a function project from an accelerator

To 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 accelerator in the language or framework of your choice and click **CHOOSE**.
3. Provide a name for your function project and your function.
4. If you are creating a Java function, select a project type.
5. Provide a Git repository to store the files for the accelerator.
6. Click **NEXT STEP**, verify the provided information, and then click **CREATE**.



7. After the Task Activity processes complete, click **DOWNLOAD ZIP FILE**.
8. 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, to generate a function project using an accelerator template and then download the project artifacts as a ZIP file:

1. Verify 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:
  - o For installations configured with a shared ingress, use `https://accelerator.DOMAIN` where `DOMAIN` is provided in the values file for the accelerator configuration.
  - o 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.

- o 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:

- o `ACCELERATOR-NAME` is the name of the function accelerator template you want to use.
- o `FUNCTION-NAME` is the name of your function project.
- o `TYPE` is the interface you want to use for your function. Available options are `http` or `cloudevents`. CloudEvents is experimental.
- o `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

To deploy and verify 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 SOURCE-IMAGE \
--type web \
--yes
--namespace YOUR-DEVELOPER-NAMESPACE
--build-env 'BP_FUNCTION=func.hello'
```

Where:

- `SOURCE-IMAGE` is a writable repository in your registry in the form `REGISTRY/IMAGE:TAG`.
    - 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”.
  - `YOUR-DEVELOPER-NAMESPACE` is the namespace you configured earlier.
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 -
-nnamespace YOUR-DEVELOPER-NAMESPACE
```

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

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 --namespace YOUR-DEVELOPER
-NAMESPACE
```

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

4. (Optional) You can test your function using a `curl` command. To do so, you must have `curl` installed on your computer. Java function POST example:

```
curl -w'\n' URL-FROM-YOUR-WORKLOAD-KNATIVE-SERVICES-SECTION \
-H "Content-Type: application/json" \
-d '{"firstName":"John", "lastName":"Doe"}'
```

For language support for the REST API, see [Supported languages and frameworks](#) earlier in this topic.

# Troubleshoot Tanzu Application Platform

These topics provide you with troubleshooting information to help resolve issues with your Tanzu Application Platform (commonly known as TAP):

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

## Troubleshoot Tanzu Application Platform

These topics provide you with troubleshooting information to help resolve issues with your Tanzu Application Platform (commonly known as TAP):

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

## Troubleshoot installing Tanzu Application Platform

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

### Developer cannot be verified when installing Tanzu CLI on macOS

You see the following error when you run Tanzu CLI commands, for example `tanzu version`, on macOS:

```
"tanzu" cannot be opened because the developer cannot be verified
```

#### Explanation

Security settings are preventing installation.

#### Solution

To resolve this issue:

1. Click **Cancel** in the macOS prompt window.
2. Open **System Preferences > Security & Privacy**.
3. Click **General**.
4. Next to the warning message for the Tanzu binary, click **Allow Anyway**.
5. Enter your system username and password in the macOS prompt window to confirm the changes.
6. In the terminal window, run:

```
tanzu version
```

- In the macOS prompt window, click **Open**.

## Access `.status.usefulErrorMessage` details

When installing Tanzu Application Platform, you receive an error message that includes the following:

```
(message: Error (see .status.usefulErrorMessage for details))
```

### Explanation

A package fails to reconcile and you must access the details in `.status.usefulErrorMessage`.

### Solution

Access the details in `.status.usefulErrorMessage` by running:

```
kubectl get packageinstall PACKAGE-NAME -n tap-install -o yaml
```

Where `PACKAGE-NAME` is the name of the package to target.

## “Unauthorized to access” error

When running the `tanzu package install` command, you receive an error message that includes the error:

```
UNAUTHORIZED: unauthorized to access repository
```

Example:

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

```
Error: package reconciliation failed: vendir: Error: Syncing directory '0':
  Syncing directory '.' with imgpkgBundle contents:
    Imgpkg: exit status 1 (stderr: Error: Checking if image is bundle: Collecting 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.yaml
-n tap-install
- Installing package 'tap.tanzu.vmware.com'
\ Getting package metadata for 'tap.tanzu.vmware.com'
| Creating service account 'tap-tap-install-sa'
/ Creating cluster admin role 'tap-tap-install-cluster-role'
| Creating cluster role binding 'tap-tap-install-cluster-rolebinding'
| Creating secret 'tap-tap-install-values'
| Creating package resource
- Waiting for 'PackageInstall' reconciliation for 'tap'
/ 'PackageInstall' resource install status: Reconciling
| 'PackageInstall' resource install status: ReconcileFailed
```

```
Please consider using 'tanzu package installed update' to update the installed package
with correct settings
```

```
Error: resource reconciliation failed: kapp: Error: waiting on reconcile packageinstal
l/tap-gui (packaging.carvel.dev/v1alpha1) namespace: tap-install:
  Finished unsuccessfully (Reconcile failed: (message: Error (see .status.usefulError
Message for details))). Reconcile failed: Error (see .status.usefulErrorMessage for de
tails)
Error: exit status 1
```

## Explanation

Often, the cause is one of the following:

- Your infrastructure provider takes longer to perform tasks than the timeout value allows.
- A race-condition between components exists. For example, a package that uses [Ingress](#) completes before the shared Tanzu ingress controller becomes available.

The VMware Carvel tools kapp-controller continues to try in a reconciliation loop in these cases. However, if the reconciliation status is `failed` then there might be a configuration issue in the provided `tap-config.yml` file.

## Solution

1. Verify if the installation is still in progress by running:

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

If the installation is still in progress, the command produces output similar to the following example, and the installation is likely to finish successfully.

```
\ Retrieving installed packages...
NAME                PACKAGE-NAME
PACKAGE-VERSION  STATUS      NAMESPACE
accelerator                accelerator.apps.tanzu.vmware.com
1.0.0                  Reconcile succeeded tap-install
api-portal                api-portal.tanzu.vmware.com
1.0.6                  Reconcile succeeded tap-install
appliveview                run.appliveview.tanzu.vmware.com
1.0.0-build.3          Reconciling tap-install
appliveview-conventions  build.appliveview.tanzu.vmware.com
1.0.0-build.3          Reconcile succeeded tap-install
buildservice                buildservice.tanzu.vmware.com
1.4.0-build.1          Reconciling tap-install
cartographer                cartographer.tanzu.vmware.com
0.1.0                  Reconcile succeeded tap-install
cert-manager                cert-manager.tanzu.vmware.com
1.5.3+tap.1            Reconcile succeeded tap-install
cnrs                        cnrs.tanzu.vmware.com
1.1.0                  Reconcile succeeded tap-install
contour                    contour.tanzu.vmware.com
1.18.2+tap.1            Reconcile succeeded tap-install
conventions-controller  controller.conventions.apps.tanzu.vmware.com
0.4.2                  Reconcile succeeded tap-install
developer-conventions  developer-conventions.tanzu.vmware.com
0.4.0-build1            Reconcile succeeded tap-install
fluxcd-source-controller fluxcd.source.controller.tanzu.vmware.com
0.16.0                  Reconcile succeeded tap-install
grype                        grype.scanning.apps.tanzu.vmware.com
1.0.0                  Reconcile succeeded tap-install
image-policy-webhook  image-policy-webhook.signing.apps.tanzu.vmware.com
1.0.0-beta.3            Reconcile succeeded tap-install
learningcenter                learningcenter.tanzu.vmware.com
```

```

0.1.0-build.6   Reconcile succeeded tap-install
learningcenter-workshops workshops.learningcenter.tanzu.vmware.com
0.1.0-build.7   Reconcile succeeded tap-install
ootb-delivery-basic ootb-delivery-basic.tanzu.vmware.com
0.5.1           Reconcile succeeded tap-install
ootb-supply-chain-basic ootb-supply-chain-basic.tanzu.vmware.com
0.5.1           Reconcile succeeded tap-install
ootb-templates   ootb-templates.tanzu.vmware.com
0.5.1           Reconcile succeeded tap-install
scanning         scanning.apps.tanzu.vmware.com
1.0.0           Reconcile succeeded tap-install
metadata-store   metadata-store.apps.tanzu.vmware.com
1.0.2           Reconcile succeeded tap-install
service-bindings service-bindings.labs.vmware.com
0.6.0           Reconcile succeeded tap-install
services-toolkit services-toolkit.tanzu.vmware.com
0.7.1           Reconcile succeeded tap-install
source-controller controller.source.apps.tanzu.vmware.com
0.2.0           Reconcile succeeded tap-install
spring-boot-conventions spring-boot-conventions.tanzu.vmware.com
0.2.0           Reconcile succeeded tap-install
tap              tap.tanzu.vmware.com
0.4.0-build.12  Reconciling        tap-install
tap-gui          tap-gui.tanzu.vmware.com
1.0.0-rc.72     Reconcile succeeded tap-install
tap-telemetry   tap-telemetry.tanzu.vmware.com
0.1.0           Reconcile succeeded tap-install
tekton-pipelines tekton.tanzu.vmware.com
0.30.0          Reconcile succeeded tap-install
    
```

If the installation has stopped running, one or more reconciliations have likely failed, as seen in the following example:

```

NAME                PACKAGE NAME
PACKAGE VERSION    DESCRIPTION
AGE
accelerator         accelerator.apps.tanzu.vmware.com
1.0.1              Reconcile succeeded
109m
api-portal          api-portal.tanzu.vmware.com
1.0.9              Reconcile succeeded
119m
appliveview        run.appliveview.tanzu.vmware.com
1.0.2-build.2     Reconcile succeeded
109m
appliveview-conventions build.appliveview.tanzu.vmware.com
1.0.2-build.2     Reconcile succeeded
109m
buildservice       buildservice.tanzu.vmware.com
1.5.0              Reconcile succeeded
119m
cartographer       cartographer.tanzu.vmware.com
0.2.1              Reconcile succeeded
117m
cert-manager       cert-manager.tanzu.vmware.com
1.5.3+tap.1       Reconcile succeeded
119m
cnrs               cnrs.tanzu.vmware.com
1.1.0              Reconcile succeeded
109m
contour            contour.tanzu.vmware.com
1.18.2+tap.1     Reconcile succeeded
117m
conventions-controller controller.conventions.apps.tanzu.vmware.com
0.5.0              Reconcile succeeded
    
```

```

117m
developer-conventions      developer-conventions.tanzu.vmware.com
0.5.0                      Reconcile succeeded
109m
fluxcd-source-controller  fluxcd.source.controller.tanzu.vmware.com
0.16.1                    Reconcile succeeded
119m
grype                      grype.scanning.apps.tanzu.vmware.com
1.0.0                    Reconcile failed: Error (see .status.usefulErrorMessage for d
etails)    109m
image-policy-webhook      image-policy-webhook.signing.apps.tanzu.vmware.com
1.0.1                    Reconcile succeeded
117m
learningcenter            learningcenter.tanzu.vmware.com
0.1.0                    Reconcile succeeded
109m
learningcenter-workshops  workshops.learningcenter.tanzu.vmware.com
0.1.0                    Reconcile succeeded
103m
metadata-store            metadata-store.apps.tanzu.vmware.com
1.0.2                    Reconcile succeeded
117m
ootb-delivery-basic       ootb-delivery-basic.tanzu.vmware.com
0.6.1                    Reconcile succeeded
103m
ootb-supply-chain-basic   ootb-supply-chain-basic.tanzu.vmware.com
0.6.1                    Reconcile succeeded
103m
ootb-templates            ootb-templates.tanzu.vmware.com
0.6.1                    Reconcile succeeded
109m
scanning                  scanning.apps.tanzu.vmware.com
1.0.0                    Reconcile succeeded
119m
service-bindings          service-bindings.labs.vmware.com
0.6.0                    Reconcile succeeded
119m
services-toolkit          services-toolkit.tanzu.vmware.com
0.7.1                    Reconcile succeeded
119m
source-controller         controller.source.apps.tanzu.vmware.com
0.2.0                    Reconcile succeeded
119m
spring-boot-conventions   spring-boot-conventions.tanzu.vmware.com
0.3.0                    Reconcile succeeded
109m
tap                       tap.tanzu.vmware.com
1.0.1                    Reconcile failed: Error (see .status.usefulErrorMessage for d
etails)    119m
tap-gui                   tap-gui.tanzu.vmware.com
1.0.2                    Reconcile succeeded
109m
tap-telemetry             tap-telemetry.tanzu.vmware.com
0.1.3                    Reconcile succeeded
119m
tekton-pipelines          tekton.tanzu.vmware.com
0.30.0                   Reconcile succeeded
119m

```

In this example, `packageinstall/grype` and `packageinstall/tap` have reconciliation errors.

- To get more details on the possible cause of a reconciliation failure, run:

```
kubectl describe packageinstall/NAME -n tap-install
```

Where `NAME` is the name of the failing package. For this example it would be `grype`.

3. Use the displayed information to search for a relevant troubleshooting issue in this topic. If none exists, and you are unable to fix the described issue yourself, please contact [support](#).
4. Repeat these diagnosis steps for any other packages that failed to reconcile.

## Failure to accept an End User License Agreement error

You cannot access Tanzu Application Platform or one of its components from VMware Tanzu Network.

### Explanation

You cannot access Tanzu Application Platform or one of its components from VMware Tanzu Network before accepting the relevant EULA in VMware Tanzu Network.

### Solution

Follow the steps in [Accept the End User License Agreements](#) in *Installing the Tanzu CLI*.

## Ingress is broken on Kind cluster

Your Contour installation cannot provide ingress to workloads when installed on a Kind cluster without a LoadBalancer solution. Your Kind cluster was created with port mappings, as described in the [Kind install guide](#).

### Explanation

In Tanzu Application Platform v1.4.13, the default configuration for `contour.envoy.service.type` is `LoadBalancer`. However, for the Envoy pods to be accessed by using the port mappings on your Kind cluster, the service must be of type `NodePort`.

### Solution

Configure `contour.envoy.service.type` to be `NodePort`. Then, configure `envoy.service.nodePorts.http` and `envoy.service.nodePorts.https` to the corresponding port mappings on your Kind node. Otherwise, the NodePort service is assigned random ports, which are not accessible through your Kind cluster.

## Troubleshoot using Tanzu Application Platform

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

## Use events to find possible causes

Events can highlight issues with components in a supply chain. For example, high occurrences of `StampedObjectApplied` or `ResourceOutputChanged` can indicate problems with trashing on a component.

To view the recent events for a workload, run:

```
kubectl describe workload.carto.run <workload-name> -n <workload-ns>
```

## Missing build logs after creating a workload

You create a workload, but no logs appear when you run:

```
tanzu apps workload tail workload-name --since 10m --timestamp
```

## Explanation

Common causes include:

- Misconfigured repository
- Misconfigured service account
- Misconfigured registry credentials

## Solution

To resolve this issue, run:

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

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

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

## Workload creation stops responding with “Builder default is not ready” message

You can see the “Builder default is not ready” message in two places:

1. The “Messages” section of the `tanzu apps workload get my-app` command.
2. The Supply Chain section of Tanzu Application Platform GUI.

This message indicates there is something wrong with the Builder (the component that builds the container image for your workload).

## Explanation

This message is typically encountered when the core component of the Builder (`kpack`) transitions into a bad state.

Although this isn’t the only scenario where this can happen, `kpack` can transition into a bad state when Tanzu Application Platform is deployed to a local `minikube` or `kind` cluster, and especially when that `minikube` or `kind` cluster is restarted.

## Solution

1. Restart `kpack` by deleting the `kpack-controller` and `kpack-webhook` pods in the `kpack` namespace. Deleting these resources triggers their recreation:
  - `kubectl delete pods --all --namespace kpack`
2. Verify status of the replacement pods:
  - `kubectl get pods --namespace kpack`
3. Verify the workload status after the new `kpack` pods `STATUS` are `Running`:
  - `tanzu apps workload get YOUR-WORKLOAD-NAME`

## “Workload already exists” error after updating the workload

When you update the workload, you receive the following error:

```
Error: workload "default/APP-NAME" already exists
Error: exit status 1
```

Where `APP-NAME` is the name of the app.

For example, when you run:

```
tanzu apps workload create tanzu-java-web-app \
--git-repo https://github.com/dbuchko/tanzu-java-web-app \
--git-branch main \
--type web \
--label apps.tanzu.vmware.com/has-tests=true \
--yes
```

You receive the following error

```
Error: workload "default/tanzu-java-web-app" already exists
Error: exit status 1
```

## Explanation

The app is running before performing a Live Update using the same app name.

## Solution

To resolve this issue, either delete the app or use a different name for the app.

## Workload creation fails due to authentication failure in Docker Registry

You might encounter an error message similar to the following when creating or updating a workload by using IDE or `apps` CLI plug-in:

```
Error: Writing 'index.docker.io/shaileshp2922/build-service/tanzu-java-web-app:lates
t': Error while preparing a transport to talk with the registry: Unable to create roun
d tripper: GET https://auth.ipv6.docker.com/token?scope=repository%3Ashaileshp2922%2Fb
uild-service%2Ftanzu-java-web-app%3Apush%2Cpull&service=registry.docker.io: unexpected
status code 401 Unauthorized: {"details":"incorrect username or password"}
```

## Explanation

This type of error frequently occurs when the URL set for `source image` (IDE) or `--source-image` flag (`apps` CLI plug-in) is not Docker registry compliant.

## Solution

1. Verify that you can authenticate directly against the Docker registry and resolve any failures by running:

```
docker login -u USER-NAME
```

2. Verify your `--source-image` URL is compliant with Docker.

The URL in this example `index.docker.io/shaileshp2922/build-service/tanzu-java-web-app` includes nesting. Docker registry, unlike many other registry solutions, does not support nesting.

- To resolve this issue, you must provide an unnested URL. For example, [index.docker.io/shaileshp2922/tanzu-java-web-app](https://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 get resource \"secrets\" in API group \"\" in the namespace \"tap-telemetry\""
```

### Explanation

The `tap-telemetry` namespace misses a role that allows the controller to list secrets in the `tap-telemetry` namespace. For more information about roles, see [Role and ClusterRole Kubernetes documentation](#).

### Solution

To resolve this issue, run:

```
kubectl patch roles -n tap-telemetry tap-telemetry-controller --type='json' -p='[{"op": "add", "path": "/rules/-", "value": {"apiGroups": [""], "resources": ["secrets"], "verbs": ["get", "list", "watch"]}]']'
```

## Debug convention might not apply

If you upgrade from Tanzu Application Platform v0.4, the debug convention can not apply to the app run image.

### Explanation

The Tanzu Application Platform v0.4 lacks SBOM data.

### Solution

Delete existing app images that were built using Tanzu Application Platform v0.4.

## Execute bit not set for App Accelerator build scripts

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 can be caused by an insufficient number of nodes or a lack of resources on nodes necessary to deploy the workloads.

### Solution

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

---

## 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 are 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 currently running.

A known rare condition during Tanzu Application Platform profiles installation can cause this. If so, you can 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 deactivate 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:

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

### Explanation

The Supply Chain Security Tools (SCST) - 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 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 can be caused by an insufficient number of nodes or a lack of resources on nodes necessary to deploy the workloads. Follow instructions from your cloud provider to scale out or scale up your cluster.

## CrashLoopBackOff from password authentication fails

SCST - Store does not start. You see the following error in the `metadata-store-app` Pod logs:

```
$ kubectl logs pod/metadata-store-app-* -n metadata-store -c metadata-store-app
...
[error] failed to initialize database, got error failed to connect to `host=metadata-store-db user=metadata-store-user database=metadata-store`: server error (FATAL: password authentication failed for user "metadata-store-user") (SQLSTATE 28P01)
```

## Explanation

The database password has changed between deployments. This is not supported.

## Solution

Redeploy the app either with the original database password or follow the latter steps to erase the data on the volume:

1. Deploy `metadata-store` app with kapp.
2. Verify that the `metadata-store-db-*` pod fails.
3. Run:

```
kubectl exec -it metadata-store-db-KUBERNETES-ID -n metadata-store /bin/bash
```

Where `KUBERNETES-ID` is the ID generated by Kubernetes and appended to the pod name.

4. To delete all database data, run:

```
rm -rf /var/lib/postgresql/data/*
```

This is the path found in `postgres-db-deployment.yaml`.

5. Delete the `metadata-store` app with kapp.
6. Deploy the `metadata-store` app with kapp.

## Password authentication fails

SCST - Store does not start. You see the following error in the `metadata-store-app` pod logs:

```
$ kubectl logs pod/metadata-store-app-* -n metadata-store -c metadata-store-app
...
[error] failed to initialize database, got error failed to connect to `host=metadata-store-db user=metadata-store-user database=metadata-store`: server error (FATAL: password authentication failed for user "metadata-store-user" (SQLSTATE 28P01))
```

## Explanation

The database password has changed between deployments. This is not supported.

## Solution

Redeploy the app either with the original database password or follow the latter steps to erase the data on the volume:

1. Deploy `metadata-store` app with kapp.
2. Verify that the `metadata-store-db-*` pod fails.
3. Run:

```
kubectl exec -it metadata-store-db-KUBERNETES-ID -n metadata-store /bin/bash
```

Where `KUBERNETES-ID` is the ID generated by Kubernetes and appended to the pod name.

4. To delete all database data, run:

```
rm -rf /var/lib/postgresql/data/*
```

This is the path found in `postgres-db-deployment.yaml`.

5. Delete the `metadata-store` app with kapp.
6. Deploy the `metadata-store` app with kapp.

## metadata-store-db pod fails to start

When SCST - Store is deployed, deleted, and then redeployed, the `metadata-store-db` pod fails to start if the database password changed during redeployment.

### Explanation

The persistent volume used by `PostgreSQL` retains old data, even though the retention policy is set to `DELETE`.

### Solution

Redeploy the app either with the original database password or follow the later steps to erase the data on the volume:

1. Deploy `metadata-store` app with kapp.
2. Verify that the `metadata-store-db-*` pod fails.
3. Run:

```
kubectl exec -it metadata-store-db-KUBERNETES-ID -n metadata-store /bin/bash
```

Where `KUBERNETES-ID` is the ID generated by Kubernetes and appended to the pod name.

4. To delete all database data, run:

```
rm -rf /var/lib/postgresql/data/*
```

This is the path found in `postgres-db-deployment.yaml`.

5. Delete the `metadata-store` app with kapp.
6. Deploy the `metadata-store` app with kapp.

## Missing persistent volume

After SCST - Store is deployed, `metadata-store-db` pod fails for missing volume while `postgres-db-pv-claim` pvc is in the `PENDING` state.

### Explanation

The cluster where SCST - Store is deployed does not have `storageclass` defined. The provisioner of `storageclass` is responsible for creating the persistent volume after `metadata-store-db` attaches `postgres-db-pv-claim`.

### Solution

1. Verify that your cluster has `storageclass` by running:

```
kubectl get storageclass
```

2. Create a `storageclass` in your cluster before deploying SCST - Store. For example:

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

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

## Failure to connect Tanzu CLI to AWS EKS clusters

When using the Tanzu CLI to connect to AWS EKS clusters, you might see one of the following errors:

- `Error: Unable to connect: connection refused. Confirm kubeconfig details and try again`
- `invalid apiVersion "client.authentication.k8s.io/v1alpha1"`

### Explanation

The cause is [Kubernetes v1.24](#) dropping support for `client.authentication.k8s.io/v1alpha1`. For more information, see [aws/aws-cli/issues/6920](#) in GitHub.

### Solution

Follow these steps to update your `aws-cli` to a supported v2.7.35 or later, and update the `kubeconfig` entry for your EKS clusters:

1. Update `aws-cli` to the latest version. For more information see [AWS documentation](#).
2. Update the `kubeconfig` entry for your EKS clusters:

```
aws eks update-kubeconfig --name ${EKS_CLUSTER_NAME} --region ${REGION}
```

3. In a new terminal window, run a Tanzu CLI command to verify the connection issue is resolved. For example:

```
tanzu apps workload list
```

Expect the command to execute without error.

## Invalid repository paths are propagated

When inputting `shared.image_registry.project_path`, invalid repository paths are propagated.

### Explanation

The key `shared.image_registry.project_path`, which takes input as `SERVER-NAME/REPO-NAME`, cannot take `/` at the end of the string.

### Solution

Do not append `/` to the end of the string.

## x509: certificate signed by unknown authority

### Explanation

Tanzu Application Platform v1.4 introduces [Shared Ingress Issuer](#) to secure ingress communication by default. The Certificate Authority for Shared Ingress Issuer is generated as self-signed. As a result, you might see one of the following errors:

- `connection refused`
- `x509: certificate signed by unknown authority`

## Solution

You can choose one of the following options to mitigate the issue:

### Option 1: Configure the Shared Ingress Issuer's Certificate Authority as a trusted Certificate Authority



#### Important

This is the recommended option for a secure instance.

Follow these steps to trust the Shared Ingress Issuer's Certificate Authority in Tanzu Application Platform:

1. Extract the ClusterIssuer's Certificate Authority.

For default installations where `ingress_issuer` is not set in `tap_values.yml`, you can extract the ClusterIssuer's Certificate Authority from cert-manager:

```
kubectl get secret tap-ingress-selfsigned-root-ca -n cert-manager -o yaml | yq
.data | cut -d' ' -f2 | head -1 | base64 -d
```

If you overrode the default `ingress_issuer` while installing Tanzu Application Platform, you must refer to your issuer's documentation to extract your ClusterIssuer's Certificate Authority instead of using the command above.

2. Add the certificate to the list of trusted certificate authorities by appending the certificate authority to the `shared.ca_cert_data` field in your `tap-values.yml`.
3. Reapply your configuration:

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

### Option 2: Deactivate the shared ingress issuer



#### Important

This option is recommended for testing purposes only.

Follow these steps to deactivate TLS for Cloud Native Runtimes, AppSSO and Tanzu Application Platform GUI:

1. Set `shared.ingress_issuer` to `""` in your `tap-values.yml`:

```
shared:
  ingress_issuer: ""
```

2. Reapply your configuration:

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

## Troubleshoot Tanzu Application Platform components

For component-level troubleshooting, see these topics:

- [Troubleshoot Tanzu Application Platform GUI](#)
- [Troubleshoot Learning Center](#)
- [Troubleshoot Service Bindings](#)
- [Troubleshoot Source Controller](#)
- [Troubleshoot Spring Boot conventions](#)
- [Troubleshoot Supply Chain Security Tools - Scan](#)
- [Troubleshoot Supply Chain Security Tools - Store](#)
- [Troubleshoot Application Live View](#)
- [Troubleshoot Cloud Native Runtimes for Tanzu](#)
- [Tanzu Build Service FAQ](#)
- [Troubleshoot Tanzu Build Service](#)
- [Troubleshoot Services Toolkit](#)



# Uninstall Tanzu Application Platform

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

To uninstall Tanzu Application Platform:

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

## Delete the packages

- If you installed Tanzu Application Platform through predefined profiles, delete the `tap` metadata package by running:

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

- If you installed any additional packages that were not in the predefined profiles, delete the individual packages by running:

1. List the installed packages by running:

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

2. Remove a package by running:

```
tanzu package installed delete PACKAGE-NAME --namespace tap-install
```

For example:

```
$ tanzu package installed delete cloud-native-runtimes --namespace tap-in
stall
| Uninstalling package 'cloud-native-runtimes' from namespace 'tap-instal
l'
/ Getting package install for 'cloud-native-runtimes'
\ Deleting package install 'cloud-native-runtimes' from namespace 'tap-in
stall'
\ Package uninstall status: Reconciling
/ Package uninstall status: Deleting
| Deleting admin role 'cloud-native-runtimes-tap-install-cluster-role'
| Deleting role binding 'cloud-native-runtimes-tap-install-cluster-rolebi
nding'
| Deleting secret 'cloud-native-runtimes-tap-install-values'
/ Deleting service account 'cloud-native-runtimes-tap-install-sa'

Uninstalled package 'cloud-native-runtimes' from namespace 'tap-install'
```

Where `PACKAGE-NAME` is the name of a package listed in step 1.

3. Repeat step 2 for each individual package installed.

## Delete the Tanzu Application Platform package repository

To delete the Tanzu Application Platform package repository:

1. Retrieve the name of the Tanzu Application Platform package repository by running:

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

For example:

```
$ tanzu package repository list --namespace tap-install
- Retrieving repositories...
NAME                REPOSITORY
STATUS              DETAILS
tanzu-tap-repository registry.tanzu.vmware.com/tanzu-application-platform/tap-packages:0.2.0 Reconcile succeeded
```

2. Remove the Tanzu Application Platform package repository by running:

```
tanzu package repository delete PACKAGE-REPO-NAME --namespace tap-install
```

Where `PACKAGE-REPO-NAME` is the name of the `packageRepository` from the earlier step.

For example:

```
$ tanzu package repository delete tanzu-tap-repository --namespace tap-install
- Deleting package repository 'tanzu-tap-repository'...
Deleted package repository 'tanzu-tap-repository' in namespace 'tap-install'
```

## Remove Tanzu CLI, plug-ins, and associated files

To completely remove the Tanzu CLI, plug-ins, and associated files, run the script for your OS:

- For Linux or MacOS, run:

```
#!/bin/zsh
rm -rf $HOME/tanzu/cli # Remove previously downloaded cli files
sudo rm /usr/local/bin/tanzu # Remove CLI binary (executable)
rm -rf ~/.config/tanzu/ # current location # Remove config directory
rm -rf ~/.tanzu/ # old location # Remove config directory
rm -rf ~/.cache/tanzu # remove cached catalog.yaml
rm -rf ~/Library/Application\ Support/tanzu-cli/* # Remove plug-ins
```

## Remove Cluster Essentials

To completely remove Cluster Essentials, see [Cluster Essentials documentation](#).

# Component documentation for Tanzu Application Platform

Tanzu Application Platform (commonly known as TAP) is a modular platform that you can enhance by installing components. Most of the Tanzu Application Platform components are documented in this section. In some cases, a component's documentation is hosted on a separate site, and you'll find a link to it in this section.

## Component documentation for Tanzu Application Platform

Tanzu Application Platform (commonly known as TAP) is a modular platform that you can enhance by installing components. Most of the Tanzu Application Platform components are documented in this section. In some cases, a component's documentation is hosted on a separate site, and you'll find a link to it in this section.

## Overview of Tanzu CLI

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

## Tanzu CLI

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

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

## Tanzu CLI Architecture

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

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

## Tanzu CLI Installation

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

To install the CLI :

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

## Tanzu CLI Command Groups

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

## Install New Plug-ins

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

1. In a terminal, run:

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

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

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

## Install Local Plug-ins

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

1. Download the plug-in `tar.gz` from the release artifacts for your distribution.

2. Extract the `tar.gz` to a location on your local machine using the extraction tool of your choice. For example, the `tar -xvf` command.
3. From that location, run:

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

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

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

## Overview of Tanzu CLI

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

### Tanzu CLI

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

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

### Tanzu CLI Architecture

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

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

### Tanzu CLI Installation

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

To install the CLI :

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

## Tanzu CLI Command Groups

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

## Install New Plug-ins

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

1. In a terminal, run:

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

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

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

## Install Local Plug-ins

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

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

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

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

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

## Overview of Tanzu CLI plug-ins

The topics in this section tell you about the following plug-ins in your Tanzu Application Platform (commonly known as TAP):

- [accelerator](#) - The Application Accelerator Tanzu CLI plug-in includes commands for developers and operators to create and use accelerators.
- [apps](#) - This Tanzu CLI plug-in provides the ability to create, view, update, and delete application workloads on any Kubernetes cluster that has the Tanzu Application Platform components installed.
- [insight](#) - The Tanzu Insight CLI plug-in enables querying vulnerability, image, and package data.

## Tanzu Apps CLI overview

This topic gives you an overview of the Tanzu Apps CLI. Use the Tanzu Apps CLI to create, view, update, and delete application workloads on any Kubernetes cluster that has the Tanzu Application Platform (commonly known as TAP) components installed.

## About workloads

Tanzu Application Platform enables developers to quickly build and test applications regardless of their familiarity with Kubernetes. Developers can turn source code into a workload that runs in a container with a URL.

A workload enables developers to choose application specifications, such as repository location, environment variables, service binding, and more.

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.

## Tutorials

To get started with Tanzu Apps CLI and workload management, see [Tutorials](#)

## How-to-guides

For more complex examples regarding Tanzu Apps CLI usage, see [How-to-guides](#)

## Tanzu Apps CLI overview

This topic gives you an overview of the Tanzu Apps CLI. Use the Tanzu Apps CLI to create, view, update, and delete application workloads on any Kubernetes cluster that has the Tanzu Application Platform (commonly known as TAP) components installed.

## About workloads

Tanzu Application Platform enables developers to quickly build and test applications regardless of their familiarity with Kubernetes. Developers can turn source code into a workload that runs in a container with a URL.

A workload enables developers to choose application specifications, such as repository location, environment variables, service binding, and more.

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.

## Tutorials

To get started with Tanzu Apps CLI and workload management, see [Tutorials](#)

## How-to-guides

For more complex examples regarding Tanzu Apps CLI usage, see [How-to-guides](#)

## Install Tanzu Apps CLI plug-in

This topic describes how to install the Tanzu Apps CLI plug-in.

## Prerequisites

Ensure that you installed or updated the Tanzu CLI, for more information, see [Install Tanzu CLI](#).

## Install Tanzu Apps CLI plug-in

Run:



```
tanzu plugin install apps --group vmware-tap/default:v1.4
```

Verify that the plug-in is installed correctly:

```
tanzu apps version
# sample output
v0.12.1
```

## Uninstall Apps CLI plug-in

Run:

```
tanzu plugin delete apps
```

## Change clusters

The Apps CLI plug-in refers to the default kubeconfig file to access a Kubernetes cluster. When you run a `tanzu apps` command, the plug-in uses the default context that is 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 CONTENT-NAME` to change the default context. All subsequent `tanzu apps` commands target the cluster defined in the new default kubeconfig context.
2. Include the `--context CONTENT-NAME` flag when running any `tanzu apps` command.



### Note

Any subsequent `tanzu apps` commands that do not include the `--context CONTENT-NAME` flag continue to use the default context set in the kubeconfig.

## Override the default kubeconfig

There are two approaches to overriding the default kubeconfig:

1. Set the environment variable `KUBECONFIG=PATH` to change the kubeconfig the Apps CLI plug-in will reference.  
All subsequent `tanzu apps` commands reference the non-default kubeconfig assigned to the environment variable.
2. Include the `--kubeconfig path` flag when running any `tanzu apps` command.



### Note

Any subsequent `tanzu apps` commands that do not include the `--context CONTEXT-NAME` flag continue to use the default context set in the kubeconfig.

For more information about kubeconfig, see [Configure Access to Multiple Clusters](#) in the Kubernetes documentation.

## Autocompletion

The Apps CLI plug-in has auto-completion support. The plug-in supports auto-completion for commands, positional arguments, flags, and flag values. Add one of the following commands to the shell config file according to your current setup:

## Bash

```
tanzu completion bash > HOME/.tanzu/completion.bash.inc
```

## Zsh

```
echo "autoload -U compinit; compinit" >> ~/.zshrc
tanzu completion zsh > "${fpath[1]}/_tanzu"
```

## Create workloads

Create workloads from one of the following sources:

- A Git repository, for example, a Git branch, Git tag, or Git commit
- An existing local project
- An image that is pulled from a registry to deploy the application.
- A Maven repository artifact.

For more information, see [Create a workload](#).

## Debug and troubleshoot workloads

Check the workload status with the `tanzu apps workload get` and `tanzu apps workload tail` commands.

Use `tanzu apps workload get` to see the workload specification, the resources attached to it, their status and any associated high-level error messages (if they exist).

Use `tanzu apps workload tail` to see testing, scanning, build, configuration, deployment, and runtime logs associated with a workload and its progression through the supply chain.

For more information about using these commands and common errors, see [Debug workloads](#).

## Create a workload

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

## Prerequisites

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

- Install [kubectl](#).
- Install Tanzu Application Platform components on a Kubernetes cluster. See [Installing Tanzu Application Platform](#).
- [Set your kubeconfig context](#) to the prepared cluster `kubectl config use-context CONTEXT_NAME`.
- Install Tanzu CLI. See [Install or update the Tanzu CLI and plug-ins](#).
- Install the Apps plug-in. See the [Install Apps plug-in](#).

- [Set up developer namespaces to use your installed packages.](#)
- For more information about the values you can provide when creating and managing the life cycle of workloads, see [Workload and Supply Chain Custom Resources](#) in the Cartographer documentation. Alternatively, run `kubectl explain workload.spec` for the Kubernetes version running on the target cluster.

## Get started with an example workload

You can create a workload from a GitHub repository or local source.

### Create a workload from GitHub repository

Use the flags `--git-repo`, `--git-branch`, `--git-tag`, and `--git-commit` flags to create a workload from an existing Git repository. This allows the [supply chain](#) to get the source from the given repository to deploy the application.

To create a named workload and specify a Git source code location, run:

```
tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanzu/application-accelerator-samples --sub-path tanzu-java-web-app --git-tag tap-1.4.0 --type web
```

Respond `y` to prompts to complete process.

Where:

- `tanzu-java-web-app` is the name of the workload.
- `--git-repo` is the location of the code to build the workload from.
- `--sub-path` is the relative path inside the repository to treat as application root.
- `--git-tag` (optional) specifies which tag in the repository to pull the code from.
- `--git-branch` (optional) specifies which branch in the repository to pull the code from.
- `--type` distinguishes the workload type.

View the full list of supported workload configuration options by running `tanzu apps workload apply --help`.

### Create a workload from local source code

Use the `--local-path` and `--source-image` flags to create a workload from an existing local project. This allows the [supplychain](#) to generate an image ([carvel-imgpkg](#)) and push it to the given registry to be used in the workload.

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

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

Respond `y` to the dialog box about publishing local source code if the image must 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 where the local source code is uploaded as an image.

### Exclude Files

When working with local source code, you can exclude files from the source code to be uploaded within the image by creating a `.tanziignore` file at the root of the source code.

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

For more information regarding the `.tanziignore` file see [tanziignorefile](#).

## Create workload from an existing image

Create a workload from an existing registry image by providing the reference to that image through the `--image` flag. The `supplychain` references the provided registry image when the workload is deployed.

For example:

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

Respond `y` to prompts to complete process.

Where:

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

## Create a workload from Maven repository artifact

Create a workload from a Maven repository artifact `Source-Controller` by setting its properties through the `--maven-*` flags when using the `supply chain`.

The Maven repository URL is set when the supply chain is created.

To create a Maven workload using the CLI provided flags, run:

```
tanzu apps workload apply my-workload \
  --maven-artifact hello-world \
  --maven-type jar
  --maven-version 0.0.1 \
  --maven-group carto.run \
  --type web -y
```

For more information about the Maven flags, see the [Maven flags command reference information](#).

For information about how to configure the Maven artifact authentication credentials, see [Maven Repository Secret](#).

## Working with YAML files

In many cases, workload life cycles are managed through CLI commands. However, there might be cases where managing the workload through direct interactions and edits of a `yaml` file is preferred. The Apps CLI plug-in supports using `yaml` files to meet the requirements.

When a workload is managed using a `yaml` file, that file **must contain a single workload definition**.

For example, a valid file looks similar to the following example:

```
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
```

```

name: tanzu-java-web-app
labels:
  app.kubernetes.io/part-of: tanzu-java-web-app
  apps.tanzu.vmware.com/workload-type: web
spec:
  source:
    git:
      url: https://github.com/vmware-tanzu/application-accelerator-samples
      ref:
        tag: tap-1.4.0
      subPath: tanzu-java-web-app

```

To create a workload from a file like the earlier example:

```
tanzu apps workload create --file my-workload-file.yaml
```



#### Note

When flags are passed in combination with `--file my-workload-file.yaml` the flag values take precedence over the associated property or values in the YAML.

The workload YAML definition can also be passed in through stdin as follows:

```
tanzu apps workload create --file - --yes
```

The console waits for input, and the content with valid `yaml` definitions for a workload can either be written or pasted. Then click **Ctrl-D** three times to start the workload creation. This can also be done with the `workload apply` command.

**Note** To pass a workload through `stdin`, the `--yes` flag is required. If not provided, the command fails.

Another way to pass a workload with the `--file` flag is using a URL, which, as mentioned before, must contain a raw file with the workload definition.

For example:

```
tanzu apps workload apply --file https://raw.githubusercontent.com/vmware-tanzu/apps-cli-plugin/main/pkg/commands/testdata/workload.yaml
```

## Bind a service to a workload

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

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

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

```
tanzu apps workload apply 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 `{service-ref-name}={apiVersion}:{kind}:{service-binding-name}`.

For more information about how to bind a service to a workload, see [Consume services on Tanzu Application Platform](#).

## Next steps

You can verify workload details and status, add environment variables, export definitions or bind services.

1. To verify a workload status and details, use `tanzu apps workload get`.

To get workload logs, use `tanzu apps workload tail`.

For more information, see [debug workload section](#).

2. To add environment variables, run:

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

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

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

4. To bind a service to a workload, see the `--service-ref` flag.

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

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

## Workload Examples

This topic provides you with examples of how to use the Tanzu Apps CLI `apps workload apply` command flags.

## Custom registry credentials

Either use a custom certificate on your system or pass the path to the certificate through flags.

To pass the certificate through flags, specify:

- `--registry-ca-cert`: This is the path of the self-signed certificate needed for the custom or private registry. This is also populated with a default value through the environment variable `TANZU_APPS_REGISTRY_CA_CERT`.
- `--registry-password`: Use this when the registry requires credentials to push. The value of this flag can also be specified through `TANZU_APPS_REGISTRY_PASSWORD`.
- `--registry-username`: Use with `--registry-password` to set the registry credentials. It can also be provided as the environment variable `TANZU_APPS_REGISTRY_USERNAME`.
- `--registry-token`: Set when the registry authentication is done through a token. The value of this flag can also be taken from `TANZU_APPS_REGISTRY_TOKEN` environment variable.

For example:

```
tanzu apps workload apply WORKLOAD --local-path PATH-TO-REPO -s registry.url.nip.io/PACKAGE/IMAGE --type web --registry-ca-cert PATH-TO-CA-CERT.nip.io.crt --registry-username USERNAME --registry-password PASSWORD
```

Alternatively, the same command can be run as:

```

```console
export TANZU_APPS_REGISTRY_CA_CERT=PATH-TO-CA-CERT.nip.io.crt
export TANZU_APPS_REGISTRY_PASSWORD=USERNAME
export TANZU_APPS_REGISTRY_USERNAME=PASSWORD

tanzu apps workload apply WORKLOAD --local-path PATH-TO-REPO -s registry.url.nip.io/PACKAGE/IMAGE

```

## –live-update and –debug

Use the `--live-update` flag to ensure that local source code changes are reflected quickly on the running workload. This is particularly valuable when iterating on features that require the workload to be deployed and running to validate.

Live update is ideally situated for running from within one of our supported IDE extensions, but it can also be utilized independently as shown in the following Spring Boot application example:

## Spring Boot application example

Prerequisites: [Tilt](#) must be installed on the client.

1. Clone the repository by running:

```
git clone https://github.com/vmware-tanzu/application-accelerator-samples
```

2. Change into the `tanzu-java-web-app` directory.
3. In `Tiltfile`, first, change the `SOURCE_IMAGE` variable to use your registry and project.
4. At the very end of the file add:

```
allow_k8s_contexts('your-cluster-name')
```

5. Inside the directory, run:

```
tanzu apps workload apply tanzu-java-web-app --live-update --local-path . -s gcr.io/PROJECT/tanzu-java-web-app-live-update -y
```

Expected output:

```

The files and directories listed in the .tanzuignore file are being excluded from the uploaded source code.
Publishing source in "." to "gcr.io/PROJECT/tanzu-java-web-app-live-update"...
  Published source

Create workload:
 1 + |---
 2 + |apiVersion: carto.run/v1alpha1
 3 + |kind: Workload
 4 + |metadata:
 5 + |  name: tanzu-java-web-app
 6 + |  namespace: default
 7 + |spec:
 8 + |  params:
 9 + |    - name: live-update
10 + |      value: "true"
11 + |  source:
12 + |    image: gcr.io/PROJECT/tanzu-java-web-app-live-update:latest@sha256:3c9fd738492a23ac532a709301fcf0c9aa2a8761b2b9347bdbab52ce9404264b
Created workload "tanzu-java-web-app"

To see logs:  "tanzu apps workload tail tanzu-java-web-app --timestamp --since 1h"

```

```
To get status: "tanzu apps workload get tanzu-java-web-app"
```

## 6. Run Tilt to deploy the workload.

```
tilt up

Tilt started on http://localhost:10350/
v0.23.6, built 2022-01-14

(space) to open the browser
(s) to stream logs (--stream=true)
(t) to open legacy terminal mode (--legacy=true)
(ctrl-c) to exit
Tilt started on http://localhost:10350/
v0.23.6, built 2022-01-14

Initial Build • (Tiltfile)
Loading Tiltfile at: /path/to/repo/tanzu-java-web-app/Tiltfile
Successfully loaded Tiltfile (1.500809ms)
tanzu-java-w... |
tanzu-java-w... | Initial Build • tanzu-java-web-app
tanzu-java-w... | WARNING: Live Update failed with unexpected error:
tanzu-java-w... |   Cannot extract live updates on this build graph structure
tanzu-java-w... | Falling back to a full image build + deploy
tanzu-java-w... | STEP 1/1 – Deploying
tanzu-java-w... |   Objects applied to cluster:
tanzu-java-w... |     → tanzu-java-web-app:workload
tanzu-java-w... |
tanzu-java-w... |   Step 1 - 8.87s (Deploying)
tanzu-java-w... |   DONE IN: 8.87s
tanzu-java-w... |
tanzu-java-w... | Tracking new pod rollout (tanzu-java-web-app-build-1-build-po
d):
tanzu-java-w... |   | Scheduled      - (...) Pending
tanzu-java-w... |   | Initialized    - (...) Pending
tanzu-java-w... |   | Ready          - (...) Pending
...
```

## -export

Use this flag to retrieve the workload definition with all the extraneous, cluster-specific, properties, and values removed. For example, the status and metadata text boxes like `creationTimestamp`. This allows you to apply the workload definition to a different environment without having to make significant edits.

This means that the workload definition includes only the text boxes that were specified by the developer that created it (`--export` preserves the essence of the developer's intent for portability).

For example, if you create a workload with:

```
tanzu apps workload apply rmq-sample-app --git-repo https://github.com/jhvhs/rabbitmq-sample --git-branch main --service-ref "rmq=rabbitmq.com/v1beta1:RabbitmqCluster:example-rabbitmq-cluster-1" -t web
```

When querying the workload with `--export`, the default export format in YAML is as follows:

```
# with yaml format
tanzu apps workload get rmq-sample-app --export
---
apiVersion: carto.run/v1alpha1
kind: Workload
```



```

metadata:
  labels:
    apps.tanzu.vmware.com/workload-type: web
name: rmq-sample-app
namespace: default
spec:
  serviceClaims:
  - name: rmq
    ref:
      apiVersion: rabbitmq.com/v1beta1
      kind: RabbitmqCluster
      name: example-rabbitmq-cluster-1
  source:
    git:
      ref:
        branch: main
      url: https://github.com/jhvhs/rabbitmq-sample

# with json format
tanzu apps workload get rmq-sample-app --export --output json
{
  "apiVersion": "carto.run/v1alpha1",
  "kind": "Workload",
  "metadata": {
    "labels": {
      "apps.tanzu.vmware.com/workload-type": "web"
    },
    "name": "rmq-sample-app",
    "namespace": "default"
  },
  "spec": {
    "serviceClaims": [
      {
        "name": "rmq",
        "ref": {
          "apiVersion": "rabbitmq.com/v1beta1",
          "kind": "RabbitmqCluster",
          "name": "example-rabbitmq-cluster-1"
        }
      }
    ],
    "source": {
      "git": {
        "ref": {
          "branch": "main"
        },
        "url": "https://github.com/jhvhs/rabbitmq-sample"
      }
    }
  }
}

```

## -output

Use this flag to retrieve the workload including all the cluster-specifics. The `--output` flag can also be used in conjunction with the `--export` flag to set the export format as `json`, `yaml`, or `yml`.

```

# with json format
tanzu apps workload get rmq-sample-app --output json # can also be used as tanzu apps
workload get rmq-sample-app -ojson
{
  "kind": "Workload",
  "apiVersion": "carto.run/v1alpha1",
  "metadata": {

```

```

"name": "rmq-sample-app",
"namespace": "default",
"uid": "3619ff6d-9e73-473a-9112-891a6d8aee9e",
"resourceVersion": "11657434",
"generation": 2,
"creationTimestamp": "2022-11-28T05:10:32Z",
"labels": {
  "apps.tanzu.vmware.com/workload-type": "web"
},
"managedFields": [
  {
    "manager": "v0.10.0+dev-002cc44e",
    "operation": "Update",
    "apiVersion": "carto.run/v1alpha1",
    "time": "2022-11-28T05:10:32Z",
    "fieldsType": "FieldsV1",
    "fieldsV1": {
      "f:metadata": {
        "f:labels": {
          ".": {},
          "f:apps.tanzu.vmware.com/workload-type": {}
        }
      },
      ...
    }
  },
  ...
]
},
...
"status": {
"observedGeneration": 2,
"conditions": [
  {
    "type": "SupplyChainReady",
    "status": "True",
    "lastTransitionTime": "2022-11-28T05:10:32Z",
    "reason": "Ready",
    "message": ""
  },
  {
    "type": "ResourcesSubmitted",
    "status": "True",
    "lastTransitionTime": "2022-11-28T05:13:33Z",
    "reason": "ResourceSubmissionComplete",
    "message": ""
  },
  ...
],
"supplyChainRef": {
  "kind": "ClusterSupplyChain",
  "name": "source-to-url"
},
"resources": [
  {
    "name": "source-provider",
    "stampedRef": {
      "kind": "GitRepository",
      "namespace": "default",
      "name": "rmq-sample-app",
      "apiVersion": "source.toolkit.fluxcd.io/v1beta1",
      "resource": "gitrepositories.source.toolkit.fluxcd.io"
    },
    "templateRef": {
      "kind": "ClusterSourceTemplate",
      "name": "source-template",

```

```

        "apiVersion": "carto.run/v1alpha1"
      },
      ...
    ]
  ...
}
...
}

## with yaml format
tanzu apps workload get rmq-sample-app --output yaml # can also be used as tanzu apps
workload get rmq-sample-app -oyaml
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  creationTimestamp: "2022-11-28T05:10:32Z"
  generation: 2
  labels:
    apps.tanzu.vmware.com/workload-type: web
  managedFields:
  - apiVersion: carto.run/v1alpha1
    ...
    manager: v0.10.0+dev-002cc44e
    operation: Update
    time: "2022-11-28T05:10:32Z"
  - apiVersion: carto.run/v1alpha1
    fieldsType: FieldsV1
    ...
    manager: cartographer
    operation: Update
    subresource: status
    time: "2022-11-28T05:10:36Z"
  name: rmq-sample-app
  namespace: default
  resourceVersion: "11657434"
  uid: 3619ff6d-9e73-473a-9112-891a6d8aee9e
spec:
  serviceClaims:
  - name: rmq
    ref:
      apiVersion: rabbitmq.com/v1beta1
      kind: RabbitmqCluster
      name: example-rabbitmq-cluster-1
  source:
    git:
      ref:
        branch: main
      url: https://github.com/jhvhs/rabbitmq-sample
status:
  conditions:
  - lastTransitionTime: "2022-11-28T05:10:32Z"
    message: ""
    reason: Ready
    status: "True"
    type: SupplyChainReady
  ...
  observedGeneration: 2
  resources:
  ...
  name: source-provider
  outputs:
  - digest: sha256:97b2cb779b4ea31339595cd204a3fec0053805eeacbbd6d6dd23af7d3000a6ae
    lastTransitionTime: "2022-11-28T05:16:01Z"

```

```

name: url
preview: |
  http://fluxcd-source-controller.flux-system.svc.cluster.local./gitrepository/default/rmq-sample-app/73c6311eefbf724fee9ad6f4524fa24ec842ff34.tar.gz
- digest: sha256:e7884b071fe1bbb2551d42a171043d061a7591e744705572136e689c2a154b7a
  lastTransitionTime: "2022-11-28T05:16:01Z"
name: revision
preview: |
  HEAD/73c6311eefbf724fee9ad6f4524fa24ec842ff34
stampedRef:
  apiVersion: source.toolkit.fluxcd.io/v1beta1
  kind: GitRepository
  name: rmq-sample-app
  namespace: default
  resource: gitrepositories.source.toolkit.fluxcd.io
templateRef:
  apiVersion: carto.run/v1alpha1
  kind: ClusterSourceTemplate
  name: source-template
- conditions:
- lastTransitionTime: "2022-11-28T05:13:25Z"
  message: ""
  reason: ResourceSubmissionComplete
  status: "True"
  type: ResourceSubmitted
...
inputs:
- name: source-provider

```

## –sub-path

Use this flag to support use cases where more than one application is in a single project or repository.

Use `--sub-path` when creating a workload from a Git repository.

```

```console
tanzu apps workload apply subpathtester --git-repo https://github.com/PATH-TO-REPO --git-branch main --type web --sub-path SUBPATH

Create workload:
1 + |---
2 + |apiVersion: carto.run/v1alpha1
3 + |kind: Workload
4 + |metadata:
5 + |  labels:
6 + |    apps.tanzu.vmware.com/workload-type: web
7 + |  name: subpathtester
8 + |  namespace: default
9 + |spec:
10 + |  source:
11 + |    git:
12 + |      ref:
13 + |        branch: main
14 + |        url: https://github.com/path-to-repo/PATH-TO-REPO
15 + |        subPath: SUBPATH
Do you want to create this workload? [yN]:
```

```

Use `--sub-path` when you create a workload from local source code. In the directory of the project you want to create the workload from:

```

```console
tanzu apps workload apply WORKLOAD --local-path . -s gcr.io/REGISTRY/WORKLOAD-IMAGE

```

```

--sub-path SUBPATH

  Publish source in "." to "gcr.io/REGISTRY/WORKLOAD-IMAGE"? It might be visible to
others who can pull images from that repository Yes
  Publishing source in "." to "gcr.io/REGISTRY/WORKLOAD-IMAGE"...
  Published source

  Create workload:
  1 + |---
  2 + |apiVersion: carto.run/v1alpha1
  3 + |kind: Workload
  4 + |metadata:
  5 + |  name: WORKLOAD
  6 + |  namespace: default
  7 + |spec:
  8 + |  source:
  9 + |    image: gcr.io/REGISTRY/my-workload-image:latest@sha256:f28c5fedd0e902
800e6df9605ce5e20a8e835df9e87b1a0aa256666ea179fc3f
 10 + |    subPath: SUBPATH
  Do you want to create this workload? [yN]:

...

```

**Note** In cases where a workload must be created from local source code, to reduce the total amount of code that is uploaded, set the `--local-path` value to point directly to the directory containing the code rather than using `--sub-path`.

## .tanziignore file

There are many files and directories in projects that are not connected to running code (these files are not part of the final running container). When creating a workload from local source code, list these unused files and directories in the `.tanziignore` file to avoid unnecessary consumption of resources when uploading the source.

When iterating on code with the `--live-update` flag enabled, changes to directories or files listed in `.tanziignore` do not trigger the automatic re-deployment of the source code.

The following are some guidelines for the `.tanziignore` file:

- The `.tanziignore` file should include a reference to itself, as it provides no value when deployed.
- Directories must not end with the system separator `/`, or `\`.
- Comments using hashtag `#` can be included.
- If the `.tanziignore` file contains files or directories that are not found in the source code, they are ignored.

## Example of a .tanziignore file

```

.tanziignore # must contain itself in order to be ignored
# This is a comment
this/is/a/folder/to/exclude

this-is-a-file.ext

```

## -dry-run

Use the `--dry-run` flag to prepare all the steps to submit a workload to the cluster but stop before sending it, and display an output of the final structure of the workload.

For example, when applying a workload from Git source:

```
tanzu apps workload apply rmq-app --git-repo https://github.com/jhvhs/rabbitmq-sample
--git-branch main --service-ref "rmq=rabbitmq.com/v1beta1:RabbitmqCluster:example-rabb
itm-cluster-1" -t web --dry-run
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  creationTimestamp: null
  labels:
    apps.tanzu.vmware.com/workload-type: web
  name: rmq-app
  namespace: default
spec:
  serviceClaims:
  - name: rmq
    ref:
      apiVersion: rabbitmq.com/v1beta1
      kind: RabbitmqCluster
      name: example-rabbitmq-cluster-1
  source:
    git:
      ref:
        branch: main
        url: https://github.com/jhvhs/rabbitmq-sample
  status:
    supplyChainRef: {}
```

Certify how a workload is created or updated in the cluster based on the current specifications passed through `--file workload.yaml` or command flags.

If there is an error applying the workload, this is shown with the `--dry-run` flag:

```
tanzu apps workload create rmq-sample-app --git-repo https://github.com/jhvhs/rabbitmq
-sample --git-branch main --service-ref "rmq=rabbitmq.com/v1beta1:RabbitmqCluster:exam
ple-rabbitmq-cluster-1" -t web --dry-run
Error: workload "default/rmq-sample-app" already exists
```

## -update-strategy

Use this flag to control whether configuration properties and values passed through `--file workload.yaml` for an existing workload `merge` with, or `replace` (overwrite), existing on-cluster properties or values set for a workload.

The `--update-strategy` flag accepts two values: `merge` (default), and `replace`.

With the default `merge`:

If the `--file workload.yaml` deletes an existing on-cluster property or value, that property is not removed from the on-cluster definition. If the `--file workload.yaml` includes a new property or value, it is added to the on-cluster workload properties/values. If the `--file workload.yaml` updates an existing value for a property, that property's value on-cluster is updated.

With `replace`:

The on-cluster workload is updated to exactly what is specified in the `--file workload.yaml` definition.

The intent of the current default merge strategy is to prevent unintentional deletions of critical properties from existing workloads.

### Note



The default value for the `--update-strategy` flag will change from `merge` to `replace` in Tanzu Application Platform v1.7.0.

Examples of the outcomes of both `merge` and `replace` update strategies are provided in the following examples:

- ```
# Export workload if there is no previous yaml definition
tanzu apps workload get spring-petclinic --export > spring-petclinic.yaml

# modify the workload definition
vi rmq-sample-app.yaml
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: spring-petclinic
labels:
  app.kubernetes.io/part-of: spring-petclinic
  apps.tanzu.vmware.com/workload-type: web
spec:
  resources:
    requests:
      memory: 1Gi
    limits:
      memory: 1Gi # delete this line
      cpu: 500m # delete this line
  source:
    git:
      url: https://github.com/sample-accelerators/spring-petclinic
      ref:
        tag: tap-1.1
```

After saving the file, to verify how both of the update strategy options behave, run:

```
tanzu apps workload apply -f ./spring-petclinic.yaml --update-strategy merge # if flag
is not specified, merge is taken as default
```

This produces the following output:

```
WARNING: Configuration file update strategy is changing. By default, provided config
uration files
will replace rather than merge existing configuration. The change will take place in t
he January 2024
Tanzu Application Platform release (use "--update-strategy" to control strategy explic
itly).

Workload is unchanged, skipping update
```

By contrast, use `replace` as follows:

```
tanzu apps workload apply -f ./spring-petclinic.yaml --update-strategy replace
```

This produces the following output:

```
WARNING: Configuration file update strategy is changing. By default, provided config
uration files
will replace rather than merge existing configuration. The change will take place in t
he January 2024
Tanzu Application Platform release (use "--update-strategy" to control strategy explic
itly).

Update workload:
```

```

...
 8, 8  | name: spring-petclinic
 9, 9  | namespace: default
10, 10 | spec:
11, 11 | resources:
12    - | limits:
13      - |   cpu: 500m
14      - |   memory: 1Gi
15, 12 | requests:
16, 13 |   memory: 1Gi
17, 14 | source:
18, 15 |   git:
...
Really update the workload "spring-petclinic"? [yN]:

```

The lines that were deleted in the YAML file are deleted as well in the workload running in the cluster. The only text boxes that remain exactly as they were created are the system populated metadata text boxes (`resourceVersion`, `uuid`, `generation`, `creationTimestamp`, `deletionTimestamp`).

## Debug workloads

This topic tells you how to use the Tanzu Apps CLI to debug workloads.

## Verify build logs

### Check build logs

After a workload is created, 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 entry.

## 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 any time during the process. Some details, such as the workload URL, are only available after the workload is running.

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.

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



## Common workload errors

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

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

### Local Path Development Error Cases

Message: Writing `registry/project/repo/workload:latest`: Writing image: Unexpected status code `401 Unauthorized` (HEAD responses have no body, use GET for details)

Cause: Apps plug-in cannot talk to the registry because the registry credentials are missing or invalid.

Resolution: Run `docker logout registry` and `docker login registry` commands and specify the valid credentials for the registry.

Message: Writing `registry/project/workload:latest`: Writing image: HEAD Unexpected status code `400 Bad Request` (HEAD responses have no body, use GET for details)

Cause: Certain registries like Harbor or GCR have a concept of `Project`. A 400 Bad request is sent when either the project does not exist, the user does not have access to it, or the path in the `--source-image` flag is missing either project or repository.

Resolution: Fix the path in the `--source-image` flag value to point to a valid repository path.

### WorkloadLabelsMissing/SupplyChainNotFound

Message: No supply chain found where full selector is satisfied by `labels: map[app.kubernetes.io/part-of:spring-petclinic]`

Cause: The labels and attributes in the workload object did not fully satisfy any installed supply chain on the cluster.

Resolution: Use the `tanzu apps cluster-supply-chain list` (alias `csc`) and `tanzu apps csc get <supply-chain-name>` commands to see the workload selection criteria for the supply chain available on the cluster. Apply any missing labels to a workload by using `tanzu apps workload apply --label required-label-name=required-label-value`. For example:

```
tanzu apps workload apply workload-name --type web
tanzu apps workload apply workload-name --label apps.tanzu.vmware.com/workload-type=web
```

### MissingValueAtPath

Message: Waiting to read value `[.status.artifact.url]` from resource `gitrepository.source.toolkit.fluxcd.io` in namespace `[ns]`

Possible Cause: The Git `url/tag/branch/commit` parameters passed in the workload are not valid.

Resolution: Fix the invalid Git parameters by using `tanzu apps workload apply`

Possible Cause: The Git repository is not accessible from the cluster

Resolution: Configure your cluster networking or your Git repository networking so that they can communicate with each other.

Possible Cause: The namespace is missing the Git secret for communicating with the private repository

Resolution: For more information, see [Git authentication](#)

## TemplateRejectedByAPIServer

Message: Unable to apply object `[ns/workload-name]` for resource `[source-provider]` in supply chain `[source-to-url]`: failed to get unstructured `[ns/workload-name]` from API server: `imagerepositories.source.apps.tanzu.vmware.com` “workload-name” is forbidden: User “system:serviceaccount:ns:default” cannot get resource “imagerepositories” in API group “source.apps.tanzu.vmware.com” in the namespace “ns”

Cause: This error happens when the service account in the workload object does not have permission to create objects that are stamped out by the supply chain.

Resolution: Set up the [Set up developer namespaces to use your installed packages](#) with the required service account and permissions.

## Review supply chain steps

After you create a workload with the `tanzu apps workload create` (or) `apply`, command, you can run the `tanzu apps workload get` command to display the current condition of each supply chain.

For example:

```
...
  Supply Chain
  name:   source-to-url

  NAME          READY  HEALTHY  UPDATED  RESOURCE
  source-provider  True   True     71m      gitrepositories.source.toolkit.fluxcd.io/spring-petclinic
  image-provider  True   True     70m      images.kpack.io/spring-petclinic
  config-provider  True   True     69m      podintents.conventions.carto.run/spring-petclinic
  app-config      True   True     69m      configmaps/spring-petclinic
  service-bindings True   True     69m      configmaps/spring-petclinic-with-claims
  api-descriptors True   True     69m      configmaps/spring-petclinic-with-api-descriptors
  config-writer   True   True     69m      runnables.carto.run/spring-petclinic-config-writer

  Delivery
  name:   delivery-basic

  NAME          READY  HEALTHY  UPDATED  RESOURCE
  source-provider  True   True     69m      imagerepositories.source.apps.tanzu.vmware.com/spring-petclinic-delivery
  deployer        True   True     69m      apps.kappctrl.k14s.io/spring-petclinic

  Messages
  No messages found.
...
```

The **Supply Chain** section displays the supply chain steps associated with the workload. If a step fails, the **READY** column value is `Unknown` or `False`, and the **HEALTHY** column value is `False`. If there is a resource in `Unknown` or `False` status, inspect it with:

```
kubectl describe RESOURCE-NAME
```

Where **RESOURCE-NAME** refers to the name of the stamped out resource, displayed in **RESOURCE** column.

For example, if `tanzu apps workload get` command returns this resource:

| NAME            | READY | HEALTHY | UPDATED | RESOURCE                                                  |
|-----------------|-------|---------|---------|-----------------------------------------------------------|
| source-provider | False | False   | 3h12m   | gitrepositories.source.toolkit.fluxcd.io/spring-petclinic |

Whatever is going on with this resource can be checked with:

```
kubectl describe gitrepositories.source.toolkit.fluxcd.io/spring-petclinic
```

The [Messages](#) section might give a hint as to what went wrong in the process. For example, a message similar to the following is shown:

```
Messages
  Workload [HealthyConditionRule]:   failed to checkout and determine revision: failed to resolve commit object for '425ae9a2a2f84d195a9f3862668e8b2abf81418a': object not found
```

This could mean that the given commit does not belong to the specified branch or does not exist in the repo.

## Additional Troubleshooting References

Additional Workload troubleshooting tips can be found on the [Troubleshoot using Tanzu Application Platform](#) page.

## Tanzu Apps CLI commands

The following topics describe the Tanzu CLI Apps plug-in commands.

- [tanzu apps clustersupplychain](#) sub-commands and details.
- [tanzu apps workload](#) sub-commands and flags usage for each:
  - [tanzu apps workload get](#)
  - [tanzu apps workload list](#)
  - [tanzu apps workload tail](#)
  - [tanzu apps workload delete](#)
  - [tanzu apps workload apply](#)

## Tanzu Apps CLI commands

The following topics describe the Tanzu CLI Apps plug-in commands.

- [tanzu apps clustersupplychain](#) sub-commands and details.
- [tanzu apps workload](#) sub-commands and flags usage for each:
  - [tanzu apps workload get](#)
  - [tanzu apps workload list](#)
  - [tanzu apps workload tail](#)
  - [tanzu apps workload delete](#)
  - [tanzu apps workload apply](#)

## tanzu apps cluster-supply-chain

This topic tells you about the Tanzu Apps CLI [cluster-supply-chain](#) command.

## Tanzu apps cluster supply chain list

The `tanzu apps clustersupplychain list` command lists the available supply chains installed in the cluster (supported clustersupplychain alias is `csc`).

Run the following command to view more detailed information about the selectors and conditions that must be met for a workload to be selected by a certain supply chain:

```
tanzu apps clustersupplychain get SUPPLYCHAIN-NAME`.
```

### Default view

The default view displays the name of the supply chain, whether it is ready or not, and its age.

For example:

```
tanzu apps clustersupplychain list
NAME                READY  AGE
basic-image-to-url  Ready  11d
source-to-url       Ready  11d

To view details: "tanzu apps cluster-supply-chain get <name>"
```

## Tanzu apps cluster supply chain get

The `tanzu apps clustersupplychain get` command gets detailed information of the cluster supply chain.

### Default view

The default view displays the status of the supply chain, and the selectors that a workload must match so it is taken by that supply chain.

For example:

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

This output indicates the attributes a workload needs to be selected by the `source-to-url` supply chain on the target cluster. For example:

- The workload must have the `--type` flag value of `web`, `server`, or `worker`.
- Or, if expressed through `workload.yaml`, the `Workload.metadata.labels` label `apps.tanzu.vmware.com/workload-type` must exist and have a value of `web`, `server`, or `worker`.

Another example is the `testing/scanning` pipeline, which has the `tekton` steps for testing and the scanning steps.

```
---
# source-test-scan-to-url: Ready
```

```

---
Supply Chain Selectors
  TYPE          KEY                                OPERATOR  VALUE
  labels        apps.tanzu.vmware.com/has-tests          true
  expressions   apps.tanzu.vmware.com/workload-type    In        web
  expressions   apps.tanzu.vmware.com/workload-type    In        server
  expressions   apps.tanzu.vmware.com/workload-type    In        worker

```

In this case, the workload must have both labels `apps.tanzu.vmware.com/has-tests: true` and `apps.tanzu.vmware.com/workload-type` set up as `web`, `server`, or `worker` to be selected for the supply chain.

## tanzu apps workload apply

This topic tells you about the Tanzu Apps CLI `tanzu apps workload apply` command.

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

The `tanzu apps workload apply` and `tanzu apps workload create` commands have the same behavior and flags with the following exceptions:

- The `tanzu apps workload create` command fails if a workload with the same name preexists on the target cluster.
- the `update-strategy` flag is only applicable to the `tanzu apps workload apply` command. The `update-strategy` flag is not applicable to the `tanzu apps workload create` command.

## Default view

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

Example

```

tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanzu/application-accelerator-samples --sub-path tanzu-java-web-app --git-tag tap-1.3 --type web
Create workload:
 1 + |---
 2 + |apiVersion: carto.run/v1alpha1
 3 + |kind: Workload
 4 + |metadata:
 5 + |  labels:
 6 + |    apps.tanzu.vmware.com/workload-type: web
 7 + |  name: tanzu-java-web-app
 8 + |  namespace: default
 9 + |spec:
10 + |  source:
11 + |    git:
12 + |      ref:
13 + |        tag: tap-1.3
14 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
15 + |      subPath: tanzu-java-web-app
Do you want to create this workload? [yN]:
Created workload "tanzu-java-web-app"

To see logs:  "tanzu apps workload tail tanzu-java-web-app --timestamp --since 1h"
To get status: "tanzu apps workload get tanzu-java-web-app"

```

In the first section, the definition of workload is displayed. It's followed by a dialog box asking `whether the workload should be created or updated`. In the last section, if a workload is created

or updated, some hints are displayed about the next steps.

## Workload Apply flags

### --annotation

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

#### Example

```
tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanzu/application-accelerator-samples --sub-path tanzu-java-web-app --git-tag tap-1.3 --type web --annotation tag=tap-1.3 --annotation name="Tanzu Java Web"
Create workload:
 1 + |---
 2 + |apiVersion: carto.run/v1alpha1
 3 + |kind: Workload
 4 + |metadata:
 5 + |  labels:
 6 + |    apps.tanzu.vmware.com/workload-type: web
 7 + |    name: tanzu-java-web-app
 8 + |    namespace: default
 9 + |spec:
10 + |  params:
11 + |    - name: annotations
12 + |      value:
13 + |        name: Tanzu Java Web
14 + |        tag: tap-1.3
15 + |    source:
16 + |      git:
17 + |        ref:
18 + |          tag: tap-1.3
19 + |        url: https://github.com/vmware-tanzu/application-accelerator-samples
20 + |        subPath: tanzu-java-web-app
```

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

#### Example

```
tanzu apps workload apply tanzu-java-web-app --annotation tag-
Update workload:
...
10, 10 |  params:
11, 11 |  - name: annotations
12, 12 |    value:
13, 13 |      name: Tanzu Java Web
14     - |      tag: tap-1.3
15, 14 |  source:
16, 15 |    git:
17, 16 |      ref:
18, 17 |        tag: tap-1.3
...
Really update the workload "tanzu-java-web-app"? [yN]:
```

### --app / -a

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

#### Example

```
tanzu apps workload apply tanzu-app --git-repo https://github.com/vmware-tanzu/application-accelerator-samples --sub-path tanzu-java-web-app --git-tag tap-1.3 --type web --app tanzu-java-web-app
Create workload:
 1 + |---
 2 + |apiVersion: carto.run/v1alpha1
 3 + |kind: Workload
 4 + |metadata:
 5 + |  labels:
 6 + |    app.kubernetes.io/part-of: tanzu-java-web-app
 7 + |    apps.tanzu.vmware.com/workload-type: web
 8 + |  name: tanzu-app
 9 + |  namespace: default
10 + |spec:
11 + |  source:
12 + |    git:
13 + |      ref:
14 + |        tag: tap-1.3
15 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
16 + |      subPath: tanzu-java-web-app
Do you want to create this workload? [yN]:
Created workload "tanzu-app"

To see logs:  "tanzu apps workload tail tanzu-app --timestamp --since 1h"
To get status: "tanzu apps workload get tanzu-app"
```

## --build-env

Sets environment variables to use in the build phase by the build resources in the supply chain.

### Example

```
tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanzu/application-accelerator-samples --sub-path tanzu-java-web-app --git-tag tap-1.3 --type web --build-env JAVA_VERSION=1.8
Create workload:
 1 + |---
 2 + |apiVersion: carto.run/v1alpha1
 3 + |kind: Workload
 4 + |metadata:
 5 + |  labels:
 6 + |    apps.tanzu.vmware.com/workload-type: web
 7 + |  name: tanzu-java-web-app
 8 + |  namespace: default
 9 + |spec:
10 + |  build:
11 + |    env:
12 + |      - name: JAVA_VERSION
13 + |        value: "1.8"
14 + |    source:
15 + |      git:
16 + |        ref:
17 + |          tag: tap-1.3
18 + |        url: https://github.com/vmware-tanzu/application-accelerator-samples
19 + |        subPath: tanzu-java-web-app
Do you want to create this workload? [yN]:
```

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

### Example

```
tanzu apps workload apply tanzu-java-web-app --build-env JAVA_VERSION-
Update workload:
```

```

...
6, 6 | apps.tanzu.vmware.com/workload-type: web
7, 7 | name: tanzu-java-web-app
8, 8 | namespace: default
9, 9 | spec:
10  - | build:
11  - |   env:
12  - |     - name: JAVA_VERSION
13  - |       value: "1.8"
14, 10 | source:
15, 11 |   git:
16, 12 |     ref:
17, 13 |       tag: tap-1.3
...
Really update the workload "tanzu-java-web-app"? [yN]:

```

## --debug

Sets the parameter variable debug to true in the workload.

### Example

```

tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanzu/application-accelerator-samples --sub-path tanzu-java-web-app --git-branch main --type web --debug
Create workload:
1 + |---
2 + |apiVersion: carto.run/v1alpha1
3 + |kind: Workload
4 + |metadata:
5 + |  labels:
6 + |    apps.tanzu.vmware.com/workload-type: web
7 + |    name: tanzu-java-web-app
8 + |    namespace: default
9 + |spec:
10 + |  params:
11 + |    - name: debug
12 + |      value: "true"
13 + |  source:
14 + |    git:
15 + |      ref:
16 + |        branch: main
17 + |        url: https://github.com/vmware-tanzu/application-accelerator-samples
18 + |        subPath: tanzu-java-web-app
Do you want to create this workload? [yN]:

```

## --dry-run

Prepares all the steps to submit the workload to the cluster and stops before sending it, showing an output of the final structure of the workload.

### Example

```

tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanzu/application-accelerator-samples --sub-path tanzu-java-web-app --git-tag tap-1.3 --type web --build-env JAVA_VERSION=1.8 --param-yaml server='port: 8080\nmanagement-port: 8181' --dry-run
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  creationTimestamp: null
  labels:
    apps.tanzu.vmware.com/workload-type: web

```



```

name: tanzu-java-web-app
namespace: default
spec:
  build:
    env:
      - name: JAVA_VERSION
        value: "1.8"
  params:
    - name: server
      value:
        management-port: 8181
        port: 8080
  source:
    git:
      ref:
        tag: tap-1.3
      url: https://github.com/vmware-tanzu/application-accelerator-samples
      subPath: tanzu-java-web-app
status:
  supplyChainRef: {}

```

### --env / -e

Sets the environment variables to the workload so the supply chain resources can use it to deploy the workload application.

#### Example

```

tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanzu/application-accelerator-samples --sub-path tanzu-java-web-app --git-tag tap-1.3 --type web --env NAME="Tanzu Java App"
Create workload:
 1 + |---
 2 + |apiVersion: carto.run/v1alpha1
 3 + |kind: Workload
 4 + |metadata:
 5 + |  labels:
 6 + |    apps.tanzu.vmware.com/workload-type: web
 7 + |  name: tanzu-java-web-app
 8 + |  namespace: default
 9 + |spec:
10 + |  env:
11 + |    - name: NAME
12 + |      value: Tanzu Java App
13 + |  source:
14 + |    git:
15 + |      ref:
16 + |        tag: tap-1.3
17 + |        url: https://github.com/vmware-tanzu/application-accelerator-samples
18 + |        subPath: tanzu-java-web-app
Do you want to create this workload? [yN]:

```

To unset an environment variable, use `-` after its name.

```

tanzu apps workload apply tanzu-java-web-app --env NAME-
Update workload:
...
 6, 6 |    apps.tanzu.vmware.com/workload-type: web
 7, 7 |  name: tanzu-java-web-app
 8, 8 |  namespace: default
 9, 9 |spec:
10   - |  env:
11   - |    - name: NAME
12   - |      value: Tanzu Java App

```

```

13, 10 | source:
14, 11 |   git:
15, 12 |     ref:
16, 13 |     tag: tap-1.3
...
Really update the workload "tanzu-java-web-app"? [yN]:

```

### --file, -f

Sets the workload specification file to create the workload. This comes from any other workload specification passed by flags to the command set or overrides what is in the file. Another way to use this flag is by using `-` in the command to receive workload definition through stdin. See [Working with YAML Files](#) for an example.

#### Example

```

tanzu apps workload apply tanzu-java-web-app -f java-app-workload.yaml --param-yaml server=${port: 9090\nmanagement-port: 9190}
Create workload:
 1 + |---
 2 + |apiVersion: carto.run/v1alpha1
 3 + |kind: Workload
 4 + |metadata:
 5 + |  labels:
 6 + |    apps.tanzu.vmware.com/workload-type: web
 7 + |  name: tanzu-java-web-app
 8 + |  namespace: default
 9 + |spec:
10 + |  build:
11 + |    env:
12 + |      - name: JAVA_VERSION
13 + |        value: "1.8"
14 + |    params:
15 + |      - name: server
16 + |        value:
17 + |          management-port: 9190
18 + |          port: 9090
19 + |    source:
20 + |      git:
21 + |        ref:
22 + |          tag: tap-1.3
23 + |        url: url: https://github.com/vmware-tanzu/application-accelerator-samples
24 + |      subPath: tanzu-java-web-app
Do you want to create this workload? [yN]:

```

### --git-repo

The Git repository from which the workload is created. You can specify `--git-tag`, or `--git-commit`.

### --git-branch

The branch in a Git repository from where the workload is created. This is specified with a commit or a tag.

#### Example

```

tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanzu/application-accelerator-samples --sub-path tanzu-java-web-app --git-branch main --type web
Create workload:
 1 + |---
 2 + |apiVersion: carto.run/v1alpha1

```

```

3 + |kind: Workload
4 + |metadata:
5 + |  labels:
6 + |    apps.tanzu.vmware.com/workload-type: web
7 + |  name: tanzu-java-web-app
8 + |  namespace: default
9 + |spec:
10 + |  source:
11 + |    git:
12 + |      ref:
13 + |        branch: main
14 + |        url: https://github.com/vmware-tanzu/application-accelerator-samples
15 + |        subPath: tanzu-java-web-app
Do you want to create this workload? [yN]:

```

### --git-tag

The tag in a Git repository from which the workload is created. This is used with `--git-commit` or `--git-branch`.

### --git-commit

Commit in Git repository from where the workload is resolved. Can be used with `--git-branch` or `--git-tag`.

### Example

```

tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanzu/application-accelerator-samples --sub-path tanzu-java-web-app --git-tag tap-1.3 --git-commit 1c4cf82e499f7e46da182922d4097908d4817320 --type web
Create workload:
1 + |---
2 + |apiVersion: carto.run/v1alpha1
3 + |kind: Workload
4 + |metadata:
5 + |  labels:
6 + |    apps.tanzu.vmware.com/workload-type: web
7 + |  name: tanzu-java-web-app
8 + |  namespace: default
9 + |spec:
10 + |  source:
11 + |    git:
12 + |      ref:
13 + |        commit: 1c4cf82e499f7e46da182922d4097908d4817320
14 + |        tag: tap-1.3
15 + |        url: https://github.com/vmware-tanzu/application-accelerator-samples
16 + |        subPath: tanzu-java-web-app
Do you want to create this workload? [yN]:

```

### --image / -i

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

### Example

```

tanzu apps workload apply tanzu-java-web-app --image private.repo.domain.com/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 + |   name: tanzu-java-web-app
 8 + |   namespace: default
 9 + |spec:
10 + |   build:
11 + |     env:
12 + |       - name: JAVA_VERSION
13 + |         value: "1.8"
14 + |   params:
15 + |     - name: server
16 + |       value:
17 + |         management-port: 9190
18 + |         port: 9090
19 + |   source:
20 + |     git:
21 + |       ref:
22 + |         tag: tap-1.3
23 + |       url: https://github.com/vmware-tanzu/application-accelerator-samples
24 + |     subPath: tanzu-java-web-app
Do you want to create this workload? [yN]:

```

### --label / -l

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

#### Example

```

tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanzu/application-accelerator-samples --sub-path tanzu-java-web-app --git-branch main --type web --label stage=production
Create workload:
 1 + |---
 2 + |apiVersion: carto.run/v1alpha1
 3 + |kind: Workload
 4 + |metadata:
 5 + |  labels:
 6 + |    apps.tanzu.vmware.com/workload-type: web
 7 + |    stage: production
 8 + |  name: tanzu-java-web-app
 9 + |  namespace: default
10 + |spec:
11 + |  source:
12 + |    git:
13 + |      ref:
14 + |        branch: main
15 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
16 + |      subPath: tanzu-java-web-app
Do you want to create this workload? [yN]:

```

To unset labels, use `-` after their name.

#### Example

```

tanzu apps workload apply tanzu-java-web-app --label stage-
Update workload:
...
 3, 3 |kind: Workload
 4, 4 |metadata:
 5, 5 |  labels:
 6, 6 |    apps.tanzu.vmware.com/workload-type: web
 7   - |    stage: production
 8, 7 |  name: tanzu-java-web-app
 9, 8 |  namespace: default
10, 9 |spec:

```

```

11, 10 | source:
...
Really update the workload "tanzu-java-web-app"? [yN]:

```

## --limit-cpu

The maximum CPU the workload pods are allowed to use.

### Example

```

tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanzu/application-accelerator-samples --sub-path tanzu-java-web-app --git-branch main --type web --limit-cpu .2
Create workload:
 1 + |---
 2 + |apiVersion: carto.run/v1alpha1
 3 + |kind: Workload
 4 + |metadata:
 5 + |  labels:
 6 + |    apps.tanzu.vmware.com/workload-type: web
 7 + |    name: tanzu-java-web-app
 8 + |    namespace: default
 9 + |spec:
10 + |  resources:
11 + |    limits:
12 + |      cpu: 200m
13 + |    source:
14 + |      git:
15 + |        ref:
16 + |          branch: main
17 + |        url: https://github.com/vmware-tanzu/application-accelerator-samples
18 + |        subPath: tanzu-java-web-app
Do you want to create this workload? [yN]:

```

## --limit-memory

The maximum memory the workload pods are allowed to use.

### Example

```

tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanzu/application-accelerator-samples --sub-path tanzu-java-web-app --git-branch main --type web --limit-memory 200Mi
Create workload:
 1 + |---
 2 + |apiVersion: carto.run/v1alpha1
 3 + |kind: Workload
 4 + |metadata:
 5 + |  labels:
 6 + |    apps.tanzu.vmware.com/workload-type: web
 7 + |    name: tanzu-java-web-app
 8 + |    namespace: default
 9 + |spec:
10 + |  resources:
11 + |    limits:
12 + |      memory: 200Mi
13 + |    source:
14 + |      git:
15 + |        ref:
16 + |          branch: main
17 + |        url: https://github.com/vmware-tanzu/application-accelerator-samples
18 + |        subPath: tanzu-java-web-app
Do you want to create this workload? [yN]:

```

## --live-update

Enable this to deploy the workload once, save changes to the code, and see those changes reflected in the workload running on the cluster.

### Example

An example with a Spring Boot application:

1. Clone the repository by running:

```
git clone https://github.com/vmware-tanzu/application-accelerator-samples
```

2. Change into the `tanzu-java-web-app` directory.
3. In `Tiltfile`, first change the `SOURCE_IMAGE` variable to use your registry and project.
4. At the very end of the file add:

```
allow_k8s_contexts('your-cluster-name')
```

5. Inside the directory, run:

```
tanzu apps workload apply tanzu-java-web-app --live-update --local-path . -s
gcr.io/my-project/tanzu-java-web-app-live-update -y
```

### Expected output:

```
The files and directories listed in the .tanuignore file are being excluded fr
om the uploaded source code.
Publishing source in "." to "gcr.io/my-project/tanzu-java-web-app-live-updat
e"...
  Published source

  Create workload:
  1 + |---
  2 + |apiVersion: carto.run/v1alpha1
  3 + |kind: Workload
  4 + |metadata:
  5 + |  name: tanzu-java-web-app
  6 + |  namespace: default
  7 + |spec:
  8 + |  params:
  9 + |    - name: live-update
 10 + |      value: "true"
 11 + |  source:
 12 + |    image: gcr.io/my-project/tanzu-java-web-app-live-update:latest@sha2
56:3c9fd738492a23ac532a709301fcf0c9aa2a8761b2b9347bdbab52ce9404264b

  Created workload "tanzu-java-web-app"

To see logs:  "tanzu apps workload tail tanzu-java-web-app --timestamp --since
1h"
To get status: "tanzu apps workload get tanzu-java-web-app"
```

6. Run Tilt to deploy the workload.

```
tilt up

Tilt started on http://localhost:10350/
v0.23.6, built 2022-01-14

(space) to open the browser
```

```

(s) to stream logs (--stream=true)
(t) to open legacy terminal mode (--legacy=true)
(ctrl-c) to exit
Tilt started on http://localhost:10350/
v0.23.6, built 2022-01-14

Initial Build • (Tiltfile)
Loading Tiltfile at: /path/to/repo/tanzu-java-web-app/Tiltfile
Successfully loaded Tiltfile (1.500809ms)
tanzu-java-w... |
tanzu-java-w... | Initial Build • tanzu-java-web-app
tanzu-java-w... | WARNING: Live Update failed with unexpected error:
tanzu-java-w... |   Cannot extract live updates on this build graph structure
tanzu-java-w... | Falling back to a full image build + deploy
tanzu-java-w... | STEP 1/1 - Deploying
tanzu-java-w... |   Objects applied to cluster:
tanzu-java-w... |     → tanzu-java-web-app:workload
tanzu-java-w... |
tanzu-java-w... |   Step 1 - 8.87s (Deploying)
tanzu-java-w... |   DONE IN: 8.87s
tanzu-java-w... |
tanzu-java-w... | Tracking new pod rollout (tanzu-java-web-app-build-1-build-po
d):
tanzu-java-w... |   | Scheduled      - (...) Pending
tanzu-java-w... |   | Initialized    - (...) Pending
tanzu-java-w... |   | Ready          - (...) Pending
...

```

## --local-path

Sets the path to a source in the local machine from where the workload creates an image to use as an application source. The local path may be a directory, a JAR, a ZIP, or a WAR file. Java/Spring Boot compiled binaries are also supported. This flag must be used with `--source-image` flag.

If Java/Spring compiled binary is passed instead of source code, the command takes less time to apply the workload since the build pack skips the compiling steps and start uploading the image.

When working with local source code, you can exclude files from the source code to be uploaded within the image by creating a file `.tanzuignore` at the root of the source code. The `.tanzuignore` file contains a list of file paths to exclude from the image including the file itself. The directories must not end with the system path separator (`/` or `\`). If the file contains directories that are not in the source code, they are ignored. Lines starting with a `#` hashtag are also ignored.

## --maven-artifact

This artifact is an output of a Maven project build. This flag must be used with `--maven-version` and `--maven-group`.

### Example

```

tanzu apps workload apply petc-mvn --maven-artifact petc --maven-version 2.6.1 --maven
-group demo.com
  Create workload:
  1 + |---
  2 + |apiVersion: carto.run/v1alpha1
  3 + |kind: Workload
  4 + |metadata:
  5 + |  name: petc-mvn
  6 + |  namespace: default
  7 + |spec:
  8 + |  params:
  9 + |    - name: maven

```

```

10 + |     value:
11 + |         artifactId: petc
12 + |         groupId: demo.com
13 + |         version: 2.6.1
Do you want to create this workload? [yN]:

```

### --maven-group

This group identifies the project across all other Maven projects.

### --maven-type

This specifies the type of artifact that the Maven project produces. This flag is optional and is set by default as `jar` by the supply chain.

### --maven-version

Definition of the current version of the Maven project.

### --source-image, -s

Registry path where the local source code is uploaded as an image.

#### Example

```

tanzu apps workload apply spring-pet-clinic --local-path /home/user/workspace/spring-p
et-clinic --source-image gcr.io/spring-community/spring-pet-clinic --type web
Publish source in "/home/user/workspace/spring-pet-clinic" to "gcr.io/spring-communi
ty/spring-pet-clinic"? It may be visible to others who can pull images from that repos
itory Yes
The files and/or directories listed in the .tanzuignore file are being excluded from t
he uploaded source code.
Publishing source in "/home/user/workspace/spring-pet-clinic" to "gcr.io/spring-commun
ity/spring-pet-clinic"...
Published source

Create workload:
 1 + |---
 2 + |apiVersion: carto.run/v1alpha1
 3 + |kind: Workload
 4 + |metadata:
 5 + |  labels:
 6 + |    apps.tanzu.vmware.com/workload-type: web
 7 + |  name: spring-pet-clinic
 8 + |  namespace: default
 9 + |spec:
10 + |  source:
11 + |    image:gcr.io/spring-community/spring-pet-clinic:latest@sha256:5feb0d9da
f3f639755d8683ca7b647027cfddc7012e80c61dcdac27f0d7856a7
Do you want to create this workload? [yN]:

```

### --namespace, -n

Specifies the namespace in which the workload is created or updated in.

#### Example

```

tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanz
u/application-accelerator-samples --sub-path tanzu-java-web-app --git-branch main --ty
pe web --namespace my-namespace
Create workload:
 1 + |---

```



```

2 + |apiVersion: carto.run/v1alpha1
3 + |kind: Workload
4 + |metadata:
5 + |  labels:
6 + |    apps.tanzu.vmware.com/workload-type: web
7 + |  name: tanzu-java-web-app
8 + |  namespace: my-namespace
9 + |spec:
10 + |  source:
11 + |    git:
12 + |      ref:
13 + |        branch: main
14 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
15 + |      subPath: tanzu-java-web-app
Do you want to create this workload? [yN]:

```

### --param / -p

Additional parameters to be sent to the supply chain, the value is sent as a string. For complex YAML and JSON objects use `--param-yaml`.

#### Example

```

tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanzu/application-accelerator-samples --sub-path tanzu-java-web-app --git-branch main --type web --param port=9090 --param management-port=9190
Create workload:
1 + |---
2 + |apiVersion: carto.run/v1alpha1
3 + |kind: Workload
4 + |metadata:
5 + |  labels:
6 + |    apps.tanzu.vmware.com/workload-type: web
7 + |  name: tanzu-java-web-app
8 + |  namespace: default
9 + |spec:
10 + |  params:
11 + |    - name: port
12 + |      value: "9090"
13 + |    - name: management-port
14 + |      value: "9190"
15 + |  source:
16 + |    git:
17 + |      ref:
18 + |        branch: main
19 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
20 + |      subPath: tanzu-java-web-app
Do you want to create this workload? [yN]:

```

To unset parameters, use `-` after their name.

#### Example

```

tanzu apps workload apply tanzu-java-web-app --param port-
Update workload:
...
7, 7 | name: tanzu-java-web-app
8, 8 | namespace: default
9, 9 |spec:
10, 10 | params:
11 - | - name: port
12 - | value: "9090"
13, 11 | - name: management-port
14, 12 | value: "9190"
15, 13 | source:

```

```

16, 14 |   git:
...
Really update the workload "tanzu-java-web-app"? [yN]:

```

### --param-yaml

Additional parameters to be sent to the supply chain, the value is sent as a complex object.

#### Example

```

tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanzu/application-accelerator-samples --sub-path tanzu-java-web-app --git-branch main --type web --param-yaml server=${port: 9090\nmanagement-port: 9190}'
Create workload:
 1 + |---
 2 + |apiVersion: carto.run/v1alpha1
 3 + |kind: Workload
 4 + |metadata:
 5 + |  labels:
 6 + |    apps.tanzu.vmware.com/workload-type: web
 7 + |  name: tanzu-java-web-app
 8 + |  namespace: default
 9 + |spec:
10 + |  params:
11 + |    - name: server
12 + |      value:
13 + |        management-port: 9190
14 + |        port: 9090
15 + |  source:
16 + |    git:
17 + |      ref:
18 + |        branch: main
19 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
20 + |      subPath: tanzu-java-web-app
Do you want to create this workload? [yN]:

```

To unset parameters, use - after their name.

#### Example

```

tanzu apps workload apply tanzu-java-web-app --param-yaml server-
Update workload:
...
 6, 6 |   apps.tanzu.vmware.com/workload-type: web
 7, 7 |   name: tanzu-java-web-app
 8, 8 |   namespace: default
 9, 9 |spec:
10  - |  params:
11  - |    - name: server
12  - |      value:
13  - |        management-port: 9190
14  - |        port: 9090
15, 10 |  source:
16, 11 |    git:
17, 12 |      ref:
18, 13 |        branch: main
...
Really update the workload "tanzu-java-web-app"? [yN]:

```

### --registry-ca-cert

Refers to the path of the self-signed certificate needed for the custom/private registry. This is also populated with a default value through environment variables. If the environment variable

`TANZU_APPS_REGISTRY_CA_CERT` is set, it's not necessary to use it in the command.

See [Custom registry credentials](#) for the currently supported environment variables.

#### Example

```
tanzu apps workload apply my-workload --local-path . -s registry.url.nip.io/my-packag
e/my-image --type web --registry-ca-cert path/to/cacert/mycert.nip.io.crt --registry-u
sername my-username --registry-password my-password
Publish source in "." to "registry.url.nip.io/my-package/my-image"? It may be visibl
e to others who can pull images from that repository Yes
Publishing source in "." to "registry.url.nip.io/my-package/my-image"...
Published source

Create workload:
 1 + |---
 2 + |apiVersion: carto.run/v1alpha1
 3 + |kind: Workload
 4 + |metadata:
 5 + |  labels:
 6 + |    apps.tanzu.vmware.com/workload-type: web
 7 + |  name: my-workload
 8 + |  namespace: default
 9 + |spec:
10 + |  source:
11 + |    image: registry.url.nip.io/my-package/my-image:latest@sha256:caeb7e3a0e
3ae0659f74d01095b6fdfe0d3c4a12856a15ac67ad6cd3b9e43648
Do you want to create this workload? [yN]:
```

#### `--registry-password`

If credentials are needed, the user name and password values are set through the `--registry-password` flag. The value of this flag can also be specified through `TANZU_APPS_REGISTRY_PASSWORD`.

#### `--registry-token`

Used for token authentication in the private registry. This flag is set as `TANZU_APPS_REGISTRY_TOKEN` environment variable.

#### `--registry-username`

Often used with `--registry-password` to set private registry credentials. Can be provided using `TANZU_APPS_REGISTRY_USERNAME` environment variable to avoid setting it every time in the command.

#### `--request-cpu`

Refers to the minimum CPU the workload pods request to use.

#### Example

```
tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanz
u/application-accelerator-samples --sub-path tanzu-java-web-app --git-branch main --ty
pe web --request-cpu .3
Create workload:
 1 + |---
 2 + |apiVersion: carto.run/v1alpha1
 3 + |kind: Workload
 4 + |metadata:
 5 + |  labels:
 6 + |    apps.tanzu.vmware.com/workload-type: web
 7 + |  name: tanzu-java-web-app
 8 + |  namespace: default
```

```

 9 + |spec:
10 + | resources:
11 + |   requests:
12 + |     cpu: 300m
13 + |   source:
14 + |     git:
15 + |       ref:
16 + |         branch: main
17 + |       url: https://github.com/vmware-tanzu/application-accelerator-samples
18 + |       subPath: tanzu-java-web-app
Do you want to create this workload? [yN]:

```

### --request-memory

Refers to the minimum memory the workload pods are requesting to use.

#### Example

```

tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanzu/application-accelerator-samples --sub-path tanzu-java-web-app --git-branch main --type web --request-memory 300Mi
Create workload:
 1 + |---
 2 + |apiVersion: carto.run/v1alpha1
 3 + |kind: Workload
 4 + |metadata:
 5 + |  labels:
 6 + |    apps.tanzu.vmware.com/workload-type: web
 7 + |  name: tanzu-java-web-app
 8 + |  namespace: default
 9 + |spec:
10 + |  resources:
11 + |    requests:
12 + |      memory: 300Mi
13 + |  source:
14 + |    git:
15 + |      ref:
16 + |        branch: main
17 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
18 + |      subPath: tanzu-java-web-app
Do you want to create this workload? [yN]:

```

### --service-account

Refers to the service account to associate with the workload. A service account provides an identity for a workload object.

#### Example

```

tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanzu/application-accelerator-samples --sub-path tanzu-java-web-app --git-branch main --type web --service-account petc-serviceaccount
Create workload:
 1 + |---
 2 + |apiVersion: carto.run/v1alpha1
 3 + |kind: Workload
 4 + |metadata:
 5 + |  labels:
 6 + |    apps.tanzu.vmware.com/workload-type: web
 7 + |  name: tanzu-java-web-app
 8 + |  namespace: default
 9 + |spec:
10 + |  serviceAccountName: petc-serviceaccount
11 + |  source:

```

```

12 + |     git:
13 + |       ref:
14 + |         branch: main
15 + |     url: https://github.com/vmware-tanzu/application-accelerator-samples
16 + |     subPath: tanzu-java-web-app
Do you want to create this workload? [yN]:

```

To unset a service account, pass empty string.

#### Example

```

tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanzu/application-accelerator-samples --sub-path tanzu-java-web-app --git-branch main --type web --service-account ""
Update workload:
...
6, 6 |     apps.tanzu.vmware.com/workload-type: web
7, 7 |     name: tanzu-java-web-app
8, 8 |     namespace: default
9, 9 | spec:
10  - |   serviceAccountName: petc-serviceaccount
11, 10 |   source:
12, 11 |     git:
13, 12 |       ref:
14, 13 |         branch: main
...
Really update the workload "tanzu-java-web-app"? [yN]:

```

#### --service-ref

Binds a service to a workload to provide the information from a service resource to an application.

**Note** For more information see [Tanzu Application Platform documentation](#).

#### Example

```

tanzu apps workload apply rmq-sample-app --git-repo https://github.com/jhvhs/rabbitmq-sample --git-branch main --service-ref "rmq=rabbitmq.com/v1beta1:RabbitmqCluster:example-rabbitmq-cluster-1"
Create workload:
1 + |---
2 + |apiVersion: carto.run/v1alpha1
3 + |kind: Workload
4 + |metadata:
5 + |  name: rmq-sample-app
6 + |  namespace: default
7 + |spec:
8 + |  serviceClaims:
9 + |    - name: rmq
10 + |      ref:
11 + |        apiVersion: rabbitmq.com/v1beta1
12 + |        kind: RabbitmqCluster
13 + |        name: example-rabbitmq-cluster-1
14 + |  source:
15 + |    git:
16 + |      ref:
17 + |        branch: main
18 + |      url: https://github.com/jhvhs/rabbitmq-sample
Do you want to create this workload? [yN]:

```

To delete service binding, use the service name followed by -.

#### Example

```
tanzu apps workload apply rmq-sample-app --service-ref rmq-
Update workload:
...
 4, 4 |metadata:
 5, 5 |  name: rmq-sample-app
 6, 6 |  namespace: default
 7, 7 |spec:
 8   - |  serviceClaims:
 9   - |    - name: rmq
10   - |      ref:
11   - |        apiVersion: rabbitmq.com/v1beta1
12   - |        kind: RabbitmqCluster
13   - |        name: example-rabbitmq-cluster-1
14, 8 |  source:
15, 9 |    git:
16,10 |      ref:
17,11 |        branch: main
...
Really update the workload "rmq-sample-app"? [yN]:
```

## --sub-path

Defines which path is used as the root path to create and update the workload.

### Example

- Git repository

```
tanzu apps workload apply subpathtester --git-repo https://github.com/path-to-r
epo/my-repo --git-branch main --type web --sub-path my-subpath
Create workload:
 1 + |---
 2 + |apiVersion: carto.run/v1alpha1
 3 + |kind: Workload
 4 + |metadata:
 5 + |  labels:
 6 + |    apps.tanzu.vmware.com/workload-type: web
 7 + |  name: subpathtester
 8 + |  namespace: default
 9 + |spec:
10 + |  source:
11 + |    git:
12 + |      ref:
13 + |        branch: main
14 + |        url: https://github.com/path-to-repo/my-repo
15 + |        subPath: my-subpath
Do you want to create this workload? [yN]:
```

- Local path
  - In the directory of the project you want to create the workload from

```
tanzu apps workload apply my-workload --local-path . -s gcr.io/my-registr
y/my-workload-image --sub-path subpath_folder
Publish source in "." to "gcr.io/my-registry/my-workload-image"? It may
be visible to others who can pull images from that repository Yes
Publishing source in "." to "gcr.io/my-registry/my-workload-image"...
Published source

Create workload:
 1 + |---
 2 + |apiVersion: carto.run/v1alpha1
 3 + |kind: Workload
 4 + |metadata:
 5 + |  name: myworkload
```

```

6 + | namespace: default
7 + |spec:
8 + | source:
9 + |   image: gcr.io/my-registry/my-workload-image:latest@sha256:f2
8c5fedd0e902800e6df9605ce5e20a8e835df9e87b1a0aa256666eal79fc3f
10 + |   subPath: subpath_folder
Do you want to create this workload? [yN]:

```

## --tail

Prints the logs of the workload creation in every step.

### Example

```

tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanzu/application-accelerator-samples --sub-path tanzu-java-web-app --git-branch main --type web --tail
Create workload:
 1 + |---
 2 + |apiVersion: carto.run/v1alpha1
 3 + |kind: Workload
 4 + |metadata:
 5 + |  labels:
 6 + |    apps.tanzu.vmware.com/workload-type: web
 7 + |  name: tanzu-java-web-app
 8 + |  namespace: default
 9 + |spec:
10 + | source:
11 + |   git:
12 + |     ref:
13 + |       branch: main
14 + |     url: https://github.com/vmware-tanzu/application-accelerator-samples
15 + |     subPath: tanzu-java-web-app
Do you want to create this workload? [yN]: y
Created workload "tanzu-java-web-app"

To see logs: "tanzu apps workload tail tanzu-java-web-app --timestamp --since 1h"
To get status: "tanzu apps workload get tanzu-java-web-app"

Waiting for workload "tanzu-java-web-app" to become ready...
+ tanzu-java-web-app-build-1-build-pod > prepare
tanzu-java-web-app-build-1-build-pod[prepare] Build reason(s): CONFIG
tanzu-java-web-app-build-1-build-pod[prepare] CONFIG:
tanzu-java-web-app-build-1-build-pod[prepare]   + env:
tanzu-java-web-app-build-1-build-pod[prepare]   + - name: BP_OCI_SOURCE
tanzu-java-web-app-build-1-build-pod[prepare]   +   value: main/d381fb658cb435a04e2271ca85bd3e8627a5e7e4
tanzu-java-web-app-build-1-build-pod[prepare]   resources: {}
tanzu-java-web-app-build-1-build-pod[prepare]   - source: {}
tanzu-java-web-app-build-1-build-pod[prepare]   + source:
tanzu-java-web-app-build-1-build-pod[prepare]   +   blob:
tanzu-java-web-app-build-1-build-pod[prepare]   +     url: http://source-controller.flux-system.svc.cluster.local./gitrepository/default/tanzu-java-web-app/1c4cf82e499f7e46da182922d4097908d4817320.tar.gz
...
...
...

```

## --tail-timestamp

Prints the logs of the workload creation in every step adding the time in which the log is occurring.

### Example

```

tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanzu/application-accelerator-samples --sub-path tanzu-java-web-app --git-branch main --type web --tail-timestamp
Create workload:
 1 + |---
 2 + |apiVersion: carto.run/v1alpha1
 3 + |kind: Workload
 4 + |metadata:
 5 + |  labels:
 6 + |    apps.tanzu.vmware.com/workload-type: web
 7 + |  name: tanzu-java-web-app
 8 + |  namespace: default
 9 + |spec:
10 + |  source:
11 + |    git:
12 + |      ref:
13 + |        branch: main
14 + |        url: https://github.com/vmware-tanzu/application-accelerator-samples
15 + |        subPath: tanzu-java-web-app
Do you want to create this workload? [yN]: y
Created workload "tanzu-java-web-app"

To see logs: "tanzu apps workload tail tanzu-java-web-app --timestamp --since 1h"
To get status: "tanzu apps workload get tanzu-java-web-app"

Waiting for workload "tanzu-java-web-app" to become ready...
+ tanzu-java-web-app-build-1-build-pod > prepare
tanzu-java-web-app-build-1-build-pod[prepare] 2022-06-15T11:28:01.348418803-05:00 Build reason(s): CONFIG
tanzu-java-web-app-build-1-build-pod[prepare] 2022-06-15T11:28:01.364719405-05:00 CONFIG:
tanzu-java-web-app-build-1-build-pod[prepare] 2022-06-15T11:28:01.364761781-05:00 + env:
tanzu-java-web-app-build-1-build-pod[prepare] 2022-06-15T11:28:01.364771861-05:00 + - name: BP_OCI_SOURCE
tanzu-java-web-app-build-1-build-pod[prepare] 2022-06-15T11:28:01.364781718-05:00 + value: main/d381fb658cb435a04e2271ca85bd3e8627a5e7e4
tanzu-java-web-app-build-1-build-pod[prepare] 2022-06-15T11:28:01.364788374-05:00 re sources: {}
tanzu-java-web-app-build-1-build-pod[prepare] 2022-06-15T11:28:01.364795451-05:00 - source: {}
tanzu-java-web-app-build-1-build-pod[prepare] 2022-06-15T11:28:01.365344965-05:00 + source:
tanzu-java-web-app-build-1-build-pod[prepare] 2022-06-15T11:28:01.365364101-05:00 + blob:
tanzu-java-web-app-build-1-build-pod[prepare] 2022-06-15T11:28:01.365372427-05:00 + url: http://source-controller.flux-system.svc.cluster.local./gitrepository/default/tanzu-java-web-app/1c4cf82e499f7e46da182922d4097908d4817320.tar.gz
...
...
...

```

## --type / -t

Sets the type of workload by adding the label `apps.tanzu.vmware.com/workload-type`, which is used as a matcher by supply chains. Use the `TANZU_APPS_TYPE` environment variable to have a default value for this flag.

### Example

```

tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanzu/application-accelerator-samples --sub-path tanzu-java-web-app --git-branch main --type web
Create workload:

```



```

1 + |---
2 + |apiVersion: carto.run/v1alpha1
3 + |kind: Workload
4 + |metadata:
5 + |  labels:
6 + |    apps.tanzu.vmware.com/workload-type: web
7 + |  name: tanzu-java-web-app
8 + |  namespace: default
9 + |spec:
10 + |  source:
11 + |    git:
12 + |      ref:
13 + |        branch: main
14 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
15 + |      subPath: tanzu-java-web-app

```

## --update-strategy



### Note

This flag is only applicable to the `tanzu apps workload apply` command. It is not applicable to the `tanzu apps workload create` command.

### Example

Assuming there is a workload created from a file, which has in its `spec` the following:

```

...
spec:
  resources:
    requests:
      memory: 1Gi
    limits:           # delete this line
      memory: 1Gi     # delete this line
      cpu: 500m       # delete this line
...

```

If the workload file is changed as specified in the comments, there are two ways to update the workload running in the cluster.

One, with `merge` update strategy.

```

tanzu apps workload apply -f ./spring-petclinic.yaml # defaulting to merge

WARNING: Configuration file update strategy is changing. By default, provided configuration files will replace rather than merge existing configuration. The change will take place in the January 2024 TAP release (use "--update-strategy" to control strategy explicitly).

Workload is unchanged, skipping update

```

The other, with `replace` update strategy, which will completely overwrite the workload in the cluster according to the new specifications in the file.

```

tanzu apps workload apply -f ./spring-petclinic.yaml --update-strategy replace

WARNING: Configuration file update strategy is changing. By default, provided configuration files will replace rather than merge existing configuration. The change will take place in the January 2024 TAP release (use "--update-strategy" to control strategy explicitly).

```

```

Update workload:
...
 8,  8  | name: spring-petclinic
 9,  9  | namespace: default
10, 10  | spec:
11, 11  | resources:
12     - |   limits:
13     - |     cpu: 500m
14     - |     memory: 1Gi
15, 12  |   requests:
16, 13  |     memory: 1Gi
17, 14  | source:
18, 15  |   git:
...
Really update the workload "spring-petclinic"? [yN]:

```

## --wait

Holds the command until the workload is ready.

### Example

```

tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanzu/application-accelerator-samples --sub-path tanzu-java-web-app --git-tag tap-1.3 --type web --wait
Update workload:
...
10, 10  | source:
11, 11  |   git:
12, 12  |     ref:
13, 13  |       branch: main
14 + |       tag: tap-1.3
14, 15  |   url: https://github.com/vmware-tanzu/application-accelerator-samples
15, 16  |   subPath: tanzu-java-web-app
Really update the workload "tanzu-java-web-app"? Yes
Updated workload "tanzu-java-web-app"

To see logs:  "tanzu apps workload tail tanzu-java-web-app --timestamp --since 1h"
To get status: "tanzu apps workload get tanzu-java-web-app"

Waiting for workload "tanzu-java-web-app" to become ready...
Workload "tanzu-java-web-app" is ready

```

## --wait-timeout

Sets a timeout to wait for the workload to become ready.

### Example

```

tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanzu/application-accelerator-samples --sub-path tanzu-java-web-app --git-tag tap-1.3-take1 --type web --wait --wait-timeout 1m
Update workload:
...
10, 10  | source:
11, 11  |   git:
12, 12  |     ref:
13, 13  |       branch: main
14     - |       tag: tap-1.3
14 + |       tag: tap-1.3-take1
15, 15  |   url: https://github.com/vmware-tanzu/application-accelerator-samples
16, 16  |   subPath: tanzu-java-web-app
Really update the workload "tanzu-java-web-app"? Yes
Updated workload "tanzu-java-web-app"

```

```
To see logs: "tanzu apps workload tail tanzu-java-web-app --timestamp --since 1h"
To get status: "tanzu apps workload get tanzu-java-web-app"

Waiting for workload "tanzu-java-web-app" to become ready...
Workload "tanzu-java-web-app" is ready
```

## --yes, -y

Assumes `--yes` on all the survey prompts.

## Example

```
tanzu apps workload apply spring-pet-clinic --local-path/home/user/workspace/spring-pe
t-clinic --source-image gcr.io/spring-community/spring-pet-clinic --type web -y
The files and/or directories listed in the .tanzuignore file are being excluded from t
he uploaded source code.
Publishing source in "/Users/dalfonso/Documents/src/java/tanzu-java-web-app" to "gcr.i
o/spring-community/spring-pet-clinic"...
  Published source

Create workload:
  1 + |---
  2 + |apiVersion: carto.run/v1alpha1
  3 + |kind: Workload
  4 + |metadata:
  5 + |  labels:
  6 + |    apps.tanzu.vmware.com/workload-type: web
  7 + |  name: spring-pet-clinic
  8 + |  namespace: default
  9 + |spec:
 10 + |  source:
 11 + |    image: gcr.io/spring-community/spring-pet-clinic:latest@sha256:5feb0d9d
af3f639755d8683ca7b647027cfddc7012e80c61dcdac27f0d7856a7
  Created workload "spring-pet-clinic"

To see logs: "tanzu apps workload tail spring-pet-clinic --timestamp --since 1h"
To get status: "tanzu apps workload get spring-pet-clinic"
```

## tanzu apps workload delete

This topic tells you about the Tanzu Apps CLI `tanzu apps workload delete` command. This command deletes workloads in a cluster. Deleting a workload does not mean the images published in the registry are deleted with it.

## Default view

A message is displayed in the terminal asking if a workload should be deleted unless the `--yes` flag is used. If you indicate “Y”, then the workload starts a deletion process inside the cluster.

```
tanzu apps workload delete spring-pet-clinic
Really delete the workload "spring-pet-clinic"? Yes
Deleted workload "spring-pet-clinic"
```

```
tanzu apps workload delete spring-pet-clinic --yes
Deleted workload "spring-pet-clinic"
```

## Workload Delete flags

**--all**

Deletes all workloads in a namespace.

```
tanzu apps workload delete --all
Really delete all workloads in the namespace "default"? (y/N) Y
Deleted workloads in namespace "default"
```

```
tanzu apps workload delete --all -n my-namespace
Really delete all workloads in the namespace "my-namespace"? Yes
Deleted workloads in namespace "my-namespace"
```

**--file, -f**

Path to a file that contains the specification of the workload to be deleted.

```
tanzu apps workload delete -f path/to/file/spring-petclinic.yaml
Really delete the workload "spring-petclinic"? Yes
Deleted workload "spring-petclinic"
```

**--namespace, -n**

Specifies the namespace in which the workload is to be deleted.

```
tanzu apps workload delete spring-petclinic -n spring-petclinic-ns
Really delete the workload "spring-petclinic"? Yes
Deleted workload "spring-petclinic"
```

**wait**

Waits until workload is deleted.

```
tanzu apps workload delete -f path/to/file/spring-petclinic.yaml --wait
Really delete the workload "spring-petclinic"? Yes
Deleted workload "spring-petclinic"
Waiting for workload "spring-petclinic" to be deleted...
Workload "spring-petclinic" was deleted
```

**--wait-timeout**

Sets a timeout to wait for workload to be deleted.

```
tanzu apps workload delete -f path/to/file/spring-petclinic.yaml --wait --wait-timeout
1m
Really delete the workload "spring-petclinic"? Yes
Deleted workload "spring-petclinic"
Waiting for workload "spring-petclinic" to be deleted...
Workload "spring-petclinic" was deleted
```

```
tanzu apps workload delete spring-petclinic -n spring-petclinic-ns --wait --wait-timeo
ut 1m
Really delete the workload "spring-petclinic"? Yes
Deleted workload "spring-petclinic"
Waiting for workload "spring-petclinic" to be deleted...
Error: timeout after 1m waiting for "spring-petclinic" to be deleted
To view status run: tanzu apps workload get spring-petclinic --namespace spring-petcli
nic-ns
Error: exit status 1
```

```
✘ exit status 1
```

`--yes, -f`

Assume yes on all the survey prompts.

```
tanzu apps workload delete spring-petclinic --yes
Deleted workload "spring-petclinic"
```

## tanzu apps workload get

This topic tells you how to use the Tanzu Apps CLI `tanzu apps workload get` command to retrieve information and status about a workload.

Some of the workload details in the command output are as follows:

- Workload name, type, and namespace.
- The source code used to build the workload (or the pre-built OCI image).
- The supply chain that processed the workload.
- The specific resources within the supply chain that interacted with the workload, and the stamped out resources associated with each of those interactions.
- The delivery workflow that the application follows.
- Any issues associated with deploying the workload.
- The *Pods* the workload generates.
- And when applicable, the knative services related to the workload.

## Default view

There are multiple sections in the workload `get` command output. The following data is displayed:

- Name of the workload and its status.
- Displays source information of workload.
- If the workload was matched with a supply chain, the information of its name and the status is displayed.
- Information and status of the individual steps that is defined in the supply chain for the workload.
- Any issue with the workload: the name and corresponding message.
- Workload related resource information and status like services claims, related pods, knative services.

At the very end of the command output, a hint to follow up commands is also displayed.



### Note

the `Supply Chain` and `Delivery` sections are included in the command output depending on whether those resources are present on the target cluster (e.g. If the target includes only build components, there would be no `Delivery` resources available and therefore the `Delivery` section would not be included in the command output.).

```

tanzu apps workload get rmq-sample-app
Overview
  name:      rmq-sample-app
  type:      web
  namespace: default

Source
  type:      git
  url:       https://github.com/jhvhs/rabbitmq-sample
  branch:    main

Supply Chain
  name:      source-to-url

NAME          READY   HEALTHY   UPDATED   RESOURCE
source-provider      True    True      7d11h     gitrepositories.source.toolkit.fluxcd.io/rmq-sample-app
image-provider      True    True      2d18h     images.kpack.io/rmq-sample-app
config-provider     True    True      7d11h     podintents.conventions.carto.run/rmq-sample-app
app-config          True    True      7d11h     configmaps/rmq-sample-app
service-bindings    True    True      7d11h     configmaps/rmq-sample-app-with-claims
api-descriptors     True    True      7d11h     configmaps/rmq-sample-app-with-api-descriptors
config-writer       True    True      2d18h     runnables.carto.run/rmq-sample-app-config-writer

Delivery
  name:      delivery-basic

NAME          READY   HEALTHY   UPDATED   RESOURCE
source-provider      True    True      7d11h     imagerepositories.source.apps.tanzu.vmware.com/rmq-sample-app-delivery
deployer           True    True      6m25s     apps.kappctrl.k14s.io/rmq-sample-app

Messages
  No messages found.

Services
CLAIM  NAME                                KIND                API VERSION
rmq    example-rabbitmq-cluster-1         RabbitmqCluster     rabbitmq.com/v1beta1

Pods
NAME                                READY   STATUS    RESTARTS   AGE
rmq-sample-app-build-1-build-pod    0/1    Completed 0           56d
rmq-sample-app-build-2-build-pod    0/1    Completed 0           46d
rmq-sample-app-build-3-build-pod    0/1    Completed 0           45d
rmq-sample-app-config-writer-54mwk-pod 0/1    Completed 0           6d12h
rmq-sample-app-config-writer-74qvp-pod 0/1    Completed 0           6d16h
rmq-sample-app-config-writer-78r5w-pod 0/1    Completed 0           45d
rmq-sample-app-config-writer-9xs5f-pod 0/1    Completed 0           46d

Knative Services
NAME          READY   URL
rmq-sample-app  Ready   http://rmq-sample-app.default.127.0.0.1.nip.io

To see logs: "tanzu apps workload tail rmq-sample-app --timestamp --since 1h"

```

--export

Exports the submitted workload in `yaml` format. This flag can also be used with the `--output` flag. With `export`, the output is shortened because some text boxes are removed.

```
tanzu apps workload get tanzu-java-web-app --export
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
labels:
  apps.tanzu.vmware.com/workload-type: web
  autoscaling.knative.dev/min-scale: "1"
name: tanzu-java-web-app
namespace: default
spec:
source:
  git:
  ref:
    tag: tap-1.3
    url: https://github.com/vmware-tanzu/application-accelerator-samples
  subPath: tanzu-java-web-app
```

### `--output/-o`

Configures how the workload is being shown. This supports the values `yaml`, `yml`, and `json`, where `yaml` and `yml` are equal. It shows the actual workload in the cluster.

- `yaml/yml`

```
tanzu apps workload get tanzu-java-web-app -o yaml
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
creationTimestamp: "2022-06-03T18:10:59Z"
generation: 1
labels:
  apps.tanzu.vmware.com/workload-type: web
  autoscaling.knative.dev/min-scale: "1"
...
spec:
source:
  git:
  ref:
    tag: tap-1.1
    url: https://github.com/vmware-tanzu/application-accelerator-samples
  subPath: tanzu-java-web-app
status:
conditions:
- lastTransitionTime: "2022-06-03T18:10:59Z"
  message: ""
  reason: Ready
  status: "True"
  type: SupplyChainReady
- lastTransitionTime: "2022-06-03T18:14:18Z"
  message: ""
  reason: ResourceSubmissionComplete
  status: "True"
  type: ResourcesSubmitted
- lastTransitionTime: "2022-06-03T18:14:18Z"
  message: ""
  reason: Ready
  status: "True"
  type: Ready
```

```

observedGeneration: 1
resources:
...
supplyChainRef:
  kind: ClusterSupplyChain
  name: source-to-url
...

```

- json

```

tanzu apps workload get tanzu-java-web-app -o json
{
  "kind": "Workload",
  "apiVersion": "carto.run/v1alpha1",
  "metadata": {
    "name": "tanzu-java-web-app",
    "namespace": "default",
    "uid": "937679ca-9c72-4e23-bfef-6334e6c003a7",
    "resourceVersion": "111637840",
    "generation": 1,
    "creationTimestamp": "2022-06-03T18:10:59Z",
    "labels": {
      "apps.tanzu.vmware.com/workload-type": "web",
      "autoscaling.knative.dev/min-scale": "1"
    },
    ...
  }
  "spec": {
    "source": {
      "git": {
        "url": "https://github.com/vmware-tanzu/application-accelerator-
-samples",
        "ref": {
          "tag": "tap-1.3"
        }
      },
      "subPath": "tanzu-java-web-app"
    },
    "status": {
      "observedGeneration": 1,
      "conditions": [
        {
          "type": "SupplyChainReady",
          "status": "True",
          "lastTransitionTime": "2022-06-03T18:10:59Z",
          "reason": "Ready",
          "message": ""
        },
        {
          "type": "ResourcesSubmitted",
          "status": "True",
          "lastTransitionTime": "2022-06-03T18:14:18Z",
          "reason": "ResourceSubmissionComplete",
          "message": ""
        },
        {
          "type": "Ready",
          "status": "True",
          "lastTransitionTime": "2022-06-03T18:14:18Z",
          "reason": "Ready",
          "message": ""
        }
      ],
      "supplyChainRef": {
        "kind": "ClusterSupplyChain",

```



```

        "name": "source-to-url"
      },
      "resources": [
        {
          "name": "source-provider",
          "stampedRef": {
            "kind": "GitRepository",
            "namespace": "default",
            "name": "tanzu-java-web-app",
            ...
          }
        }
      ]
    }
  }
  ...
}
...
}

```

## --namespace/-n

Specifies the namespace where the workload is deployed.

```
tanzu apps workload get tanzu-java-web-app -n development
```

### Overview

```

name:      tanzu-java-web-app
type:      web
namespace: development

```

### Source

```

type:      git
url:       https://github.com/vmware-tanzu/application-accelerator-samples
sub-path:  tanzu-java-web-app
tag:       tap-1.3

```

### Supply Chain

```
name: source-to-url
```

| NAME             | READY | HEALTHY | UPDATED | RESOURCE                                                    |
|------------------|-------|---------|---------|-------------------------------------------------------------|
| source-provider  | True  | True    | 31m     | gitrepositories.source.toolkit.fluxcd.io/tanzu-java-web-app |
| image-provider   | True  | True    | 30m     | images.kpack.io/tanzu-java-web-app                          |
| config-provider  | True  | True    | 30m     | podintents.conventions.carto.run/tanzu-java-web-app         |
| app-config       | True  | True    | 30m     | configmaps/tanzu-java-web-app                               |
| service-bindings | True  | True    | 30m     | configmaps/tanzu-java-web-app-with-claims                   |
| api-descriptors  | True  | True    | 30m     | configmaps/tanzu-java-web-app-with-api-descriptors          |
| config-writer    | True  | True    | 30m     | runnables.carto.run/tanzu-java-web-app-config-writer        |

### Delivery

```
name: delivery-basic
```

| NAME            | READY | HEALTHY | UPDATED | RESOURCE                                                                   |
|-----------------|-------|---------|---------|----------------------------------------------------------------------------|
| source-provider | True  | True    | 30m     | imagerepositories.source.apps.tanzu.vmware.com/tanzu-java-web-app-delivery |
| deployer        | True  | True    | 30m     | apps.kappctrl.k14s.io/tanzu-java-web-app                                   |

### Messages

```
No messages found.
```

### Pods

| NAME                                       | READY | STATUS    | RESTARTS | AGE   |
|--------------------------------------------|-------|-----------|----------|-------|
| tanzu-java-web-app-build-11-build-pod      | 0/1   | Completed | 0        | 6d12h |
| tanzu-java-web-app-build-12-build-pod      | 0/1   | Completed | 0        | 22h   |
| tanzu-java-web-app-build-3-build-pod       | 0/1   | Completed | 0        | 60d   |
| tanzu-java-web-app-config-writer-655rb-pod | 0/1   | Completed | 0        | 21d   |
| tanzu-java-web-app-config-writer-7h8bn-pod | 0/1   | Completed | 0        | 6d12h |
| tanzu-java-web-app-config-writer-7xr6m-pod | 0/1   | Completed | 0        | 60d   |
| tanzu-java-web-app-config-writer-g9gp8-pod | 0/1   | Completed | 0        | 45d   |

#### Knative Services

| NAME               | READY | URL                                                |
|--------------------|-------|----------------------------------------------------|
| tanzu-java-web-app | Ready | http://tanzu-java-web-app.default.127.0.0.1.nip.io |

To see logs: "tanzu apps workload tail tanzu-java-web-app --namespace development --timestamp --since 1h"

## tanzu apps workload list

This topic tells you about the Tanzu Apps CLI `tanzu apps workload list` command.

The `tanzu apps workload list` command gets the workloads present in the cluster, either in the current namespace, in another namespace, or all namespaces.

### Default view

The default view for workload list is a table with the workloads present in the cluster in the specified namespace. This table has, in each row, the name of the workload, the app it is related to, its status and how long it's been in the cluster.

For example, in the default namespace

```
tanzu apps workload list
```

| NAME                | TYPE    | APP                | READY                 | AGE  |
|---------------------|---------|--------------------|-----------------------|------|
| nginx4              | web     | <empty>            | Ready                 | 7d9h |
| petclinic2          | web     | <empty>            | Ready                 | 29h  |
| rmq-sample-app      | web     | <empty>            | Ready                 | 164m |
| rmq-sample-app4     | web     | <empty>            | WorkloadLabelsMissing | 29d  |
| spring-pet-clinic   | web     | <empty>            | Unknown               | 166m |
| spring-petclinic2   | web     | spring-petclinic   | Unknown               | 29d  |
| spring-petclinic3   | <empty> | spring-petclinic   | Ready                 | 29d  |
| tanzu-java-web-app  | web     | tanzu-java-web-app | Ready                 | 40m  |
| tanzu-java-web-app2 | web     | tanzu-java-web-app | Ready                 | 20m  |

## >Workload List flags

`--all-namespaces, -A`

Shows workloads in all namespaces in cluster.

```
tanzu apps workload list -A
```

| NAMESPACE | TYPE | NAME           | APP     | READY |
|-----------|------|----------------|---------|-------|
| AGE       |      |                |         |       |
| default   | web  | nginx4         | <empty> | Ready |
| 7d9h      |      |                |         |       |
| default   | web  | petclinic2     | <empty> | Ready |
| 30h       |      |                |         |       |
| default   | web  | rmq-sample-app | <empty> | Ready |
| 179m      |      |                |         |       |

|                    |     |                     |                    |                          |
|--------------------|-----|---------------------|--------------------|--------------------------|
| default<br>29d     | web | rmq-sample-app4     | <empty>            | WorkloadLabelsMissing    |
| default<br>3h1m    | web | spring-pet-clinic   | <empty>            | Unknown                  |
| default<br>29d     | web | spring-petclinic2   | spring-petclinic   | Unknown                  |
| default<br>29d     | web | spring-petclinic3   | spring-petclinic   | Ready                    |
| default<br>40m     | web | tanzu-java-web-app  | tanzu-java-web-app | Ready                    |
| default<br>20m     | web | tanzu-java-web-app2 | tanzu-java-web-app | Ready                    |
| nginx-ns<br>ver 8d | web | nginx2              | <empty>            | TemplateRejectedByAPISer |
| nginx-ns<br>ver 8d | web | nginx4              | <empty>            | TemplateRejectedByAPISer |

### --app

Shows workloads which app is the one specified in the command.

```
tanzu apps workload list --app spring-petclinic
```

| NAME              | TYPE | READY   | AGE |
|-------------------|------|---------|-----|
| spring-petclinic2 | web  | Unknown | 29d |
| spring-petclinic3 | web  | Ready   | 29d |

### --namespace, -n

Lists all the workloads present in the specified namespace.

```
tanzu apps workload list -n my-namespace
```

| NAME | TYPE | APP     | READY                       | AGE |
|------|------|---------|-----------------------------|-----|
| app1 | web  | <empty> | TemplateRejectedByAPIServer | 8d  |
| app2 | web  | <empty> | Ready                       | 8d  |
| app3 | web  | <empty> | Unknown                     | 8d  |

### --output, -o

Allows to list all workloads in the specified namespace in yaml, yml or json format.

- yaml/yml

```
---
- apiVersion: carto.run/v1alpha1
  kind: Workload
  metadata:
    creationTimestamp: "2022-05-17T22:06:49Z"
    generation: 1
    labels:
      app.kubernetes.io/part-of: tanzu-java-web-app
      apps.tanzu.vmware.com/workload-type: web
    managedFields:
      ...
      ...
    manager: cartographer
    operation: Update
    time: "2022-05-17T22:06:52Z"
  name: tanzu-java-web-app2
  namespace: default
  resourceVersion: "6071972"
  uid: 7fbcd40d-4eb3-41dc-a1db-657b64148708
```

```

spec:
  source:
    git:
      ref:
        tag: tap-1.3
        url: https://github.com/vmware-tanzu/application-accelerator-sample
s
      subPath: tanzu-java-web-app
...
...
---
- apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  creationTimestamp: "2022-05-17T22:06:49Z"
  generation: 1
  labels:
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/workload-type: web
  managedFields:
    ...
    ...
  manager: cartographer
  operation: Update
  time: "2022-05-17T22:06:52Z"
name: tanzu-java-web-app
namespace: default
resourceVersion: "6071972"
uid: 7fbcd40d-4eb3-41dc-a1db-657b64148708
spec:
  source:
    git:
      ref:
        tag: tap-1.3
        url: https://github.com/vmware-tanzu/application-accelerator-sample
s
      subPath: tanzu-java-web-app
...
...

```

- json

```

[
  {
    "kind": "Workload",
    "apiVersion": "carto.run/v1alpha1",
    "metadata": {
      "name": "tanzu-java-web-app2",
      "namespace": "default",
      "uid": "7fbcd40d-4eb3-41dc-a1db-657b64148708",
      "resourceVersion": "6071972",
      "generation": 1,
      "creationTimestamp": "2022-05-17T22:06:49Z",
      "labels": {
        "app.kubernetes.io/part-of": "tanzu-java-web-app",
        "apps.tanzu.vmware.com/workload-type": "web"
      }
    },
    ...
  }
  ...
],
[
  {
    "kind": "Workload",
    "apiVersion": "carto.run/v1alpha1",
    "metadata": {
      "name": "tanzu-java-web-app",

```

```

        "namespace": "default",
        "uid": "7fbcd40d-4eb3-41dc-a1db-657b64148708",
        "resourceVersion": "6071972",
        "generation": 1,
        "creationTimestamp": "2022-05-17T22:06:49Z",
        "labels": {
            "app.kubernetes.io/part-of": "tanzu-java-web-app",
            "apps.tanzu.vmware.com/workload-type": "web"
        },
        ...
    }
    ...
},
...
...
]

```

## tanzu apps workload tail

This topic tells you about the Tanzu Apps CLI `tanzu apps workload tail` command.

The `tanzu apps workload tail` checks the runtime logs of a workload.

### Default view

Without timestamp set, workload tail will show the stage where it is and the log related.

```

+ spring-pet-clinic-build-1-build-pod > prepare
+ spring-pet-clinic-build-1-build-pod > detect
+ spring-pet-clinic-build-1-build-pod > analyze
+ spring-pet-clinic-build-1-build-pod > build
+ spring-pet-clinic-build-1-build-pod > restore
spring-pet-clinic-build-1-build-pod[detect] ===== Output: tanzu-buildpacks/poetry@
0.1.0 =====
spring-pet-clinic-build-1-build-pod[detect] pyproject.toml must include [tool.poetry.d
ependencies.python], see https://python-poetry.org/docs/pyproject/#dependencies-and-de
v-dependencies
spring-pet-clinic-build-1-build-pod[analyze] Restoring data for sbom from previous ima
ge
spring-pet-clinic-build-1-build-pod[detect] err: tanzu-buildpacks/poetry@0.1.0 (1)
spring-pet-clinic-build-1-build-pod[detect] ===== Output: tanzu-buildpacks/poetry@
0.1.0 =====
spring-pet-clinic-build-1-build-pod[detect] pyproject.toml must include [tool.poetry.d
ependencies.python], see https://python-poetry.org/docs/pyproject/#dependencies-and-de
v-dependencies
spring-pet-clinic-build-1-build-pod[detect] err: tanzu-buildpacks/poetry@0.1.0 (1)
spring-pet-clinic-build-1-build-pod[detect] 10 of 38 buildpacks participating
spring-pet-clinic-build-1-build-pod[detect] paketo-buildpacks/ca-certificates 3.1.0
spring-pet-clinic-build-1-build-pod[detect] paketo-buildpacks/bellsoft-liberica 9.2.0
spring-pet-clinic-build-1-build-pod[detect] paketo-buildpacks/syft 1.10.0
spring-pet-clinic-build-1-build-pod[detect] paketo-buildpacks/gradle 6.4.1
spring-pet-clinic-build-1-build-pod[detect] paketo-buildpacks/maven 6.4.0
spring-pet-clinic-build-1-build-pod[detect] paketo-buildpacks/executable-jar 6.1.0
spring-pet-clinic-build-1-build-pod[detect] paketo-buildpacks/apache-tomcat 7.2.0
spring-pet-clinic-build-1-build-pod[detect] paketo-buildpacks/dist-zip 5.2.0
spring-pet-clinic-build-1-build-pod[detect] paketo-buildpacks/spring-boot 5.8.0
spring-pet-clinic-build-1-build-pod[detect] paketo-buildpacks/image-labels 4.1.0
...
...
...

```

## >Workload Tail flags

**--component**

Set the component from which the tail command should stream the logs. The values that the flag can take depend on the final deployed pods label `app.kubernetes.io/component`, for example, `build`, `run` and, `config-writer`

```
tanzu apps workload tail pet-clinic --component build

pet-clinic-build-1-build-pod[export] Adding label 'io.buildpacks.project.metadata'
pet-clinic-build-1-build-pod[export] Adding label 'org.opencontainers.image.title'
pet-clinic-build-1-build-pod[export] Adding label 'org.opencontainers.image.version'
pet-clinic-build-1-build-pod[export] Adding label 'org.springframework.boot.version'
pet-clinic-build-1-build-pod[export] Adding label 'org.opencontainers.image.source'
pet-clinic-build-1-build-pod[export] Setting default process type 'web'
pet-clinic-build-1-build-pod[export] Saving gcr.io/dalfonso-tanzu-dev-frmrk/pet-clinic-default...
pet-clinic-build-1-build-pod[export] *** Images (sha256:2ae6154c4433d870a330a0c2fc825340c3ead2603e3d1526e47c47cb6297fffe):
pet-clinic-build-1-build-pod[export]          gcr.io/dalfonso-tanzu-dev-frmrk/pet-clinic-default
pet-clinic-build-1-build-pod[export]          gcr.io/dalfonso-tanzu-dev-frmrk/pet-clinic-default:b1.20220603.181107
pet-clinic-build-1-build-pod[export] Adding cache layer 'paketo-buildpacks/bellsoft-liberica:jdk'
pet-clinic-build-1-build-pod[export] Adding cache layer 'paketo-buildpacks/syft:syft'
pet-clinic-build-1-build-pod[export] Adding cache layer 'paketo-buildpacks/maven:application'
pet-clinic-build-1-build-pod[export] Adding cache layer 'paketo-buildpacks/maven:cache'
pet-clinic-build-1-build-pod[export] Adding cache layer 'cache.sbom'
```

**--namespace, -n**

Specifies the namespace where the workload was deployed to get logs from.

```
tanzu apps workload tail pet-clinic -n development

pet-clinic-00004-deployment-6445565f7b-ts815[workload] 2022-06-14 16:28:52.684 INFO 1
--- [          main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.63]
+ pet-clinic-build-3-build-pod > export
pet-clinic-00004-deployment-6445565f7b-ts815[workload] 2022-06-14 16:28:52.699 INFO 1
--- [          main] o.a.c.c.C.[Tomcat-1].[localhost].[/] : Initializing Spring embedded WebApplicationContext
pet-clinic-00004-deployment-6445565f7b-ts815[workload] 2022-06-14 16:28:52.699 INFO 1
--- [          main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 131 ms
pet-clinic-00004-deployment-6445565f7b-ts815[workload] 2022-06-14 16:28:52.755 INFO 1
--- [          main] o.s.b.a.e.web.EndpointLinksResolver : Exposing 13 endpoint(s) beneath base path '/actuator'
pet-clinic-00004-deployment-6445565f7b-ts815[workload] 2022-06-14 16:28:53.059 INFO 1
--- [          main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8081 (http) with context path ''
pet-clinic-00004-deployment-6445565f7b-ts815[workload] 2022-06-14 16:28:53.074 INFO 1
--- [          main] o.s.s.petclinic.PetClinicApplication : Started PetClinicApplication in 8.373 seconds (JVM running for 8.993)
pet-clinic-00004-deployment-6445565f7b-ts815[workload] 2022-06-14 16:28:53.229 INFO 1
--- [nio-8081-exec-1] o.a.c.c.C.[Tomcat-1].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
pet-clinic-00004-deployment-6445565f7b-ts815[workload] 2022-06-14 16:28:53.229 INFO 1
--- [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
pet-clinic-00004-deployment-6445565f7b-ts815[workload] 2022-06-14 16:28:53.231 INFO 1
```



```
I6IDeWmDBcbiAgICAgIHn1cnZpY2VBY2NvdW50TmFtZTogZGVmYXVsdFxuIn0=
pet-clinic-config-writer-9fbk6-pod[step-main] + base64 --decode
pet-clinic-config-writer-9fbk6-pod[step-main] ++ cat files.json
+ pet-clinic-config-writer-kpmc6-pod > place-tools
pet-clinic-config-writer-9fbk6-pod[step-main] ++ jq -r 'to_entries | .[] | @sh "mkdir
-p $(dirname \(.key)) && echo \(.value) > \(.key) "'
+ pet-clinic-config-writer-kpmc6-pod > step-main
+ pet-clinic-config-writer-kpmc6-pod > step-init
+ pet-clinic-config-writer-kpmc6-pod > place-scripts
pet-clinic-config-writer-9fbk6-pod[step-main] + eval 'mkdir -p $(dirname \''delivery.
yaml\''') && echo \''apiVersion: serving.knative.dev/v1'
pet-clinic-config-writer-9fbk6-pod[step-main] kind: Service
pet-clinic-config-writer-9fbk6-pod[step-main] metadata:
pet-clinic-config-writer-9fbk6-pod[step-main]   name: pet-clinic
pet-clinic-config-writer-9fbk6-pod[step-main]   labels:
pet-clinic-config-writer-9fbk6-pod[step-main]     apps.tanzu.vmware.com/workload-type:
web
pet-clinic-config-writer-9fbk6-pod[step-main]     autoscaling.knative.dev/min-scale:
"1"
pet-clinic-config-writer-9fbk6-pod[step-main]     app.kubernetes.io/component: run
pet-clinic-config-writer-9fbk6-pod[step-main]     carto.run/workload-name: pet-clinic
```

## --timestamp, -t

Adds the timestamp to the beginning of each log message

```
tanzu apps workload tail pet-clinic -t

pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645910625-0
5:00
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645942876-0
5:00
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645951930-0
5:00
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645957151-0
5:00
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645961411-0
5:00
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645967316-0
5:00
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645971010-0
5:00
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645976591-0
5:00
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645986474-0
5:00
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645990521-0
5:00
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645994112-0
5:00
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645998053-0
5:00
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.646001577-0
5:00
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.646005296-0
5:00 :: Built with Spring Boot :: 2.6.8
```

## Tanzu Accelerator CLI overview

The Tanzu Accelerator Tanzu CLI includes commands for developers and operators to create and use accelerators.



## Server API connections for operators and developers

The Tanzu Accelerator CLI must connect to a server for all provided commands except for the `help` and `version` commands.

Operators typically use **create**, **update**, and **delete** commands for managing accelerators in a Kubernetes context. They also use the **fragment** commands to manage accelerator fragments. These commands require a Kubernetes context where the operator is already authenticated and is authorized to create and edit the accelerator resources. Operators can also use the **get** and **list** commands by using the same authentication. For any of these commands, the operator can specify the `--context` flag to access accelerators in a specific Kubernetes context.

Developers use the **list**, **get**, and **generate** commands for using accelerators available in an Application Accelerator server. Developers use the `--server-url` to point to the Application Accelerator server they want to use. The URL depends on the configuration settings for Application Accelerator:

- For installations configured with a **shared ingress**, use `https://accelerator.<domain>` where `domain` defaults to the `shared.ingress_domain` value provided in the values file of Tanzu Application Platform.
- For installations using a **LoadBalancer**, look up the External IP address by using:

```
kubectl get -n accelerator-system service/acc-server
```

Use `http://<External-IP>` as the URL.

- For any other configuration, you can use port forwarding by using:

```
kubectl port-forward service/acc-server -n accelerator-system 8877:80
```

Use `http://localhost:8877` as the URL.

The developer can set an `ACC_SERVER_URL` environment variable to avoid having to provide the same `--server-url` flag for every command. Run `export ACC_SERVER_URL=<URL>` for the terminal session in use. If the developer explicitly specifies the `--server-url` flag, it overrides the `ACC_SERVER_URL` environment variable if it is set.

## Installation

For information about installing the Tanzu CLI Accelerator plug-in, see [Install Accelerator CLI plug-in](#).

## Command reference

For information about available commands, see [Command Reference](#).

## Tanzu Accelerator CLI overview

The Tanzu Accelerator Tanzu CLI includes commands for developers and operators to create and use accelerators.

## Server API connections for operators and developers

The Tanzu Accelerator CLI must connect to a server for all provided commands except for the `help` and `version` commands.

Operators typically use **create**, **update**, and **delete** commands for managing accelerators in a Kubernetes context. They also use the **fragment** commands to manage accelerator fragments. These commands require a Kubernetes context where the operator is already authenticated and is authorized to create and edit the accelerator resources. Operators can also use the **get** and **list** commands by using the same authentication. For any of these commands, the operator can specify the `--context` flag to access accelerators in a specific Kubernetes context.

Developers use the **list**, **get**, and **generate** commands for using accelerators available in an Application Accelerator server. Developers use the `--server-url` to point to the Application Accelerator server they want to use. The URL depends on the configuration settings for Application Accelerator:

- For installations configured with a **shared ingress**, use `https://accelerator.<domain>` where `domain` defaults to the `shared.ingress_domain` value provided in the values file of Tanzu Application Platform.
- For installations using a **LoadBalancer**, look up the External IP address by using:

```
kubectl get -n accelerator-system service/acc-server
```

Use `http://<External-IP>` as the URL.

- For any other configuration, you can use port forwarding by using:

```
kubectl port-forward service/acc-server -n accelerator-system 8877:80
```

Use `http://localhost:8877` as the URL.

The developer can set an `ACC_SERVER_URL` environment variable to avoid having to provide the same `--server-url` flag for every command. Run `export ACC_SERVER_URL=<URL>` for the terminal session in use. If the developer explicitly specifies the `--server-url` flag, it overrides the `ACC_SERVER_URL` environment variable if it is set.

## Installation

For information about installing the Tanzu CLI Accelerator plug-in, see [Install Accelerator CLI plug-in](#).

## Command reference

For information about available commands, see [Command Reference](#).

## Install Tanzu Accelerator CLI

This topic tells you how to install the Tanzu Accelerator CLI.



### Note

Follow the steps in this topic if you do not want to use a profile to install Tanzu Accelerator CLI. For more information about profiles, see [About Tanzu Application Platform components and profiles](#).

## Prerequisites

Before you install the Tanzu Accelerator CLI:

- Follow the instructions to [Install or update the Tanzu CLI and plug-ins](#).

## Install

To install the Tanzu Accelerator CLI:

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

```
tanzu plugin install accelerator
```

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

```
tanzu accelerator version
```

A version will be displayed in the output.

If the following error is displayed during installation:

```
Error: could not find plug-in "accelerator" in any known repositories
✘ could not find plug-in "accelerator" in any known repositories
```

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

```
plugins:
...
  - name: accelerator
    description: Manage accelerators in a Kubernetes cluster
    versions: []
```

## Command reference

This topic provides you with a list of the Tanzu Accelerator CLI commands.

- [tanzu accelerator](#)
  - [tanzu accelerator apply](#)
  - [tanzu accelerator create](#)
  - [tanzu accelerator delete](#)
  - [tanzu accelerator fragment](#)
  - [tanzu accelerator fragment create](#)
  - [tanzu accelerator fragment delete](#)
  - [tanzu accelerator fragment get](#)
  - [tanzu accelerator fragment list](#)
  - [tanzu accelerator fragment update](#)
  - [tanzu accelerator generate](#)
  - [tanzu accelerator generate-from-local](#)
  - [tanzu accelerator get](#)
  - [tanzu accelerator list](#)
  - [tanzu accelerator push](#)
  - [tanzu accelerator update](#)

## Command reference

This topic provides you with a list of the Tanzu Accelerator CLI commands.

- [tanzu accelerator](#)
  - [tanzu accelerator apply](#)
  - [tanzu accelerator create](#)
  - [tanzu accelerator delete](#)
  - [tanzu accelerator fragment](#)
  - [tanzu accelerator fragment create](#)
  - [tanzu accelerator fragment delete](#)
  - [tanzu accelerator fragment get](#)
  - [tanzu accelerator fragment list](#)
  - [tanzu accelerator fragment update](#)
  - [tanzu accelerator generate](#)
  - [tanzu accelerator generate-from-local](#)
  - [tanzu accelerator get](#)
  - [tanzu accelerator list](#)
  - [tanzu accelerator push](#)
  - [tanzu accelerator update](#)

## tanzu accelerator

This topic tells you how to use the Tanzu Accelerator CLI `tanzu accelerator` command to manage accelerators in a Kubernetes cluster.

## Options

```

--context name      name of the kubeconfig context to use (default is current-co
ntext defined by kubeconfig)
-h, --help          help for accelerator
--kubeconfig file  kubeconfig file (default is $HOME/.kube/config)

```

## SEE ALSO

- [tanzu accelerator apply](#) - Apply accelerator resource
- [tanzu accelerator create](#) - Create a new accelerator
- [tanzu accelerator delete](#) - Delete an accelerator
- [tanzu accelerator fragment create](#) - Create a fragment
- [tanzu accelerator generate](#) - Generate project from accelerator
- [tanzu accelerator generate-from-local](#) - Generate a project from local or registered accelerators/fragments
- [tanzu accelerator get](#) - Get accelerator information
- [tanzu accelerator list](#) - List accelerators
- [tanzu accelerator push](#) - Push local path to source image

- [tanzu accelerator update](#) - Update an accelerator

## tanzu accelerator

This topic tells you how to use the Tanzu Accelerator CLI `tanzu accelerator` command to manage accelerators in a Kubernetes cluster.

### Options

```
--context name      name of the kubeconfig context to use (default is current-co
ntext defined by kubeconfig)
-h, --help          help for accelerator
--kubeconfig file  kubeconfig file (default is $HOME/.kube/config)
```

### SEE ALSO

- [tanzu accelerator apply](#) - Apply accelerator resource
- [tanzu accelerator create](#) - Create a new accelerator
- [tanzu accelerator delete](#) - Delete an accelerator
- [tanzu accelerator fragment create](#) - Create a fragment
- [tanzu accelerator generate](#) - Generate project from accelerator
- [tanzu accelerator generate-from-local](#) - Generate a project from local or registered accelerators/fragments
- [tanzu accelerator get](#) - Get accelerator information
- [tanzu accelerator list](#) - List accelerators
- [tanzu accelerator push](#) - Push local path to source image
- [tanzu accelerator update](#) - Update an accelerator

## tanzu accelerator apply

### tanzu accelerator apply

This topic tells you how to use the Tanzu Accelerator CLI `tanzu accelerator apply` command to create or update accelerators.

### Synopsis

Create or update accelerator resource using specified manifest file.

```
tanzu accelerator apply [flags]
```

### Examples

```
tanzu accelerator apply --filename <path-to-resource-manifest>
```

### Options

```
-f, --filename string  path of manifest file for the resource
-h, --help             help for apply
```

```
-n, --namespace string namespace for the resource (default "accelerator-system")
```

## Options inherited from parent commands

```
--context name name of the kubeconfig context to use (default is current-context defined by kubeconfig)
--kubeconfig file kubeconfig file (default is $HOME/.kube/config)
```

## SEE ALSO

- [tanzu accelerator](#) - Manage accelerators in a Kubernetes cluster

## tanzu accelerator create

This topic tells you how to use the Tanzu Accelerator CLI `tanzu accelerator create` command to create a new accelerator.

## Synopsis

Create a new accelerator resource with specified configuration.

Accelerator configuration options include: - Git repository URL and branch/tag where accelerator code and metadata is defined - Metadata like description, display-name, tags and icon-url

The Git repository option is required. Metadata options are optional and will override any values for the same options specified in the accelerator metadata retrieved from the Git repository.

```
tanzu accelerator create [flags]
```

## Examples

```
tanzu accelerator create <accelerator-name> --git-repository <URL> --git-branch <branch>
```

## Options

```
--description string description of this accelerator
--display-name string display name for the accelerator
--git-branch string Git repository branch to be used
--git-repo string Git repository URL for the accelerator
--git-sub-path string Git repository subPath to be used
--git-tag string Git repository tag to be used
-h, --help help for create
--icon-url string URL for icon to use with the accelerator
--interval string interval for checking for updates to Git or image repository
--local-path string path to the directory containing the source for the accelerator
-n, --namespace string namespace for accelerator system (default "accelerator-system")
--secret-ref string name of secret containing credentials for private Git or image repository
--source-image string name of the source image for the accelerator
--tags strings tags that can be used to search for accelerators
```

## Options inherited from parent commands

```
--context name      name of the kubeconfig context to use (default is current-co
ncontext defined by kubeconfig)
--kubeconfig file  kubeconfig file (default is $HOME/.kube/config)
```

## SEE ALSO

- [tanzu accelerator](#) - Manage accelerators in a Kubernetes cluster

## tanzu accelerator delete

This topic tells you how to use the Tanzu Accelerator CLI `tanzu accelerator delete` command to delete an accelerator.

## Synopsis

Delete the accelerator resource with the specified name.

```
tanzu accelerator delete [flags]
```

## Examples

```
tanzu accelerator delete <accelerator-name>
```

## Options

```
-h, --help          help for delete
-n, --namespace string namespace for accelerator system (default "accelerator-syst
em")
```

## Options inherited from parent commands

```
--context name      name of the kubeconfig context to use (default is current-co
ncontext defined by kubeconfig)
--kubeconfig file  kubeconfig file (default is $HOME/.kube/config)
```

## SEE ALSO

- [tanzu accelerator](#) - Manage accelerators in a Kubernetes cluster

## tanzu accelerator fragment

This topic tells you how to use the Tanzu Accelerator CLI `tanzu accelerator fragment` command to manage fragments.

## Synopsis

Commands to manage accelerator fragments

## Examples

```
tanzu accelerator fragment --help
```

## Options

```
-h, --help    help for fragment
```

## Options inherited from parent commands

```
--context name      name of the kubeconfig context to use (default is current-co
ntext defined by kubeconfig)
--kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
```

## SEE ALSO

- [tanzu accelerator](#) - Manage accelerators in a Kubernetes cluster
- [tanzu accelerator fragment create](#) - Create a new accelerator fragment
- [tanzu accelerator fragment delete](#) - Delete an accelerator fragment
- [tanzu accelerator fragment get](#) - Get accelerator fragment information
- [tanzu accelerator fragment list](#) - List accelerator fragments
- [tanzu accelerator fragment update](#) - Update an accelerator fragment

## tanzu accelerator fragment create

This topic tells you how to use the Tanzu Accelerator CLI `tanzu accelerator fragment create` command to create a new accelerator fragment.

## Synopsis

Create a new accelerator fragment resource with specified configuration.

Accelerator configuration options include: - Git repository URL and branch/tag where accelerator code and metadata is defined - Metadata like description, display-name, tags and icon-url

The Git repository option is required. Metadata options are optional and will override any values for the same options specified in the accelerator metadata retrieved from the Git repository.

```
tanzu accelerator fragment create [flags]
```

## Example

```
tanzu accelerator fragment create <fragment-name> --git-repository <URL> --git-branch
<branch> --git-sub-path <sub-path>
```

## Options

```
--display-name string  display name for the accelerator
--git-branch string     Git repository branch to be used
--git-repo string       Git repository URL for the accelerator
--git-sub-path string   Git repository subPath to be used
--git-tag string        Git repository tag to be used
-h, --help              help for create
--interval string       interval for checking for updates to Git or image reposi
tory
-n, --namespace string  namespace for accelerator system (default "accelerator-s
```



```
system")
  --secret-ref string    name of secret containing credentials for private Git or
image repository
```

## Options inherited from parent commands

```
--context name          name of the kubeconfig context to use (default is current-co
nfig defined by kubeconfig)
--kubeconfig file      kubeconfig file (default is $HOME/.kube/config)
```

## SEE ALSO

- [tanzu accelerator fragment](#) - Fragment commands

## tanzu accelerator fragment delete

## tanzu accelerator fragment delete

This topic tells you how to use the Tanzu Accelerator CLI `tanzu accelerator fragment delete` command to delete an accelerator fragment.

## Synopsis

Delete the accelerator fragment resource with the specified name.

```
tanzu accelerator fragment delete [flags]
```

## Examples

```
tanzu accelerator fragment delete <fragment-name>
```

## Options

```
-h, --help                help for delete
-n, --namespace string    namespace for accelerator system (default "accelerator-syst
em")
```

## Options inherited from parent commands

```
--context name          name of the kubeconfig context to use (default is current-co
nfig defined by kubeconfig)
--kubeconfig file      kubeconfig file (default is $HOME/.kube/config)
```

## SEE ALSO

- [tanzu accelerator fragment](#) - Fragment commands

## tanzu accelerator fragment get

This topic tells you how to use the Tanzu Accelerator CLI `tanzu accelerator fragment get` command to get accelerator fragment information.

## Synopsis

Get accelerator fragment information.

```
tanzu accelerator fragment get [flags]
```

## Examples

```
tanzu accelerator get <fragment-name>
```

## Options

```
-h, --help           help for get
-n, --namespace string namespace for accelerator system (default "accelerator-system")
```

## 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)
```

## SEE ALSO

- [tanzu accelerator fragment](#) - Fragment commands

## tanzu accelerator fragment list

This topic tells you how to use the Tanzu Accelerator CLI `tanzu accelerator fragment list` to list accelerator fragments.

## Synopsis

List all accelerator fragments.

```
tanzu accelerator fragment list [flags]
```

## Examples

```
tanzu accelerator fragment list
```

## Options

```
-h, --help           help for list
-n, --namespace string namespace for accelerator system (default "accelerator-system")
```

## 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)
```

## SEE ALSO

- [tanzu accelerator fragment](#) - Fragment commands

## tanzu accelerator fragment update

This topic tells you how to use the Tanzu Accelerator CLI `tanzu accelerator fragment update` command to update an accelerator fragment.

## Synopsis

Update an accelerator fragment resource with the specified name using the specified configuration.

Accelerator configuration options include: - Git repository URL and branch/tag where accelerator code and metadata is defined - Metadata like display-name

The update command also provides a `--reconcile` flag that will force the accelerator fragment to be refreshed with any changes made to the associated Git repository.

```
tanzu accelerator fragment update [flags]
```

## Examples

```
tanzu accelerator update <accelerator-name> --description "Lorem Ipsum"
```

## Options

```
--display-name string  display name for the accelerator fragment
--git-branch string    Git repository branch to be used
--git-repo string      Git repository URL for the accelerator fragment
--git-sub-path string  Git repository subPath to be used
--git-tag string       Git repository tag to be used
-h, --help             help for update
--interval string     interval for checking for updates to Git repository
-n, --namespace string namespace for accelerator fragments (default "accelerato
r-system")
--reconcile           trigger a reconciliation including the associated GitRep
ository resource
--secret-ref string   name of secret containing credentials for private Git re
pository
```

## Options inherited from parent commands

```
--context name        name of the kubeconfig context to use (default is current-co
nfig)
--kubeconfig file     kubeconfig file (default is $HOME/.kube/config)
```

## SEE ALSO

- [tanzu accelerator fragment](#) - Fragment commands

## tanzu accelerator generate

## tanzu accelerator generate

This topic tells you how to use the Tanzu Accelerator CLI `tanzu accelerator generate` command to generate a project from an accelerator.

## Synopsis

Generate a project from an accelerator using provided options and download project artifacts as a ZIP file.

Generation options are provided as a JSON string and should match the metadata options that are specified for the accelerator used for the generation. The options can include “projectName” which defaults to the name of the accelerator. This “projectName” will be used as the name of the generated ZIP file.

You can see the available options by using the “tanzu accelerator get ” command.

Here is an example of an options JSON string that specifies the “projectName” and an “includeKubernetes” boolean flag:

```
--options '{"projectName":"test", "includeKubernetes": true}'
```

You can also provide a file that specifies the JSON string using the `-options-file` flag.

The generate command needs access to the Application Accelerator server. You can specify the `-server-url` flag or set an `ACC_SERVER_URL` environment variable. If you specify the `-server-url` flag it overrides the `ACC_SERVER_URL` environment variable if it is set.

```
tanzu accelerator generate [flags]
```

## Examples

```
tanzu accelerator generate <accelerator-name> --options '{"projectName":"test"}'
```

## Options

```
-h, --help                help for generate
--options string          options JSON string
--options-file string     path to file containing options JSON string
--output-dir string      directory that the zip file will be written to
--server-url string       the URL for the Application Accelerator server
```

## Options inherited from parent commands

```
--context name           name of the kubeconfig context to use (default is current-co
nfig)
--kubeconfig file        kubeconfig file (default is $HOME/.kube/config)
```

## SEE ALSO

- [tanzu accelerator](#) - Manage accelerators in a Kubernetes cluster

## Tanzu accelerator generate-from-local

This topic tells you how to use the Tanzu Accelerator CLI `tanzu accelerator generate-from-local` command to generate a project from a combination of registered and local artifacts.

## Synopsis

Use local files and registered accelerators or fragments to create a project, and download the project artifacts as ZIP files.

Options values are provided as a JSON object and match the declared options that are specified for the accelerator used for the generation. The options can include `projectName` which by default is set to the name of the accelerator. This `projectName` is used as the name of the generated ZIP file.

Here is an example of an options JSON string that specifies the `projectName` and an `includeKubernetes` Boolean flag:

```
--options '{"projectName":"test", "includeKubernetes": true}'
```

You can also provide a file that specifies the JSON string using the `--options-file` flag.

The `generate-from-local` command needs access to the Application Accelerator server. You can specify the `--server-url` flag or set an `ACC_SERVER_URL` environment variable. If you specify the `--server-url` flag it overrides the `ACC_SERVER_URL` environment variable if it is set.

## Examples

Generate a project using a combination of local and registered assets

```
tanzu accelerator generate-from-local \
--accelerator-path java-rest=workspace/java-rest \ # Use a local accelerator
--fragment-paths java-version=workspace/version \ # Use a local fragment
--fragment-names tap-workload \ # Use a registered fragment
--options '{"projectName":"test"}' \
--output-dir "./generated-java-rest-app"
```

## Options

```
--accelerator-name string          name of the registered accelerator to use
--accelerator-path "key=value" pair key value pair of the name and path to the d
irectory containing the accelerator
-f, --force                        force clean and rewrite of output-dir
--fragment-names strings           names of the registered fragments to use
--fragment-paths stringToString    key value pairs of the name and path to the
directory containing each fragment (default [])
-h, --help                          help for generate-from-local
--options string                   options JSON string (default "{}")
--options-file string              path to file containing options JSON string
-o, --output-dir string            the directory that the project will be creat
ed in (defaults to the project name)
--server-url string                the URL for the Application Accelerator serv
er
```

## Options inherited from parent commands

```
--context name                    name of the kubeconfig context to use (default is current-cont
ext defined by kubeconfig)
--kubeconfig file                 kubeconfig file (default is $HOME/.kube/config)
```

## tanzu accelerator get

This topic tells you how to use the Tanzu Accelerator CLI `tanzu accelerator get` command to get accelerator information.

## Synopsis

Get accelerator information.

You can choose to get the accelerator from the Application Accelerator server using `--server-url` flag or from a Kubernetes context using `--from-context` flag. The default is to get accelerators from the Kubernetes context. To override this, you can set the `ACC_SERVER_URL` environment variable with the URL for the Application Accelerator server you want to access.

```
tanzu accelerator get [flags]
```

## Examples

```
tanzu accelerator get <accelerator-name> --from-context
```

## Options

```

--from-context      retrieve resources from current context defined in kubecon
fig
-h, --help          help for get
-n, --namespace string namespace for accelerator system (default "accelerator-sys
tem")
--server-url string the URL for the Application Accelerator server

```

## Options inherited from parent commands

```

--context name      name of the kubeconfig context to use (default is current-co
nfig)
--kubeconfig file   kubeconfig file (default is $HOME/.kube/config)

```

## SEE ALSO

- [tanzu accelerator](#) - Manage accelerators in a Kubernetes cluster

## tanzu accelerator list

This topic tells you how to use the Tanzu Accelerator CLI `tanzu accelerator list` command to list accelerators.

## Synopsis

List all accelerators.

You can choose to list the accelerators from the Application Accelerator server using `--server-url` flag or from a Kubernetes context using `--from-context` flag. The default is to list accelerators from the Kubernetes context. To override this, you can set the `ACC_SERVER_URL` environment variable with the URL for the Application Accelerator server you want to access.

```
tanzu accelerator list [flags]
```

## Examples

```
tanzu accelerator list
```

## Options

```

--from-context      retrieve resources from current context defined in kubecon
fig
-h, --help          help for list
-n, --namespace string namespace for accelerator system (default "accelerator-sys
tem")
--server-url string the URL for the Application Accelerator server
-t, --tags strings  accelerator tags to match against

```

## Options inherited from parent commands

```

--context name      name of the kubeconfig context to use (default is current-co
nfig)
--kubeconfig file  kubeconfig file (default is $HOME/.kube/config)

```

## SEE ALSO

- [tanzu accelerator](#) - Manage accelerators in a Kubernetes cluster

## tanzu accelerator push

## tanzu accelerator push

This topic tells you how to use the Tanzu Accelerator CLI `tanzu accelerator push` command to push source code from local path to source image.

## Synopsis

Push source code from local path to source image used by an accelerator

```
tanzu accelerator push [flags]
```

## Examples

```
tanzu accelerator push --local-path <local path> --source-image <image>
```

## Options

```

-h, --help          help for push
--local-path string path to the directory containing the source for the acce
lerator
--source-image string name of the source image for the accelerator

```

## Options inherited from parent commands

```

--context name      name of the kubeconfig context to use (default is current-co
nfig)
--kubeconfig file  kubeconfig file (default is $HOME/.kube/config)

```

## SEE ALSO

- [tanzu accelerator](#) - Manage accelerators in a Kubernetes cluster

## tanzu accelerator update

This topic tells you how to use the Tanzu Accelerator CLI `tanzu accelerator update` command to update an accelerator.

### Synopsis

Update an accelerator resource with the specified name using the specified configuration.

Accelerator configuration options include: - Git repository URL and branch/tag where accelerator code and metadata is defined - Metadata like description, display-name, tags and icon-url

The update command also provides a `-reconcile` flag that will force the accelerator to be refreshed with any changes made to the associated Git repository.

```
tanzu accelerator update [flags]
```

### Examples

```
tanzu accelerator update <accelerator-name> --description "Lorem Ipsum"
```

### Options

```
--description string      description of this accelerator
--display-name string     display name for the accelerator
--git-branch string       Git repository branch to be used
--git-repo string         Git repository URL for the accelerator
--git-sub-path string     Git repository subPath to be used
--git-tag string          Git repository tag to be used
-h, --help                help for update
--icon-url string         URL for icon to use with the accelerator
--interval string        interval for checking for updates to Git or image repository
-n, --namespace string    namespace for accelerator system (default "accelerator-system")
--reconcile                trigger a reconciliation including the associated GitRepository resource
--secret-ref string       name of secret containing credentials for private Git or image repository
--source-image string     name of the source image for the accelerator
--tags strings            tags that can be used to search for accelerators
```

### Options inherited from parent commands

```
--context name            name of the kubeconfig context to use (default is current-context defined by kubeconfig)
--kubeconfig file        kubeconfig file (default is $HOME/.kube/config)
```

### SEE ALSO

- [tanzu accelerator](#) - Manage accelerators in a Kubernetes cluster

## Overview of Tanzu Insight plug-in



The Tanzu Insight CLI plug-in helps you query vulnerability, image, and package data.

Follow these steps to install, configure, and use your Tanzu Insight CLI plug-in:

**Note:** Prior to using the CLI plug-in, you must install the Supply Chain Security Tools - Store, either as its own package, or as part of Tanzu Application Platform View profile.

1. If the `insight` plug-in is not already installed, see [Install the Tanzu Insight plug-in](#)
2. [Configure insight](#)

Once `tanzu insight` CLI plug-in is set up:

1. [Add data](#)
2. [Query data](#)

## Overview of Tanzu Insight plug-in

The Tanzu Insight CLI plug-in helps you query vulnerability, image, and package data.

Follow these steps to install, configure, and use your Tanzu Insight CLI plug-in:

**Note:** Prior to using the CLI plug-in, you must install the Supply Chain Security Tools - Store, either as its own package, or as part of Tanzu Application Platform View profile.

1. If the `insight` plug-in is not already installed, see [Install the Tanzu Insight plug-in](#)
2. [Configure insight](#)

Once `tanzu insight` CLI plug-in is set up:

1. [Add data](#)
2. [Query data](#)

## Install your Tanzu Insight CLI plug-in

This topic tells you how to install your Tanzu Insight CLI plug-in.

By following the [instructions](#) to install the Tanzu CLI and all the plug-ins, the Tanzu Insight plug-in is also installed.



### 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
```

2. Follow the steps in [Configure the Tanzu Insight CLI plug-in](#).

## Configure your Tanzu Insight CLI plug-in

This topic tells you how to configure your Tanzu Insight CLI plug-in.

## Set the target and certificate authority (CA) certificate

**Note** These instructions are for the recommended configuration where ingress is enabled. For instructions on non-ingress setups, see [Configure target endpoint and certificate](#) for more details.

Set the endpoint host to `metadata-store.INGRESS-DOMAIN`, such as `metadata-store.example.domain.com`. Where `INGRESS-DOMAIN` is the value of the `ingress_domain` property in your deployment yaml.

**Note** In a multi-cluster setup, a DNS record is **required** for the domain. The below instructions for single cluster setup do not apply, skip to Set Target section.

### Single Cluster setup

In a single-cluster setup, a DNS record is still recommended. However, if no accessible DNS record exists for the domain, edit the `/etc/hosts` file to add a local record:

```
ENVOY_IP=$(kubectl get svc envoy -n tanzu-system-ingress -o jsonpath="{.status.loadBalancer.ingress[0].ip}")

# Replace with your domain
METADATA_STORE_DOMAIN="metadata-store.example.domain.com"

# Delete any previously added entry
sudo sed -i ' ' "$METADATA_STORE_DOMAIN/d" /etc/hosts

echo "$ENVOY_IP $METADATA_STORE_DOMAIN" | sudo tee -a /etc/hosts > /dev/null
```

## Set Target

To get the certificate, run:

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

Set the target by running:

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



#### Important

The `tanzu insight config set-target` does not initiate a test connection. Use `tanzu insight health` to test connecting using the configured endpoint and CA certificate. Neither commands test whether the access token is correct. For that you must use the plug-in to [add data](#) and [query data](#).

## Set the access token

When using the `insight` plug-in, you must set the `METADATA_STORE_ACCESS_TOKEN` environment variable, or use the `--access-token` flag. VMware discourages using the `--access-token` flag as the token appears in your shell history.

The following command retrieves the access token from the default `metadata-store-read-write-client` service account and stores it in `METADATA_STORE_ACCESS_TOKEN`:

```
export METADATA_STORE_ACCESS_TOKEN=$(kubectl get secrets metadata-store-read-write-client -n metadata-store -o jsonpath="{.data.token}" | base64 -d)
```

## Verify the connection

Verify that your configuration is correct and you can make a connection using `tanzu insight health`.



### Important

The `tanzu insight health` command tests the configured endpoint and CA certificate. However, it does not test whether the access token is correct. For that, you must use the plug-in to [add](#) and [query](#) data.

For example:

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

## Query vulnerabilities, images, and packages

This topic tells you how to query the database to understand vulnerability, image, and dependency relationships. The Tanzu Insight CLI plug-in queries the database for vulnerability scan reports or Software Bill of Materials (commonly known as SBOM) files.

## Supported use cases

The following are use cases supported by the CLI:

- What packages and CVEs exist in a particular image? ([image](#))
- What 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 for more information see [Tanzu Insight Details](#).

## Example #1: What packages and 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@3.1.2-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 and CVEs does my source code contain?

### Determining source code org, repo, and commit SHA

To query a source scan for vulnerabilities, you need a Git org and Git repository, or the commit SHA. Find these by examining the source scan resource.

Run:

```
kubectl describe sourcescan WORKLOAD-NAME -n WORKLOAD-NAMESPACE
```

For example:

```
kubectl describe sourcescan tanzu-java-web-app -n my-apps
```

In the resource look for the `Spec.Blob` field. Within, there's `Revision` and `URL`.

For example:

```
Spec:
  Blob:
    Revision:    master/c7e4c27ba43250a4b7c46f030355c108aa73cc39
    URL:         http://source-controller.flux-system.svc.cluster.local./gitrepository/my-apps/tanzu-java-web-app-gitops/c7e4c27ba43250a4b7c46f030355c108aa73cc39.tar.gz

```

In the earlier example, the URL is parsed and split into the org and repo. Revision is parsed as the commit SHA.

- Org is parsed as `gitrepository`
- Repo is parsed as `my-apps/tanzu-java-web-app-gitops/c7e4c27ba43250a4b7c46f030355c108aa73cc39.tar.gz`
- Commit SHA is parsed as `master/c7e4c27ba43250a4b7c46f030355c108aa73cc39`

Use this information to perform your search.

## Source code query with repo and org

Run:

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

Where:

- **REPO** specifies the repository. For example, java-web-app, my-apps/java-web-app/c7ls8bakd87sakjda8d7.tar.gz
- **ORG** is the source code's organization. For example, gitrepository, gitrepository-kj32ka18

For example:

```
$ tanzu insight source get --repo my-apps/java-web-app/c7ls8bakd87sakjda8d7.tar.gz --org gitrepository
ID: 1
Repository: my-apps/java-web-app/c7ls8bakd87sakjda8d7.tar.gz
Commit: c7e4c27ba43250a4b7c46f030355c108aa73cc39
Organization: gitrepository
Packages:
    1. go.uber.org/atomic@v1.7.0
    CVEs:
        1. CVE-2022-42322 (Low)
    2. golang.org/x/crypto@v0.0.0-20220518034528-6f7dac969898
    3. github.com/valyala/bytebufferpool@v1.0.0
```

## Source code query with commit SHA

Run:

```
tanzu insight source get --commit COMMIT
```

Where:

- **COMMIT** specifies the commit. For example, d7e4c27ba43250a4b7c46f030355c108aa73cc39, main/d7e4c27ba43250a4b7c46f030355c108aa73cc39

For example:

```
$ tanzu insight source get --commit b66668e
ID: 2
Repository: kpack
Commit: b66668e
Organization: pivotal
Packages:
    1. cloud.google.com/go/kms@v1.0.0
    2. github.com/BurntSushi/toml@v3.1.1
    CVEs:
        1. CVE-2021-30999 (Low)
    3. github.com/Microsoft/go-winio@v0.5.2
```

## Example #3: What dependencies are affected by a specific CVE?

Run:

```
tanzu insight vulnerabilities get --cveid CVE-IDENTIFIER
```

Where:

- `CVE-IDENTIFIER` is the CVE identifier, for example, CVE-2021-30139.

For example:

```
$ tanzu insight vulnerabilities get --cveid CVE-2010-4051
1. CVE-2010-4051 (Low)
Packages:
  1. libc-bin@2.28-10
  2. libc-l10n@2.28-10
  3. libc6@2.28-10
  4. locales@2.28-10
```

## Add data

For more information about manually adding data, see [Add Data](#).

## Add data to your Supply Chain Security Tools - Store

This topic tells you how to add vulnerability scan reports or Software Bill of Materials (commonly known as SBOM) files to your Supply Chain Security Tools (commonly known as SCST) - Store.

### Supported formats and file types

Currently, only CycloneDX XML and JSON files are accepted.

Source commits and image files have been tested. Additional file types might work, but are not fully supported (for example, JAR files).



#### Note

If you are not using a source commit or image file, you must ensure the `component.version` field in the CycloneDX file is non-null.

## Generate a CycloneDX file

A CycloneDX file is needed to post data. Supply Chain Security Tools - Scan outputs CycloneDX files automatically. For more information, see [Supply Chain Security Tools - Scan](#).

To generate a file to post manually, use Grype or another tool in the [CycloneDX Tool Center](#).

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

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

```
grype REPO:TAG -o cyclonedx > IMAGE-CVE-REPORT
```

Where:

- `REPO` is the name of your repository
- `TAG` is the name of a tag
- `IMAGE-CVE-REPORT` is the resulting file name of the Grype image scan report

For example:

```
$ gype 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.

## Tanzu insight CLI plug-in command reference

This topic tells you about the Tanzu Insight CLI plug-in.

### Synopsis

This Tanzu Insight CLI is used to post data and query the Supply Chain Security Tools - Store through its secure REST API. Source commit and image vulnerability reports can be uploaded using CycloneDX format (XML and JSON) and SPDX format (JSON). Source commit, image, package, and vulnerabilities can be queried and outputted in CycloneDX XML, JSON, and human-readable text formats.

### Options

```
-h, --help help for tanzu insight
```

### See also

- [Tanzu insight config](#) - Config commands
- [Tanzu insight health](#) - Checks if endpoint is reachable
- [Tanzu insight image](#) - Image commands
- [Tanzu insight package](#) - Package commands
- [Tanzu insight source](#) - Source commands
- [Tanzu insight version](#) - Display Tanzu Insight version
- [Tanzu insight vulnerabilities](#) - Vulnerabilities commands

## tanzu insight config set-target

### tanzu insight config set-target

This topic tells you how to use the Tanzu Insight CLI `tanzu insight config set-target` command to set your 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

This topic tells you how to use the Tanzu Insight CLI `tanzu insight config` command to get help for the configuration commands.

## 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

This topic tells you how to use the Tanzu Insight CLI `tanzu insight health` command to check if an 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

This topic tells you how to use the Tanzu Insight CLI `tanzu insight image` command to get help for the image commands.

## 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

This topic tells you how to use the Tanzu Insight CLI `tanzu insight image add` command to add an image report.

```
tanzu insight image add [--cyclonedxtype <json|xml>] [--spdxtype json] --path <filepath>
h>
```

If report type is not specified, it will be defaulted to `--cyclonedxtype=xml`

## Examples

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

## Options

```
--cyclonedxtype string  cyclonedx file type(xml/json, default: xml)
-h, --help             help for add
--path string          path to file
--spdxtype string      spdx file type(json)
```

## See also

- [Tanzu insight image](#) - Image commands

## tanzu insight image get

This topic tells you how to use the Tanzu Insight CLI `tanzu insight image get` command to get an 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

This topic tells you how to use the Tanzu Insight CLI `tanzu insight image packages` command to get the 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

This topic tells you how to use the Tanzu Insight CLI `tanzu insight image vulnerabilities` command to get the 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

This topic tells you how to use the Tanzu Insight CLI `tanzu insight package` command to get help for the package commands.

## 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

This topic tells you how to use the Tanzu Insight CLI `tanzu insight package get` command to get the 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

This topic tells you how to use the Tanzu Insight CLI `tanzu insight package images` command to get the 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

This topic tells you how to use the Tanzu Insight CLI `tanzu insight package sources` command to get the 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

This topic tells you how to use the Tanzu Insight CLI `tanzu insight package vulnerabilities` command to get the 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

This topic tells you how to use the Tanzu Insight CLI `tanzu insight source` command to get help for the source commands.

## 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

This topic tells you how to use the Tanzu Insight CLI `tanzu insight source add` command to add a source report.

```
tanzu insight source add [--cyclonedxtype <json|xml>] [--spdxtype json] --path <filepath>
```

If report type is not specified, it defaults to `--cyclonedxtype=xml`

## Examples

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

## Options

```
--cyclonedxtype string  cyclonedx file type (xml/json, default: xml)
-h, --help             help for add
--path string          path to file
--spdxtype string      spdx file type (json)
```

## See also

- [Tanzu insight source](#) - Source commands

## tanzu insight source get

This topic tells you how to use the Tanzu Insight CLI `tanzu insight source get` command to 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

This topic tells you how to use the Tanzu Insight CLI `tanzu insight source packages` command to get the 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 0blb659907 --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

This topic tells you how to use the Tanzu Insight CLI `tanzu insight source vulnerabilities` command to get the source vulnerabilities.

## Synopsis

Get source vulnerabilities. You can specify either commit or repository.



```
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

This topic tells you how to use the Tanzu Insight CLI `tanzu insight version` command to display the 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

This topic tells you how to use the Tanzu Insight CLI `tanzu insight vulnerabilities` command to get help for the vulnerabilities commands.

## 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

This topic tells you how to use the Tanzu Insight CLI `tanzu insight vulnerabilities get` command to get a 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

This topic tells you how to use the Tanzu Insight CLI `tanzu insight vulnerabilities images` command to get the 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

This topic tells you how to use the Tanzu Insight CLI `tanzu insight vulnerabilities packages` command to get the 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

This topic tells you how to use the Tanzu Insight CLI `tanzu insight vulnerabilities sources` command to get the 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 of Default roles for Tanzu Application Platform

Tanzu Application Platform (commonly known as TAP) v1.4 includes:

- Six 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

Four roles are for users:

- app-editor
- app-viewer
- app-operator
- service-operator

Two roles are for service accounts associated with the Tanzu Supply Chain:

- workload
- deliverable

The default roles provide an opinionated starting point for the most common permissions that users need when using Tanzu Application Platform. However, as described in the [Kubernetes documentation](#) about RBAC, you can create customized roles and permissions that better meet your needs. Aggregated cluster roles are used to build VMware Tanzu Application Platform default roles.

Cluster admins must be careful when creating Roles or ClusterRoles. When changing roles or adding new roles that carry one of the labels used by the default roles, the roles are automatically updated to the aggregation state. It can lead to unintentional changes in functions and permissions to all users.

The default roles are installed with every Tanzu Application Platform profile except for [view](#). For an overview of the different roles and their permissions, see [Role Descriptions](#).

## Working with roles using the RBAC CLI plug-in

For more information about working with roles, see [Bind a user or group to a default role](#).

## Disclaimer

[Tanzu Application Platform GUI](#) does not make use of the described roles. Instead, it provides the user with view access for each cluster.

## Overview of Default roles for Tanzu Application Platform

Tanzu Application Platform (commonly known as TAP) v1.4 includes:

- Six 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

Four roles are for users:

- app-editor
- app-viewer
- app-operator
- service-operator

Two roles are for service accounts associated with the Tanzu Supply Chain:

- workload
- deliverable

The default roles provide an opinionated starting point for the most common permissions that users need when using Tanzu Application Platform. However, as described in the [Kubernetes documentation](#) about RBAC, you can create customized roles and permissions that better meet your needs. Aggregated cluster roles are used to build VMware Tanzu Application Platform default roles.

Cluster admins must be careful when creating Roles or ClusterRoles. When changing roles or adding new roles that carry one of the labels used by the default roles, the roles are automatically updated to the aggregation state. It can lead to unintentional changes in functions and permissions to all users.

The default roles are installed with every Tanzu Application Platform profile except for `view`. For an overview of the different roles and their permissions, see [Role Descriptions](#).

## Working with roles using the RBAC CLI plug-in

For more information about working with roles, see [Bind a user or group to a default role](#).

## Disclaimer

[Tanzu Application Platform GUI](#) does not make use of the described roles. Instead, it provides the user with view access for each cluster.

## Set up authentication for your Tanzu Application Platform deployment

There are multiple ways to set up authentication for your Tanzu Application Platform (commonly known as TAP) deployment. You can manage authentication at the infrastructure level with your Kubernetes provider, such as Tanzu Kubernetes Grid, EKS, AKS, or GKE.

VMware recommends Pinniped for integrating your identity management into Tanzu Application Platform on multicloud. It provides many supported integrations for widely used identity providers. To use Pinniped, see [Installing Pinniped on Tanzu Application Platform](#) and [Log in by using Pinniped](#).

See [Integrating Azure Active Directory](#) for Azure Active Directory Integration.

## Tanzu Kubernetes Grid

For Tanzu Kubernetes Grid clusters, Pinniped is the default identity solution and is installed as a core package. For more information, see [Core Packages](#) and [Enable Identity Management in an Existing Deployment](#) in the Tanzu Kubernetes Grid documentation.

## Set up authentication for your Tanzu Application Platform deployment

There are multiple ways to set up authentication for your Tanzu Application Platform (commonly known as TAP) deployment. You can manage authentication at the infrastructure level with your Kubernetes provider, such as Tanzu Kubernetes Grid, EKS, AKS, or GKE.

VMware recommends Pinniped for integrating your identity management into Tanzu Application Platform on multicloud. It provides many supported integrations for widely used identity providers. To use Pinniped, see [Installing Pinniped on Tanzu Application Platform](#) and [Log in by using Pinniped](#).

See [Integrating Azure Active Directory](#) for Azure Active Directory Integration.

## Tanzu Kubernetes Grid

For Tanzu Kubernetes Grid clusters, Pinniped is the default identity solution and is installed as a core package. For more information, see [Core Packages](#) and [Enable Identity Management in an Existing Deployment](#) in the Tanzu Kubernetes Grid documentation.

## Install Pinniped on Tanzu Application Platform

[Pinniped](#) is used to support authentication on Tanzu Application Platform (commonly known as TAP). This topic tells you how to install Pinniped on a single cluster of Tanzu Application Platform.



### Note

This topic only provides an example of one possible installation method for Pinniped on Tanzu Application Platform by using the default Contour ingress controller included in the platform. See [Pinniped documentation](#) for more information about the specific installation method that suits your environment.

Use this topic to learn how to deploy two Pinniped components into the cluster:

- **Pinniped Supervisor:** An OIDC server which allows users to authenticate with an external identity provider (IDP). It hosts an API for the concierge component to fulfill authentication requests.
- **Pinniped Concierge:** A credential exchange API that takes a credential from an identity source, for example, Pinniped Supervisor, proprietary IDP, as input. The Pinniped Concierge authenticates the user by using the credential, and returns another credential that is parsable by the host Kubernetes cluster or by an impersonation proxy that acts on behalf of the user.

## Prerequisites

Meet these prerequisites:

- Install the package `certmanager`. This is included in Tanzu Application Platform.
- Install the package `contour`. This is included in Tanzu Application Platform.
- Create a `workspace` directory to function as your workspace.

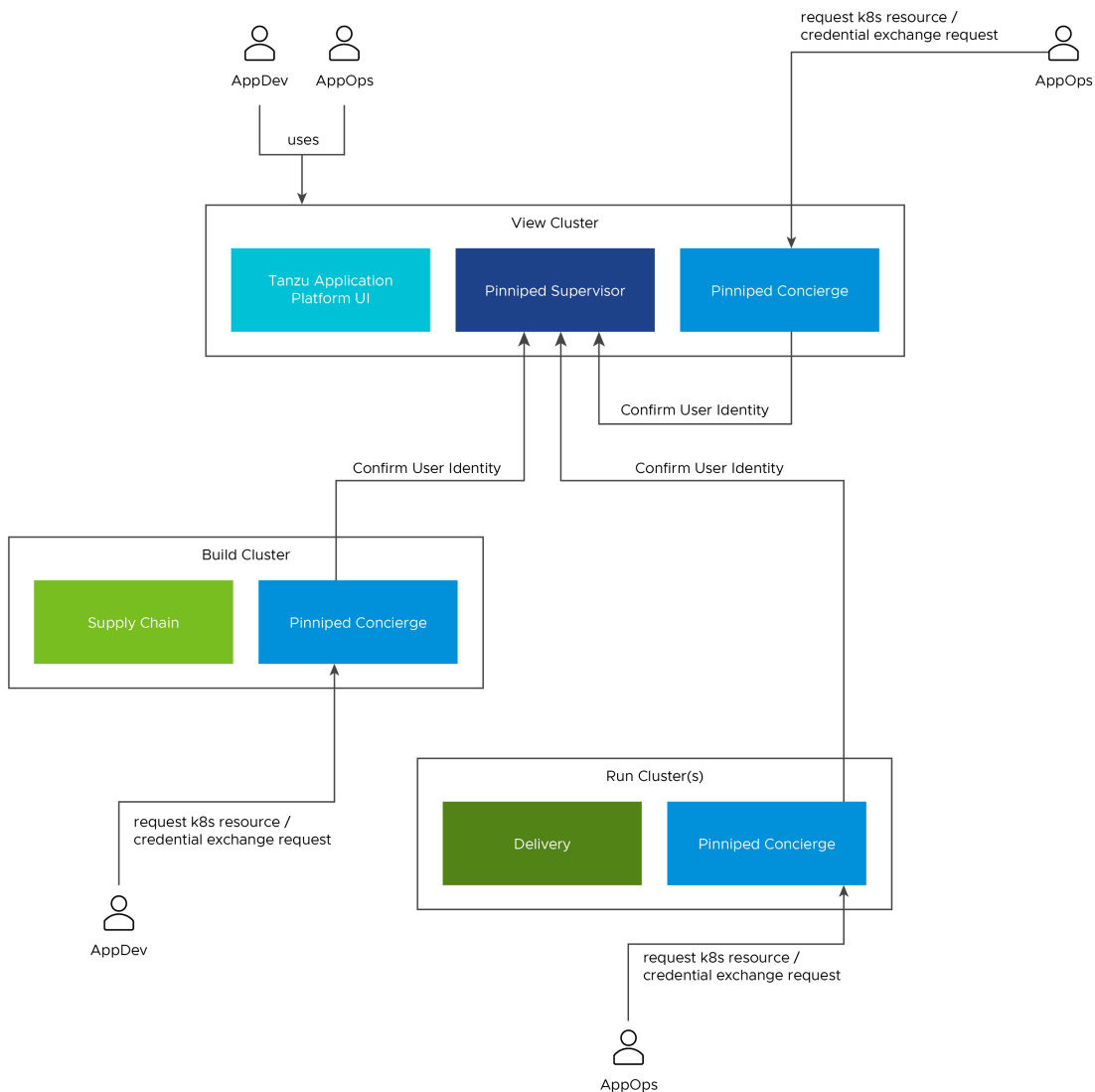
## Environment planning

If you run Tanzu Application Platform on a single cluster, both Pinniped Supervisor and Pinniped Concierge are installed to this cluster.

When running a multicluster setup, you must decide which cluster to deploy the Supervisor onto. Furthermore, every cluster must have the Concierge deployed. Pinniped Supervisor runs as a central component that is consumed by multiple Pinniped Concierge instances. As a result, Pinniped Supervisor must be deployed to a single cluster that meets the [prerequisites](#). You can deploy Pinniped Supervisor to the View Cluster of your Tanzu Application Platform, because it is a central single instance cluster. For more information, see [Overview of multicluster Tanzu Application Platform](#).

You must deploy the Pinniped Concierge to every cluster that you want to enable authentication for, including the View Cluster itself.

See the following diagram for a possible deployment model:



For more information about the Pinniped architecture and deployment model, see [Pinniped documentation](#).

## Install Pinniped Supervisor by using Let's Encrypt

Follow these steps to install `pinniped-supervisor`:

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

### Create Certificates (`letsencrypt` or `cert-manager`)

Choose a fully qualified domain name (FQDN) that can resolve to the Contour instance in the `tanzu-system-ingress` namespace. The FQDN `pinniped-supervisor.example.com` is used in the following sections.

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

```
---
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-staging
  namespace: cert-manager
spec:
  acme:
    email: "EMAIL"
    privateKeySecretRef:
      name: letsencrypt-staging
    server: https://acme-staging-v02.api.letsencrypt.org/directory
    solvers:
    - http01:
        ingress:
          class: contour
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: pinniped-supervisor-cert
  namespace: pinniped-supervisor
spec:
  secretName: pinniped-supervisor-tls-cert
  dnsNames:
  - "DNS-NAME"
  issuerRef:
    name: letsencrypt-staging
    kind: ClusterIssuer
```

Where:

- `EMAIL` is the user email address for `letsencrypt`. For example, `your-mail@example.com`
- `DNS-NAME` is the domain in which the `pinniped-supervisor` is published. For example, `pinniped-supervisor.example.com`



## Create Ingress resources

Create a Service and Ingress resource to make the `pinniped-supervisor` accessible from outside the cluster.

To do so, create the following resources and save them into `workspace/pinniped-supervisor/ingress.yaml`:

```
---
apiVersion: v1
kind: Service
metadata:
  name: pinniped-supervisor
  namespace: pinniped-supervisor
spec:
  ports:
    - name: pinniped-supervisor
      port: 8443
      protocol: TCP
      targetPort: 8443
  selector:
    app: pinniped-supervisor

---
apiVersion: projectcontour.io/v1
kind: HTTPProxy
metadata:
  name: pinniped-supervisor
  namespace: pinniped-supervisor
spec:
  virtualhost:
    fqdn: "DNS-NAME"
    tls:
      passthrough: true
  routes:
    - services:
        - name: pinniped-supervisor
          port: 8443
```

Where:

- `DNS-NAME` is the domain in which the `pinniped-supervisor` is published. For example, `pinniped-supervisor.example.com`
- `tls.passthrough: true` specifies that the TLS connection is forwarded to and terminated in the supervisor pod.

## Create the `pinniped-supervisor` configuration

Create a `FederationDomain` to link the concierge to the supervisor instance and configure an `OIDCIdentityProvider` to connect the supervisor to your OIDC Provider. The following example uses `auth0` as the `OIDCIdentityProvider`. For more information about how to configure different identity providers, including OKTA, GitLab, OpenLDAP, Dex, Microsoft AD and more, see [Pinniped documentation](#).

To create the `pinniped-supervisor` configuration, create the following resources and save them into `workspace/pinniped-supervisor/oidc_identity_provider.yaml`:

```
apiVersion: idp.supervisor.pinniped.dev/v1alpha1
kind: OIDCIdentityProvider
metadata:
  namespace: pinniped-supervisor
  name: auth0
```

```

spec:
  # Specify the upstream issuer URL associated with your auth0 application.
  issuer: https://"APPLICATION-SUBDOMAIN".auth0.com/

  # Specify how to form authorization requests.
  authorizationConfig:
    additionalScopes: ["openid", "email"]
    allowPasswordGrant: false

  # Specify how claims are mapped to Kubernetes identities. This varies by provider.
  claims:
    username: email
    groups: groups

  # Specify the name of the Kubernetes Secret that contains your
  # application's client credentials (created as follows).
  client:
    secretName: auth0-client-credentials

---
apiVersion: v1
kind: Secret
metadata:
  namespace: pinniped-supervisor
  name: auth0-client-credentials
type: secrets.pinniped.dev/oidc-client
stringData:
  clientID: "AUTH0-CLIENT-ID"
  clientSecret: "AUTH0-CLIENT-SECRET"

---
apiVersion: config.supervisor.pinniped.dev/v1alpha1
kind: FederationDomain
metadata:
  name: pinniped-supervisor-federation-domain
  namespace: pinniped-supervisor
spec:
  issuer: "DNS-NAME"
  tls:
    secretName: pinniped-supervisor-tls-cert

```

Where:

- `APPLICATION-SUBDOMAIN` is the application specific subdomain that is assigned after the application registration.
- `AUTH0-CLIENT-ID` and `AUTH0-CLIENT-SECRET` are the credentials retrieved from the application registration.
- `DNS-NAME` is the domain in which the `pinniped-supervisor` is published. For example, `pinniped-supervisor.example.com`

## Apply the resources

After creating the resource files, you can install them into the cluster. Follow these steps to deploy them as a [kapp application](#):

1. Install the `pinniped-supervisor` by running:

```
kapp deploy -y --app pinniped-supervisor -f pinniped-supervisor -f https://get.pinniped.dev/v0.22.0/install-pinniped-supervisor.yaml
```



### Note

To keep the security patches up to date, you must install the most recent version of Pinniped. See [Vmware Tanzu Pinniped Releases](#) in GitHub for more information.

2. Get the external IP address of Ingress by running:

```
kubectl -n tanzu-system-ingress get svc/envoy -o jsonpath='{.status.loadBalancer.ingress[0].ip}'
```

3. If not already covered by the Tanzu Application Platform wildcard DNS entry, add an entry to the DNS system to bind the external IP address with.

## Switch to production issuer ([letsencrypt](#) or [cert-manager](#))

Follow these steps to switch to a [letsencrypt](#) production issuer so the generated TLS certificate is recognized as valid by web browsers and clients:

1. Edit the ClusterIssuer for [letsencrypt](#) and add TLS certificate resource for [pinniped-supervisor](#) by creating or updating the following resources and saving them into [workspace/pinniped-supervisor/certificates.yaml](#):

```
---
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-prod
  namespace: cert-manager
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    email: "EMAIL"
    privateKeySecretRef:
      name: letsencrypt-prod
    solvers:
    - http01:
        ingress:
          class: contour
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: pinniped-supervisor-cert
  namespace: pinniped-supervisor
spec:
  secretName: pinniped-supervisor-tls-cert
  dnsNames:
  - "DNS-NAME"
  issuerRef:
    name: letsencrypt-prod
    kind: ClusterIssuer
```

Where:

- o [EMAIL](#) is the user email address for [letsencrypt](#). For example, [your-mail@example.com](#)
  - o [DNS-NAME](#) is the domain in which the [pinniped-supervisor](#) is published. For example, [pinniped-supervisor.example.com](#)
2. Create or update the [pinniped-supervisor](#) kapp application:

```
kapp deploy -y --app pinniped-supervisor -f pinniped-supervisor -f https://get.pinniped.dev/v0.22.0/install-pinniped-supervisor.yaml
```

## Install Pinniped Supervisor Private CA

Follow these steps to install `pinniped-supervisor`:

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

### Create Certificate Secret

Choose a fully qualified domain name (FQDN) that can resolve to the Contour instance in the `tanzu-system-ingress` namespace. Create a certificate by using a CA that the clients trust. This FQDN can be under the `ingress_domain` in the TAP values file, or a dedicated DNS entry. The FQDN `pinniped-supervisor.example.com` is used in the following sections.

After the certificate files are available, they must be encoded to base64 format in a single-line layout. For example, you can encode the certificate file `my.crt` by running:

```
cat my.crt | base64 -w 0
```

Create the following resource and save it into `workspace/pinniped-supervisor/ingress.yaml`:

```
---
apiVersion: v1
kind: Secret
metadata:
  name: pinniped-supervisor-tls-cert
  namespace: pinniped-supervisor
type: kubernetes.io/tls
data:
  tls.crt: PRIVATE-KEY
  tls.key: PUBLIC-KEY
```

Where:

- `PRIVATE-KEY` is the base64 encoded public key.
- `PUBLIC-KEY` is the base64 encoded public key.

### Create Ingress resources

Create a Service and Ingress resource to make the `pinniped-supervisor` accessible from outside the cluster.

To do so, create the following resources and save them into `workspace/pinniped-supervisor/ingress.yaml`:

```
---
apiVersion: v1
kind: Service
metadata:
  name: pinniped-supervisor
  namespace: pinniped-supervisor
spec:
```

```

ports:
- name: pinniped-supervisor
  port: 8443
  protocol: TCP
  targetPort: 8080
selector:
  app: pinniped-supervisor

---
apiVersion: projectcontour.io/v1
kind: HTTPProxy
metadata:
  name: pinniped-supervisor
  namespace: pinniped-supervisor
spec:
  virtualhost:
    fqdn: "DNS-NAME"
    tls:
      passthrough: true
  routes:
  - services:
    - name: pinniped-supervisor
      port: 8443

```

Where:

- `DNS-NAME` is the domain in which the `pinniped-supervisor` is published. For example, `pinniped-supervisor.example.com`
- `tls.passthrough: true` specifies that the TLS connection is forwarded to and terminated in the supervisor pod.

## Create the `pinniped-supervisor` configuration

Create a `FederationDomain` to link the concierge to the supervisor instance and configure an `OIDCIdentityProvider` to connect the supervisor to your OIDC Provider. The following example uses `auth0` as the `OIDCIdentityProvider`. For more information about how to configure different identity providers, including OKTA, GitLab, OpenLDAP, Dex, Microsoft AD and more, see [Pinniped documentation](#).

To create the `pinniped-supervisor` configuration, create the following resources and save them into `workspace/pinniped-supervisor/oidc_identity_provider.yaml`:

```

apiVersion: idp.supervisor.pinniped.dev/v1alpha1
kind: OIDCIdentityProvider
metadata:
  namespace: pinniped-supervisor
  name: auth0
spec:
  # Specify the upstream issuer URL associated with your auth0 application.
  issuer: https://"APPLICATION-SUBDOMAIN".auth0.com/

  # Specify how to form authorization requests.
  authorizationConfig:
    additionalScopes: ["openid", "email"]
    allowPasswordGrant: false

  # Specify how claims are mapped to Kubernetes identities. This varies by provider.
  claims:
    username: email
    groups: groups

  # Specify the name of the Kubernetes Secret that contains your
  # application's client credentials (created as follows).

```

```

client:
  secretName: auth0-client-credentials

---
apiVersion: v1
kind: Secret
metadata:
  namespace: pinniped-supervisor
  name: auth0-client-credentials
type: secrets.pinniped.dev/oidc-client
stringData:
  clientID: "AUTH0-CLIENT-ID"
  clientSecret: "AUTH0-CLIENT-SECRET"

---
apiVersion: config.supervisor.pinniped.dev/v1alpha1
kind: FederationDomain
metadata:
  name: pinniped-supervisor-federation-domain
  namespace: pinniped-supervisor
spec:
  issuer: "DNS-NAME"
  tls:
    secretName: pinniped-supervisor-tls-cert

```

Where:

- **APPLICATION-SUBDOMAIN** is the application specific subdomain that is assigned after the application registration.
- **AUTH0-CLIENT-ID** and **AUTH0-CLIENT-SECRET** are the credentials retrieved from the application registration.
- **DNS-NAME** is the domain in which the pinniped-supervisor is published. For example, `pinniped-supervisor.example.com`

## Apply the resources

After creating the resource files, you can install them into the cluster. Follow these steps to deploy them as a **kapp application**:

1. Install the supervisor by running:

```
kapp deploy -y --app pinniped-supervisor -f pinniped-supervisor -f https://get.pinniped.dev/v0.22.0/install-pinniped-supervisor.yaml
```



### Note

To keep the security patches up to date, you must install the most recent version of Pinniped. See [Vmware Tanzu Pinniped Releases](#) in GitHub for more information.

2. Get the external IP address of Ingress by running:

```
kubectl -n tanzu-system-ingress get svc/envoy -o jsonpath='{.status.loadBalancer.ingress[0].ip}'
```

3. If not already covered by a Tanzu Application Platform wildcard DNS entry, add an entry to the DNS system to bind the external IP address with.

## Install Pinniped Concierge

To install Pinniped Concierge:

1. Switch tooling to the desired cluster.
2. Deploy the Pinniped Concierge by running:

```
kapp deploy -y --app pinniped-concierge \
  -f https://get.pinniped.dev/v0.22.0/install-pinniped-concierge.yaml
```

3. Get the CA certificate of the supervisor by running the following command against the cluster running the `pinniped-supervisor`:

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



### Note

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

4. Create the following resource to `workspace/pinniped-concierge/jwt_authenticator.yaml`:

```
---
apiVersion: authentication.concierge.pinniped.dev/v1alpha1
kind: JWTAuthenticator
metadata:
  name: pinniped-jwt-authenticator
spec:
  issuer: "DNS-NAME"
  audience: concierge
  tls:
    certificateAuthorityData: "CA-DATA"
```

If you use the `letsencrypt` production issuer, you can omit the `tls` section:

```
---
apiVersion: authentication.concierge.pinniped.dev/v1alpha1
kind: JWTAuthenticator
metadata:
  name: pinniped-jwt-authenticator
spec:
  issuer: "DNS-NAME"
  audience: concierge
```

Where:

- o `DNS-NAME` is the domain in which the `pinniped-supervisor` is published. For example, `pinniped-supervisor.example.com`
  - o `CA-DATA` is the public key of the signing CA or the public key of the Pinniped httpoxy certificate.
5. 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 [Log in by using Pinniped](#).

## Integrate your Azure Active Directory

This topic tells you how to integrate your Azure Active Directory (commonly known as AD).

### Integrate Azure AD with a new or existing AKS without Pinniped

Perform the following procedures to integrate Azure AD with a new or existing AKS without Pinniped.

#### Prerequisites

Meet these prerequisites:

- Download and install the [Azure CLI](#)
- Download and install the [Tanzu CLI](#)
- Download and install the [Tanzu CLI RBAC plug-in](#)

#### Set up a platform operator

To set up a platform operator:

1. Navigate to the **Azure Active Directory Overview** page.
2. Select **Groups** under the **Manage** side menu.
3. Identify or create an admin group for the AKS cluster.
4. Retrieve the object ID of the admin group.
5. Take one of the following actions.
  - Create an AKS Cluster with Azure AD enabled by running:

```
az group create --name RESOURCE-GROUP --location LOCATION
az aks create -g RESOURCE-GROUP -n MANAGED-CLUSTER --enable-aad --aad-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](#).

## Set up the Azure AD app

To set up the Azure AD app:

1. Navigate to the **Azure Active Directory Overview** page.
2. Select **App registrations** under the **Manage** side menu.
3. Select **New Registration**.
4. Enter the name of the application. For example, `gke-pinniped-supervisor-app`.
5. Under **Supported account types**, select **Accounts in this organisational directory only (VMware, Inc. only - Single tenant)**.
6. Under **Redirect URI**, select **Web** as the platform.
7. Enter the call URI to the supervisor. For example, `https://pinniped-supervisor.example.com/callback`.
8. Select **Register** to create the app.
9. If not already redirected, navigate to the app settings page.
10. Select **Token configuration** under the **Manage** menu.
11. Select **Add groups claim** > **All groups (includes distribution lists but not groups assigned to the application)**.
12. Select **Add** to create the group claim.
13. Select the app name in the breadcrumb navigation to return to the app settings page.
14. Select the **Endpoints** tab and record the value in the **OpenID Connect metadata document** field.
15. Return to the app settings page.
16. Record the **Application (client) ID**.
17. Select **Certificates & secrets** under the **Manage** menu.
18. Create a new client secret and record this value.
19. Add the following YAML to `oidc_identity_provider.yaml`.

```
---
apiVersion: idp.supervisor.pinniped.dev/v1alpha1
kind: OIDCIdentityProvider
metadata:
  namespace: pinniped-supervisor
  name: azure-ad
spec:
  # Specify the upstream issuer URL.
  issuer: ISSUER-URL

  authorizationConfig:
    additionalScopes: ["openid", "email", "profile"]
```

```

allowPasswordGrant: false

# Specify how claims are mapped to Kubernetes identities.
claims:
  username: preferred_username
  groups: groups

# Specify the name of the Kubernetes Secret that contains your
# application's client credentials (created below).
client:
  secretName: azure-ad-client-credentials
---
apiVersion: v1
kind: Secret
metadata:
  namespace: pinniped-supervisor
  name: azure-ad-client-credentials
  type: secrets.pinniped.dev/oidc-client
stringData:
  clientID: "AZURE-AD-CLIENT-ID"
  clientSecret: "AZURE-AD-CLIENT-SECRET"

```

Where:

- `ISSUER-URL` is the OpenID Connect metadata document URL you recorded earlier, but without the trailing `/.well-known/openid-configuration`
- `AZURE-AD-CLIENT-ID` is the Azure AD client ID you recorded earlier
- `AZURE-AD-CLIENT-SECRET` is the Azure AD client secret you recorded earlier

20. Apply your changes from the kubectl CLI by running:

```
kubectl apply workspace/pinniped-supervisor/oidc_identity_provider.yaml
```

## Set up the Tanzu Application Platform default role group

To set up a Tanzu Application Platform default role group:

1. Navigate to the **Azure Active Directory Overview** page.
2. Select **Groups** under the **Manage** side menu.
3. Identify or create a list of groups in the Azure AD for each of the Tanzu Application Platform default roles (`app-operator`, `app-viewer`, and `app-editor`).
4. Retrieve the corresponding object IDs for each group.
5. Add users to the groups accordingly.
6. For each object ID retrieved earlier, use the Tanzu CLI RBAC plug-in to bind the `object id` group to a role by running:

```
tanzu rbac binding add -g OBJECT-ID -r TAP-ROLE -n NAMESPACE
```

Where:

- `OBJECT-ID` is the object ID
- `TAP-ROLE` is the Tanzu Application Platform role
- `NAMESPACE` is the namespace

## Set up kubeconfig

Follow these steps to set up kubeconfig:

1. Set up `kubeconfig` using the Pinniped CLI by running:

```
pinniped get kubeconfig --kubeconfig-context YOUR-KUBECONFIG-CONTEXT > /tmp/concierge-kubeconfig
```

Where `YOUR-KUBECONFIG-CONTEXT` is your your kubeconfig context.

2. Run any kubectl command to trigger a browser login. For example:

```
export KUBECONFIG="/tmp/concierge-kubeconfig"
kubectl get pods
```

## Role descriptions for Tanzu Application Platform

This topic is a high level overview of each default role. For more information about the specific permissions of each role for every Tanzu Application Platform (commonly known as TAP) component, see [Detailed role permissions for Tanzu Application Platform](#).

### app-editor

The app-editor role can create, edit, and delete a Tanzu workload or deliverable.

Assign this role to a user, for example an app developer, to give permissions to create running workloads on the cluster. This allows them to deploy their applications. This role allows the user to:

- View, create, update, or delete a Tanzu workload or deliverable. This includes viewing the logs of the pods spun up through the Tanzu workload and tracing a commit through the build process.
- Download the images associated with their Tanzu workload so they can test images locally, or create a Tanzu workload from it instead of starting from source code in a repository.
- View and use Application Accelerator templates.
- View, create, update, or delete a Tanzu workload binding with an existing service.

### app-viewer

The app-viewer role cannot create, edit, or delete a Tanzu workload or deliverable.

This role has a subset of the permissions of the app-editor role. Use it if you do not want a user to create, edit, or delete a Tanzu workload or deliverable, but they need to view its status. For example, give these permissions to an application developer that requires visibility into the state of their Tanzu workload or micro-service, but does not have the permissions to deploy it, such as to production or staging environments. This role cannot bind services with a Tanzu workload.

### app-operator

The app-operator role can create, edit, and delete supply chain resources.

Assign this role to a user who defines the activities within a supply chain or the path to production. For example, building, testing, or scanning. This role can view, create, update, or delete Tanzu supply chain resources, including Tanzu Build Service control plane resources such as:

- kpack's builder, stack, and store
- Scanning resources
- Grype
- The metadata store

If this person must create Tanzu workloads, you can bind the user with the app-editor role as well.

## service-operator

The service-operator role can create, edit, and delete service instances, service instance classes, and resource claim policies to permit the claimability of service instances across one or more namespaces.

Assign this role to a user who is responsible for the life cycle (create, edit and delete) of service instances. This role can also view resource claims across all namespaces as well as query for the list of claimable service instances in a given namespace.

## workload

This role provides the service account associated with the Tanzu workload the permissions needed to execute the activities in the supply chain. This role is for a "robot" versus a user.

## deliverable

This role gives the delivery "robot" service account the permissions needed to create running workloads. This role is not for a user.

## Role descriptions for Tanzu Application Platform

This topic is a high level overview of each default role. For more information about the specific permissions of each role for every Tanzu Application Platform (commonly known as TAP) component, see [Detailed role permissions for Tanzu Application Platform](#).

## app-editor

The app-editor role can create, edit, and delete a Tanzu workload or deliverable.

Assign this role to a user, for example an app developer, to give permissions to create running workloads on the cluster. This allows them to deploy their applications. This role allows the user to:

- View, create, update, or delete a Tanzu workload or deliverable. This includes viewing the logs of the pods spun up through the Tanzu workload and tracing a commit through the build process.
- Download the images associated with their Tanzu workload so they can test images locally, or create a Tanzu workload from it instead of starting from source code in a repository.
- View and use Application Accelerator templates.
- View, create, update, or delete a Tanzu workload binding with an existing service.

## app-viewer

The app-viewer role cannot create, edit, or delete a Tanzu workload or deliverable.

This role has a subset of the permissions of the app-editor role. Use it if you do not want a user to create, edit, or delete a Tanzu workload or deliverable, but they need to view its status. For example, give these permissions to an application developer that requires visibility into the state of their Tanzu workload or micro-service, but does not have the permissions to deploy it, such as to production or staging environments. This role cannot bind services with a Tanzu workload.

## app-operator

The app-operator role can create, edit, and delete supply chain resources.

Assign this role to a user who defines the activities within a supply chain or the path to production. For example, building, testing, or scanning. This role can view, create, update, or delete Tanzu supply chain resources, including Tanzu Build Service control plane resources such as:

- kpack's builder, stack, and store
- Scanning resources
- Grype
- The metadata store

If this person must create Tanzu workloads, you can bind the user with the app-editor role as well.

## service-operator

The service-operator role can create, edit, and delete service instances, service instance classes, and resource claim policies to permit the claimability of service instances across one or more namespaces.

Assign this role to a user who is responsible for the life cycle (create, edit and delete) of service instances. This role can also view resource claims across all namespaces as well as query for the list of claimable service instances in a given namespace.

## workload

This role provides the service account associated with the Tanzu workload the permissions needed to execute the activities in the supply chain. This role is for a "robot" versus a user.

## deliverable

This role gives the delivery "robot" service account the permissions needed to create running workloads. This role is not for a user.

## Detailed role permissions for Tanzu Application Platform

This topic tells you the specific permissions of each role for every Tanzu Application Platform (commonly known as TAP) component.

## Native Kubernetes Resources

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

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

```
resources: ["ingresses"]
verbs: ["get", "list", "watch"]
```

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

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

## App Accelerator

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

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

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

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

## Cartographer

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

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

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

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

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

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

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

```
- apiGroups: ["carto.run"]
  resources: ["clusterconfigtemplates", "clusterconfigtemplates", "clusterdeliveries", "clusterdeploymenttemplates", "clusterimagetemplates", "clusterruntemplates", "clustersourc
etemplates", "clustersupplychains", "clustertemplates"]
  verbs: ["get", "list", "watch", "create", "patch", "update", "delete", "deletecollection"]
```

## Cloud Native Runtimes

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

```
- apiGroups: ["apps"]
  resources: ["deployments","replicasets","statefulsets"]
  verbs: ["get","list","watch"]
- apiGroups: ["batch"]
  resources: ["cronjobs","jobs"]
  verbs: ["get","list","watch"]
- apiGroups: ["networking.k8s.io"]
  resources: ["ingresses"]
  verbs: ["get","list","watch"]
- apiGroups: ["eventing.knative.dev"]
  resources: ["brokers","triggers"]
  verbs: ["get","list","watch"]
- apiGroups: ["serving.knative.dev"]
  resources: ["configurations","services","revisions","routes"]
  verbs: ["get","list","watch"]
- apiGroups: ["sources.*"]
  resources: ["(many)"]
  verbs: ["get","list","watch"]
```

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

```
- apiGroups: ["eventing.knative.dev"]
  resources: ["brokers"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["sources.*"]
  resources: ["(many)"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
```

## Convention Service

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

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

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

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

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

```
- apiGroups: ["conventions.carto.run"]
  resources: ["clusterpodconventions"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
```



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

## Developer Conventions

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

```
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get","list","watch"]
- apiGroups: [""]
  resources: ["pods/exec","pods/portforward"]
  verbs: ["get","list","create"]
- apiGroups: ["carto.run"]
  resources: ["workloads"]
  verbs: ["get","list","watch"]
```

## OOTB Templates

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

```
- apiGroups: [""]
  resources: ["configmaps"]
  verbs: ["get","list","watch"]
- apiGroups: ["carto.run"]
  resources: ["deliverables","runnables"]
  verbs: ["get","list","watch"]
- apiGroups: ["conventions.carto.run"]
  resources: ["podintents"]
- apiGroups: ["conventions.apps.tanzu.vmware.com"]
  resources: ["podintents"]
  verbs: ["get","list","watch"]
- apiGroups: ["kappctrl.k14s.io"]
  resources: ["apps"]
  verbs: ["get","list","watch"]
- apiGroups: ["kpack.io"]
  resources: ["images"]
  verbs: ["get","list","watch"]
- apiGroups: ["scanning.apps.tanzu.vmware.com"]
  resources: ["imagescans","sourcescans"]
  verbs: ["get","list","watch"]
- apiGroups: ["servicebinding.io"]
  resources: ["servicebindings"]
  verbs: ["get","list","watch"]
- apiGroups: ["services.apps.tanzu.vmware.com"]
  resources: ["resourceclaims"]
  verbs: ["get","list","watch"]
- apiGroups: ["serving.knative.dev"]
  resources: ["services"]
  verbs: ["get","list","watch"]
- apiGroups: ["source.apps.tanzu.vmware.com"]
  resources: ["imagerepositories","mavenartifacts"]
  verbs: ["get","list","watch"]
- apiGroups: ["source.toolkit.fluxcd.io"]
  resources: ["gitrepositories"]
  verbs: ["get","list","watch"]
- apiGroups: ["tekton.dev"]
  resources: ["pipelineruns","taskruns"]
  verbs: ["get","list","watch"]
```

**apps.tanzu.vmware.com/aggregate-to-workload: "true"**

```

- apiGroups: ["carto.run"]
  resources: ["deliverables","runnables"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["conventions.carto.run"]
  resources: ["podintents"]
- apiGroups: ["conventions.apps.tanzu.vmware.com"]
  resources: ["podintents"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["kpack.io"]
  resources: ["images"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["scanning.apps.tanzu.vmware.com"]
  resources: ["imagescans","sourcescans"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["source.apps.tanzu.vmware.com"]
  resources: ["imagerepositories","mavenartifacts"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["source.toolkit.fluxcd.io"]
  resources: ["gitrepositories"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["tekton.dev"]
  resources: ["pipelineruns","taskruns"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]

```

**apps.tanzu.vmware.com/aggregate-to-deliverable: "true"**

```

- apiGroups: [""]
  resources: ["configmaps"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["kappctrl.k14s.io"]
  resources: ["apps"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["servicebinding.io"]
  resources: ["servicebindings"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["services.apps.tanzu.vmware.com"]
  resources: ["resourceclaims"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["serving.knative.dev"]
  resources: ["services"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["source.apps.tanzu.vmware.com"]
  resources: ["imagerepositories","mavenartifacts"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["source.toolkit.fluxcd.io"]
  resources: ["gitrepositories"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]

```

## Service Bindings

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

```

- apiGroups: ["servicebinding.io"]
  resources: ["servicebindings"]
  verbs: ["get","list","watch"]

```

## Services Toolkit

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

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

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

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

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

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

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

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

## Source Controller

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

```
- apiGroups: ["source.apps.tanzu.vmware.com"]
  resources: ["imagerepositories", "mavenartifacts"]
  verbs: ["get", "list", "watch"]
```

## Supply Chain Security Tools — Scan

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

```
- apiGroups: ["scanning.apps.tanzu.vmware.com"]
  resources: ["imagescans", "scanpolicies", "scantemplates", "sourcescans"]
  verbs: ["get", "list", "watch"]
```

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

```
- apiGroups: ["scanning.apps.tanzu.vmware.com"]
  resources: ["scanpolicies", "scantemplates"]
  verbs: ["get", "list", "watch", "create", "patch", "update", "delete", "deletecollection"]
```

## Tanzu Build Service

`apps.tanzu.vmware.com/aggregate-to-app-editor: "true"`

```
- apiGroups: ["kpack.io"]
  resources: ["builds"]
```

```
verbs: ["patch"]
```

`apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"`

```
- apiGroups: ["kpack.io"]
  resources: ["builds","builders","images"]
  verbs: ["get","list","watch"]
```

`apps.tanzu.vmware.com/aggregate-to-app-viewer-cluster-access: "true"`

```
- apiGroups: ["kpack.io"]
  resources: ["clusterbuilders","clusterstacks","clusterstores"]
  verbs: ["get","list","watch"]
```

`apps.tanzu.vmware.com/aggregate-to-app-operator: "true"`

```
- apiGroups: ["kpack.io"]
  resources: ["builders"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
```

`apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access`

```
- apiGroups: ["kpack.io"]
  resources: ["clusterbuilders","clusterstacks","clusterstores"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
```

## Tekton

`apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"`

```
- apiGroups: ["tekton.dev"]
  resources: ["pipelineresources","pipelineruns","pipelines","taskruns","tasks"]
  verbs: ["get","list","watch"]
```

`apps.tanzu.vmware.com/aggregate-to-app-viewer-cluster-access: "true"`

```
- apiGroups: ["tekton.dev"]
  resources: ["clustertasks"]
  verbs: ["get","list","watch"]
```

`apps.tanzu.vmware.com/aggregate-to-app-operator: "true"`

```
- apiGroups: ["tekton.dev"]
  resources: ["pipelineresources","pipelines","tasks"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
```

`apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access`

```
- apiGroups: ["tekton.dev"]
  resources: ["clustertasks"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
```

## Bind a user or group to a default role

You can choose one of the following two approaches to bind a user or group to a default role:

- Use the Tanzu Application Platform RBAC CLI plug-in, which only supports binding Tanzu Application Platform (commonly known as TAP) default roles.
- Use Kubernetes role-based access control (RBAC) role binding.

VMware recommends that you use the Tanzu Application Platform RBAC CLI plug-in. This CLI plug-in simplifies the process by binding the cluster-scoped resource permissions at the same time as the namespace-scoped resource permissions, where applicable, for each default role. The following sections cover the Tanzu Application Platform RBAC CLI plug-in.

## Prerequisites

1. Download the latest Tanzu CLI version.
2. Download the Tanzu Application Platform RBAC CLI plug-in `tar.gz` file from [Tanzu Network](#).
3. Ensure you have admin access to the cluster.
4. Ensure you have configured an authentication solution for the cluster. You can use [Pinniped](#) or the authentication service native to your Kubernetes distribution.

## Install the Tanzu Application Platform RBAC CLI plug-in

Follow these steps to install the Tanzu Application Platform RBAC CLI plug-in:



### Caution

The Tanzu Application Platform RBAC CLI plug-in is currently in beta and is intended for evaluation and test purposes only.

1. Untar the `tar.gz` file:

```
tar -zxvf NAME-OF-THE-TAR
```

2. Install the Tanzu Application Platform RBAC CLI plug-in locally on your operating system:

### macOS

```
tanzu plugin install rbac --local darwin-amd64
```

### Linux

```
tanzu plugin install rbac --local linux-amd64
```

### Windows

```
tanzu plugin install rbac --local windows-amd64
```

## (Optional) Use a different kubeconfig location

You can use a different kubeconfig location by running:

```
tanzu rbac --kubeconfig PATH-OF-KUBECONFIG binding add --user USER --role ROLE --namespace NAMESPACE
```

**Note**

The environment variable `KUBECONFIG` is not implemented. You must use the `--kubeconfig` flag to enter a different location. Otherwise the default `~/.kube/config` is used.

For example:

```
$ tanzu rbac --kubeconfig /tmp/pinniped_kubeconfig.yaml binding add --user username@vmware.com --role app-editor --namespace user-ns
```

## Add the specified user or group to a role

Add a user or group to a role by running:

```
tanzu rbac binding add --user USER --role ROLE --namespace NAMESPACE
```

```
tanzu rbac binding add --group GROUP --role ROLE --namespace NAMESPACE
```

For example:

```
$ tanzu rbac binding add --user username@vmware.com --role app-editor --namespace user-ns
```

## Get a list of users and groups from a role

Get a list of users and groups from a role by running:

```
tanzu rbac binding get --role ROLE --namespace NAMESPACE
```

For example:

```
$ tanzu rbac binding get --role app-editor --namespace user-ns
```

## Remove the specified user or group from a role

Remove a user or group from a role by running:

```
tanzu rbac binding delete --user USER --role ROLE --namespace NAMESPACE
```

```
tanzu rbac binding delete --group GROUP --role ROLE --namespace NAMESPACE
```

For example:

```
$ tanzu rbac binding delete --user username@vmware.com --role app-editor --namespace user-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[] | sel
ect(.subjects[]?.name == \"\${NAME}\") | .roleRef.name" | xargs -n1 kubectl desc
ribe clusterroles
```

2. Get a list of user or group for a specific role:

```
tanzu rbac binding get --role ROLE --namespace NAMESPACE
```

## Log in to Tanzu Application Platform by using Pinniped

This topic tells you how to log in to your Tanzu Application Platform (commonly known as TAP) by using Pinniped.

As a prerequisite, the administrator must provide users access to resources by using [rolebindings](#). It can be done with the `tanzu rbac` plug-in. For more information, see [Bind a user or group to a default role](#).

To log in to your cluster by using Pinniped, follow these steps:

1. Install the Pinniped CLI.

For more information, see [Pinniped documentation](#).



### Important

The latest compatible version of Pinniped CLI is required not only for the administrator to generate the `kubeconfig`, but also for the user to log in with the provided configuration.

2. Generate and distribute `kubeconfig` to users.
3. Login with the provided `kubeconfig`.

## Download the Pinniped CLI



You must use a Pinniped CLI version that matches the installed Concierge or Supervisor. Use one of the following links to download the Pinniped CLI version `0.22.0`:

- [Mac OS with AMD64](#)
- [Linux with AMD64](#)
- [Windows with AMD64](#)

You must install the command-line tool on your `$PATH`, such as `/usr/local/bin` on macOS or Linux. You must also mark the file as executable.

## Generate and distribute kubeconfig to users

As an administrator, you can generate the kubeconfig by using the following command:

```
pinniped get kubeconfig --kubeconfig-context <your-kubeconfig-context> > /tmp/concierge-kubeconfig
```

Distribute this `kubeconfig` to your users so they can login by using `pinniped`.

## Login with the provided kubeconfig

As a user of the cluster, you need the `kubeconfig` provided by your admin and the Pinniped CLI installed on your local machine to log in. Logging in is required to request information from the cluster. You can execute any resource request with `kubectl` to enter the authentication flow. For example:

```
kubectl --kubeconfig /tmp/concierge-kubeconfig get pods
```

If you do not want to explicitly use `--kubeconfig` in every command, you can also export an environment variable to set the `kubeconfig` path in your shell session.

```
export KUBECONFIG="/tmp/concierge-kubeconfig"
kubectl get pods
```

This command enables `pinniped` to print a URL for you to visit in the browser. You can then log in, copy the authentication code and paste it back to the terminal. After the login succeeds, you either see the resources or a message indicating that you have no permission to access the resources.

If you use a Windows machine, the command referenced in the generated `kubeconfig` might not work. In this case, you must change the path under `user.exec.command` in the `kubeconfig` to point to the install path of the Pinniped CLI.

## Additional resources about Tanzu Application Platform authentication and authorization

Use this topic to learn additional information about authentication and authorization for Tanzu Application Platform (commonly known as TAP).

See [Default roles for Tanzu Application Platform overview](#) to get started.

## Install

Default roles are released as part of Tanzu Application Platform. Alternatively, you can also install default roles independently. See [Install default roles independently](#) for more information.

**Note**

The `tanzu rbac` CLI plug-in requires a [separate installation](#).

## Additional resources about Tanzu Application Platform authentication and authorization

Use this topic to learn additional information about authentication and authorization for Tanzu Application Platform (commonly known as TAP).

See [Default roles for Tanzu Application Platform overview](#) to get started.

## Install

Default roles are released as part of Tanzu Application Platform. Alternatively, you can also install default roles independently. See [Install default roles independently](#) for more information.

**Note**

The `tanzu rbac` CLI plug-in requires a [separate installation](#).

## Install default roles independently for your Tanzu Application Platform

This topic tells you how to install default roles for Tanzu Application Platform (commonly known as TAP) without deploying a TAP profile.

**Note**

Follow the steps in this topic if you do not want to use a profile to install default roles. For more information about profiles, see [Components and installation profiles](#).

## Prerequisites

Before installing default roles, complete all prerequisites to install Tanzu Application Platform. For more information, see [Prerequisites](#).

## Install

To install default roles:

1. List version information for the package by running:

```
tanzu package available list tap-auth.tanzu.vmware.com --namespace tap-install
```

For example:

```
$ tanzu package available list tap-auth.tanzu.vmware.com --namespace tap-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:

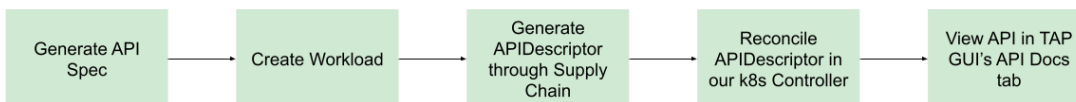
```
$ tanzu package install tap-auth \
  --package-name tap-auth.tanzu.vmware.com \
  --version 1.0.1 \
  --namespace tap-install
```

## Overview of API Auto Registration

This topic provides an overview of API Auto Registration for Tanzu Application Platform.

### Overview

API Auto Registration automates the registration of API specification defined in a workload's configuration. The registered API specification is accessible in Tanzu Application Platform GUI without any additional steps. An automated workflow using a supply chain, leverages API Auto Registration to create and manage a Kubernetes Custom Resource (CR) of kind `APIDescriptor`. A Kubernetes controller periodically reconciles the CR and updates the API entity in Tanzu Application Platform GUI to achieve automated API specification registration from origin workloads. You might also use API Auto Registration without supply chain automation, with other GitOps processes, or by directly applying an `APIDescriptor` CR to the cluster.



### Getting started

For information about API Auto Registration architecture, or the `APIDescriptor` CR and API entities in Tanzu Application Platform GUI, see [Key Concepts](#).

For information about configuring iterate, run, and full Tanzu Application Platform cluster profiles, see [Configure API Auto Registration](#).

For information about generating API specifications and registering them with Tanzu Application Platform GUI catalog, see [Use API Auto Registration](#).

For information about other profiles, install the `api-auto-registration` package. See [Install API Auto Registration](#).

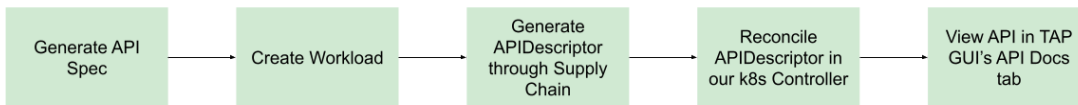
For information about troubleshooting and debugging API Auto Registration, see [Troubleshooting](#).

## Overview of API Auto Registration

This topic provides an overview of API Auto Registration for Tanzu Application Platform.

## Overview

API Auto Registration automates the registration of API specification defined in a workload's configuration. The registered API specification is accessible in Tanzu Application Platform GUI without any additional steps. An automated workflow using a supply chain, leverages API Auto Registration to create and manage a Kubernetes Custom Resource (CR) of kind `APIDescriptor`. A Kubernetes controller periodically reconciles the CR and updates the API entity in Tanzu Application Platform GUI to achieve automated API specification registration from origin workloads. You might also use API Auto Registration without supply chain automation, with other GitOps processes, or by directly applying an `APIDescriptor` CR to the cluster.



## Getting started

For information about API Auto Registration architecture, or the `APIDescriptor` CR and API entities in Tanzu Application Platform GUI, see [Key Concepts](#).

For information about configuring iterate, run, and full Tanzu Application Platform cluster profiles, see [Configure API Auto Registration](#).

For information about generating API specifications and registering them with Tanzu Application Platform GUI catalog, see [Use API Auto Registration](#).

For information about other profiles, install the `api-auto-registration` package. See [Install API Auto Registration](#).

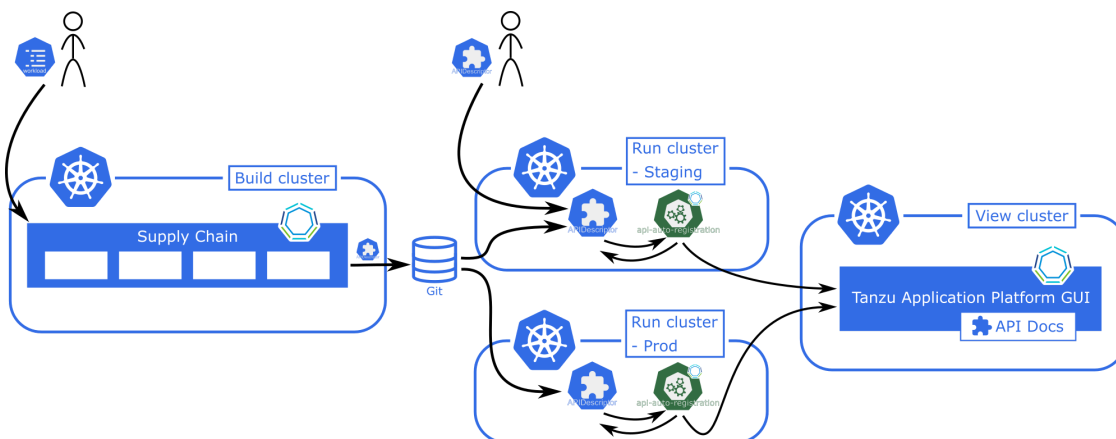
For information about troubleshooting and debugging API Auto Registration, see [Troubleshooting](#).

## Key Concepts for API Auto Registration

This topic explains key concepts you use with API Auto Registration.

## API Auto Registration Architecture

You can use the full potential of API Auto Registration by using a distributed environment, like the one in this diagram:



## APIDescriptor Custom Resource Explained

To use API Auto Registration, you must create a custom resource of type `APIDescriptor`. The information from this custom resource is used to construct an API entity in Tanzu Application Platform GUI.

This custom resource exposes the following text boxes:

```
apiVersion: apis.apps.tanzu.vmware.com/v1alpha1
kind: APIDescriptor
metadata:
  name:           # name of your APIDescriptor
  namespace:     # optional namespace of your APIDescriptor
spec:
  type:          # type of the API spec. oneOf(openapi, grpc, asynccapi, graphql)
  description:   # description for the API exposed
  system:        # system that the API is part of
  owner:         # person/team that owns the API
  location:
    path:        # sub-path where the API spec is available
    baseURL:     # base URL object where the API spec is available. oneOf(url, ref)
    url:         # static absolute base URL
    ref:         # object ref to oneOf(HTTPProxy, Knative Service, Ingress)
    apiVersion:
    kind:
    name:
    namespace:
```

The text boxes cause specific behavior in Tanzu Application Platform GUI:

- The system and owner are copied to the API entity. You might have to separately create and add the `System` and `Group` kind to the catalog.
- Tanzu Application Platform GUI uses the namespace for the API entity where the `APIDescriptor` CR is applied. This causes the API entity's name, system, and owner to all be in that namespace.
- To explicitly use a system or owner in a different namespace, you can specify that in the `system: my-namespace/my-other-system` Or `owner: my-namespace/my-other-team` text boxes.
- If the system or owner you are trying to link doesn't have a namespace specified, you can qualify them with the `default` namespace. For example, `system: default/my-default-system`

## With an Absolute URL

To create an `APIDescriptor` with a static `baseURL.url`, you must apply the following YAML to your cluster.

```
apiVersion: apis.apps.tanzu.vmware.com/v1alpha1
kind: APIDescriptor
metadata:
  name: sample-absolute-url
spec:
  type: openapi
  description: A set of API endpoints to manage the resources within the petclinic app.
  system: spring-petclinic
  owner: team-petclinic
  location:
    path: "/v3/api-docs.yaml"
```

```

baseUrl:
  url: https://myservice.com

```

## With an Object Ref

You can use an object reference, instead of hard coding the URL, to point to a HTTPProxy, Knative Service, or Ingress.

## With an HTTPProxy Object Ref

This section includes an example YAML that points to an HTTPProxy from which the controller extracts the `.spec.virtualhost.fqdn` as the `baseUrl`.

```

apiVersion: apis.apps.tanzu.vmware.com/v1alpha1
kind: APIDescriptor
metadata:
  name: sample-contour-ref
spec:
  type: openapi
  description: A set of API endpoints to manage the resources within the petclinic ap
  p.
  system: spring-petclinic
  owner: team-petclinic
  location:
    path: "/test/openapi"
    baseUrl:
      ref:
        apiVersion: projectcontour.io/v1
        kind: HTTPProxy
        name: my-httpproxy
        namespace: my-namespace # optional

```

## With a Knative Service Object Ref

To use a Knative Service, your controller reads the `status.url` as the `baseUrl`. For example:

```

# all other fields similar to the above example
baseUrl:
  ref:
    apiVersion: serving.knative.dev/v1
    kind: Service
    name: my-knative-service
    namespace: my-namespace # optional

```

## With an Ingress Object Ref

To use an Ingress instead, your controller reads the URL from the `jsonPath` specified. When `jsonPath` is left empty, your controller reads the `"{.spec.rules[0].host}"` as the URL. For example:

```

# all other fields similar to the above example
baseUrl:
  ref:
    apiVersion: networking.k8s.io/v1
    kind: Ingress
    name: my-ingress
    jsonPath: "{.spec.rules[1].host}"
    namespace: my-namespace # optional

```

## APIDescriptor Status Fields

When processing an APIDescriptor several fields are added to the `status`. One of these is `conditions`, which provide information useful for troubleshooting. The conditions are explained in the [Troubleshooting Guide](#).

In addition to `conditions` the `status` contains a couple of other useful fields. The following is a list of these fields with a brief explanation of what they contain.

```
status:
  registeredEntityURL: # Url of the corresponding API Entity in TAP GUI
  registeredTapUID:   # Unique identifier for the corresponding API Entity in TAP G
  UI
  resolvedAPISpec:   # Full API Spec as retrieved by Api Auto Registration
```

## Install API Auto Registration

This topic describes how you can install API Auto Registration from the Tanzu Application Platform package repository.



### Note

Follow the steps in this topic if you do not want to use a profile to install API Auto Registration. For more information about profiles, see [Components and installation profiles](#).

## Tanzu Application Platform prerequisites

Before installing API Auto Registration, complete all prerequisites to install Tanzu Application Platform. See [Tanzu Application Platform Prerequisites](#).

## Using with TLS

Starting in Tanzu Application Platform v1.4, TLS is turned on by default for several components. API Auto Registration automatically trusts the CA for the shared `ingress_issuer` when using the default ClusterIssuer `tap-ingress-selfsigned`. This change means that a `Certificate` is automatically generated using this issuer.

If you do not want a `Certificate` to generate automatically, you can set the `auto_generate_cert` flag to `false` in the values file. To replace the default with a custom ingress issuer, see [Security and compliance](#). Whenever you do not use the default ClusterIssuer `tap-ingress-selfsigned`, do not automatically generate certificates, or use other custom CAs, you must manually set the certificate. See [Troubleshooting](#).

## Install

To install the API Auto Registration package:

1. List version information for the package by running:

```
tanzu package available list apis.apps.tanzu.vmware.com --namespace tap-install
```

For example:

```
$ tanzu package available list apis.apps.tanzu.vmware.com --namespace tap-install
```

```
- Retrieving package versions for apis.apps.tanzu.vmware.com...
NAME                                VERSION  RELEASED-AT
apis.apps.tanzu.vmware.com  0.1.0   2022-08-30 19:00:00 -0500 -05
apis.apps.tanzu.vmware.com  0.2.0   2022-11-24 12:20:00 -0500 -05
```

## 2. (Optional) Gather values schema.

Display values schema of the package:

```
tanzu package available get apis.apps.tanzu.vmware.com/VERSION-NUMBER --values-
schema --namespace tap-install
```

Where `VERSION-NUMBER` is the version of the package listed in the earlier step.

For example:

```
tanzu package available get apis.apps.tanzu.vmware.com/0.2.2 --values-schema --
namespace tap-install

Retrieving package details for apis.apps.tanzu.vmware.com/0.2.2...
KEY                                DEFAULT                                TYPE
DESCRIPTION
ca_cert_data                        string
Optional: PEM-encoded certificate data for the controller to trust TLS connecti
ons with a custom CA.
ingress_issuer                       string
Optional: Name of the default cluster issuer used to generate certificates.
auto_generate_cert                   true                                   boolean
Flag that indicates if a cert-manager certificate should be generated using
the ingress_issuer. Only applies if the ingress_issuer is specified.
cluster_name                         dev                                   string
Name of the cluster used for setting the API entity lifecycle in TAP GUI. The v
alue should be unique for each run cluster.
sync_period                           5m                                   string
Time period used for reconciling an APIDescriptor.
tap_gui_url                          http://server.tap-gui.svc.cluster.local:7000 string
FQDN URL for TAP GUI.
replicas                              1                                   integer
Number of controller replicas to deploy.
resources.limits.cpu                  500m                                  string
CPU limit of the controller.
resources.limits.memory               500Mi                                  string
Memory limit of the controller.
resources.requests.cpu                 20m                                    string
CPU request of the controller.
resources.requests.memory              100Mi                                   string
Memory request of the controller.
logging_profile                       production                             string
Logging profile for controller. If set to development, use console logging with
full stack traces, else use JSON logging.
```

## 3. Locate the Tanzu Application Platform GUI URL.

When running on a full profile Tanzu Application Platform cluster, the default value of Tanzu Application Platform GUI URL is sufficient. You can edit this to match the externally available FQDN of Tanzu Application Platform GUI to display the entity URL in the externally accessible APIDescriptor status.

When installed in a run cluster or with a profile where Tanzu Application Platform GUI is not installed in the same cluster, you must set the `tap_gui_url` parameters correctly for successful entity registration with Tanzu Application Platform GUI.

You can locate the `tap_gui_url` by going to the view cluster with the Tanzu Application Platform GUI you want to register the entity with:



```
kubectl get secret tap-values -n tap-install -o jsonpath="{.data['tap-values\.yaml']}" | base64 -d | yq '.tap_gui.app_config.app.baseUrl'
```

- (Optional) VMware recommends creating `api-auto-registration-values.yaml`.

To overwrite the default values when installing the package, create a `api-auto-registration-values.yaml` file:

```
tap_gui_url: https://tap-gui.view-cluster.com
cluster_name: staging-us-east
auto_generate_cert: false
ca_cert_data: |
  -----BEGIN CERTIFICATE-----
  MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...
  -----END CERTIFICATE-----
sync_period: 2m
```

- Install the package using the Tanzu CLI:

```
tanzu package install api-auto-registration
--package-name apis.apps.tanzu.vmware.com
--namespace tap-install
--version $VERSION
--values-file api-auto-registration-values.yaml
```

- Verify the package installation by running:

```
tanzu package installed get api-auto-registration -n tap-install
```

Verify that `STATUS` is `Reconcile succeeded`:

```
kubectl get pods -n api-auto-registration
```

- Verify that applying an `APIDescriptor` resource to your cluster causes the `STATUS` showing `Ready`:

```
kubectl apply -f - <<EOF
apiVersion: apis.apps.tanzu.vmware.com/v1alpha1
kind: APIDescriptor
metadata:
  name: sample-api-descriptor-with-absolute-url
spec:
  type: openapi
  description: A sample APIDescriptor to validate package installation successful
  system: test-installation
  owner: test-installation
  location:
    path: "/api/v3/openapi.json"
    baseURL:
      url: https://petstore3.swagger.io
EOF
```

Verify that the `APIDescriptor` status shows `Ready`:

```
kubectl get apidescriptors
NAME                                STATUS
sample-api-descriptor-with-absolute-url  Ready

kubectl get apidescriptors -owide
NAME                                STATUS  TAP GUI ENTITY URL  API
SPEC URL
```

```
sample-api-descriptor-with-absolute-url    Ready    <url-to-the-entity>    <ur
l-to-the-api-spec>
```

If the status does not show **Ready**, you can inspect the reason with the detailed message shown by running:

```
kubectl get apidescriptor sample-api-descriptor-with-absolute-url -o jsonpath
='{.status.conditions[?(@.type=="Ready")].message}'
```

Verify that the entity is created in your Tanzu Application Platform GUI: [TAP-GUI-URL/catalog/default/api/sample-api-descriptor-with-absolute-url](#)

## Use API Auto Registration

This topic describes how you can use API Auto Registration.



### Note

The run profile requires you to [update the install values](#) before proceeding. For iterate and full profiles, the default values work but you might prefer to update them. For information about profiles, see [About Tanzu Application Platform profiles](#).

API Auto Registration requires the following:

1. A location exposing a dynamic or static API specification.
2. An APIDescriptor Custom Resource (CR) with that location created in the cluster.
3. (Optional) Configure Cross-Origin Resource Sharing (CORS) for OpenAPI specifications.

To generate OpenAPI Spec:

- [By creating a simple Spring Boot app](#)
- [By scaffolding a new project using App Accelerator Template](#)
- [In an existing Spring Boot project](#)

To create APIDescriptor Custom Resource:

- [Using Out Of The Box Supply Chains](#)
- [Using Custom Supply Chains](#)
- [Using other GitOps processes or Manually](#)

To configure:

- [CORS for viewing OpenAPI Spec in TAP GUI](#)

## Generate OpenAPI Spec

### Using a Spring Boot app with a REST service

You can use a [Spring Boot example app](#) built using [Building a RESTful Web Service guide](#), and has the [Springdoc](#) dependency.

Example of a workload using the Spring Boot app:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: simple-rest-app
```

```

labels:
  ...
  apis.apps.tanzu.vmware.com/register-api: "true"
spec:
  source:
    ...
  params:
    - name: api_descriptor
      value:
        type: openapi
        location:
          path: "/v3/api-docs"
        system: dev
        owner: team-a
        description: "A set of API endpoints."

```

## Using App Accelerator Template

If you are creating a new application exposing an API, you might use the [java-rest-service](#) App Accelerator template to get a pre-built app that includes a `workload.yaml` with a basic REST API. From your Tanzu Application Platform GUI Accelerators tab, search for the accelerator and scaffold it according to your needs.

## Using an existing Spring Boot project using springdoc

If you have an existing Spring Boot app that exposes an API, you can generate OpenAPI specifications using springdoc. See the [springdoc documentation](#)

After you have springdoc configured and an OpenAPI automatically generated, you can choose one of the three methods of creating the APIDescriptor custom resource. VMware recommends having your Spring Boot app to be managed using Workloads and the Out-Of-The-Box (OOTB) supply chain. See the [Use Out-Of-The-Box \(OOTB\) supply chains](#) for further instructions. Alternatively, if you want to use custom supply chains, see [Using Custom Supply Chains](#). Lastly, if you want to use a different GitOps process or manage the APIDescriptor CR manually, see the [Using other GitOps processes or Manually](#) section.

## Create APIDescriptor Custom Resource

### Use Out-Of-The-Box (OOTB) supply chains

All the Out-Of-The-Box (OOTB) supply chains are modified so that they can use API Auto Registration. If you want your workload to be auto registered, you must make modifications to your workload YAML:

1. Add the label `apis.apps.tanzu.vmware.com/register-api: "true"`.
2. Add a parameter of `type api_descriptor`:

```

params:
  - name: api_descriptor
    value:
      type: openapi # We currently support any of openapi, aysncapi, graphql, grpc
      location:
        path: "/v3/api-docs" # The path to the api documentation
        owner: team-petclinic # The team that owns this
        description: "A set of API endpoints to manage the resources within the petclinic app."

```

There are 2 different options for the location:

- The default supply chains use Knative to deploy your applications. In this event the only location information you must send is the path to the API documentation. The controller can figure out the base URL for you.
- You can hardcode the URL using the `baseURL` property. The controller uses a combination of this `baseURL` and your path to retrieve the YAML.

Example workload that exposes a Knative service:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: petclinic-knative
  labels:
    ...
    apis.apps.tanzu.vmware.com/register-api: "true"
spec:
  source:
    ...
  params:
    - name: api_descriptor
      value:
        type: openapi
        location:
          path: "/v3/api-docs"
        system: pet-clinics
        owner: team-petclinic
        description: "A set of API endpoints to manage the resources within the petclinic app."
```

Example of a workload with a hardcoded URL to the API documentation:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: petclinic-hard-coded
  labels:
    ...
    apis.apps.tanzu.vmware.com/register-api: "true"
spec:
  source:
    ...
  params:
    - name: api_descriptor
      value:
        type: openapi
        location:
          baseURL: http://petclinic-hard-coded.my-apps.tapdemo.vmware.com/
          path: "/v3/api-docs"
        owner: team-petclinic
        system: pet-clinics
        description: "A set of API endpoints to manage the resources within the petclinic app."
```

After the supply chain runs, it creates an `APIDescriptor` custom resource. This resource is what Tanzu Application Platform uses to auto register your API. See [APIDescriptor explained](#).

## Using Custom Supply Chains

If you are creating custom supply chains, you can still use API Auto Registration. To write a supply chain pipeline, use `ClusterConfigTemplate` by the name of `config-template` in your pipeline. To

write a custom task, verify how the template is written to read parameters, interpret baseURL from Knative Services, and construct APIDescriptor CRs.

In the Delivery pipeline, you must directly create an APIDescriptor custom resource. You must grant permissions to create the CR from the delivery pipeline.

For information about APIDescriptors, see [Key Concepts](#).

## Using other GitOps processes or Manually

Using your GitOps process, or manually, you must stamp out an APIDescriptor CR and apply it in the cluster you choose. Be sure specify all the required fields for an APIDescriptor CR to reconcile.

For information about APIDescriptors, see [Key Concepts](#).

## Additional configuration

### Setting up CORS for OpenAPI specifications

The agent, usually a browser, uses the [CORS](#) protocol to verify whether the current origin uses an API. To use the “Try it out” feature for OpenAPI specifications from the API Documentation plugin, you must configure CORS to allow successful requests.

Your API must be configured to allow CORS Requests from Tanzu Application Platform GUI. How you accomplish this varies based on the programming language and framework you are using. If you are using Spring, see [CORS support in spring framework](#).

At a high level, the Tanzu Application Platform GUI domain must be accepted as valid cross-origin by your API.

Verify the following:

- **Origins allowed** header: [Access-Control-Allow-Origin](#): A list of comma-separated values. This list must include your Tanzu Application Platform GUI host.
- **Methods allowed** header: [Access-Control-Allow-Method](#): Must allow the method used by your API. Also confirm that your API supports preflight requests, a valid response to the OPTIONS HTTP method.
- **Headers allowed** header: [Access-Control-Allow-Headers](#): If the API requires any header, you must include it in the API configuration or your authorization server.

## Troubleshoot API Auto Registration

This topic contains ways that you can troubleshoot API Auto Registration.

### Debug API Auto Registration

This topic includes commands for debugging or troubleshooting the APIDescriptor CR.

1. Get the details of APIDescriptor CR.

```
kubectl get apidescriptor <api-apidescriptor-name> -owide
```

2. Find the status of the APIDescriptor CR.

```
kubectl get apidescriptor <api-apidescriptor-name> -o jsonpath='{.status.conditions}'
```

3. Read logs from the [api-auto-registration](#) controller.

```
kubectl -n api-auto-registration logs deployment.apps/api-auto-registration-controller
```

4. Patch an APIDescriptor that is stuck in Deleting mode.

This might happen if the controller package is uninstalled before you clean up the APIDescriptor resources. You can reinstall the package and delete all the APIDescriptor resources first, or run the following command for each stuck APIDescriptor resource.

```
kubectl patch apidescriptor <api-apidescriptor-name> -p '{"metadata":{"finalizers":null}}' --type=merge
```



#### Note

If you manually remove the finalizers from the APIDescriptor resources, you can have stale API entities within Tanzu Application Platform GUI that you must manually deregister.

## APIDescriptor CRD shows message of `connection refused` but service is up and running

In Tanzu Application Platform v1.4 and later, if your workloads use ClusterIssuer for the TLS configuration or your API specifications location URL is secured using a custom CA, you might encounter the following message.

Your APIDescription CRD shows a status and message similar to:

```
Message:          Get "https://spring-petclinic.example.com/v3/api-docs": dial tcp 12.34.56.78:443: connect: connection refused
Reason:          FailedToRetrieve
Status:          False
Type:            APISpecResolved
Last Transition Time: 2022-11-28T09:59:13Z
```

This might be due to your workloads using a custom Ingress issuer. To solve this issue, either:

- Configure `ca_cert_data` following the instructions in [Configure CA Cert Data](#).
- Deactivate TLS by setting `shared.ingress_issuer: ""`. VMware discourages this method. Deactivating TLS reduces your ability to test plugin functionality and iterate quickly.

### Configure CA Cert Data

1. Obtain the PEM Encoded crt file for your ClusterIssuer or TLS setup . You use this to update the `api-auto-registration` package.
2. If you installed the API Auto Registration package through predefined profiles, you must update the `tap-values.yaml` and update the Tanzu Application Platform installation. Place the PEM encoded certificate into the `shared.ca_cert_data` key of the values file. See [Install your Tanzu Application Platform profile](#). Run the following command to update the package.

```
tanzu package installed update tap -p tap.tanzu.vmware.com -v <TAP_VERSION> --values-file tap-values.yaml -n tap-install
```

Where `TAP-VERSION` is the version of Tanzu Application Platform installed.

3. If you installed the API Auto Registration package as standalone, you must update the `api-auto-registration-values.yaml` and then update the package. Place the PEM encoded

certificate into the `ca_cert_data` key of the values file. Run to update the package.

```
tanzu package installed update api-auto-registration --version <API_AUTO_REGISTRATION_VERSION> --namespace tap-install --values-file api-auto-registration-values.yaml
```

You can find the available api-auto-registration versions by running:

```
tanzu package available list -n tap-install | grep 'API Auto Registration'
```

## APIDescriptor CRD shows message of `x509: certificate signed by unknown authority` but service is running

Your APIDescription CRD shows a status and message similar to:

```
Message:          Put "https://tap-gui.tap.my-cluster.tapdemo.vmware.com/api/catalog/immediate/entities": x509: certificate signed by unknown authority
Reason:           Error
Status:           False
Type:             Ready
Last Transition Time: 2022-11-28T09:59:13Z
```

This is the same issue as `connection refused` described earlier.

## Overview of API Validation and Scoring

API Validation and Scoring focuses on scanning and validating an OpenAPI specification. The API specification is generated from the [API Auto Registration](#). After an API is registered, the API specification goes through static scan analysis and is validated. Based on the validation, a scoring is provided to indicate the quality and health of the API specification as it relates to Documentation, OpenAPI best practices, and Security. The Validation Analysis card on the API overview page displays the summary of the scores. To learn more details about the scores, you can go to the detailed view by clicking the **MORE DETAILS** link.

API Validation and Scoring helps you to ensure your APIs are secure and robust, by providing feedback and recommendations early on in the software development life cycle. Based on the feedback and recommendations, you can edit your API specifications, improve the scores and the posture of your APIs and better understand how well the APIs are implemented.

## Overview of API Validation and Scoring

API Validation and Scoring focuses on scanning and validating an OpenAPI specification. The API specification is generated from the [API Auto Registration](#). After an API is registered, the API specification goes through static scan analysis and is validated. Based on the validation, a scoring is provided to indicate the quality and health of the API specification as it relates to Documentation, OpenAPI best practices, and Security. The Validation Analysis card on the API overview page displays the summary of the scores. To learn more details about the scores, you can go to the detailed view by clicking the **MORE DETAILS** link.

API Validation and Scoring helps you to ensure your APIs are secure and robust, by providing feedback and recommendations early on in the software development life cycle. Based on the feedback and recommendations, you can edit your API specifications, improve the scores and the posture of your APIs and better understand how well the APIs are implemented.

## Install API Validation and Scoring

This topic tells you how to install API Validation and Scoring from the Tanzu Application Platform (commonly known as TAP) package repository.

## Prerequisites

Before installing API Validation and Scoring, complete the following prerequisites:

1. Create a [Tanzu Network](#) account to download Tanzu Application Platform packages.
2. Provision Kubernetes cluster v1.22, v1.23 or v1.24 on Amazon Elastic Kubernetes Service.



### Note

The Installation of API scoring and validation package must be done on a new cluster without any existing Tanzu Application Platform installations.

3. [Install Tanzu CLI](#).
4. [Install kapp](#).
5. Install Kubernetes CLI. For more information, see [Install Tools](#) in the Kubernetes documentation.
6. [Deploy Cluster Essentials](#)

## Resource requirements

To deploy API Validation and Scoring package, your cluster must have at least:

- 5 nodes.
- 4 vCPUs available per node.
- 16 GB of RAM available per node.
- 100 GB of disk space available across all nodes.

## Relocate images to a registry

VMware recommends relocating the images from VMware Tanzu Network registry to your own container image registry before attempting installation. API Validation and Scoring depends on VMware Tanzu Network for continued operation. If you don't relocate the images, VMware Tanzu Network offers no uptime guarantees. The option to skip relocation is documented for evaluation and proof-of-concept only.

To relocate images from the VMware Tanzu Network registry to your registry:

1. Install Docker if it is not already installed.
2. Log in to your image registry by running:

```
docker login MY-REGISTRY
```

Where `MY-REGISTRY` is your own container registry.

3. Log in to the VMware Tanzu Network registry with your VMware Tanzu Network credentials by running:

```
docker login registry.tanzu.vmware.com
```

4. Set up environment variables for installation use by running:



```
export INSTALL_REGISTRY_USERNAME=MY-REGISTRY-USER
export INSTALL_REGISTRY_HOSTNAME=MY-REGISTRY
export APIX_VERSION=VERSION-NUMBER
export INSTALL_REPO=TARGET-REPOSITORY
```

Where:

- `MY-REGISTRY-USER` is the user with write access to `MY-REGISTRY`
- `MY-REGISTRY-PASSWORD` is the password for `MY-REGISTRY-USER`
- `MY-REGISTRY` is your own container registry.
- `VERSION-NUMBER` is your API Validation and Scoring package version. For example, `0.2.5`
- `TARGET-REPOSITORY` is your target repository, a folder/repository on `MY-REGISTRY` that serves as the location for the installation files for API Validation and Scoring.

### 5. Install the Carvel tool `imgpkg` CLI.

To query for the available `imgpkg` CLI versions on VMWare Tanzu Network Registry, run:

```
imgpkg tag list -i registry.tanzu.vmware.com/tanzu-application-platform/apix |
grep -v sha | sort -V
```

### 6. Relocate the images with the `imgpkg` CLI by running:

```
imgpkg copy -b registry.tanzu.vmware.com/tanzu-application-platform/apix:${APIX_VERSION} --to-repo ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/apix
```

## Add the API Validation and Scoring package repository

Tanzu CLI packages are available on repositories. Adding the API Validation and Scoring package repository makes the packages available for installation.

[Relocate images to a registry](#) is strongly recommended but not required for installation. If you skip this step, you can use the following values to replace the corresponding variables:

- `INSTALL_REGISTRY_HOSTNAME` is `registry.tanzu.vmware.com`
- `INSTALL_REPO` is `tanzu-application-platform`
- `INSTALL_REGISTRY_USERNAME` and `INSTALL_REGISTRY_PASSWORD` are the credentials to run `docker login registry.tanzu.vmware.com`
- `APIX_VERSION` is your API Validation and Scoring package version. For example, `0.2.5`

To add the API Validation and Scoring package repository to your cluster:

1. Create a namespace called `apix-install` for deploying API Validation and Scoring package by running:

```
kubectl create ns apix-install
```

This namespace keeps the objects grouped together logically.

2. Create a secret for adding the API Validation and Scoring package repository:

```
tanzu secret registry add tap-registry --username ${INSTALL_REGISTRY_USERNAME}
--password ${INSTALL_REGISTRY_PASSWORD} --server ${INSTALL_REGISTRY_HOSTNAME} -
-export-to-all-namespaces --yes --namespace apix-install
```

3. Add the API Validation and Scoring package repository to the cluster by running:

```
tanzu package repository add apix-repository \
--url ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/apix:${APIX_VERSION} \
--namespace apix-install
```

4. Verify the package installation by running:

```
tanzu package available list -n apix-install
```

If the package installed, expect to see the output that resembles the following:

| NAME                       | DISPLAY-NAME | SHORT_DESCRIPTION          | L |
|----------------------------|--------------|----------------------------|---|
| ATEST-VERSION              |              |                            |   |
| apix.apps.tanzu.vmware.com | apix         | apix.apps.tanzu.vmware.com |   |
| 0.2.5                      |              |                            |   |

5. Get the status of the API Validation and Scoring package repository by running:

```
tanzu package repository get apix-repository --namespace apix-install
```

For example:

```
~ % tanzu package repository get apix-repository --namespace apix-install
NAME:      apix-repository
VERSION:   796582
REPOSITORY: projects.registry.vmware.com/mazinger/apix
TAG:       0.2.5
STATUS:    Reconcile succeeded
REASON:
```

Verify the **STATUS** is **Reconcile succeeded**

## Install

Follow these steps to install the API Validation and Scoring package:

1. To overwrite the default values when installing the package, create the `apix-values.yaml` file:

```
apix:
  host: "HOST"
  backstage:
    host: "BACKSTAGE-HOST"
    port: "BACKSTAGE-PORT"
```

Where:

- o **HOST** is the hostname of the API Validation and Scoring GUI. It can be left empty "" to use the default value.
- o **BACKSTAGE-HOST** is the Tanzu Application Platform GUI or Backstage host that you want to point to. For example, <https://tap-gui.view-cluster.com>
- o **BACKSTAGE-PORT** is the Tanzu Application Platform GUI or Backstage port that you want to point to. For example, [443](#)

2. Install the API Validation and Scoring package using the Tanzu CLI by running:

```
tanzu package install apix -n apix-install -p apix.apps.tanzu.vmware.com -v ${APIX_VERSION} -f apix-values.yaml
```

3. Verify that **STATUS** is **Reconcile succeeded** by running:

```
tanzu package installed get apix -n apix-install
```

If your package successfully reconciled, expect to see the output that resembles the following::

```
NAME:                apix
PACKAGE-NAME:        apix.apps.tanzu.vmware.com
PACKAGE-VERSION:     0.2.5
STATUS:              Reconcile succeeded
CONDITIONS:          [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

## Uninstall

Uninstall the API Validation and Scoring package by running:

```
tanzu package installed delete apix -n apix-install
```

For example:

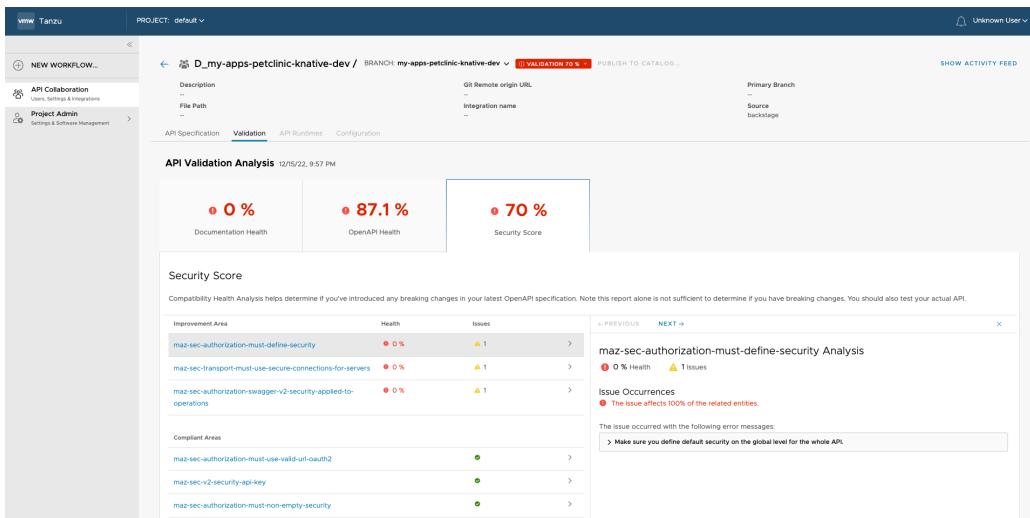
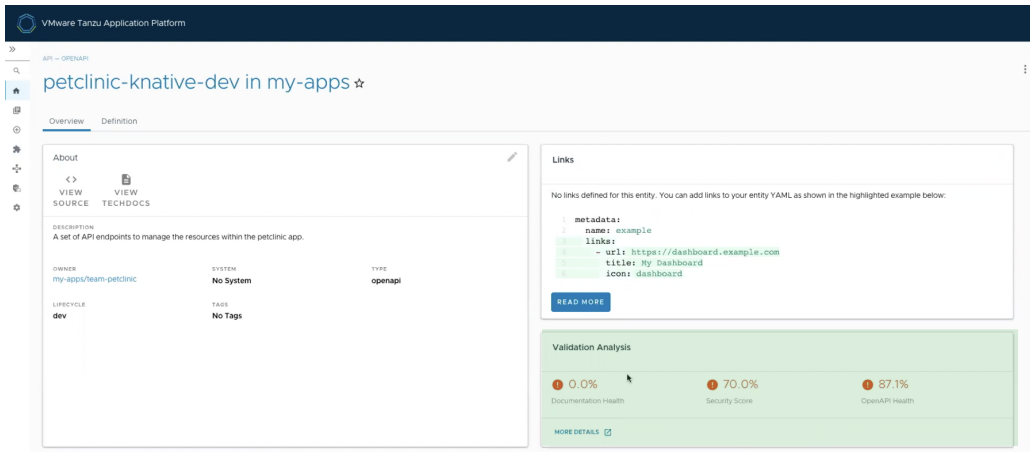
```
% tanzu package installed delete apix -n apix-install
Deleting installed package 'apix' in namespace 'apix-install'. Are you sure? [y/N]: y
Uninstalling package 'apix' from namespace 'apix-install'
Getting package install for 'apix'
Deleting package install 'apix' from namespace 'apix-install'
'PackageInstall' resource deletion status: Deleting
Deleting admin role 'apix-apix-install-cluster-role'
Deleting role binding 'apix-apix-install-cluster-rolebinding'
Deleting secret 'apix-apix-install-values'
Deleting service account 'apix-apix-install-sa'
Uninstalled package 'apix' from namespace 'apix-install'
```

## Use API Validation and Scoring to score your auto-registered API

## Use API Validation and Scoring to score your auto-registered API

This topic tells you how an [Auto Registered API](#) is scored:

- See [Use API Auto Registration](#) to deploy the workload.
- Navigate to the Tanzu Application Platform GUI to view the API .
- The **Overview** tab of your API in Tanzu Application Platform GUI shows the API scores.
- To view more details about the Validation Analysis and the required improvements for your API, click **MORE DETAILS**.



## Overview of API portal for VMware Tanzu

You can use API portal for VMware Tanzu to find APIs you can use in your own applications. You can view detailed API documentation and try out an API to meet your needs. API portal assembles its dashboard and detailed API documentation views by ingesting OpenAPI documentation from the source URLs. An API portal operator can add any number of OpenAPI source URLs in a single instance.

## Getting started

To install the package without the predefined profiles of Tanzu Application Platform, see [Install API portal](#).

For information about API portal for VMware Tanzu, see [API portal for VMware Tanzu](#).

For information about configuring the package, see [Configuring API portal for VMware Tanzu on Kubernetes](#).

API portal for VMware Tanzu supports:

- Authentication through Single Sign-On (SSO)
- API keys configuration and management
- Secure communication using TLS

## Overview of API portal for VMware Tanzu

You can use API portal for VMware Tanzu to find APIs you can use in your own applications. You can view detailed API documentation and try out an API to meet your needs. API portal assembles its dashboard and detailed API documentation views by ingesting OpenAPI documentation from the source URLs. An API portal operator can add any number of OpenAPI source URLs in a single instance.

## Getting started

To install the package without the predefined profiles of Tanzu Application Platform, see [Install API portal](#).

For information about API portal for VMware Tanzu, see [API portal for VMware Tanzu](#).

For information about configuring the package, see [Configuring API portal for VMware Tanzu on Kubernetes](#).

API portal for VMware Tanzu supports:

- Authentication through Single Sign-On (SSO)
- API keys configuration and management
- Secure communication using TLS

## Install API portal for VMware Tanzu

This topic tells you how to install and update Tanzu API portal for VMware Tanzu from the Tanzu Application Platform (commonly known as TAP) package repository.



### Note

Follow the steps in this topic if you do not want to use a profile to install API portal. For more information about profiles, see [Components and installation profiles](#).

## Prerequisites

Before installing API portal:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see [Prerequisites](#).

## Install

To install the API portal package:

1. Confirm what versions of API portal are available to install by running:

```
tanzu package available list -n tap-install api-portal.tanzu.vmware.com
```

For example:

```
$ tanzu package available list api-portal.tanzu.vmware.com --namespace tap-inst
all
- Retrieving package versions for api-portal.tanzu.vmware.com...
NAME                                VERSION  RELEASED-AT
api-portal.tanzu.vmware.com         1.0.3    2021-10-13T00:00:00Z
```

2. (Optional) Gather values schema.

```
tanzu package available get api-portal.tanzu.vmware.com/VERSION-NUMBER --values
-schema --namespace tap-install
```

Where `VERSION-NUMBER` is the version of the package listed earlier.

For example:

```
$ tanzu package available get api-portal.tanzu.vmware.com/1.0.3 --values-schema
--namespace tap-install

Retrieving package details for api-portal.tanzu.vmware.com/1.0.3...
```

3. (Optional) VMware recommends creating `api-portal-values.yaml`.

To overwrite the default values when installing the package, create a `api-portal-values.yaml` file following the values schema.

4. Install API portal by running:

```
tanzu package install api-portal -n tap-install -p api-portal.tanzu.vmware.com
-v $VERSION --values-file api-portal-values.yaml
```

For example:

```
$ tanzu package install api-portal -n tap-install -p api-portal.tanzu.vmware.co
m -v 1.0.3 --values-file api-portal-values.yaml

/ Installing package 'api-portal.tanzu.vmware.com'
| Getting namespace 'api-portal'
| Getting package metadata for 'api-portal.tanzu.vmware.com'
| Creating service account 'api-portal-api-portal-sa'
| Creating cluster admin role 'api-portal-api-portal-cluster-role'
| Creating cluster role binding 'api-portal-api-portal-cluster-rolebinding'
/ Creating package resource
- Package install status: Reconciling

Added installed package 'api-portal' in namespace 'tap-install'
```

5. Verify the package installation by running:

```
tanzu package installed get api-portal -n tap-install
```

Verify that `STATUS` is `Reconcile succeeded`:

```
kubectl get pods -n api-portal
```

## Update the installation values for the `api-portal` package

To update the installation values for the `api-portal` package:

1. To overwrite the default values, create new values, or update the existing values, you need an `api-portal-values.yaml` file. If you do not already have an existing values file, you can extract the existing values by running:

```
tanzu package installed get api-portal -n tap-install -f api-portal-values.yaml
```

For reference, you can view the schema of the package:

```
tanzu package available get apis.apps.tanzu.vmware.com/VERSION-NUMBER --values-schema --namespace tap-install
```

Where `VERSION-NUMBER` is the version of the package listed in the earlier step.

For example:

```
tanzu package available get api-portal.tanzu.vmware.com/1.2.5 --values-schema --namespace tap-install
```

2. Now, update the package using the Tanzu CLI:

```
tanzu package installed update api-auto-registration
--package-name apis.apps.tanzu.vmware.com
--namespace tap-install
--version $VERSION
--values-file api-portal-values.yaml
```

3. If you installed the API portal package as part of Tanzu Application Platform, you must update the `tap-values.yaml` and update the installation of Tanzu Application Platform. See [Install your Tanzu Application Platform profile](#).
4. If you installed the API portal package as part of Tanzu Application Platform, you must update the `tap-values.yaml` and update the TAP installation. See [Install your Tanzu Application Platform profile](#).



#### Note

You can update API portal as part of upgrading Tanzu Application Platform. See [Upgrading Tanzu Application Platform](#).

## Overview of Application Accelerator

This topic tells you about the Application Accelerator component and architecture in Tanzu Application Platform (commonly known as TAP).

### Overview

Application Accelerator for VMware Tanzu helps you bootstrap developing your applications and deploying them in a discoverable and repeatable way.

Enterprise Architects author and publish accelerator projects that provide developers and operators in their organization ready-made, conforming code and configurations.

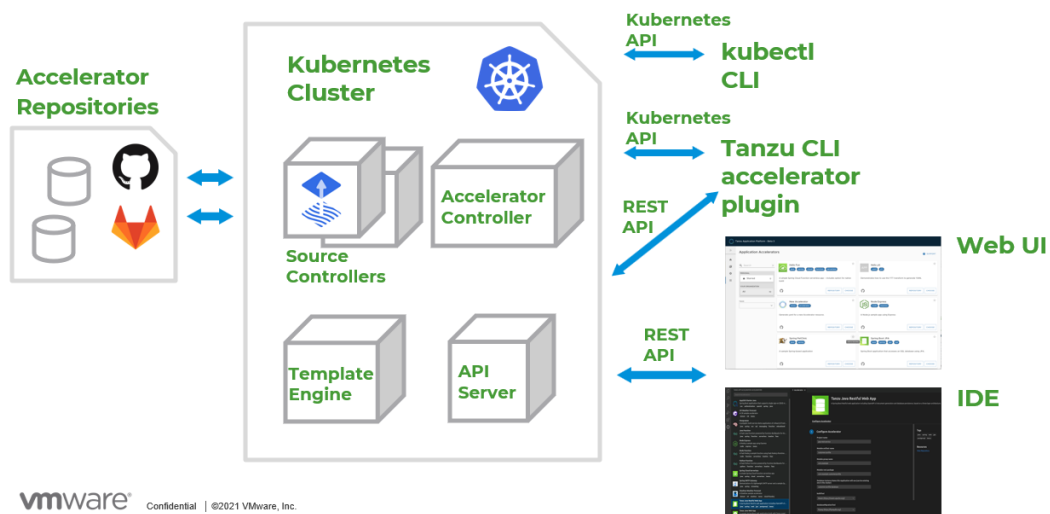
Published accelerator projects are maintained in Git repositories. You can then use Application Accelerator to create new projects based on those accelerator projects.

The Application Accelerator user interface(UI) enables you to discover available accelerators, configure them, and generate new projects to download.

### Architecture

The following diagram of Accelerator components illustrates the Application Accelerator architecture.

## Accelerator Components



## How does Application Accelerator work?

Application Accelerator allows you to generate new projects from files in Git repositories. An `accelerator.yaml` file in the repository declares input options for the accelerator. This file also contains instructions for processing the files when you generate a new project.

Accelerator custom resources (CRs) control which repositories appear in both the Tanzu Application Platform Application Accelerator UI and in the Application Accelerator extension for VS Code. You can maintain CRs by using Kubernetes tools such as `kubectl` or by using the Tanzu CLI accelerator commands. The Accelerator controller reconciles the CRs with a Flux2 Source Controller to fetch files from GitHub or GitLab.

The Application Accelerator web UI gives you a searchable list of accelerators to choose from. After you select an accelerator, the UI presents text boxes for the options that are defined within the `accelerator.yaml` of the selected accelerator.

Application Accelerator sends the input values to the Accelerator Engine for processing. (Optional) The user can choose to have a new Git repository created as part of the project creation process. The Engine then returns the project in a ZIP file. If the project was generated using the Application Accelerator extension for VS Code, the project automatically be extracted to the directory location of your choice on your local machine. You can then open the project in your favorite integrated development environment (IDE) to develop further.

## Next steps

Learn more about:

- [Creating Accelerators](#)

## Overview of Application Accelerator

This topic tells you about the Application Accelerator component and architecture in Tanzu Application Platform (commonly known as TAP).

## Overview



Application Accelerator for VMware Tanzu helps you bootstrap developing your applications and deploying them in a discoverable and repeatable way.

Enterprise Architects author and publish accelerator projects that provide developers and operators in their organization ready-made, conforming code and configurations.

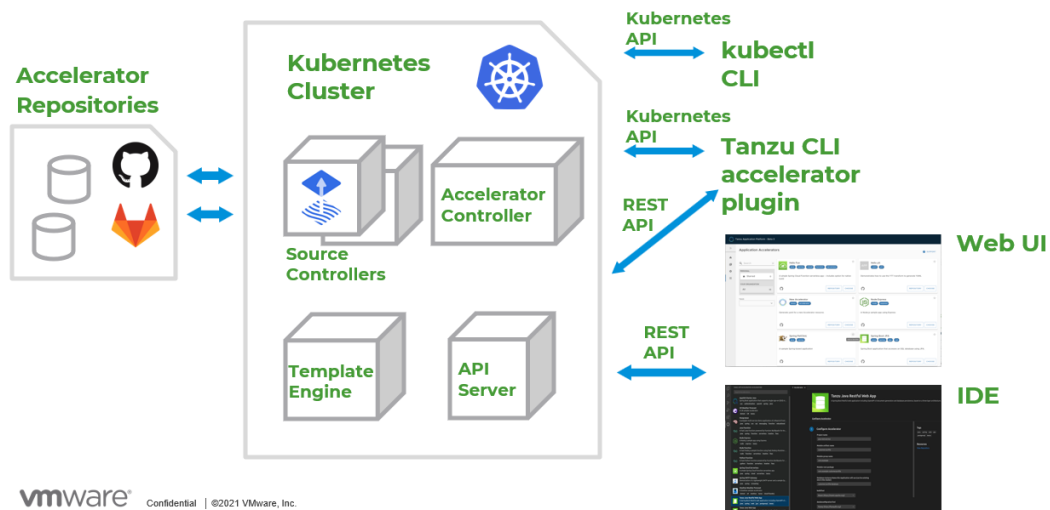
Published accelerator projects are maintained in Git repositories. You can then use Application Accelerator to create new projects based on those accelerator projects.

The Application Accelerator user interface(UI) enables you to discover available accelerators, configure them, and generate new projects to download.

## Architecture

The following diagram of Accelerator components illustrates the Application Accelerator architecture.

### Accelerator Components



## How does Application Accelerator work?

Application Accelerator allows you to generate new projects from files in Git repositories. An `accelerator.yaml` file in the repository declares input options for the accelerator. This file also contains instructions for processing the files when you generate a new project.

Accelerator custom resources (CRs) control which repositories appear in both the Tanzu Application Platform Application Accelerator UI and in the Application Accelerator extension for VS Code. You can maintain CRs by using Kubernetes tools such as `kubectl` or by using the Tanzu CLI accelerator commands. The Accelerator controller reconciles the CRs with a Flux2 Source Controller to fetch files from GitHub or GitLab.

The Application Accelerator web UI gives you a searchable list of accelerators to choose from. After you select an accelerator, the UI presents text boxes for the options that are defined within the `accelerator.yaml` of the selected accelerator.

Application Accelerator sends the input values to the Accelerator Engine for processing. (Optional) The user can choose to have a new Git repository created as part of the project creation process. The Engine then returns the project in a ZIP file. If the project was generated using the Application Accelerator extension for VS Code, the project automatically be extracted to the directory location of your choice on your local machine. You can then open the project in your favorite integrated development environment (IDE) to develop further.

## Next steps

Learn more about:

- [Creating Accelerators](#)

## Install Application Accelerator

This topic tells you how to install Application Accelerator from the Tanzu Application Platform (commonly known as TAP) package repository.



### Note

Follow the steps in this topic if you do not want to use a profile to install Application Accelerator. For more information about profiles, see [About Tanzu Application Platform components and profiles](#).

## Prerequisites

Before installing Application Accelerator:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see [Prerequisites](#).
- Install Flux SourceController on the cluster. See [Install Flux CD Source Controller](#).
- Install Source Controller on the cluster. See [Install Source Controller](#).

## Configure properties and resource use

When you install the Application Accelerator, you can configure the following optional properties from within your `.yaml` configuration file:

| Property                      | Default                                                            | Description                                                                                   |
|-------------------------------|--------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|
| registry.secret_ref           | registry.tanzu<.vmware.com                                         | The secret used for accessing the registry where the App-Accelerator images are located       |
| server.service_type           | ClusterIP                                                          | The service type for the acc-ui-server service including LoadBalancer, NodePort, or ClusterIP |
| server.watched_namespace      | accelerator-system                                                 | The namespace the server watches for accelerator resources                                    |
| server.engine_invocation_url  | http://acc-engine.accelerator-system.svc.cluster.local/invocations | The URL to use for invoking the accelerator engine                                            |
| engine.service_type           | ClusterIP                                                          | The service type for the acc-engine service including LoadBalancer, NodePort, or ClusterIP    |
| engine.max_direct_memory_size | 32M                                                                | The maximum size for the Java <code>-XX:MaxDirectMemorySize</code> 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                                           |

| Property                                       | Default              | Description                                                                                                                     |
|------------------------------------------------|----------------------|---------------------------------------------------------------------------------------------------------------------------------|
| domain                                         | tap.example.com      | Top-level domain to use for ingress configuration, default is <code>shared.ingress_domain</code>                                |
| tls.secret_name                                | tls                  | The name of the secret                                                                                                          |
| tls.namespace                                  | tanzu-system-ingress | The namespace for the secret                                                                                                    |
| telemetry.retain_invocation_events_for_no_days | 30                   | The number of days to retain recorded invocation events resources                                                               |
| telemetry.record_invocation_events             | true                 | The system records each engine invocation when generating files for an accelerator?                                             |
| git_credentials.secret_name                    | git-credentials      | The name to use for the secret storing Git credentials for accelerators                                                         |
| git_credentials.username                       | null                 | The user name to use in secret storing Git credentials for accelerators                                                         |
| git_credentials.password                       | null                 | The password to use in secret storing Git credentials for accelerators                                                          |
| git_credentials.ca_file                        | null                 | The CA certificate data to use in secret storing Git credentials for accelerators                                               |
| managed_resources.enable                       | false                | Whether to enable the App used to control managed accelerator resources                                                         |
| managed_resources.git_url                      | none                 | Required if managed_resources are enabled. Git repository URL containing manifests for managed accelerator resources            |
| managed_resources.git_ref                      | origin/main          | Required if managed_resources are enabled. Git ref to use for repository containing manifests for managed accelerator resources |
| managed_resources.git_sub_path                 | null                 | Git subPath to use for repository containing manifests for managed accelerator resources                                        |
| managed_resources.git_secret_ref               | git-credentials      | Secret name to use for repository containing manifests for managed accelerator resources                                        |

VMware recommends that you do not override the default setting for `registry.secret_ref`, `server.engine_invocation_url`, or `engine.service_type`. These properties are only used to configure non-standard installations.

The following table is the resource use configurations for the components of Application Accelerator.

| Component      | Resource requests         | Resource limits           |
|----------------|---------------------------|---------------------------|
| acc-controller | CPU: 100m<br>memory: 20Mi | CPU: 100m<br>memory: 30Mi |
| acc-server     | CPU: 100m<br>memory: 20Mi | CPU: 100m<br>memory: 30Mi |
| acc-engine     | CPU: 500m<br>memory: 1Gi  | CPU: 500m<br>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  1.4.0    2022-12-08 12:00:00 -0500 EST
```

2. (Optional) To make changes to the default installation settings, run:

```
tanzu package available get "accelerator.apps.tanzu.vmware.com/${ACCELERATOR_VERSION_NUMBER}" --values-schema --namespace tap-install
```

Where `ACCELERATOR-VERSION-NUMBER` is the version of the Accelerator package that was listed earlier.

For example:

```
tanzu package available get accelerator.apps.tanzu.vmware.com/1.4.0 --values-schema --namespace tap-install
```

For more information about values schema options, see the properties listed 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.

4. (Optional) For clusters that do not support the `LoadBalancer` service type, override the default value for `server.service_type`. For example:

```
server:
  service_type: "ClusterIP"
  watched_namespace: "accelerator-system"
samples:
  include: true
```

5. Install the package by running:

```
tanzu package install app-accelerator -p accelerator.apps.tanzu.vmware.com -v ${ACCELERATOR_VERSION_NUMBER} -n tap-install -f app-accelerator-values.yaml
```

Where `ACCELERATOR-VERSION-NUMBER` is the version of the Application Accelerator package included with the Tanzu Application Platform installation.

For example:

```
$ tanzu package install app-accelerator -p accelerator.apps.tanzu.vmware.com -v 1.4.0 -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'
```

- Verify the package install by running:

```
tanzu package installed get app-accelerator -n tap-install
```

For example:

```
$ tanzu package installed get app-accelerator -n tap-install
| Retrieving installation details for cc...
NAME:                app-accelerator
PACKAGE-NAME:        accelerator.apps.tanzu.vmware.com
PACKAGE-VERSION:     1.4.0
STATUS:              Reconcile succeeded
CONDITIONS:          [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`.

- To see the IP address for the Application Accelerator API when the `server.service_type` is set to `LoadBalancer`, run:

```
kubectl get service -n accelerator-system
```

This lists an external IP address for use with the `--server-url` Tanzu CLI flag for the Accelerator plug-in `generate & generate-from-local` command.

## Troubleshooting

Depending on the error output, there are some actions that you can take.

### Verify installed packages

The package might be already installed. Verify this by running:

```
tanzu package installed list -n tap-install
```

Look for any package called `accelerator.apps.tanzu.vmware.com`.

### Look at resource events

The error might be within the custom resources such as `accelerator`, `Git repository`, `fragment`, and so on. These errors are checked by using Kubernetes command line interface tool (`kubectl`).

Here is an example using the custom resource `accelerator`:

```
kubectl get acc -n accelerator-system.
```

It displays the output:

| NAME                | READY | REASON | AGE  |
|---------------------|-------|--------|------|
| appsso-starter-java | True  | Ready  | 5h2m |
| where-for-dinner    | True  | Ready  | 5h2m |
| java-function       | True  | Ready  | 5h2m |
| java-rest-service   | True  | Ready  | 5h2m |
| java-server-side-ui | True  | Ready  | 5h2m |

|                          |       |           |      |
|--------------------------|-------|-----------|------|
| node-express             | True  | Ready     | 5h2m |
| node-function            | False | Not-Ready | 5h2m |
| python-function          | True  | Ready     | 5h2m |
| spring-cloud-serverless  | True  | Ready     | 5h2m |
| spring-smtp-gateway      | True  | Ready     | 5h2m |
| tanzu-java-web-app       | True  | Ready     | 5h2m |
| tap-initialize           | True  | Ready     | 5h2m |
| weatherforecast-csharp   | True  | Ready     | 5h2m |
| weatherforecast-steeltoe | True  | Ready     | 5h2m |

To verify the error event, run:

```
kubectl get acc node-function -n accelerator-system -o yaml
```

You can then look at the event section for more information about the error.

## Configure Application Accelerator

This topic tells you about advanced configuration options available for Application Accelerator in Tanzu Application Platform (commonly known as TAP). This includes configuring Git-Ops style deployments of accelerators and configurations for use with non-public repositories and in air-gapped environments.

### Overview

Accelerators are created either using the Tanzu CLI or by applying a YAML manifest using `kubectl`. Another option is [Using a Git-Ops style configuration for deploying a set of managed accelerators](#).

Application Accelerator pulls content from accelerator source repositories using either the “Flux SourceController” or the “Tanzu Application Platform Source Controller” components. If the repository used is accessible anonymously from a public server, you do not have to configure anything additional. Otherwise, provide authentication as explained in [Using non-public repositories](#). There are also options for making these configurations easier explained in [Configuring tap-values.yaml with Git credentials secret](#)

## Using a Git-Ops style configuration for deploying a set of managed accelerators

To enable a Git-Ops style of managing resources used for deploying accelerators, there is a new set of properties for the Application Accelerator configuration. The resources are managed using a Carvel `kapp-controller` App in the `accelerator-system` namespace that watches a Git repository containing the manifests for the accelerators. This means that you can make changes to the manifests, or to the accelerators they point to, and the changes are reconciled and reflected in the deployed resources.

You can specify the following accelerator configuration properties when installing the Application Accelerator. The same properties are provided in the `accelerator` section of the `tap-values.yaml` file:

```
accelerator:
  managed_resources:
    enable: true
  git:
    url: GIT-REPO-URL
    ref: origin/main
    sub_path: null
    secret_ref: git-credentials
```

Where:

- `GIT-REPO-URL` is the URL of a Git repository that contains manifest YAML files for the accelerators that you want to have managed. The URL must start with `https://` or `git@`. You can specify a `sub_path` if necessary and also a `secret_ref` if the repository requires authentication. If not needed, then leave these additional properties out.

For more information, see [Configure tap-values.yaml with Git credentials secret](#) and [Creating a manifest with multiple accelerators and fragments](#) in this topic.

## Functional and Organizational Considerations

Any accelerator manifest that is defined under the `GIT-REPO-URL` and optional `sub_path` is selected by the `kapp-controller` app. If there are multiple manifests at the defined `GIT-REPO-URL`, they are all watched for changes and displayed to the user as a merged catalog.

For example: if you have two manifests containing multiple accelerator or fragment definitions, `manifest-1.yaml`, and `manifest-2.yaml`, on the same path in the organizational considerations. The resulting catalog is (`manifest-1.yaml` + `manifest-2.yaml`).

## Examples for creating accelerators

### A minimal example for creating an accelerator

A minimal example might look like the following manifest:

`spring-cloud-serverless.yaml`

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: spring-cloud-serverless
spec:
  git:
    url: https://github.com/vmware-tanzu/application-accelerator-samples
    subPath: spring-cloud-serverless
    ref:
      branch: main
```

This example creates an accelerator named `spring-cloud-serverless`. The `displayName`, `description`, `iconUrl`, and `tags` text boxes are populated based on the content under the `accelerator` key in the `accelerator.yaml` file found in the `main` branch of the Git repository at [Application Accelerator Samples](#) under the sub-path `spring-cloud-serverless`. For example:

`accelerator.yaml`

```
accelerator:
  displayName: Spring Cloud Serverless
  description: A simple Spring Cloud Function serverless app
  iconUrl: https://raw.githubusercontent.com/simple-starters/icons/master/icon-cloud.png
  tags:
  - java
  - spring
  - cloud
  - function
  - serverless
  - tanzu
  ...
```

To create this accelerator with `kubectl`, run:

```
kubectl apply --namespace --accelerator-system --filename spring-cloud-serverless.yaml
```

Or, you can use the Tanzu CLI and run:

```
tanzu accelerator create spring-cloud-serverless --git-repo https://github.com/vmware-tanzu/application-accelerator-samples.git --git-branch main --git-sub-path spring-cloud-serverless
```

## An example for creating an accelerator with customized properties

You can specify the `displayName`, `description`, `iconUrl`, and `tags` text boxes and this overrides any values provided in the accelerator's Git repository. The following example explicitly sets those text boxes and the `ignore` text box:

`my-spring-cloud-serverless.yaml`

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: my-spring-cloud-serverless
spec:
  displayName: My Spring Cloud Serverless
  description: My own Spring Cloud Function serverless app
  iconUrl: https://raw.githubusercontent.com/simple-starters/icons/master/icon-cloud.png
  tags:
    - spring
    - cloud
    - function
    - serverless
  git:
    ignore: ".git/, bin/"
    url: https://github.com/vmware-tanzu/application-accelerator-samples
    subPath: spring-cloud-serverless
    ref:
      branch: test
```

To create this accelerator with `kubectl`, run:

```
kubectl apply --namespace --accelerator-system --filename my-spring-cloud-serverless.yaml
```

To use the Tanzu CLI, run:

```
tanzu accelerator create my-spring-cloud-serverless --git-repo https://github.com/vmware-tanzu/application-accelerator-samples --git-branch main --git-sub-path spring-cloud-serverless \
  --description "My own Spring Cloud Function serverless app" \
  --display-name "My Spring Cloud Serverless" \
  --icon-url https://raw.githubusercontent.com/simple-starters/icons/master/icon-cloud.png \
  --tags "spring,cloud,function,serverless"
```



### Note

It is not possible to provide the `git.ignore` option with the Tanzu CLI.

## Creating a manifest with multiple accelerators and fragments

You might have a manifest that contains multiple accelerators or fragments. For example:

`accelerator-collection.yaml`



```

---
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: spring-cloud-serverless
spec:
  git:
    url: https://github.com/vmware-tanzu/application-accelerator-samples
    subPath: spring-cloud-serverless
    ref:
      branch: main
---
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: tanzu-java-web-app
spec:
  git:
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
    subPath: tanzu-java-web-app
    ref:
      branch: main

```

For a larger example of this, see [Sample Accelerators Main](#). Optionally, use this to create an initial catalog of accelerators and fragments during a fresh Application Accelerator install.

## Configure `tap-values.yaml` with Git credentials secret



### Note

For how to create a new OAuth Token for optional Git repository creation, see [Create an Application Accelerator Git repository during project creation](#).

When deploying accelerators using Git repositories that requires authentication or are installed with custom CA certificates, you must provide some additional authentication values in a secret. The examples in the next section provide more details. This section describes how to configure a Git credentials secret that is used in later Git-based examples.

You can specify the following accelerator configuration properties when installing Application Accelerator. The same properties are provided in the `accelerator` section of the `tap-values.yaml` file:

```

accelerator:
  git_credentials:
    secret_name: git-credentials
    username: GIT-USER-NAME
    password: GIT-CREDENTIALS
    ca_file: CUSTOM-CA-CERT

```

Where:

- `GIT-USER-NAME` is the user name for authenticating with the Git repository.
- `GIT-CREDENTIALS` is the password or access token used for authenticating with the Git repository. VMware recommends using an access token for this.
- `CUSTOM-CA-CERT` is the certificate data needed when accessing the Git repository.

This is an example of this part of a `tap-values.yaml` configuration:

```

accelerator:
  git_credentials:
    secret_name: git-credentials
    username: testuser
    password: s3cret
    ca_file: |
      -----BEGIN CERTIFICATE-----
      .
      .
      . < certificate data >
      .
      .
      -----END CERTIFICATE-----

```

You can specify the custom CA certificate data using the shared config value `shared.ca_cert_data` and it propagates to all components that can make use of it, including the App Accelerator configuration. The example earlier produces an output such as this using the shared value:

```

shared:
  ca_cert_data: |
    -----BEGIN CERTIFICATE-----
    .
    .
    . < certificate data >
    .
    .
    -----END CERTIFICATE-----

accelerator:
  git_credentials:
    secret_name: git-credentials
    username: testuser
    password: s3cret

```

## Using non-public repositories

For GitHub repositories that aren't accessible anonymously, you must provide credentials in a Secret.

- For HTTPS repositories the secret must contain user name and password fields. The password field can contain a personal access token instead of an actual password. For more information, see [Fluxcd/source-controller basic access authentication](#).
- For HTTPS with self-signed certificates, you can add a `.data.caFile` value to the secret created for HTTPS authentication. For more information, see [fluxcd/source-controller HTTPS Certificate Authority](#).
- For SSH repositories, the secret must contain identity, identity.pub, and known\_hosts text boxes. For more information, see [fluxcd/source-controller SSH authentication](#).
- For Image repositories that aren't publicly available, an image pull secret might be provided. For more information, see [Kubernetes documentation on using imagePullSecrets](#).

## Examples for a private Git repository

### Example using http credentials

To create an accelerator using a private Git repository, first create a secret with the HTTP credentials.

**Note**

For better security, use an access token as the password.

```
kubectl create secret generic https-credentials \
  --namespace accelerator-system \
  --from-literal=username=<user> \
  --from-literal=password=<access-token>
```

Verify that your secret was created by running:

```
kubectl get secret --namespace accelerator-system https-credentials -o yaml
```

The output is similar to:

```
apiVersion: v1
kind: Secret
metadata:
  name: https-credentials
  namespace: accelerator-system
type: Opaque
data:
  username: <BASE64>
  password: <BASE64>
```

After you created and verified the secret, you can create the accelerator by using the `spec.git.secretRef.name` property:

`private-acc.yaml`

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: private-acc
spec:
  displayName: private
  description: Accelerator using a private repository
  git:
    url: REPOSITORY-URL
    ref:
      branch: main
    secretRef:
      name: https-credentials
```

For https credentials, the `REPOSITORY-URL` must use `https://` as the URL scheme.

If you are using the Tanzu CLI, add the `--secret-ref` flag to your `tanzu accelerator create` command and provide the name of the secret for that flag.

**Example using http credentials with self-signed certificate**

To create an accelerator using a private Git repository with a self-signed certificate, create a secret with the HTTP credentials and the certificate.

**Note**

For better security, use an access token as the password.

```
kubectl create secret generic https-ca-credentials \
  --namespace accelerator-system \
  --from-literal=username=<user> \
  --from-literal=password=<access-token> \
  --from-file=caFile=<path-to-CA-file>
```

Verify that your secret was created by running:

```
kubectl get secret --namespace accelerator-system https-ca-credentials -o yaml
```

The output is similar to:

```
apiVersion: v1
kind: Secret
metadata:
  name: https-ca-credentials
  namespace: accelerator-system
type: Opaque
data:
  username: <BASE64>
  password: <BASE64>
  caFile: <BASE64>
```

After you have the secret created, you can create the accelerator by using the `spec.git.secretRef.name` property:

`private-acc.yaml`

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: private-acc
spec:
  displayName: private
  description: Accelerator using a private repository
  git:
    url: REPOSITORY-URL
    ref:
      branch: main
    secretRef:
      name: https-ca-credentials
```



### Important

For https credentials, the `REPOSITORY-URL` must use `https://` as the URL scheme.

If you are using the Tanzu CLI, add the `--secret-ref` flag to your `tanzu accelerator create` command and provide the name of the secret for that flag.

### Example using SSH credentials

To create an accelerator using a private Git repository, create a secret with the SSH credentials such as this example:

```
ssh-keygen -q -N "" -f ./identity
ssh-keyscan github.com > ./known_hosts
kubectl create secret generic ssh-credentials \
  --namespace accelerator-system \
  --from-file=./identity \
```

```
--from-file=./identity.pub \
--from-file=./known_hosts
```

If you have a key file already created, skip the `ssh-keygen` and `ssh-keyscan` steps and replace the values for the `kubectl create secret` command. Such as:

- `--from-file=identity=<path to your identity file>`
- `--from-file=identity.pub=<path to your identity.pub file>`
- `--from-file=known_hosts=<path to your know_hosts file>`

Verify that your secret was created by running:

```
kubectl get secret --namespace accelerator-system ssh-credentials -o yaml
```

The output is similar to :

```
apiVersion: v1
kind: Secret
metadata:
  name: ssh-credentials
  namespace: accelerator-system
type: Opaque
data:
  identity:<BASE64>
  identity.pub: <BASE64>
  known_hosts: <BASE64>
```

To use this secret when creating an accelerator, provide the secret name in the `spec.git.secretRef.name` property:

`private-acc-ssh.yaml`

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: private-acc
spec:
  displayName: private
  description: Accelerator using a private repository
  git:
    url: REPOSITORY-URL
    ref:
      branch: main
    secretRef:
      name: ssh-credentials
```

When using SSH credentials, the `REPOSITORY-URL` must include the user name as part of the URL. For example: `ssh://user@example.com:22/repository.git`. For more information, see [Flux documentation](#).

If you are using the Tanzu CLI, add the `--secret-ref` flag to your `tanzu accelerator create` command and provide the name of the secret for that flag.

## Examples for a private source-image repository

If your registry uses a self-signed certificate then you must add the CA certificate data to the configuration for the “Tanzu Application Platform Source Controller” component. Add it under `source_controller.ca_cert_data` in your `tap-values.yaml` file that is used during installation.

`tap-values.yaml`

```
source_controller:
  ca_cert_data: |-
    -----BEGIN CERTIFICATE-----
    .
    .
    . < certificate data >
    .
    .
    -----END CERTIFICATE-----
```

### Example using image-pull credentials

To create an accelerator using a private source-image repository, create a secret with the image-pull credentials:

```
create secret generic registry-credentials \
  --namespace accelerator-system \
  --from-literal=username=<user> \
  --from-literal=password=<password>
```

Verify that your secret was created by running:

```
kubectl get secret --namespace accelerator-system registry-credentials -o yaml
```

The output is similar to:

```
apiVersion: v1
kind: Secret
metadata:
  name: registry-credentials
  namespace: accelerator-system
type: Opaque
data:
  username: <BASE64>
  password: <BASE64>
```

After you have the secret created, you can create the accelerator by using the `spec.git.secretRef.name` property:

`private-acc.yaml`

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: private-acc
spec:
  displayName: private
  description: Accelerator using a private repository
  source:
    image: "registry.example.com/test/private-acc-src:latest"
    imagePullSecrets:
      - name: registry-credentials
```

If you are using the Tanzu CLI, add the `--secret-ref` flag to your `tanzu accelerator create` command and provide the name of the secret for that flag.

## Configure ingress timeouts when some accelerators take longer to generate

If Tanzu Application Platform is configured to use an ingress for Tanzu Application Platform GUI and the Accelerator Server, then it might detect a timeout during accelerator generation. This can happen if the accelerator takes a longer time to generate than the default timeout. When this happens, Tanzu Application Platform GUI appears to continue to run for an indefinite period. In the IDE extension, it shows a 504 error. To mitigate this, you can increase the timeout value for the HTTPProxy resources used for the ingress by applying secrets with overlays to edit the HTTPProxy resources.

## Configure an ingress timeout overlay secret for each HTTPProxy

For Tanzu Application Platform GUI, create the following overlay secret in the `tap-install` namespace:

```
apiVersion: v1
kind: Secret
metadata:
  name: patch-tap-gui-timeout
  namespace: tap-install
stringData:
  patch.yaml: |
    #@ load("@ytt:overlay", "overlay")
    #@overlay/match by=overlay.subset({"kind": "HTTPProxy", "metadata": {"name": "tap-gui"}})
    ---
    spec:
      routes:
        #@overlay/match by=overlay.subset({"services": [{"name": "server"}]})
        #@overlay/match-child-defaults missing_ok=True
        - timeoutPolicy:
            idle: 30s
            response: 30s
```

For Accelerator Server (used for IDE extension), create the following overlay secret in the `tap-install` namespace:

```
apiVersion: v1
kind: Secret
metadata:
  name: patch-accelerator-timeout
  namespace: tap-install
stringData:
  patch.yaml: |
    #@ load("@ytt:overlay", "overlay")
    #@overlay/match by=overlay.subset({"kind": "HTTPProxy", "metadata": {"name": "accelerator"}})
    ---
    spec:
      routes:
        #@overlay/match by=overlay.subset({"services": [{"name": "acc-server"}]})
        #@overlay/match-child-defaults missing_ok=True
        - timeoutPolicy:
            idle: 30s
            response: 30s
```

## Apply the timeout overlay secrets in tap-values.yaml

Add the following `package_overlays` section to `tap-values.yaml` before installing or updating Tanzu Application Platform:

```
package_overlays:
- name: tap-gui
```

```

secrets:
- name: patch-tap-gui-timeout
- name: accelerator
secrets:
- name: patch-accelerator-timeout

```

## Configuring skipping TLS verification for access to Source Controller

You can configure the Flux or Tanzu Application Platform Source Controller to use Transport Layer Security (TLS) and use custom certificates. In that case, configure the Accelerator System to skip the TLS verification for calls to access the sources by providing the following property in the `accelerator` section of the `tap-values.yaml` file:

```

sources:
  skip_tls_verify: true

```

## Enabling TLS for Accelerator Server

To enable TLS for the Accelerator Server, the following properties must be provided in the `accelerator` section of the `tap-values.yaml` file:

```

server:
  tls:
    enabled: true
    key: SERVER-PRIVATE-KEY
    crt: SERVER-CERTIFICATE

```

Where:

- `SERVER-PRIVATE-KEY` is the pem encoded server private key.
- `SERVER-CERTIFICATE` is the pem encoded server certificate.

Here is a sample `tap-values.yaml` configuration with TLS enabled for Accelerators Server:

```

server:
  tls:
    enabled: true
    key: |
      -----BEGIN PRIVATE KEY-----
      .
      . < private key data >
      .
      -----END PRIVATE KEY-----
    crt: |
      -----BEGIN CERTIFICATE-----
      .
      . < certificate data >
      .
      -----END CERTIFICATE-----

```

## Configuring skipping TLS verification of Engine calls for Accelerator Server

If you configure the Accelerator Engine to use TLS and use custom certificates, then you can configure the Accelerator Server to skip the TLS verification for calls to the Engine by providing the following property in the `accelerator` section of the `tap-values.yaml` file:



```
server:
  engine_skip_tls_verify: true
```

## Enabling TLS for Accelerator Engine

To enable TLS for the Accelerator Engine, the following properties are provided in the `accelerator` section of the `tap-values.yaml` file:

```
engine:
  tls:
    enabled: true
    key: ENGINE-PRIVATE-KEY
    crt: ENGINE-CERTIFICATE
```

Where:

- `ENGINE-PRIVATE-KEY` is the pem encoded acc-engine private key.
- `ENGINE-CERTIFICATE` is the pem encoded acc-engine certificate.

Here is a sample `tap-values.yaml` configuration with TLS enabled for Accelerators Engine:

```
engine:
  tls:
    enabled: true
    key: |
      -----BEGIN PRIVATE KEY-----
      .
      . < private key data >
      .
      -----END PRIVATE KEY-----
    crt: |
      -----BEGIN CERTIFICATE-----
      .
      . < certificate data >
      .
      -----END CERTIFICATE-----
```

## Next steps

- [Creating accelerators](#)

## Create accelerators

This topic tells you how to create an accelerator in Tanzu Application Platform GUI.

An accelerator contains your conforming code and configurations that developers can use to create new projects that by default follow the standards defined in your accelerators.

## Prerequisites

The following prerequisites are required to create an accelerator:

- Application Accelerator is installed. For information about installing Application Accelerator, see [Installing Application Accelerator for VMware Tanzu](#).
- You can access Tanzu Application Platform GUI from a browser or use the Application Accelerator extension for VS Code.
  - For more information about Tanzu Application Platform GUI, see [Overview of Tanzu Application Platform GUI](#).

- For more information about Application Accelerator extension for VS Code, see [Application Accelerator Visual Studio Code extension](#).
- kubectl is installed and authenticated with admin rights for your target cluster.

## Getting started

You can use any Git repository to create an accelerator. You need the URL of the repository to create an accelerator.

For this example, the Git repository is `public` and contains a `README.md` file. These are options available when you create repositories on GitHub.

Use the following procedure to create an accelerator based on this Git repository:

- Clone your Git repository.
- Create a file named `accelerator.yaml` in the root directory of this Git repository.
- Add the following content to the `accelerator.yaml` file:

```

accelerator:
  displayName: Simple Accelerator
  description: Contains just a README
  imageUrl: https://images.freecreatives.com/wp-content/uploads/2015/05/smiley-5
59124_640.jpg
  tags:
  - simple
  - getting-started

```

Feel free to use a different icon if it uses a reachable URL.

- Add the new `accelerator.yaml` file, commit this change and push to your Git repository.

## Publishing the new accelerator

- To publish your new accelerator, run:

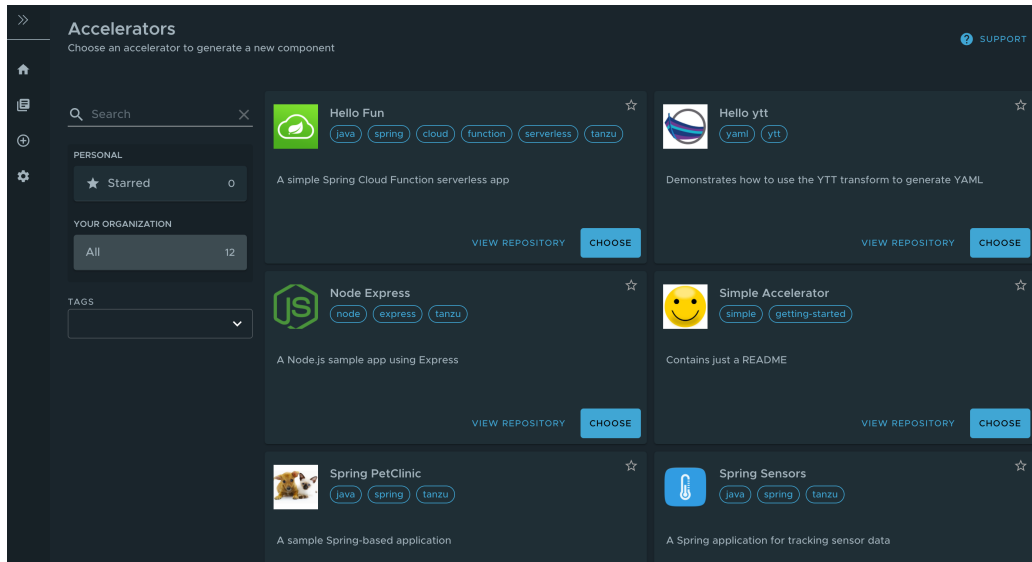
```

tanzu accelerator create simple --git-repository ${GIT_REPOSITORY_URL} --git-br
anch ${GIT_REPOSITORY_BRANCH}

```

Where:

- `GIT-REPOSITORY-URL` is the URL for your Git repository where the accelerator is located.
  - `GIT-REPOSITORY-BRANCH` is the name of the branch where you pushed the new `accelerator.yaml` file.
- Refresh Tanzu Application Platform GUI or the Application Accelerator extension in VS Code to reveal the newly published accelerator. It might take a few seconds to refresh the catalog and add an entry for your new accelerator.



Alternatively, use the Tanzu CLI to create a separate manifest file and apply it to the cluster.

1. Create a `simple-manifest.yaml` file and add the following content, filling in with your Git repository and branch values.

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: simple
  namespace: accelerator-system
spec:
  git:
    url: YOUR-GIT-REPOSITORY-URL
    ref:
      branch: YOUR-GIT-BRANCH
```

2. To apply the `simple-manifest.yaml`, run this command in your terminal in the directory where you created this file:

```
tanzu accelerator apply -f simple-manifest.yaml
```

## Using local-path for publishing accelerators

You can publish an accelerator directly from a local directory on your system. This helps when authoring accelerators and allows you to avoid having to commit every small change to a remote Git repository. You can also specify `--interval` so the accelerator is reconciled quicker when VMware push new changes.

```
tanzu accelerator create simple --local-path ${ACCELERATOR_PATH} --source-image ${SOURCE_IMAGE_REPO} --interval 10s
```

Where:

- `ACCELERATOR-PATH` is the path to the accelerator source. It is a fully qualified or a relative path. If your current directory is already the directory where your source is, then use `“.”`.
- `SOURCE-IMAGE-REPO` is the name of the OCI image repository where you want to push the new accelerator source. If using Docker Hub, use something such as `docker.io/YOUR_DOCKER_ID/simple-accelerator-source`.

After you have made any additional changes, you can push the latest to the same OCI image repository using:

```
tanzu accelerator push --local-path ${ACCELERATOR_PATH} --source-image ${SOURCE_IMAGE_REPO}
```

The accelerator now reflects the new content after approximately a 10-second wait as specified in the previous command.

## Using accelerator fragments

Accelerator fragments are reusable accelerator components that can provide options, files, or transforms. They can be imported from accelerators using an `import` entry and the transforms from the fragment can be referenced in an `InvokeFragment` transform in the accelerator that is declaring the import. For additional details see [InvokeFragment transform](#).

The accelerator samples include three fragments - `java-version`, `tap-initialize`, and `live-update`. See [vmware-tanzu/application-accelerator-samples/fragments](#) Git repository for the content of these fragments.

To discover what fragments are available to use, run:

```
tanzu accelerator fragment list
```

Look at the `java-version` fragment as an example. It contains the following `accelerator.yaml` file:

```
accelerator:
  options:
  - name: javaVersion
    inputType: select
    label: Java version to use
    choices:
    - value: "1.8"
      text: Java 8
    - value: "11"
      text: Java 11
    - value: "17"
      text: Java 17
    defaultValue: "11"
    required: true

engine:
  merge:
  - include: [ "pom.xml" ]
    chain:
    - type: ReplaceText
      regex:
        pattern: "<java.version>.*<"
        with: "'<java.version>' + #javaVersion + '<"
  - include: [ "build.gradle" ]
    chain:
    - type: ReplaceText
      regex:
        pattern: "sourceCompatibility = .*"
        with: "'sourceCompatibility = '" + #javaVersion + "'"
  - include: [ "config/workload.yaml" ]
    chain:
    - type: ReplaceText
      condition: "#javaVersion == '17'"
      substitutions:
      - text: "spec:"
        with: "'spec:\n  build:\n    env:\n      - name: BP_JVM_VERSION\n    valu
e: \"17\"'"
```

This fragment contributes the following to any accelerator that imports it:

1. An option named `javaVersion` with three choices `Java 8`, `Java 11`, and `Java 17`
2. Three `ReplaceText` transforms:
  - If the accelerator has a `pom.xml` file, then what is specified for `<java.version>` is replaced with the chosen version.
  - If the accelerator has a `build.gradle` file, then what is specified for `sourceCompatibility` is replaced with the chosen version.
  - If the accelerator has a `config/workload.yaml` file, and the user selected “Java 17” then a build environment entry of `BP_JVM_VERSION` is inserted into the `spec:` section.

## Deploying accelerator fragments

To deploy new fragments to the accelerator system, use the new `tanzu accelerator fragment create` CLI command or apply a custom resource manifest file with either `kubectl apply` or the `tanzu accelerator apply` commands.

The resource manifest for the `java-version` fragment looks like this:

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Fragment
metadata:
  name: java-version
  namespace: accelerator-system
spec:
  displayName: Select Java Version
  git:
    ref:
      tag: GIT_TAG_VERSION
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
    subPath: fragments/java-version
```

Where `GIT-TAG-VERSION` is the Git tag of the `java-version` fragment. For example, `tap-1.4.0` is a valid Git tag for the `java-version` fragment.

To create the fragment, save the above manifest in a `java-version.yaml` file) and run:

```
tanzu accelerator apply -f ./java-version.yaml
```



### Note

The `accelerator apply` command can be used to apply both Accelerator and Fragment resources.

To avoid having to create a separate manifest file, run:

```
tanzu accelerator fragment create java-version \
  --git-repo https://github.com/vmware-tanzu/application-accelerator-samples.git \
  --git-tag ${GIT_TAG_VERSION} \
  --git-sub-path fragments/java-version
```

Where `GIT-TAG-VERSION` is the Git tag of the `java-version` fragment. For example, `tap-1.4.0` is a valid Git tag for the `java-version` fragment.

Now you can use this `java-version` fragment in an accelerator:

```

accelerator:
  displayName: Hello Fragment
  description: A sample app
  tags:
  - java
  - spring
  - cloud
  - tanzu

  imports:
  - name: java-version

engine:
  merge:
  - include: ["**/*"]
  - type: InvokeFragment
    reference: java-version

```

The earlier accelerator imports the `java-version` which, as seen earlier, provides an option to select the Java version to use for the project. It then instructs the engine to run the transforms provided in the fragment that updates the Java version used in `pom.xml` or `build.gradle` files from the accelerator.

For more detail on the use of fragments, see [InvokeFragment transform](#).

## Next steps

Learn how to:

- Write an [accelerator.yaml](#).
- Configure accelerators with [Accelerator Custom Resources](#).
- Manipulate files using [Transforms](#).
- Use [SpEL in the accelerator.yaml file](#).

## Create accelerators

This topic tells you how to create an accelerator in Tanzu Application Platform GUI.

An accelerator contains your conforming code and configurations that developers can use to create new projects that by default follow the standards defined in your accelerators.

## Prerequisites

The following prerequisites are required to create an accelerator:

- Application Accelerator is installed. For information about installing Application Accelerator, see [Installing Application Accelerator for VMware Tanzu](#).
- You can access Tanzu Application Platform GUI from a browser or use the Application Accelerator extension for VS Code.
  - For more information about Tanzu Application Platform GUI, see [Overview of Tanzu Application Platform GUI](#).
  - For more information about Application Accelerator extension for VS Code, see [Application Accelerator Visual Studio Code extension](#).
- kubectl is installed and authenticated with admin rights for your target cluster.

## Getting started

You can use any Git repository to create an accelerator. You need the URL of the repository to create an accelerator.

For this example, the Git repository is `public` and contains a `README.md` file. These are options available when you create repositories on GitHub.

Use the following procedure to create an accelerator based on this Git repository:

1. Clone your Git repository.
2. Create a file named `accelerator.yaml` in the root directory of this Git repository.
3. Add the following content to the `accelerator.yaml` file:

```

accelerator:
  displayName: Simple Accelerator
  description: Contains just a README
  imageUrl: https://images.freecreatives.com/wp-content/uploads/2015/05/smiley-5
59124_640.jpg
  tags:
    - simple
    - getting-started

```

Feel free to use a different icon if it uses a reachable URL.

4. Add the new `accelerator.yaml` file, commit this change and push to your Git repository.

## Publishing the new accelerator

1. To publish your new accelerator, run:

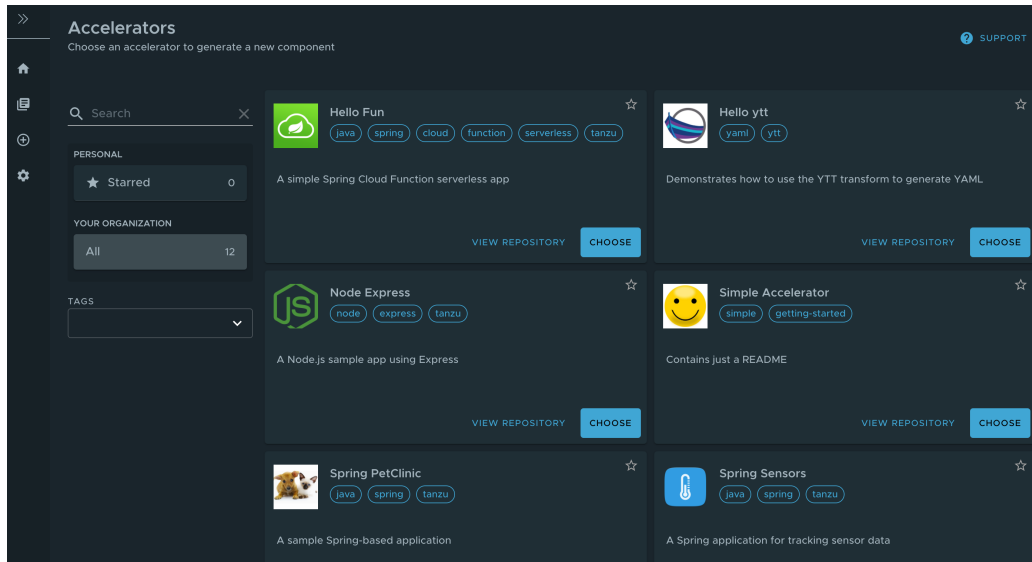
```

tanzu accelerator create simple --git-repository ${GIT_REPOSITORY_URL} --git-br
anch ${GIT_REPOSITORY_BRANCH}

```

Where:

- `GIT-REPOSITORY-URL` is the URL for your Git repository where the accelerator is located.
  - `GIT-REPOSITORY-BRANCH` is the name of the branch where you pushed the new `accelerator.yaml` file.
2. Refresh Tanzu Application Platform GUI or the Application Accelerator extension in VS Code to reveal the newly published accelerator. It might take a few seconds to refresh the catalog and add an entry for your new accelerator.



Alternatively, use the Tanzu CLI to create a separate manifest file and apply it to the cluster.

1. Create a `simple-manifest.yaml` file and add the following content, filling in with your Git repository and branch values.

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: simple
  namespace: accelerator-system
spec:
  git:
    url: YOUR-GIT-REPOSITORY-URL
    ref:
      branch: YOUR-GIT-BRANCH
```

2. To apply the `simple-manifest.yaml`, run this command in your terminal in the directory where you created this file:

```
tanzu accelerator apply -f simple-manifest.yaml
```

## Using local-path for publishing accelerators

You can publish an accelerator directly from a local directory on your system. This helps when authoring accelerators and allows you to avoid having to commit every small change to a remote Git repository. You can also specify `--interval` so the accelerator is reconciled quicker when VMware push new changes.

```
tanzu accelerator create simple --local-path ${ACCELERATOR_PATH} --source-image ${SOURCE_IMAGE_REPO} --interval 10s
```

Where:

- `ACCELERATOR-PATH` is the path to the accelerator source. It is a fully qualified or a relative path. If your current directory is already the directory where your source is, then use `“.”`.
- `SOURCE-IMAGE-REPO` is the name of the OCI image repository where you want to push the new accelerator source. If using Docker Hub, use something such as `docker.io/YOUR_DOCKER_ID/simple-accelerator-source`.

After you have made any additional changes, you can push the latest to the same OCI image repository using:



```
tanzu accelerator push --local-path ${ACCELERATOR_PATH} --source-image ${SOURCE_IMAGE_REPO}
```

The accelerator now reflects the new content after approximately a 10-second wait as specified in the previous command.

## Using accelerator fragments

Accelerator fragments are reusable accelerator components that can provide options, files, or transforms. They can be imported from accelerators using an `import` entry and the transforms from the fragment can be referenced in an `InvokeFragment` transform in the accelerator that is declaring the import. For additional details see [InvokeFragment transform](#).

The accelerator samples include three fragments - `java-version`, `tap-initialize`, and `live-update`. See [vmware-tanzu/application-accelerator-samples/fragments](#) Git repository for the content of these fragments.

To discover what fragments are available to use, run:

```
tanzu accelerator fragment list
```

Look at the `java-version` fragment as an example. It contains the following `accelerator.yaml` file:

```
accelerator:
  options:
  - name: javaVersion
    inputType: select
    label: Java version to use
    choices:
    - value: "1.8"
      text: Java 8
    - value: "11"
      text: Java 11
    - value: "17"
      text: Java 17
    defaultValue: "11"
    required: true

engine:
  merge:
  - include: [ "pom.xml" ]
    chain:
    - type: ReplaceText
      regex:
        pattern: "<java.version>.*<"
        with: "'<java.version>' + #javaVersion + '<"
  - include: [ "build.gradle" ]
    chain:
    - type: ReplaceText
      regex:
        pattern: "sourceCompatibility = .*"
        with: "'sourceCompatibility = '" + #javaVersion + "'"
  - include: [ "config/workload.yaml" ]
    chain:
    - type: ReplaceText
      condition: "#javaVersion == '17'"
      substitutions:
      - text: "spec:"
        with: "'spec:\n  build:\n    env:\n      - name: BP_JVM_VERSION\n    valu
e: \"17\"'"
```

This fragment contributes the following to any accelerator that imports it:

1. An option named `javaVersion` with three choices `Java 8`, `Java 11`, and `Java 17`
2. Three `ReplaceText` transforms:
  - If the accelerator has a `pom.xml` file, then what is specified for `<java.version>` is replaced with the chosen version.
  - If the accelerator has a `build.gradle` file, then what is specified for `sourceCompatibility` is replaced with the chosen version.
  - If the accelerator has a `config/workload.yaml` file, and the user selected “Java 17” then a build environment entry of `BP_JVM_VERSION` is inserted into the `spec:` section.

## Deploying accelerator fragments

To deploy new fragments to the accelerator system, use the new `tanzu accelerator fragment create` CLI command or apply a custom resource manifest file with either `kubectl apply` or the `tanzu accelerator apply` commands.

The resource manifest for the `java-version` fragment looks like this:

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Fragment
metadata:
  name: java-version
  namespace: accelerator-system
spec:
  displayName: Select Java Version
  git:
    ref:
      tag: GIT_TAG_VERSION
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
    subPath: fragments/java-version
```

Where `GIT-TAG-VERSION` is the Git tag of the `java-version` fragment. For example, `tap-1.4.0` is a valid Git tag for the `java-version` fragment.

To create the fragment, save the above manifest in a `java-version.yaml` file) and run:

```
tanzu accelerator apply -f ./java-version.yaml
```



### Note

The `accelerator apply` command can be used to apply both Accelerator and Fragment resources.

To avoid having to create a separate manifest file, run:

```
tanzu accelerator fragment create java-version \
  --git-repo https://github.com/vmware-tanzu/application-accelerator-samples.git \
  --git-tag ${GIT_TAG_VERSION} \
  --git-sub-path fragments/java-version
```

Where `GIT-TAG-VERSION` is the Git tag of the `java-version` fragment. For example, `tap-1.4.0` is a valid Git tag for the `java-version` fragment.

Now you can use this `java-version` fragment in an accelerator:

```

accelerator:
  displayName: Hello Fragment
  description: A sample app
  tags:
  - java
  - spring
  - cloud
  - tanzu

  imports:
  - name: java-version

engine:
  merge:
  - include: ["**/*"]
  - type: InvokeFragment
    reference: java-version

```

The earlier accelerator imports the `java-version` which, as seen earlier, provides an option to select the Java version to use for the project. It then instructs the engine to run the transforms provided in the fragment that updates the Java version used in `pom.xml` or `build.gradle` files from the accelerator.

For more detail on the use of fragments, see [InvokeFragment transform](#).

## Next steps

Learn how to:

- Write an [accelerator.yaml](#).
- Configure accelerators with [Accelerator Custom Resources](#).
- Manipulate files using [Transforms](#).
- Use [SpEL in the accelerator.yaml file](#).

## Create an accelerator.yaml file in Application Accelerator

This topic tells you how to use Application Accelerator to create an `accelerator.yaml` file in Tanzu Application Platform (commonly known as TAP).

By including an `accelerator.yaml` file in your Accelerator repository, you can declare input options that users fill in using a form in the UI. Those option values control processing by the template engine before it returns the zipped output files. For more information, see the [Sample accelerator](#).

When there is no `accelerator.yaml`, the repository still works as an accelerator but the files are passed unmodified to users.

`accelerator.yaml` has two top-level sections: `accelerator` and `engine`.

## Accelerator

This section documents how an accelerator is presented to users in the web UI. For example:

```

accelerator:
  displayName: Hello Fun
  description: A simple Spring Cloud Function serverless app
  iconUrl: https://raw.githubusercontent.com/simple-starters/icons/master/icon-cloud.png
  tags:
  - java

```

```

- spring

options:
- name: deploymentType
  inputType: select
  choices:
  - value: none
    text: Skip Kubernetes deployment
  - value: k8s-simple
    text: Kubernetes deployment and service
  - value: knative
    text: Knative service
  defaultValue: k8s-simple
  required: true

```

## Accelerator metadata

These properties are in accelerator listings such as the web UI:

- **displayName:** A human-readable name.
- **description:** A more detailed description.
- **iconUrl:** A URL pointing to an icon image.
- **tags:** A list of tags used to filter accelerators.

## Accelerator options

The list of options is passed to the UI to create input text boxes for each option.

The following option properties are used by both the UI and the engine.

- **name:** Each option must have a unique, camelCase name. The option value entered by a user is made available as a [SPeL](#) variable name. For example, `#deploymentType`.
- **dataType:** Data types that work with the UI are:
  - `string`
  - `boolean`
  - `number`
  - Custom types defined in the accelerator `types` section
  - Arrays of these such as `[string]`, `[number]`, and so on.

Most input types return a string, which is the default. Use Boolean values with `checkbox`.

- **defaultValue:** This literal value pre-populates the option. Ensure that its type matches the `dataType`. For example, use `["text 1", "text 2"]` for the `dataType` `[string]`. Options without a `defaultValue` can trigger a processing error if the user doesn't provide a value for that option.
- **validationRegex:** When present, a regex validates the string representation of the option value *when set*. It doesn't apply when the value is blank. As a consequence, don't use the regex to enforce prerequisites. See **required** for that purpose.

This regex is used in several layers in Application Accelerator, built using several technologies, for example, JavaScript and Java. So refrain from using "exotic" regex features. Also, the regex applies to portions of the value by default. That is, `[a-z ]+` matches `Hello world` despite the capital `H`. To apply it to the whole value (or just start/end), anchor it using `^` and `$`.

Finally, backslashes in a YAML string using double quotes must be escaped, so to match a number, write `validationRegex: "\d+"` or use another string style.

The following option properties are for UI purposes only.

- **label:** A human-readable version of the `name` identifying the option.
- **description:** A tooltip to accompany the input.
- **inputType:**
  - `text`: The default input type.
  - `textarea`: Single text value with larger input allowing line breaks.
  - `checkbox`: Ideal for Boolean values or multi-value selection from choices.
  - `select`: Single-value selection from choices using a drop-down menu.
  - `radio`: Alternative single-value selection from choices using buttons.
- **choices:** This is a list of predefined choices. Users can select from the list in the UI. Choices are supported by `checkbox`, `select`, and `radio`. Each choice must have a `text` property for the displayed text, and a `value` property for the value that the form returns for that choice. The list is presented in the UI in the same order as it is declared in `accelerator.yaml`.
- **required:** `true` forces users to enter a value in the UI.
- **dependsOn:** This is a way to control visibility by specifying the `name` and optional `value` of another input option. When the other option has a value exactly equal to `value`, or `true` if no `value` is specified, then the option with `dependsOn` is visible. Otherwise, it is hidden. Ensure that the value matches the `dataType` of the `dependsOn` option. For example, a multi-value option (`dataType = [string]`) such as a `checkbox` uses `[matched-value]` to trigger another option when `matched-value` (and only `matched-value`) is selected. See the following section for more information about `dependsOn`.

### DependsOn and multi-value dataType

`dependsOn` tests for strict equality, even for multi-valued options. This means that a multi-valued option must not be used to trigger several other options unfolding, one for each value. Instead, use several single-valued options:

Instead of

```
options:
- name: toppings
  dataType: [string]
  inputType: checkbox
  choices:
    - value: vegetables
      text: Vegetables
    - value: meat
      text: Meat
    ...
- name: vegType
  dependsOn:
    name: toppings
    value: [vegetables] # or vegetables, this won't do what you want either
- name: meatType
  dependsOn:
    name: toppings
    value: [meat]
...
```

do this:

```

options:
  - name: useVeggies
    dataType: boolean
    inputType: checkbox
    label: Vegetables
  - name: useMeat
    dataType: boolean
    inputType: checkbox
    label: Meat
  - name: vegType
    dependsOn:
      name: useVeggies
      value: true
  - name: meatType
    dependsOn:
      name: useMeat
      value: true
  ...

```

## Examples

The later screenshot and `accelerator.yaml` file snippet that follows demonstrates each `inputType`. You can also see the GitHub sample [demo-input-types](#).

The screenshot displays a dark-themed user interface with a list of options on the left and their corresponding input fields on the right. Each option is followed by a collapse icon (a circle with a minus sign). The options and their values are:

- Options**: text (Text value)
- toggle**:  On
- depends on toggle**:  text value
- textarea**: Text line 1, Text line 2
- checkbox**:  Checkbox choice 1,  Checkbox choice 2,  Checkbox choice 3
- depends on checkbox**:  text value
- select**: Select choice 2
- radio**:  Radio choice 1,  Radio choice 2,  Radio choice 3
- tag**: tag1 x tag2 x + Tag
- multi-text**: text value 1, text value 2

```

accelerator:
  displayName: Demo Input Types
  description: "Accelerator with options for each inputType"
  iconUrl: https://raw.githubusercontent.com/simple-starters/icons/master/icon-cloud.png
  tags: ["demo", "options"]

  options:

```

```

- name: text
  display: true
  defaultValue: Text value

- name: toggle
  display: true
  dataType: boolean
  defaultValue: true

- name: dependsOnToggle
  label: 'depends on toggle'
  description: Visibility depends on the value of the toggle option being true.
  dependsOn:
    name: toggle
  defaultValue: text value

- name: textarea
  inputType: textarea
  display: true
  defaultValue: |
    Text line 1
    Text line 2

- name: checkbox
  inputType: checkbox
  display: true
  dataType: [string]
  defaultValue:
    - value-2
  choices:
    - text: Checkbox choice 1
      value: value-1
    - text: Checkbox choice 2
      value: value-2
    - text: Checkbox choice 3
      value: value-3

- name: dependsOnCheckbox
  label: 'depends on checkbox'
  description: Visibility depends on the checkbox option containing exactly value va
  lue-2.
  dependsOn:
    name: checkbox
    value: [value-2]
  defaultValue: text value

- name: select
  inputType: select
  display: true
  defaultValue: value-2
  choices:
    - text: Select choice 1
      value: value-1
    - text: Select choice 2
      value: value-2
    - text: Select choice 3
      value: value-3

- name: radio
  inputType: radio
  display: true
  defaultValue: value-2
  choices:
    - text: Radio choice 1
      value: value-1
    - text: Radio choice 2

```

```

    value: value-2
  - text: Radio choice 3
    value: value-3

engine:
  type: YTT

```

## Engine

The engine section describes how to take the files from the accelerator root directory and transform them into the contents of a generated project file.

The YAML notation here defines what is called a transform. A transform is a function on a set of files. It uses a set of files as input. It produces a modified set of files as output derived from this input.

Different types of transforms do different tasks:

- Filtering the set of files: that is, removing, or keeping files that match certain criteria.
- Changing the contents of files. For example, replacing some strings in the files.
- Renaming or moving files: that is, changing the paths of the files.

The notation also provides the composition operators `merge` and `chain`, which enable you to create more complex transforms by composing simpler transforms together.

The following is an example of what is possible. To learn the notation, see [Introduction to transforms](#).

## Engine example

```

engine:
  include:
    ["**/*.md", "**/*.xml", "**/*.gradle", "**/*.java"]
  exclude:
    ["**/secret/**"]
  let:
    - name: includePoms
      expression:
        "#buildType == 'Maven'"
    - name: includeGradle
      expression: "#buildType == 'Gradle'"
  merge:
    - condition:
        "#includeGradle"
      include: ["*.gradle"]
    - condition: "#includePoms"
      include: ["pom.xml"]
    - include: ["**/*.java", "README.md"]
  chain:
    - type: ReplaceText
      substitutions:
        - text: "Hello World!"
          with: "#greeting"
  chain:
    - type: RewritePath
      regex: (.*)simpleboot(.)
      rewriteTo: "#g1 + #packageName + #g2"
    - type: ReplaceText
      substitutions:
        - text: simpleboot
          with: "#packageName"

```



```
onConflict:
  Fail
```

## Engine notation descriptions

This section describes the notations in the preceding example.

`engine` is the global transformation node. It produces the final set of files to be zipped and returned from the accelerator. As input, it receives all the files from the accelerator repository root. The properties in this node dictate how this set of files is transformed into a final set of files zipped as the accelerator result.

`engine.include` filters the set of files, retaining only those matching a list of path patterns. This ensures that that the accelerator only detects files in the repository that match the list of patterns.

`engine.exclude` further restricts which files are detected. The example ensures files in any directory called `secret` are never detected.

`engine.let` defines additional variables and assigns them values. These derived symbols function such as options, but instead of being supplied from a UI widget, they are computed by the accelerator itself.

`engine.merge` executes each of its children in parallel. Each child receives a copy of the current set of input files. These are files remaining after applying the `include` and `exclude` filters. Each of the children therefore produces a set of files. All the files from all the children are then combined, as if overlaid on top of each other in the same directory. If more than one child produces a file with the same path, the transform resolves the conflict by dropping the file contents from the earlier child and keeping the contents from the later child.

`engine.merge.chain` specifies additional transformations to apply to the set of files produced by this child. In the example, `ReplaceText` is only applied to Java files and `README.md`.

`engine.chain` applies transformation to all files globally. The chain has a list of child transformations. These transformations are applied after everything else in the same node. This is the global node. The children in a chain are applied sequentially.

`engine.onConflict` specifies how conflict is handled when an operation, such as merging, produces multiple files at the same path: - `Fail` raises an error when there is a conflict. - `UseFirst` keeps the contents of the first file. - `UseLast` keeps the contents of the last file. - `Append` keeps both by using `cat <first-file> <second-file>`.

## Advanced accelerator use

Additional advanced features can be leveraged when writing an `accelerator.yaml`. For more information see, [Creating dynamic parameters using custom types](#)

## Use transforms in Application Accelerator

This topic tells you about using transforms with Application Accelerator.

When the accelerator engine executes the accelerator, it produces a ZIP file containing a set of files. The purpose of the `engine` section is to describe precisely how the contents of that ZIP file is created.

```
accelerator:
  ...
engine:
  <transform-definition>
```

## Why transforms?

When you run an accelerator, the contents of the accelerator produce the result. It is made up of subsets of the files taken from the accelerator `<root>` directory and its subdirectories. You can copy the files as is, or transform them in a number of ways before adding them to the result.

As such, the YAML notation in the `engine` section defines a transformation that takes as input a set of files (in the `<root>` directory of the accelerator) and produces as output another set of files, which are put into the ZIP file.

Every transform has a `type`. Different types of transform have different behaviors and different YAML properties that control precisely what they do.

In the following example, a transform of type `Include` is a filter. It takes as input a set of files and produces as output a subset of those files, retaining only those files whose path matches any one of a list of `patterns`.

If the accelerator has something like this:

```
engine:
  type: Include
  patterns: ['**/*.java']
```

This accelerator produces a ZIP file containing all the `.java` files from the accelerator `<root>` or its subdirectories but nothing else.

Transforms can also operate on the contents of a file, instead of merely selecting it for inclusion.

For example:

```
type: ReplaceText
substitutions:
- text: hello-fun
  with: "#artifactId"
```

This transform looks for all instances of a string `hello-fun` in all its input files and replaces them with an `artifactId`, which is the result of evaluating a SpEL expression.

## Combining transforms

From the preceding examples, you can see that transforms such as `ReplaceText` and `Include` are too “primitive” to be useful by themselves. They are meant to be the building blocks of more complex accelerators.

To combine transforms, provide two operators called `Chain` and `Merge`. These operators are recursive in the sense that they compose a number of child transforms to create a more complex transform. This allows building arbitrarily deep and complex trees of nested transform definitions.

The following example shows what each of these two operators does and how they are used together.

### Chain

Because transforms are functions whose input and output are of the same type (a set of files), you can take the output of one function and feed it as input to another. This is what `Chain` does. In mathematical terms, `Chain` is *function composition*.

You might, for example, want to do this with the `ReplaceText` transform. Used by itself, it replaces text strings in *all* the accelerator input files. What if you wanted to apply this replacement to only a

subset of the files? You can use an `Include` filter to select only a subset of files of interest and chain that subset into `ReplaceText`.

For example:

```
type: Chain
transformations:
- type: Include
  patterns: ['**/pom.xml']
- type: ReplaceText
  substitutions:
  - text: hello-fun
    with: "#artifactId"
```

## Merge

Chaining `Include` into `ReplaceText` limits the scope of `ReplaceText` to a subset of the input files. Unfortunately, it also eliminates all other files from the result.

For example:

```
engine:
  type: Chain
  transformations:
  - type: Include
    patterns: ['**/pom.xml']
  - type: ReplaceText
    substitutions:
    - text: hello-fun
      with: "#artifactId"
```

The preceding accelerator produces a ZIP file that only contains `pom.xml` files and nothing else.

What if you also wanted other files in that ZIP? Perhaps you want to include some Java files as well, but don't want to apply the same text replacement to them.

You might be tempted to write something such as:

```
engine:
  type: Chain
  transformations:
  - type: Include
    patterns: ['**/pom.xml']
  - type: ReplaceText
    ...
  - type: Include
    patterns: ['**/*.java']
```

However, that doesn't work. If you chain non-overlapping includes together like this, the result is an empty result set. The reason is that the first include retains only `pom.xml` files. These files are fed to the next transform in the chain. The second include only retains `.java` files, but because there are only `pom.xml` files left in the input, the result is an empty set.

This is where `Merge` comes in. A `Merge` takes the outputs of several transforms executed independently on the same input sourceset and combines or merges them together into a single sourceset.

For example:

```
engine:
  type: Merge
  sources:
  - type: Chain
```

```

- type: Include
  patterns: ['**/pom.xml']
- type: ReplaceText
  ...
- type: Include
  patterns: ['**/*.java']

```

The preceding accelerator produces a result that includes both:

- The `pom.xml` files with some text replacements applied to them.
- Verbatim copies of all the `.java` files.

## Shortened notation

It becomes cumbersome and verbose to combine transforms such as `Include`, `Exclude`, and `ReplaceText` with explicit `Chain` and `Merge` operators. Also, there is a common composition pattern to using them. Specifically, select an interesting subset using includes/excludes, apply a chain of additional transformations to the subset, and merge the result with the results of other transforms.

That is why there is a swiss army knife transform (known the `Combo` transform) that combines `Include`, `Exclude`, `Merge`, and `Chain`.

For example:

```

type: Combo
include: ['**/*.txt', '**/*.md']
exclude: ['**/secret/**']
merge:
- <transform-definition>
- ...
chain:
- <transform-definition>
- ...

```

Each of the properties in this `Combo` transform is optional if you specify at least one.

Notice how each of the properties `include`, `exclude`, `merge`, and `chain` corresponds to the name of a type of transform, only spelled with lowercase letters.

If you specify only one of the properties, the `Combo` transform behaves exactly as if you used that type of transformation by itself.

For example:

```
merge: ...
```

Behaves the same as:

```
type: Merge
sources: ...
```

When you do specify multiple properties at the same time, the `Combo` transform composes them together in a “logical way” combining `Merge` and `Chain` under the hood.

For example:

```
include: ['**/*.txt', '**.md']
chain:
- type: ReplaceText
  ...

```

Is the same as:

```

type: Chain
transformations:
- type: Include
  patterns: ['**/*.txt', '**.md']
- type: Chain
  transformations:
  - type: ReplaceText
  ...

```

When you use all of the properties of `Combo` at once:

```

include: I
exclude: E
merge:
- S1
- S2
chain:
- T1
- T2

```

This is equivalent to:

```

type: Chain
transformations:
- type: Include
  patterns: I
- type: Exclude
  patterns: E
- type: Merge
  sources:
  - S1
  - S2
- T1
- T2

```

## A Combo of one?

You can use the `Combo` as a convenient shorthand for a single type of annotation. However, though you *can* use it to combine multiple types, and though that is its main purpose, that doesn't mean you *have* to.

For example:

```
include: ["**/*.java"]
```

This is a `Combo` transform (remember, `type: Combo` is optional), but rather than combining multiple types of transforms, it only defines the `include` property. This makes it behave exactly as an `Include` transform:

```

type: Include
patterns: ["**/*.java"]

```

It is usually more convenient to use a `Combo` transform to denote a single `Include`, `Exclude`, `Chain`, or `Merge` transform, because it is slightly shorter to write it as a `Combo` than writing it with an explicit `type` property.

## A common pattern with merge transforms

It is a common and useful pattern to use merges with overlapping contents to apply a transformation to a subset of files and then replace these changed files within a bigger context.

For example:

```
engine:
  merge:
    - include: ["**/*"]
    - include: ["**/pom.xml"]
  chain:
    - type: ReplaceText
      substitutions: ...
```

The preceding accelerator copies all files from accelerator `<root>` while applying some text replacements only to `pom.xml` files. Other files are copied verbatim.

Here in more detail is how this works:

- **Transform A** is applied to the files from accelerator `<root>`. It selects all files, including `pom.xml` files.
- **Transform B** is *also* applied to the files from accelerator `<root>`. Again, `Merge` passes the same input independently to each of its child transforms. Transform B selects `pom.xml` files and replaces some text in them.

So both **Transform A** and **Transform B** output `pom.xml` files. The fact that both result sets contain the same file, and with different contents in them in this case, is a conflict that has to be resolved. By default, `Combo` follows a simple rule to resolve such conflicts: take the contents from the last child. Essentially, it behaves as if you overlaid both result sets one after another into the same location. The contents of the latter overwrite any previous files placed there by the earlier.

In the preceding example, this means that while both **Transform A** and **Transform B** produce contents for `pom.xml`, the contents from **Transform B** “wins.” So you get the version of the `pom.xml` that has text replacements applied to it rather than the verbatim copy from **Transform A**.

## Conditional transforms

Every `<transform-definition>` can have a `condition` attribute.

```
- condition: "#k8sConfig == 'k8s-resource-simple'"
  include: [ "kubernetes/app/*.yaml" ]
  chain:
    - type: ReplaceText
      substitutions:
        - text: hello-fun
          with: "#artifactId"
```

When a transform’s condition is `false`, that transform is “deactivated.” This means it is replaced by a transform that “does nothing.” However, doing nothing can have different meanings depending on the context:

- When in the context of a `Merge`, a deactivated transform behaves like something that returns an empty set. A `Merge` adds things together using a kind of union; adding an empty set to union essentially does nothing.
- When in the context of a `Chain` however, a deactivated transform behaves like the `identity` function instead (that is, `lambda (x) => x`). When you chain functions together, a value is passed through all functions in succession. So each function in the chain has the chance to “do something” by returning a different modified value. If you are a function in a chain, to do nothing means to return the input you received unchanged as your output.

If this all sounds confusing, fortunately there is a basic guideline for understanding and predicting the effect of a deactivated transform in the context of your accelerator definition. Namely, if a transform’s condition evaluates to false, pretend it isn’t there. In other words, your accelerator

behaves as if you deleted (or commented out) that transform’s YAML text from the accelerator definition file.

The following examples illustrate both cases.

## Conditional ‘Merge’ transform

This example, **transform A**, has a conditional transform in a `Merge` context:

```
merge:
- condition: "#k8sConfig == 'k8s-resource-simple'"
  include: [ "kubernetes/app/*.yaml" ]
  chain:
  ...
- include: [ "pom.xml" ]
  chain:
  ...
```

If the condition of **transform A** is `false`, it is replaced with an “empty set” because it is used in a `Merge` context. This has the same effect as if the whole of **transform A** was deleted or commented out:

```
merge:
- include: [ "pom.xml" ]
  chain:
  ...
```

In this example, if the condition is `false`, only `pom.xml` file is in the result.

## Conditional ‘Chain’ transform

In the following example, some conditional transforms are used in a `Chain` context:

```
merge:
- include: [ '**/*.json' ]
  chain:
- type: ReplaceText
  condition: '#customizeJson'
  substitutions: ...
- type: JsonPrettyPrint
  condition: '#prettyJson'
  indent: '#jsonIndent'
```

The `JsonPrettyPrint` transform type is purely hypothetical. There *could* be such a transform, but VMware doesn’t currently provide it.

In the preceding example, both **transform A** and **transform B** are conditional and used in a `Chain` context. **Transform A** is chained after the `include` transform. Whereas **transform B** is chained after **transform A**. When either of these conditions is `false`, the corresponding transform behaves like the identity function. Namely, whatever set of files it receives as input is exactly what it returns as output.

This behavior accords with our guideline. For example, if **transform A**’s condition is `false`, it behaves as if **transform A** wasn’t there. **Transform A** is chained after `include` so it receives the `include`’s result, returns it unchanged, and this is passed to **transform B**. In other words, the result of the `include` is passed as is to **transform B**. This is precisely what would happen were **transform A** not there.

## A small gotcha with using conditionals in merge transforms

As mentioned earlier, it is a useful pattern to use merges with overlapping contents. Yet you must be careful using this in combination with conditional transforms.

For example:

```
engine:
  merge:
    - include: ["**/*"]
    - include: ["**/pom.xml"]
  chain:
    - type: ReplaceText
      substitutions: ...
```

Now add a little twist. Say you only wanted to include pom files if the user selects a `useMaven` option. You might be tempted to add a 'condition' to **transform B** to deactivate it when that option isn't selected:

```
engine:
  merge:
    - include: "**/*"
    - condition: '#useMaven'
      include: ["**/pom.xml"]
  chain:
    - type: ReplaceText
      substitutions: ...
```

However, this doesn't do what you might expect. The final result *still* contains `pom.xml` files. To understand why, recall the guideline for deactivated transforms: If a transform is deactivated, pretend it isn't there. So when `#useMaven` is `false`, the example reduces to:

```
engine:
  merge:
    - include: ["**/*"]
```

This accelerator copies all files from accelerator `<root>`, *including* `pom.xml`.

There are several ways to avoid this pitfall. One is to ensure the `pom.xml` files are not included in **transform A** by explicitly excluding them:

```
...
- include: ["**/*"]
  exclude: ["**/pom.xml"]
...
```

Another way is to apply the exclusion of `pom.xml` conditionally in a `Chain` after the main transform:

```
engine:
  merge:
    - include: ["**/*"]
    - include: ["**/pom.xml"]
  chain:
    - type: ReplaceText
      substitutions: ...
  chain:
    - condition: '!#useMaven'
      exclude: ['**/pom.xml']
```

## Merge conflict

The representation of the set of files upon which transforms operate is "richer" than what you can physically store on a file system. A key difference is that in this case, the set of files allows for



multiple files with the same path to exist at the same time. When files are initially read from a physical file system, or a ZIP file, this situation does not arise. However, as transforms are applied to this input, it can produce results that have more than one file with the same path and yet different contents.

Earlier examples illustrated this happening through a `merge` operation. Again, for example:

```
merge:
- include: ["**/*"]
- include: ["**/pom.xml"]
  chain:
  - type: ReplaceText
    substitutions: ...
```

The result of the preceding `merge` is two files with path `pom.xml`, assuming there was a `pom.xml` file in the input. **Transform A** produces a `pom.xml` that is a verbatim copy of the input file. **Transform B** produces a modified copy with some text replaced in it.

It is impossible to have two files on a disk with the same path. Therefore, this conflict must be resolved before you can write the result to disk or pack it into a ZIP file.

As the example shows, merges are likely to give rise to these conflicts, so you might call this a “merge conflict.” However, such conflicts can also arise from other operations. For example,

`RewritePath`:

```
type: RewritePath
regex: '.*.md'
rewriteTo: "'docs/README.md'"
```

This example renames any `.md` file to `docs/README.md`. Assuming the input contains more than one `.md` file, the output contains multiple files with path `docs/README.md`. Again, this is a conflict, because there can only be one such file in a physical file system or ZIP file.

## Resolving “merge” conflicts

By default, when a conflict arises, the engine doesn’t do anything with it. Our internal representation for a set of files allows for multiple files with the same path. The engine carries on manipulating the files as is. This isn’t a problem until the files must be written to disk or a ZIP file. If a conflict is still present at that time, an error is raised.

If your accelerator produces such conflicts, they must be resolved before writing files to disk. To this end, VMware provides the `UniquePath` transform. This transform allows you to specify what to do when more than one file has the same path. For example:

```
chain:
- type: RewritePath
  regex: '.*.md'
  rewriteTo: "'docs/README.md'"
- type: UniquePath
  strategy: Append
```

The result of the above transform is that all `.md` files are gathered up and concatenated into a single file at path `docs/README.md`. Another possible resolution strategy is to keep only the contents of one of the files. See [Conflict Resolution](#).

`Combo` transform also comes with some convenient built-in support for conflict resolution. It automatically selects the `UseLast` strategy if none is explicitly supplied. So in practice, you rarely, if ever, need to explicitly specify a conflict resolution strategy.

## File ordering

As mentioned earlier, our set of files representation is richer than the files on a typical file system in that it allows for multiple files with the same path. Another way in which it is richer is that the files in the set are “ordered.” That is, a `FileSet` is more like an ordered list than an unordered set.

In most situations, the order of files in a `FileSet` doesn’t matter. However, in conflict resolution it *is* significant. If you look at the preceding `RewritePath` example again, you might wonder about the order in which the various `.md` files are appended to each other. This ordering is determined by the order of the files in the input set.

So what is that order? In general, when files are read from disk to create a `FileSet`, you cannot assume a specific order. Yes, the files are read and processed in a sequential order, but the actual order is not well defined. It depends on implementation details of the underlying file system. The accelerator engine therefore does not ensure a specific order in this case. It only ensures that it *preserves* whatever ordering it receives from the file system, and processes files in accord with that order.

As an accelerator author, better to avoid relying on the file order produced from reading directly from a file system. So it’s better to avoid doing something like the preceding `RewritePath` example, *unless* you do not care about the ordering of the various sections of the produced `README.md` file.

If you do care and want to control the order explicitly, you use the fact that `Merge` processes its children in order and reflects this order in the resulting output set of files. For example:

```
chain:
  - merge:
    - include: ['README.md']
    - include: ['DEPLOYMENT.md']
      chain:
        - type: RewritePath
          rewriteTo: "README.md"
  - type: UniquePath
    strategy: Append
```

In this example, `README.md` from the first child of `merge` definitely comes before `DEPLOYMENT.md` from the second child of `merge`. So you can control the merge order directly by changing the order of the merge children.

## Next steps

This introduction focused on an intuitive understanding of the `<transform-definition>` notation. This notation defines precisely how the accelerator engine generates new project content from the files in the accelerator root.

To learn more, read the following more detailed documents:

- An exhaustive [Reference](#) of all built-in transform types
- A sample, commented [accelerator.yaml](#) to learn from a concrete example

## Use custom types in Application Accelerator

This topic tells you how to declare new `types` in `accelerator.yaml`

Use these types for options declaration, in addition to the built-in types `string`, `number`, and `boolean`.

In `accelerator.yaml`, use the `types` entry (inside the top-level `accelerator` section) to define custom types.

The name must be an initial capital letter.

In the following example, the `struct` type definition is syntactically equivalent to a sequence of option definitions:

```

accelerator:
  options:
    ...
  types:
    - name: Task
      struct:
        - name: title
          dataType: string
          label: Title
          description: A sample title
        - name: details
          label: Task details
          description: Enter the task details
        - name: done
          dataType: boolean
          label: Done?
          defaultValue: false

```

This example creates a new *type* that is available for the `dataType` property of any option. For example,

```

accelerator:
  options:
    - name: myTask
      dataType: Task
  types:
    ...

```

UIs render similar to the following:

### Title\*

A sample title

### Task details

Enter the task details

Done?

and associate the entered values to the `myTask` top-level name, resulting in the following example values submission (here represented using JSON notation):

```

{
  "myTask": { // Note the use of a nested object here
    "title": "Get job done!",
    "details": "Needs this asap",
    "done": false
  }
}

```

The type of the `myTask` value is `object` (in Javascript/JSON parlance) and `Map<String, ?>` when seen from the Java engine side.

The earlier example is technically possible with the custom types feature, but brings little benefit over having three options named to achieve the same result, for example, `myTaskTitle`, `myTaskDetails`, and `myTaskDone`. The value of custom types is when they are used in sequence types, allowing you to enter an unbounded list of structured data:

```

accelerator:
  options:
    - name: myTasks
      dataType: [Task]
  types:
    ...

```

Which might result in the following example submission (JSON):

```

{
  "myTasks": [ // Note the use of JSON array
    { // with elem 0 being an object
      "details": "something",
      "done": true,
      "title": "The Title"
    },
    { // and elem 1 as well, etc
      "details": "something else",
      "done": false,
      "title": "The other Title"
    }
  ]
}

```

## Limitations

A `struct` custom type declaration is made of an ordered series of option definitions. The support and semantics for individual text boxes of option-definition-like elements when used in the type *declaration* are stated in the following example.

When *referencing* a custom type in an option definition, some previously valid properties of an option definition might become irrelevant or unsupported. This is stated in the following example:

```

accelerator:
  types:
    - name: MyType
      struct:
        - name: someField # the "option name" will become a 'property' of the newly
          created type
          dataType: string # is the type of this single property. Typically, will be
          a simple
                                # scalar type like string or number
          defaultValue: foo # supported and is the default if not overridden at usage
          point by the option's defaultValue
          description: something # will become the description for the field's widget
          choices:
                                # supported
            - value: v
              text: t
          validationRegex:      # validates that single property
          label:                 # will become the "title" of the widget
          inputType:             # supported
          required:              # supported
          dependsOn:            # supported against other properties of THIS struct
          .. other fields
      options:
        - name: anOptionThatUsesACustomType
          dataType: MyType

```

```

defaultValue: # supported, should then be an object (or array thereof)
description: # supported, is the description of the whole option (as opposed to
individual fields)
label: # supported, idem
choices: # NOT supported
  - value: v
    text: t
validationRegex: # NOT supported
inputType: # NOT supported
required: # technically supported, useless in practice
dependsOn: # OK to depend on another option

```

## Interaction with SpEL

Everywhere that SpEL is used in the engine syntax, accelerator authors might use SpEL syntax for accessing properties or array elements:

```
#myTasks[2]['done']
```

Do not use array indexing either with a literal number or a variable, as the purpose of the list of the custom types feature is that you don't know the data length in advance. For more information about idiomatic uses of repeated structured data, see [Loop Transform](#).

## Interaction with Composition

Using composition alongside custom types has the following advantages/disadvantages:

- You might want to **leverage** types declared in an imported fragment
- There might be a type **name clash** between a host accelerator/fragment and an imported fragment, because the imported fragment author is unaware of how the fragment is to be used.

For more information about the syntax to customize the imported types names, see [Use fragments in Application Accelerator](#).

## Use fragments in Application Accelerator

This topic tells you how to use fragments in Application Accelerator.

### Introduction

Despite their benefits, writing and maintaining, accelerators can become repetitive and verbose as new accelerators are added. Some create a project different from the next with similar aspects, requiring some form of copy-paste.

To alleviate this concern, Application Accelerators support a feature named Composition that allows the re-use of parts of an accelerator, called **fragments**.

### Introducing fragments

A **fragment** looks exactly the same as an accelerator:

- It is made of a set of files.
- It contains an `accelerator.yaml` descriptor with options, declarations, and a root transform.

There are differences however. Namely:

- Fragments are declared to the system differently. They are filed as **fragment** custom resources.
- They deal with files differently. Because fragments deal with their own files and files from the accelerator using them, they use dedicated conflict resolution **strategies** (more on this later).

Fragments may be thought of as “functions” in programming languages. After being defined and referenced, they are “called” at various points in the main accelerator. The composition feature is designed with ease of use and “common use first” in mind, so these “functions” are typically called with as little noise as possible. You can also call them complex or different values.

Composition relies on two building blocks that play hand in hand:

- The `imports` section at the top of an accelerator manifest.
- The, `InvokeFragment` transform, to be used alongside any other transform.

## | The `imports` section explained

To be usable in composition, a fragment **MUST** be *imported* into the dedicated section of an accelerator manifest:

```
accelerator:
  name: my-awesome-accelerator
  options:
    - name: flavor
      dataType: string
      defaultValue: Strawberry
  imports:
    - name: my-first-fragment
    - name: another-fragment
  engine:
    ...
```

The effect of importing a fragment this way is twofold:

- It makes its files available to the engine (therefore importing a fragment is required).
- It exposes all of its options as options of the accelerator as if they were defined by the accelerator itself.

So in the earlier example, if the `my-first-fragment` fragment had the following `accelerator.yaml` file:

```
accelerator
  name: my-first-fragment
  options:
    - name: optionFromFragment
      dataType: boolean
      description: this option comes from the fragment
  ...
```

Then it is as if the `my-awesome-accelerator` accelerator defined it:

```
accelerator:
  name: my-awesome-accelerator
  options:
    - name: flavor
      dataType: string
      defaultValue: Strawberry
    - name: optionFromFragment
      dataType: boolean
```

```

    description: this option comes from the fragment
  imports:
    - name: my-first-fragment
    - name: another-fragment
  engine:
    ...

```

All the metadata about options (type, default value, description, choices if applicable, *etc.*) come along when imported.

Because of this, users are prompted for a value for those options that come from fragments, as if they were options of the accelerator.

## Using the `InvokeFragment` Transform

The second part at play in the composition is the `InvokeFragment` Transform.

As with any other transform, it may be used anywhere in the `engine` tree and receives files that are “visible” at that point. Those files, alongside those that make up the fragment, are made available to the fragment logic. If the fragment wants to choose between two versions of a file for a path, two new conflict resolution `strategies` are available: `FavorForeign` and `FavorOwn`.

The behavior of the `InvokeFragment` transform is very straightforward: after having validated options that the fragment expects (and maybe after having set default values for options that support one), it executes the whole transform of the fragment *as if it was written in place of `InvokeFragment`*.

See the `InvokeFragment` [reference page](#) for more explanations, examples, and configuration options. This topic now focuses on additional features of the `imports` section that are seldom used but still available to cover more complex use cases.

## Back to the `imports` section

The complete definition of the `imports` section is as follows, with features in increasing order of “complexity”:

```

accelerator:
  name: ...
  options:
    - name: ...
    ...
  imports:
    - name: some-fragment

    - name: another-fragment
      expose:
        - name: "*"
      exposeTypes:
        - name: "*"

    - name: yet-another-fragment
      expose:
        - name: someOption

        - name: someOtherOption
          as: aDifferentName
      exposeType:
        - name: SomeType

        - name: SomeOtherType
          as: ADifferentName

```

```
engine:
  ...
```

As shown earlier, the `imports` section calls a list of fragments to import. By default, all their options and types become options/type of the accelerator. Those options appear *after* the options defined by the accelerator, in the order the fragments are imported in.

It is even possible for a fragment to import another fragment, the semantics being the same as when an accelerator imports a fragment. This is a way to break apart a fragment even further if needed.

When importing a fragment, you can select which options of the fragment to make available as options of the accelerator. **This feature should only be used when a name clash arises in option names.**

The semantics of the `expose` block are as follows:

- For every `name/as` pair, don't use the original (`name`) of the option but instead, use the alias (`as`). Other metadata about the option is left unchanged.
- If the special `name: "*" (which is NOT a legit option name usually) appears, all imported option names that are not remapped (the index at which the * appears in the YAML list is irrelevant) might be exposed with their original name.`
- The default value for `expose` is `[{name: "*"}]`, that is, by default exposes all options with their original name.
- As soon as a single remap rule appears, the default is overridden. For example, to override some names AND expose the others unchanged, the `*` must be explicitly re-added.
- To explicitly un-expose ALL options from an imported fragment, an empty array may be used and overrides the default: `expose: []`.

Similarly, you can also select which `custom types` of the fragment to make available as types of the accelerator. **This feature should only be used when a name clash arises in types names.**

The semantics of the `exposeTypes` block are as follows:

- For every `name/as` pair, don't use the original (`name`) of the type but instead, use the alias (`as`). Options that used the original name are automatically "rewritten" to use the new name.
- If the special `name: "*" appears, which is NOT usually a legit type name, all imported other type names that are not remapped are exposed with their original name. The index at which the * appears in the YAML list is irrelevant.`
- The default value for `exposeTypes` is `[{name: "*"}]`, that is, by default exposes all types with their original name.
- As soon as a single remap rule appears, the default is overridden. For example, to override some names AND expose the others unchanged, the `*` must be explicitly re-added.
- To explicitly un-expose ALL types from an imported fragment, an empty array may be used, which overrides the default: `exposeTypes: []`.

## Using `dependsOn` in the `imports` section

Lastly, as a convenience for the conditional use of fragments, you can make an exposed imported option *depend on* another option, as in the following example:

```
imports:
  - name: tap-initialize
    expose:
      - name: gitRepository
```



```

as: gitRepository
dependsOn:
  name: deploymentType
  value: workload
- name: gitBranch
as: gitBranch
dependsOn:
  name: deploymentType
  value: workload

```

This plays well with the use of `condition`, as in the following example:

```

...
engine:
  ...
  type: InvokeFragment
  condition: "#deploymentType == 'workload'"
  reference: tap-initialize``

```

## Discovering fragments using Tanzu CLI accelerator plug-in

Using the accelerator plug-in for Tanzu CLI, you can view a list of available fragments. Run:

```
tanzu accelerator fragment list
```

To see a list of available accelerator fragments. For example:

| NAME                               | READY | REPOSITORY                                                                                                                                                                       |
|------------------------------------|-------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| app-sso-client                     | true  | source-image: dev.registry.tanzu.vmware.com/app-accelerator/fragments/app-sso-client@sha256:ed5cf5544477d52d4c7baf3a76f71a112987856e77558697112e46947ada9241                     |
| java-version                       | true  | source-image: dev.registry.tanzu.vmware.com/app-accelerator/fragments/java-version@sha256:df99a5ace9513dc8d083fb5547e2a24770dfb08ec11b6591e98497a329b969d                        |
| live-update                        | true  | source-image: dev.registry.tanzu.vmware.com/app-accelerator/fragments/live-update@sha256:c2eda015b0f811b0eeaa587b6f2c5410ac87d40701906a357cca0decb3f226a4                        |
| spring-boot-app-sso-auth-code-flow | true  | source-image: dev.registry.tanzu.vmware.com/app-accelerator/fragments/spring-boot-app-sso-auth-code-flow@sha256:78604c96dd52697ea0397d3933b42f5f5c3659cbcdc0616ff2f57be558650499 |
| tap-initialize                     | true  | source-image: dev.registry.tanzu.vmware.com/app-accelerator/fragments/tap-initialize@sha256:7a3ae8f9277ef633200622dbf9d0f5a07dea25351ac3dbf803ea2fa759e3baac                     |
| tap-workload                       | true  | source-image: dev.registry.tanzu.vmware.com/app-accelerator/fragments/tap-workload@sha256:8056ad9f05388883327d9bbe457e6a0122dc452709d179f683eceb6d848338d0                       |

The `tanzu accelerator fragment get <fragment-name>` command shows all the options defined for the fragment and also any accelerators or other fragments that import this fragment. Run:

```
tanzu accelerator fragment get java-version
```

The following output is displayed:

```

name: java-version
namespace: accelerator-system
displayName: Select Java Version
ready: true
options:
- choices:
  - text: Java 8
    value: "1.8"

```

```

- text: Java 11
  value: "11"
- text: Java 17
  value: "17"
defaultValue: "11"
inputType: select
label: Java version to use
name: javaVersion
required: true
artifact:
  message: Resolved revision: dev.registry.tanzu.vmware.com/app-accelerator/fragments/
java-version@sha256:df99a5ace9513dc8d083fb5547e2a24770dfb08ec11b6591e98497a329b969d
  ready: true
  url: http://source-controller-manager-artifact-service.source-system.svc.cluster.local/
imagerepository/accelerator-system/java-version-frag-97nwp/df99a5ace9513dc8d083fb5547e2a24770dfb08ec11b6591e98497a329b969d.tar.gz
imports:
  None
importedBy:
  accelerator/java-rest-service
  accelerator/java-server-side-ui
  accelerator/spring-cloud-serverless

```

This shows the `options` and `importedBy` with a list of three accelerators that import this fragment.

Correspondingly, the `tanzu accelerator get <accelerator-name>` shows the fragments that an accelerator imports. Run:

```
tanzu accelerator get java-rest-service
```

The following output is shown:

```

name: java-rest-service
namespace: accelerator-system
description: A Spring Boot Restful web application including OpenAPI v3 document generation and database persistence, based on a three-layer architecture.
displayName: Tanzu Java Restful Web App
iconUrl: data:image/png;base64,...abbreviated...
source:
  image: dev.registry.tanzu.vmware.com/app-accelerator/samples/java-rest-service@sha256:c098bb38b50d8bbead0a1b1e9be5118c4fdce3e260758533c38487b39ae0922d
  secret-ref: [{reg-creds}]
tags:
- java
- spring
- web
- jpa
- postgresql
- tanzu
ready: true
options:
- defaultValue: customer-profile
  inputType: text
  label: Module artifact name
  name: artifactId
  required: true
- defaultValue: com.example
  inputType: text
  label: Module group name
  name: groupId
  required: true
- defaultValue: com.example.customerprofile
  inputType: text
  label: Module root package
  name: packageName

```

```

required: true
- defaultValue: customer-profile-database
  inputType: text
  label: Database Instance Name this Application will use (can be existing one in
    the cluster)
  name: databaseName
  required: true
- choices:
  - text: Maven (https://maven.apache.org/)
    value: maven
  - text: Gradle (https://gradle.org/)
    value: gradle
  defaultValue: maven
  inputType: select
  name: buildTool
  required: true
- choices:
  - text: Flyway (https://flywaydb.org/)
    value: flyway
  - text: Liquibase (https://docs.liquibase.com/)
    value: liquibase
  defaultValue: flyway
  inputType: select
  name: databaseMigrationTool
  required: true
- dataType: boolean
  defaultValue: false
  label: Expose OpenAPI endpoint?
  name: exposeOpenAPIEndpoint
- defaultValue: ""
  dependsOn:
    name: exposeOpenAPIEndpoint
  inputType: text
  label: System API Belongs To
  name: apiSystem
- defaultValue: ""
  dependsOn:
    name: exposeOpenAPIEndpoint
  inputType: text
  label: Owner of API
  name: apiOwner
- defaultValue: ""
  dependsOn:
    name: exposeOpenAPIEndpoint
  inputType: text
  label: API Description
  name: apiDescription
- choices:
  - text: Java 8
    value: "1.8"
  - text: Java 11
    value: "11"
  - text: Java 17
    value: "17"
  defaultValue: "11"
  inputType: select
  label: Java version to use
  name: javaVersion
  required: true
- dataType: boolean
  defaultValue: true
  dependsOn:
    name: buildTool
    value: maven
  inputType: checkbox
  label: Include TAP IDE Support for Java Workloads

```

```

name: liveUpdateIDESupport
- defaultValue: dev.local
dependsOn:
  name: liveUpdateIDESupport
description: The prefix for the source image repository where source can be stored
  during development
inputType: text
label: The source image repository prefix to use when pushing the source
name: sourceRepositoryPrefix
artifact:
  message: Resolved revision: dev.registry.tanzu.vmware.com/app-accelerator/samples/java-rest-service@sha256:c098bb38b50d8bbead0a1b1e9be5118c4fdce3e260758533c38487b39ae0922d
  ready: true
  url: http://source-controller-manager-artifact-service.source-system.svc.cluster.local./imagerepository/accelerator-system/java-rest-service-acc-wcn8x/c098bb38b50d8bbead0a1b1e9be5118c4fdce3e260758533c38487b39ae0922d.tar.gz
imports:
  java-version
  live-update
  tap-workload

```

The `imports` section at the end shows the fragments that this accelerator imports. The `options` section shows all options defined for this accelerator. This includes all options defined in the imported fragments, for example, the options for the Java version imported from the `java-version` fragment.

## Transforms reference

This topic provides you with a list and brief description of the available Application Accelerator transforms in Tanzu Application Platform (commonly known as TAP).

## Available transforms

You can use:

- [Combo](#) as a shortcut notation for many common operations. It combines the behaviors of many of the other transforms.
- [Include](#) to select files to operate on.
- [Exclude](#) to select files to operate on.
- [Merge](#) to work on subsets of inputs and to gather the results at the end.
- [Chain](#) to apply several transforms in sequence using function composition.
- [Let](#) to introduce new scoped variables to the model.
- [InvokeFragment](#) allows re-using various fragments across accelerators.
- [ReplaceText](#) to perform simple token replacement in text files.
- [RewritePath](#) to move files around using regular expression (regex) rules.
- [OpenRewriteRecipe](#) to apply [Rewrite](#) recipes, such as package rename.
- [YTT](#) to run the `ytt` tool on its input files and gather the result.
- [UseEncoding](#) to set the encoding to use when handling files as text.
- [UniquePath](#) to decide what to do when several files end up on the same path.
- [Loop](#) to iterate over a list and apply a transform for each element.

## See also

- [Conflict Resolution](#)

## Transforms reference

This topic provides you with a list and brief description of the available Application Accelerator transforms in Tanzu Application Platform (commonly known as TAP).

## Available transforms

You can use:

- [Combo](#) as a shortcut notation for many common operations. It combines the behaviors of many of the other transforms.
- [Include](#) to select files to operate on.
- [Exclude](#) to select files to operate on.
- [Merge](#) to work on subsets of inputs and to gather the results at the end.
- [Chain](#) to apply several transforms in sequence using function composition.
- [Let](#) to introduce new scoped variables to the model.
- [InvokeFragment](#) allows re-using various fragments across accelerators.
- [ReplaceText](#) to perform simple token replacement in text files.
- [RewritePath](#) to move files around using regular expression (regex) rules.
- [OpenRewriteRecipe](#) to apply [Rewrite](#) recipes, such as package rename.
- [YTT](#) to run the `ytt` tool on its input files and gather the result.
- [UseEncoding](#) to set the encoding to use when handling files as text.
- [UniquePath](#) to decide what to do when several files end up on the same path.
- [Loop](#) to iterate over a list and apply a transform for each element.

## See also

- [Conflict Resolution](#)

## Combo transform

This topic tells you about the Application Accelerator [Combo](#) transform in Tanzu Application Platform (commonly known as TAP).

The [Combo](#) transform combines the behaviors of [Include](#), [Exclude](#), [Merge](#), [Chain](#), [UniquePath](#), and [Let](#).

## Syntax reference

Here is the full syntax of [Combo](#):

```
type: Combo                                # This can be omitted, because Combo is the default transform type.
let:  # See Let.
- name: <string>
  expression: <SpEL expression>
```

```

- name: <string>
  expression: <SpEL expression>
condition: <SpEL expression>
include: [<ant pattern>] # See Include.
exclude: [<ant pattern>] # See Exclude.
merge: # See Merge.
- <m1-transform>
- <m2-transform>
- ...
chain: # See Chain.
- <c1-transform>
- <c2-transform>
- ...
applyTo: [<ant pattern>] # See Chain
onConflict: <conflict resolution> # See UniquePath.

```

## Behavior

The `Combo` transform properties have default values, are optional, and you must use at least one property.

When you configure the `Combo` transform with all properties, it behaves as follows:

1. Applies the `include` as if it were the first element of a `Chain`. The default value is `['**']`; if not present, all files are retained.
2. Applies the `exclude` as if it were the second element of the chain. The default value is `[]`; if not present, no files are excluded. Only files that match the `include`, but are not excluded by the `exclude`, remain.
3. Feeds all those files as input to all transforms declared in the `merge` property, exactly as `Merge` does. The result of that `Merge`, which is the third transform in the big chain, is another set of files. If there are no elements in `merge`, the previous result is directly fed to the next step.
4. The result of the merge step is prone to generate duplicate entries for the same `path`. It's implicitly forwarded to a `UniquePath` check, configured with the `onConflict` strategy. The default policy is to retain files appearing later. The results of the transforms that appear later in the `merge` block “win” against results appearing earlier.
5. Passes that result as the input to the `chain` defined by the `chain` property. The combo chain is prolonged with the elements defined in `chain`. If there are no elements in `chain`, it's as if the previous result was used directly. If the `applyTo` property is set, it applies to the sub-chain (and that sub-chain only).
6. If the `let` property is defined in the `Combo`, the whole execution is wrapped inside a `Let` that exposes its derived symbols.

To recap in pseudo code, a giant `Combo` behaves like this:

```

Let(symbols, in:
  Chain(
    include,
    exclude,
    Chain(Merge(<m1-transform>, <m2-transform>, ...), UniquePath(onConflict)),
    Chain(<applyTo>, <c1-transform>, <c2-transform>, ...)
  )
)

```

You rarely use at any one time all the features that `Combo` offers. Yet `Combo` is a good way to author other common building blocks without having to write their `type: x` in full.

For example, this:

```
include: ['**/*.txt']
```

is a perfectly valid way to achieve the same effect as this:

```
type: Include
patterns: ['**/*.txt']
```

Similarly, this:

```
chain:
- type: T1
  ...
- type: T2
  ...
```

is often preferred over the more verbose:

```
type: Chain
transformations:
- type: T1
  ...
- type: T2
  ...
```

As with other transforms, the order of declaration of properties has no impact. We've used a convention that mimics the actual behavior for clarity, but the following applies **T1** and **T2** on all `.yaml` files even though VMware has placed the `include` section after the `merge` section.

```
merge:
- type: T1
- type: T2
include: ["*.yaml"]
```

In other words, `Combo` applies `include` filters before `merge` irrespective of the physical order of the keys in the YAML text. It's a good practice to place the `include` key before the `merge` key. This makes the accelerator definition more readable, but has no effect on its execution order.

## Examples

The following are typical use cases for `Combo`.

To apply separate transformations to separate sets of files. For example, to all `.yaml` files and to all `.xml` files:

```
merge:                                # This uses the Merge syntax in a first Combo.
- include: ['*.yaml']                 # This actually nests a second Combo inside the first.
  chain:
    - type: T1
    - type: T2
- include: ['*.yaml']                 # Here comes a third Combo, used as the 2nd child inside
the first
  chain:
    - type: T3
    - type: T4
```

To apply **T1** then **T2** on all `.yaml` files that are not in any `secret` directory:

```
include: ['**/*.yaml']
exclude: ['**/secret/**']
chain:
```

```
- type: T1
  ..
- type: T2
  ..
```

## Include transform

This topic tells you about the Application Accelerator [Include](#) transform in Tanzu Application Platform (commonly known as TAP).

The [Include](#) transform retains files based on their [path](#), letting in *only* those files whose path matches at least one of the configured [patterns](#). The contents of files, and any of their other characteristics, are unaffected.

[Include](#) is a basic building block seldom used as is, which makes sense if composed inside a [Chain](#) or a [Merge](#). It is often more convenient to leverage the shorthand notation offered by [Combo](#).

## Syntax reference

```
type: Include
patterns: [<ant pattern>]
condition: <SpEL expression>
```

## Examples

```
type: Chain
transformations:
- type: Include
  patterns: ["**/*.yaml"]
- type: # At this point, only yaml files are affected
```

## See also

- [Exclude](#)
- [Combo](#)

## Exclude transform

This topic tells you about the Application Accelerator [Exclude](#) transform in Tanzu Application Platform (commonly known as TAP).

The [Exclude](#) transform retains files based on their [path](#), allowing all files except ones with a path that matches at least one of the configured [patterns](#). The contents of files, and any of their other characteristics are unaffected.

[Exclude](#) is a basic building block seldom used *as is*, which makes sense if composed inside a [Chain](#) or a [Merge](#). It is often more convenient to leverage the shorthand notation offered by [Combo](#).

## Syntax reference

```
type: Exclude
patterns: [<ant pattern>]
condition: <SpEL expression>
```



## Examples

```
type: Chain
transformations:
  - type: Exclude
    patterns: ["**/secret/**"]
  - type: # At this point, no file matching **/secret/** is affected.
```

## See also

- [Include](#)
- [Combo](#)

## Merge transform

This topic tells you about the Application Accelerator [Merge](#) transform in Tanzu Application Platform (commonly known as TAP).

The [Merge](#) transform feeds a copy of its input to several other transforms and merges the results together using set union.

A [Merge](#) of **T1**, **T2**, and **T3** applied to input **I** results in **T1(I) ∪ T2(I) ∪ T3(I)**.

An empty merge produces nothing ( $\emptyset$ ).

## Syntax reference

```
type: Merge
sources:
  - <transform>
  - <transform>
  - <transform>
  - ...
condition: <SpEL expression>
```

## See also

- [Combo](#) is often used to express a [Merge](#) **and** other transformations in a shorthand syntax.

## Chain transform

This topic tells you about the Application Accelerator [Chain](#) transform in Tanzu Application Platform (commonly known as TAP).

The [Chain](#) transform uses function composition to produce its final output.

## Syntax reference

```
type: Chain
transformations:
  - <transform>
  - <transform>
  - <transform>
  - ...
applyTo: [<ant pattern>]
condition: <SpEL expression>
```

## Behavior

A chain of **T1** then **T2** then **T3** first applies transform **T1**. It then applies **T2** to the output of **T1**, and finally applies **T3** to the output of that. In other words, **T3** to **T2** to **T1**.

An empty chain acts as function identity.

If the optional `applyTo` property is set, then the chained transformations are only applied to files with paths that match the `applyTo` patterns. Files with paths that don't match are left untouched and merged back with the other results to form the final result of the `Chain` transform.

## Let transform

This topic tells you about the Application Accelerator `Let` transform in Tanzu Application Platform (commonly known as TAP).

The `Let` transform wraps another transform, creating a new scope that extends the existing scope.

SpEL expressions inside the `Let` can access variables from both the existing scope and the new scope.

Variables defined by the `Let` should not shadow existing variables. If they do, those existing variables won't be accessible.

## Syntax reference

```
type: Let
symbols:
- name: <string>
  expression: <SpEL expression>
- ...
in: <transform> # <- new symbols are visible in here
```

## Execution

The `Let` adds variables to the new scope by computation of `SpEL expressions`.

```
engine:
  let:
  - name: <string>
    expression: <SpEL expression>
  - ...
```

Both a `name` and an `expression` must define each symbol where:

- `name` must be a camelCase string name. If a `let symbol` happens to have the same name as a symbol already defined in the surrounding scope, then the local symbol shadows the symbol from the surrounding scope. This makes the variable from the surrounding scope inaccessible in the remainder of the `Let` but doesn't alter its original value.
- `expression` must be a valid SpEL expression expressed as a YAML string. Be careful when using the `#` symbol for variable evaluation, because this is the comment marker in YAML. So SpEL expressions in YAML must enclose strings in quotes or rely on block style. For more information about block style, see [Block Style Productions](#).

Symbols defined in the `Let` are evaluated in the new scope in the order they are defined. This means that symbols lower in the list can make use of the variables defined higher in the list but not the other way around.

## See also

- [Combo](#) provides a way to declare a `Let` scope and other transforms in a short syntax.

## Loop transform

This topic tells you about the Application Accelerator `Loop` transform in Tanzu Application Platform (commonly known as TAP).

The `Loop` transform iterates over elements in a list and applies the provided transform for every element in that list.

When `doAsMerge` is used, a copy of the `Loop` transform's input is passed to each transform and the outputs from each transform are merged using a set union.

When `doAsChain` is used, each transform is executed sequentially, receiving the previous transform's output as its input. The first transform is to receive the `Loop` transform's input as its input.

## Syntax reference

```
type: Loop
on: <SpEL expression>
var: <string>
index: <string>
doAsChain: <transform>
doAsMerge: <transform>
```

- `on` must be a SpEL expression that evaluates a list. This is the list of elements to be iterated over.
- `var` is the name of the variable to be assigned to the current element on each iteration. (optional)
- `index` is the variable's name to be assigned to the index of the current element on each iteration. (optional)
- `doAsMerge` is the transform to be executed for every element in the list, on a copy of the `Loop` transform's input.
- `doAsChain` is the transform to be executed for every element in the list, passing the output of the transform as input to the next transform.

Both `var` and `index` are optional.

Only one of the `doAsMerge` or `doAsChain` variables is to be used in a `Loop` transform.

## Behavior

Consider the following when choosing `doAsMerge` or `doAsChain`:

`doAsMerge` executes the transform on the same input files for every iteration and merges the resulting outputs. It is best suited when a transform is executed multiple times on the same input and does not have conflicts.

`doAsChain` executes the transform on the initial input files once and then passes the resulting output to the second iteration and so on. It is best suited when a transform must detect any changes that occurred in the previous iteration.

## Examples

Create a new directory for every module in `modules` (a list of strings) based on the contents of the “template” directory.

```
type: Loop
on: "#modules"
var: m
doAsMerge:
  type: RewritePath
  regex: "template/(.*)"
  rewriteTo: "#m + '/' + #g1"
```

Add every artifactId in `artifacts` (a list of strings) as a Spring plug-in.

```
type: Loop
on: "#artifacts"
var: a
doAsChain:
  type: OpenRewriteRecipe
  recipe: org.openrewrite.maven.AddPlugin
  options:
    groupId: "'org.springframework'"
    artifactId: "#a"
    version: "'5.7.1'"
```

You can use `Loop` in combination with custom types, for example:

```
accelerator:
  types:
    - name: MavenPlugin
      struct:
        - name: groupId
        - name: artifactId
        - name: version
      options:
        - name: pluginsToAdd
          dataType: [MavenPlugin] # End users will be able to enter a collection of GAV tuples
  engine:
    include: [pom.xml]
    chain:
      - type: Loop
        on: pluginsToAdd # Iterate on the pluginsToAdd collection
        var: p           # The variable "p" will contain each tuple in turn
        doAsChain:      # Will apply the second execution to the result of the first, and so on...
          type: OpenRewriteRecipe
          recipe: org.openrewrite.maven.AddPlugin
          options:
            groupId: "#p['groupId']"
            artifactId: "#p['artifactId']"
            version: "#p['version']"
```

For more information, see [Using Custom Types](#).

## InvokeFragment transform

This topic tells you about the Application Accelerator `InvokeFragment` transform in Tanzu Application Platform (commonly known as TAP).

The `InvokeFragment` performs transformations defined in an imported Fragment, allowing re-use across accelerators.

## Syntax reference

```

type: InvokeFragment
reference: <imported-fragment>
let: # See Let
  - name: <string>
    expression: <SpEL expression>
  ...
anchor: [<file path>]

```

## Behavior

Assuming some fragment `my-fragment` has been imported in the accelerator (thus exposing the options it defines as options of the current accelerator), the following construct invokes `my-fragment`:

```

type: InvokeFragment
reference: my-fragment

```

This passes all input files (depending where this invocation sits in the “tree”) to the invoked fragment, which can then manipulate them alongside its own files. The result of the invocation becomes the result of this transform.

## Variables

At the point of invocation, all currently defined variables are made visible to the invoked fragment. Therefore, if it was `import`-ed in the most straightforward manner, a fragment defining an option `myOption` is defining an option named `myOption` at the accelerator level, and the value provided by the user is visible at the time of invocation.

To override a value, or if an imported option has been exposed under a different name, or not at all, you can use a `let` construct when using `InvokeFragment`. This behaves as the `Let` transform: for the duration of the fragment invocation, the variables defined by `let` now have their newly defined values. Outside the scope of the invocation, the regular model applies.

## </a/>Files

The set of files coming from the invoking accelerator and made visible to the fragment is the set of files that “reach” the point of invocation. For example, in the following case:

```

include: ["somedir/**"]
chain:
  - type: InvokeFragment
    reference: my-fragment

```

All files that the fragment invocation “sees” are files in the `somedir/` subdirectory. If the `my-fragment` has not been written accordingly, this can be problematic. Chances are that this re-usable fragment expects files to be present at the root of the project tree and work on them.

To better cope with this typical situation, the `InvokeFragment` transform exposes the optional `anchor` configuration property. Continuing with the earlier example, by using `anchor: somedir`, then all files coming from the current accelerator are exposed as if their `path` had the `somedir/` prefix removed. When it comes to gathering the result of the invocation though, all resulting files are re-introduced with a prefix prepended to their `path` (this applies to **all** files produced by the fragment, not just the ones originating from the accelerator).

The value of the `anchor` property must not start nor end with a slash (/) character.

## Examples

Let's start with a full-featured example showcasing the interaction between the `imports` section and `InvokeFragment`

```

accelerator:
  name: my-accelerator
  options:
    - name: someOption
      dataType: number
  imports:
    - name: my-fragment

engine:
  merge:
    - include: ["..."]
    - ...
    - chain:
      - include: ["**/pom.xml"]
      - type: InvokeFragment
        reference: my-fragment
  
```

Assuming `my-fragment` is defined like so

```

accelerator:
  name: my-fragment
  options:
    - name: indentationLevel
      dataType: number
      defaultValue: 2
  transform:
  chain:
    - include: ["**/*.xml"]
    - type: SomeTransform
    ...
  
```

Then users will be presented with two options: `someOption` and `indentationLevel`, as if `indentationLevel` was defined in the host accelerator.

Moreover, the behavior of the calling accelerator is exactly as if the body of the fragment transform was inserted in-place of `InvokeFragment`:

```

accelerator:
  name: my-accelerator
  options:
    - name: someOption
      dataType: number
    - name: indentationLevel
      dataType: number
      defaultValue: 2

engine:
  merge:
    - include: ["..."]
    - ...
    - chain:
      - include: ["**/pom.xml"]
      - chain:
        - include: ["**/*.xml"]
        - type: SomeTransform
        ...
  
```

Now you can imagine some scenarios to better clarify all configuration properties.

You can pretend, for some reason, that you don't want to use the value entered in the `indentationLevel` option for the fragment, but twice the value provided for `someOption`. The `InvokeFragment` block can be rewritten such as this:

```
type: InvokeFragment
reference: my-fragment
let:
  - name: indentationLevel
    value: '2 * #someOption'
```

Now for some other crazy example to better explain the interactions. If the invocation in the accelerator looked like this:

```
engine:
  merge:
    - include: ["..."]
    - ...
    - chain:
      - include: ["**/README.md"]
      - type: InvokeFragment
        reference: my-fragment
```

Then there is absolutely zero visible effect, because this is forwarding only `README.md` files to the fragment and the fragment is itself using a filter on `*.xml` files.

## See also

- [Let](#)
- [RewritePath](#)

## ReplaceText transform

This topic tells you about the Application Accelerator `ReplaceText` transform in Tanzu Application Platform (commonly known as TAP).

The `ReplaceText` transform allows replacing one or several text tokens in files as they are being copied to their destination. The replacement values are the result of dynamic evaluation of [SpEL expressions](#).

This transform is text-oriented and requires knowledge of how to interpret the stream of bytes that make up the file contents into text. All files are assumed to use `UTF-8` encoding by default, but you can use the [UseEncoding](#) transform upfront to specify a different charset to use on some files.

You can use `ReplaceText` transform in one of two ways:

- To replace several literal text tokens.
- To define the replacement behavior using a single regular expression, in which case the replacement SpEL expression can leverage the regex capturing group syntax.

## Syntax reference

Syntax reference for replacing several literal text tokens:

```
type: ReplaceText
substitutions:
  - text: STRING
```

```

    with: SPEL-EXPRESSION
  - text: STRING
    with: SPEL-EXPRESSION
  - ..
condition: SPEL-EXPRESSION

```

Syntax reference for defining the replacement behavior using a *single* regular expression:

Regex is used to match the entire document. To match on a per line basis, enable multiline mode by including (?m) in the regex.

```

type: ReplaceText
regex:
  pattern: REGULAR-EXPRESSION
  with: SPEL-EXPRESSION
condition: SPEL-EXPRESSION

```

In both cases, the SpEL expression can use the special `#files` helper object. This enables the replacement string to consist of the contents of an accelerator file. See the following [example](#).

Another set of helper objects are functions of the form `xxx2Yyyy()` where `xxx` and `yyy` can take the value `camel`, `kebab`, `pascal`, or `snake`. For example, `camel2Snake()` enables changing from `camelCase` to `snake_case`.

## Examples

Replacing the hardcoded string `"hello-world-app"` with the value of variable `#artifactId` in all `.md`, `.xml`, and `.yaml` files.

```

include: ['**/*.md', '**/*.xml', '**/*.yaml']
chain:
  - type: ReplaceText
    substitutions:
      - text: "hello-world-app"
        with: "#artifactId"

```

Doing the same in the `README-fr.md` and `README-de.md` files, which are encoded using the `ISO-8859-1` charset:

```

include: ['README-fr.md', 'README-de.md']
chain:
  - type: UseEncoding
    encoding: 'ISO-8859-1'
  - type: ReplaceText
    substitutions:
      - text: "hello-world-app"
        with: "#artifactId"

```

Similar to the preceding example, but making sure the value appears as kebab case, while the entered `#artifactId` is using camel case:

```

include: ['**/*.md', '**/*.xml', '**/*.yaml']
chain:
  - type: ReplaceText
    substitutions:
      - text: "hello-world-app"
        with: "#camel2Kebab(#artifactId)"

```

Replacing the hardcoded string `"REPLACE-ME"` with the contents of file named after the value of the `#platform` option in `README.md`:



```
include: ['README.md']
chain:
  - type: ReplaceText
    substitutions:
      - text: "REPLACE-ME"
        with: "#files.contentsOf('snippets/install-' + #platform + '.md')"
```

## See also

- [UseEncoding](#)

## RewritePath transform

This topic tells you about the Application Accelerator [RewritePath](#) transform in Tanzu Application Platform (commonly known as TAP).

The [RewritePath](#) transform allows you to change the name and path of files without affecting their content.

## Syntax reference

```
type: RewritePath
regex: <string>
rewriteTo: <SpEL expression>
matchOrFail: <boolean>
```

For each input file, [RewritePath](#) attempts to match its `path` by using the regular expression (regex) defined by the `regex` property. If the regex matches, [RewritePath](#) changes the `path` of the file to the evaluation result of `rewriteTo`.

`rewriteTo` is an expression that has access to the overall engine model and to variables defined by capturing groups of the regular expression. Both *named capturing groups* (`?<example>[a-z]*`) and regular *index-based* capturing groups are supported. `g0` contains the whole match, `g1` contains the first capturing group, and so on.

If the regex doesn't match, the behavior depends on the `matchOrFail` property:

- If set to `false`, which is the default, the file is left untouched.
- If set to `true`, an error occurs. This prevents misconfiguration if you expect all files coming in to match the regex. For more information about typical interactions between [RewritePath](#) and [Chain + Include](#), see the following section, [Interaction with Chain and Include](#).

The default value for `regex` is the following regular expression, which provides convenient access to some named capturing groups:

```
^(?<folder>.*\/)?(?<filename>([^\]+?)|)(?=(?<ext>\.[^\/.]*?)?)$)
```

Using `some/deep/nested/file.xml` as an example, the preceding regular expression captures:

- **folder:** The full folder path the file is in. In this example, `some/deep/nested/`.
- **filename:** The full name of the file, including extension *if present*. In this example, `file.xml`.
- **ext:** The last dot and extension in the filename, *if present*. In this example, `.xml`.

The default value for `rewriteTo` is the expression `#folder + #filename`, which doesn't rewrite paths.

## Examples

The following moves all files from `src/main/java` to `sub-module/src/main/java`:

```
type: RewritePath
regex: src/main/java/(.*)
rewriteTo: "'sub-module/src/main/java' + #g1" # 'sub-module/' + #g0 works too
```

The following flattens all files found inside the `sub-path` directory and its subdirectories, and puts them into the `flattened` folder:

```
type: RewritePath
regex: sub-path/(.*)*(?<filename>[^/]+)
rewriteTo: "'flattened' + #filename" # 'flattened' + #g2 would work too
```

The following turns all paths into lowercase:

```
type: RewritePath
rewriteTo: "#g0.toLowerCase()"
```

## Interaction with Chain and Include

It's common to define pipelines that perform a `Chain` of transformations on a subset of files, typically selected by `Include/Exclude`:

```
- include: "**/*.java"
- chain:
  - # do something here
  - # and then here
```

If one of the transformations in the chain is a `RewritePath` operation, chances are you want the rewrite to apply to *all* files matched by the `Include`. For those typical configurations, you can set the `matchOrFail` guard to `true` to ensure the `regex` you provide indeed matches all files coming in.

## See also

- Use [UniquePath](#) to ensure rewritten paths don't clash with other files, or to decide which path to select if they do clash.

## OpenRewriteRecipe transform

This topic tells you about the Application Accelerator `OpenRewriteRecipe` transform in Tanzu Application Platform (commonly known as TAP).

The `OpenRewriteRecipe` transform allows you to apply any `Open Rewrite Recipe` to a set of files and gather the results.

The following Open Rewrite Recipes are supported:

- [Java recipes](#)
- [Maven recipes](#)
- [XML recipes](#)
- [YAML recipes](#)
- [JSON recipes](#)
- [Properties recipes](#)

The engine leverages v7.30.1 of Open Rewrite and parses Java files using the grammar for Java 11.

## Syntax reference

```

type: OpenRewriteRecipe
recipe: <string> # Full qualified classname of the recipe
options:
  <string>: <SpEL expression> # Keys and values depend on the class of the recipe
  <string>: <SpEL expression> # Refer to the documentation of said recipe
  ...

```

## Example

The following example applies the [ChangePackage](#) Recipe to a set of Java files in the `com.acme` package and moves them to the value of `#companyPkg`. This is more powerful than using [RewritePath](#) and [ReplaceText](#), as it reads the syntax of files and correctly deals with imports, fully compared to non-fully qualified names, and so on.

```

chain:
  - include: ["**/*.java"]
  - type: OpenRewriteRecipe
    recipe: org.openrewrite.java.ChangePackage
    options:
      oldPackageName: "com.acme"
      newPackageName: "#companyPkg"

```

## YTT transform

This topic tells you about the Application Accelerator [YTT](#) transform in Tanzu Application Platform (commonly known as TAP).

The [YTT](#) transform starts the [YTT](#) template engine as an external process.

## Syntax reference

```

type: YTT
extraArgs: # optional
  - <SPEL-EXPRESSION-1>
  - <SPEL-EXPRESSION-2>
  - ...

```

The [YTT](#) transform’s YAML notation does not require any parameters. When invoked without parameters, which is the typical use case, the YTT transform’s input is determined entirely by two things only:

1. The input files fed into the transform.
2. The current values for options and derived symbols.

## Execution

YTT is invoked as an external process with the following command line:

```

ytt -f <input-folder> \
  --data-values-file <symbols.json> \
  --output-files <output-folder> \
  <extra-args>

```

The `<input-folder>` is a temporary directory into which the input files are “materialized.” That is, the set of files passed to the YTT transform as input is written out into this directory to allow the

YTT process to read them.

The `<symbols.json>` file is a temporary JSON file, which the current option values and derived symbols are materialized in the form of a JSON map. This allows YTT templates in the `<input-folder>` to make use of these symbols during processing.

The `<output-folder>` is a fresh temporary directory that is empty at the time of invocation. In a typical scenario, upon completion, the output directory contains files generated by YTT.

The `<extra-args>` are additional command line arguments obtained by evaluating the SPEL expressions from the `extraArgs` attribute.

When the `ytt` process completes with a 0 exit code, this is considered a successful execution and the contents of the output directory is taken to be the result of the YTT transform.

When the `ytt` process completes with a non 0 exit code, the execution of the YTT transform is considered to have failed and an exception is raised.

## Examples

### Basic invocation

When you want to execute `ytt` on the contents of the entire accelerator repository, use the YTT transform as your only transform in the engine declaration.

```
accelerator:
  ...
engine:
  type: YTT
```

To do anything beyond calling YTT, compose YTT into your accelerator flow using merge or chain combinators. This is exactly the same as composing any other type of transform.

For example, when you want to define some derived symbols as well as merge the results from YTT with results from other parts of your accelerator transform, you can reference this example:

```
engine:
  let: # Define derived symbols visible to all transforms (including YTT)
    - name: theAnswer
      expression: "41 + 1"
  merge:
    - include: ["deploy/**/*.yaml"] # select some yaml files to process with YTT
      chain: # Chain selected yaml files to YTT
        - type: YTT
    - ... include/generate other stuff to be merged alongside yaml generated by YTT...
```

The preceding example uses a combination of [Chain](#) and [Merge](#). You can use either [Merge](#) or [Chain](#) or both to compose YTT into your accelerator flow. Which one you choose depends on how you want to use YTT as part of your larger accelerator.

### Using `extraArgs`

The `extraArgs` passes additional command line arguments to YTT. This adds file marks. See [File Marks](#) in the Carvel documentation.

For example, the following runs YTT and renames the `foo/demo.yaml` file in its output to `bar/demo.yaml`.

```
engine:
  type: YTT
  extraArgs: ["--file-mark", "'foo/demo.yaml:path=bar/demo.yaml'"]
```

The `extraArgs` attribute expects SPEL expressions. Take care to use proper escaping of literal strings using double and single quotes (that is, `""LITERAL-STRING""`).

## UseEncoding transform

This topic tells you about the Application Accelerator `UseEncoding` transform in Tanzu Application Platform (commonly known as TAP).

When considering files in textual form, for example, when doing text replacement with the `ReplaceText` transform, the engine must decide which `encoding` to use.

By default, `UTF-8` is assumed. If any files must be handled differently, use the `UseEncoding` transform to annotate them with an explicit encoding.

`UseEncoding` returns an error if you apply encoding to files that have already been explicitly configured with a particular encoding.

## Syntax reference

```
type: UseEncoding
encoding: <encoding>      # As recognized by the java java.nio.charset.Charset class
condition: <SpEL expression>
```

Supported encoding names include, for example, `UTF-8`, `US-ASCII`, and `ISO-8859-1`.

## Example use

`UseEncoding` is typically used as an upfront transform to, for example, `ReplaceText` in a chain:

```
type: Chain      # Or using "Combo"
transformations:
- type: UseEncoding
  encoding: ISO-8859-1
- type: ReplaceText
  substitutions:
  - text: "hello"
    with: "#howToSayHello"
```

## See also

- [ReplaceText](#)

## UniquePath transform

This topic tells you about the Application Accelerator `UniquePath` transform in Tanzu Application Platform (commonly known as TAP).

You can use the `UniquePath` transform to ensure there are no `path` conflicts between files transformed. You can often use this at the tail of a `Chain`.

## Syntax reference

```
type: UniquePath
strategy: <conflict resolution>
condition: <SpEL expression>
```

## Examples

The following example concatenates the file that was originally named `DEPLOYMENT.md` to the file `README.md`:

```
chain:
  - merge:
    - include: ['README.md']
    - include: ['DEPLOYMENT.md']
      chain:
        - type: RewritePath
          rewriteTo: "README.md"
  - type: UniquePath
    strategy: Append
```

## See also

- `UniquePath` uses a [Conflict Resolution](#) strategy to decide what to do when several input files use the same `path`.
- `Combo` implicitly embeds a `UniquePath` after the `Merge` defined by its `merge` property.

## Conflict resolution

This topic tells you how to resolve conflicts that Application Accelerator transforms in Tanzu Application Platform (commonly known as TAP) might produce.

For example, if you're using `Merge` (or `Combo`'s `merge` syntax) or `RewritePath`, a transform can produce several files at the same `path`. The engine then must take an action: Should it keep the last file? Report an error? Concatenate the files together?

Such conflicts can arise for a number of reasons. You can avoid or resolve them by configuring transforms with a *conflict resolution*. For example:

- `Combo` uses `UseLast` by default, but you can configure it to do otherwise.
- You can explicitly end a transform `Chain` with a `UniquePath`, which by default uses `Fail`. This is customizable.

## Syntax reference

```
type: Combo      # often omitted
merge:
  - <transform>
chain:
  - <transform>
  - ...
onConflict: <conflict resolution> # defaults to 'UseLast'
```

```
type: Chain      # or implicitly using Combo
transformations:
  - <transform>
  - <transform>
  - type: UniquePath
    strategy: <conflict resolution> # defaults to 'Fail'
```

## Available strategies

The following values and behaviors are available:

- **Fail**: Stop processing on the first file that exhibits `path` conflicts.
- **UseFirst**: For each conflicting file, the file produced first (typically by a transform appearing earlier in the YAML definition) is retained.
- **UseLast**: For each conflicting file, the file produced last (typically by a transform appearing later in the YAML definition) is retained.
- **Append**: The conflicting versions of files are concatenated (as if using `cat file1 file2 ...`), with files produced first appearing first.
- **FavorOwn**: *Only makes sense in the context of `composition`*. Selects the version of the file that comes from the current executing fragment if possible, falls back to the caller version otherwise.
- **FavorForeign**: *Only makes sense in the context of `composition`*. Selects the version of the file that was provided by the caller if present, falls back to the file originating from this fragment's fileset otherwise.

## See also

- [Combo](#)
- [UniquePath](#)

## Use SpEL with Application Accelerator

This topic tells you about some common Spring Expression Language (SpEL) use cases for the `accelerator.yaml` file in Application Accelerator.

For more information, see [Spring Expression Language](#) documentation.

## Variables

You can reference all the values added as options in the `accelerator` section from the YAML file as variables in the `engine` section. You can access the value using the syntax `#<option name>`:

```
options:
- name: foo
  dataType: string
  inputType: text
...
engine:
- include: ["some/file.txt"]
  chain:
- type: ReplaceText
  substitutions:
- text: bar
  with: "#foo"
```

This sample replaces every occurrence of the text `bar` in the file `some/file.txt` with the contents of the `foo` option.

## Implicit variables

Some variables are made available to the model by the engine, including:

- `artifactId` is a built-in value derived from the `projectName` passed in from the UI with spaces replaced by “\_”. If that value is empty, it is set to `app`.

- `files` is a helper object that currently exposes the `contentsOf(<path>)` method. For more information, see [ReplaceText](#).
- `camel2Kebab` and other variations of the form `xxx2Yyyy` is a series of helper functions for dealing with changing case of words. For more information, see [ReplaceText](#).

## Conditionals

You can use Boolean options for conditionals in your transformations.

```
options:
  - name: numbers
    inputType: select
    choices:
      - text: First Option
        value: first
      - text: Seconf Option
        value: second
    defaultValue: first
  ...
engine:
  - include: ["some/file.txt"]
    condition: "#numbers == 'first'"
    chain:
      - type: ReplaceText
        substitutions:
          - text: bar
            with: "#foo"
```

This replaces the text only if the selected option is the first one.

## Rewrite path concatenation

```
options:
  - name: renameTo
    dataType: string
    inputType: text
  ...
engine:
  - include: ["some/file.txt"]
    chain:
      - type: RewritePath
        rewriteTo: "'somewhere/' + #renameTo + '.txt'"
```

## Regular expressions

Regular expressions allow you to use patterns as a matcher for strings. Here is a small example of what you can do with them:

```
options:
  - name: foo
    dataType: string
    inputType: text
    defaultValue: abcZ123
  ...
engine:
  - include: ["some/file.txt"]
    condition: "#foo.matches('[a-z]+Z\d+')"
    chain:
      - type: ReplaceText
        substitutions:
```



```
- text: bar
  with: "#foo"
```

This example uses RegEx to match a string of letters that ends with a capital Z and any number of digits. If this condition is fulfilled, the text is replaced in the file, `file.txt`.

## Dealing with string arrays

Options with a `dataType` of `[string]` come out as an array of strings.

"To use them and for example format the result as a bulleted list, you can use the Java static `String.join()` method. For example:

```
accelerator:
  options:
    - name: meals
      dataType: [string]
      inputType: checkbox
      choices:
        - value: fish
        - value: chips
        - value: BLT
    ...
  engine:
    type: ReplaceText
    substitutions:
      - text: recipe
        with: "' * ' + T(java.lang.String).join('\n * ', #meals)"
```

## Accelerator custom resource definition

This topic tells you about the Application Accelerator custom resource definition.

### The `Accelerator` custom resource definition (CRD) defines any accelerator resources to be made

available to the Application Accelerator for VMware Tanzu system. It is a namespaced CRD, meaning that any resources created belong to a namespace. For the resource to be available to the Application Accelerator system, it must be created in the namespace that the Application Accelerator UI server is configured to watch.

The `Fragment` custom resource definition (CRD) defines any accelerator fragment resources to be made available to the Application Accelerator for VMware Tanzu system. It is a namespaced CRD, meaning that any resources created belong to a namespace. For the resource to be available to the Application Accelerator system, it must be created in the namespace that the Application Accelerator UI server is configured to watch.

## API definitions

The `Accelerator` CRD is defined with the following properties:

| Property | Value                             |
|----------|-----------------------------------|
| Name     | Accelerator                       |
| Group    | accelerator.apps.tanzu.vmware.com |
| Version  | v1alpha1                          |

| Property  | Value |
|-----------|-------|
| ShortName | acc   |

## Accelerator CRD Spec

The `Accelerator` CRD `spec` defined in the `AcceleratorSpec` type has the following fields:

| Field                   | Description                                                                                                                                                                                                                                                                                | Required/Optional |
|-------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|
| displayName             | A short descriptive name used for an Accelerator.                                                                                                                                                                                                                                          | Optional (*)      |
| description             | A longer description of an Accelerator.                                                                                                                                                                                                                                                    | Optional (*)      |
| iconUrl                 | A URL for an image to represent the Accelerator in a UI.                                                                                                                                                                                                                                   | Optional (*)      |
| tags                    | An array of strings defining attributes of the Accelerator that can be used in a search.                                                                                                                                                                                                   | Optional (*)      |
| git                     | Defines the accelerator source Git repository.                                                                                                                                                                                                                                             | Optional (***)    |
| git.url                 | The repository URL, can be a HTTP/S or SSH address.                                                                                                                                                                                                                                        | Optional (***)    |
| git.gitImplementation   | Determines which git client library to use. The default setting is to go-git. Valid values are ('go-git', 'libgit2').                                                                                                                                                                      | Optional (**)     |
| git.ignore              | Overrides the set of excluded patterns in the .sourceignore format (which is the same as .gitignore). If not provided, a default of .git/ is used.                                                                                                                                         | Optional (**)     |
| git.interval            | The interval at which to inquire for repository updates. If not provided the default setting is 10 minutes. There is an additional refresh interval (currently 10s) involved before accelerators can appear in the UI. There might be a 10s delay before changes are reflected in the UI.* | Optional (**)     |
| git.ref                 | Git reference to checkout and monitor for changes, the default is main branch.                                                                                                                                                                                                             | Optional (**)     |
| git.ref.branch          | The Git branch to checkout, the default is main.                                                                                                                                                                                                                                           | Optional (**)     |
| git.ref.commit          | The Git commit SHA to checkout, if specified tag filters are ignored.                                                                                                                                                                                                                      | Optional (**)     |
| git.ref.semver          | The Git tag semver expression, takes precedence over tag.                                                                                                                                                                                                                                  | Optional (**)     |
| git.ref.tag             | The Git tag to checkout, takes precedence over branch.                                                                                                                                                                                                                                     | Optional (**)     |
| git.secretRef           | The secret name containing the Git credentials. For HTTPS repositories, the secret must contain user name and password fields. For SSH repositories, the secret must contain identity, identity.pub, and known_hosts fields.                                                               | Optional (**)     |
| git.subPath             | SubPath is the folder inside the git repository to consider as the root of the accelerator or fragment. Defaults at the root of the repository.                                                                                                                                            | Optional          |
| source                  | Defines the source image repository.                                                                                                                                                                                                                                                       | Optional (***)    |
| source.image            | Image is a reference to an image in a remote registry.                                                                                                                                                                                                                                     | Optional (***)    |
| source.imagePullSecrets | ImagePullSecrets contains the names of the Kubernetes Secrets containing registry login information to resolve image metadata                                                                                                                                                              | Optional          |
| source.interval         | The interval at which to check for repository updates.                                                                                                                                                                                                                                     | Optional          |

| Field                     | Description                                                                                                                                          | Required/Optional |
|---------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|
| source.serviceAccountName | ServiceAccountName is the name of the Kubernetes ServiceAccount used to authenticate the image pull if the service account has attached pull secrets | Optional          |

The `Fragment` CRD is defined with the following properties:

| Property  | Value                             |
|-----------|-----------------------------------|
| Name      | Fragment                          |
| Group     | accelerator.apps.tanzu.vmware.com |
| Version   | v1alpha1                          |
| ShortName | frag                              |

## Fragment CRD Spec

The `Fragment` CRD *spec* defined in the `FragmentSpec` type has the following fields:

| Field          | Description                                                                                                                                                                                                                  | Required/Optional |
|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|
| displayName    | DisplayName is a short descriptive name used for a Fragment.                                                                                                                                                                 | Optional          |
| git            | Defines the fragment source Git repository.                                                                                                                                                                                  | Required          |
| git.url        | The repository URL, can be a HTTP/S or SSH address.                                                                                                                                                                          | Required          |
| git.ignore     | Overrides the set of excluded patterns in the <code>.sourceignore</code> format (which is the same as <code>.gitignore</code> ). If not provided, a default of <code>.git/</code> is used.                                   | Optional (**)     |
| git.interval   | The interval at which to inquire for repository updates. If not provided the default is 10 min.                                                                                                                              | Optional (**)     |
| git.ref        | Git reference to checkout and monitor for changes, the default is main branch.                                                                                                                                               | Optional (**)     |
| git.ref.branch | The Git branch to checkout, defaults to main.                                                                                                                                                                                | Optional (**)     |
| git.ref.commit | The Git commit SHA to checkout, if specified tag filters are ignored.                                                                                                                                                        | Optional (**)     |
| git.ref.semver | The Git tag semver expression, takes precedence over tag.                                                                                                                                                                    | Optional (**)     |
| git.ref.tag    | The Git tag to checkout, takes precedence over branch.                                                                                                                                                                       | Optional (**)     |
| git.secretRef  | The secret name containing the Git credentials. For HTTPS repositories, the secret must contain user name and password fields. For SSH repositories, the secret must contain identity, identity.pub, and known_hosts fields. | Optional (**)     |
| git.subPath    | SubPath is the directory inside the Git repository to consider as the root of the accelerator or fragment. Defaults at the root of the repository.                                                                           | Optional          |

\* Any optional text boxes marked with an asterisk (\*) are populated from a text box of the same name in the `accelerator` definition in the `accelerator.yaml` file if that is present in the Git repository for the accelerator.

\*\* Any fields marked with a double asterisk (\*\*) are part of the Flux GitRepository CRD that is documented in the Flux Source Controller [Git Repositories](#) documentation.

\*\*\* Any fields marked with a triple asterisk (\*\*\*) are optional but either `git` or `source` is required to specify the repository to use. If `git` is specified, the `git.url` is required, and if `source` is specified,

`source.image` is required.

## Excluding files

The `git.ignore` field defaults to `.git/`, which is different from the defaults provided by the Flux Source Controller GitRepository implementation. You can override this, and provide your own exclusions. For more information, see [fluxcd/source-controller Excluding files](#).

## Test accelerators in Application Accelerator

This topic tells you how to test an updated accelerator, or fragment that is not registered in your Tanzu Application Platform (commonly known as TAP) cluster.

## Generating a project from local sources

When you are authoring your accelerator, you can test it before committing any changes.

With the `tanzu accelerator generate-from-local` command, you can run your accelerator (or fragment), including any changes you have locally, specify a set of options and view the generated project.

You can run the accelerator using the components on your Tanzu Application Platform cluster, without impacting the state of the Tanzu Application Platform cluster.

To do so, ensure that you have the following prerequisites:

- The Tanzu CLI is installed, with the Application Accelerator plug-in. For details about installing the Tanzu CLI and plug-ins, see [Tanzu CLI](#).
- The server URL is pointing to the Tanzu Application Platform cluster you want to test with. For details about setting the server URL, see [Application Accelerator CLI plug-in overview](#).

For example, to use the accelerator that is located at the path `workspace/java-rest`:

```
tanzu accelerator generate-from-local --accelerator-path java-rest=workspace/java-rest
--fragment-names tap-workload,java-version --options '{"projectName":"test"}' --output
-dir generated-project
```

This generates the project in the local directory `generated-project`, using the accelerator located at `workspace/java-rest`, the fragments `tap-workload` and `java-version` which are assumed to be already registered in the Tanzu Application Platform cluster and the option `projectName` set to `test`.

For example, to use the fragment named `java-version` that is located at the path `workspace/version`:

```
tanzu accelerator generate-from-local --accelerator-name java-rest --fragment-paths ja
va-version=workspace/version --fragment-names tap-workload --options '{"projectNam
e":"test"}' --output-dir generated-project
```

This generates the project in the local directory `generated-project`, using the accelerator `java-rest` and the fragment `tap-workload` which are assumed to be already registered in the Tanzu Application Platform cluster, the fragment named `java-version` located at `workspace/version`, and the option `projectName` set to `test`.

For the full documentation for the `generate-from-local` command, see reference [Tanzu accelerator generate-from-local](#).

No changes are made to the Tanzu Application Platform cluster that is provided with the server URL. No new accelerators/fragments are registered or modified. A Tanzu Application Platform cluster is required to ensure that there is consistency between the version that is used for testing

and the version that is used when the accelerator is registered. Furthermore, it allows using registered fragments and accelerators as dependencies for the local accelerator/fragment.

## CI/CD Pipeline

As you iterate on an accelerator, you can have some automated assertions run before any changes to the accelerator are accepted.

The process for generating a project from the committed source files is the same as described earlier.

When the generated project is available, you can run various assertions on it:

```
cd generated-project
test -f build.gradle
./gradlew test
```

If you have multiple assertions, you might choose to run a predefined script:

```
cd generated-project
../assertions/validate-generate-project.sh
```

You might choose to generate multiple projects from the same accelerator, providing different options for each and running different assertions on each generated project.

### (Optional) Getting the Tanzu CLI in a CI/CD pipeline

If the Tanzu CLI is already available in your CI/CD pipeline you can skip this section.

VMware provides an example script that is agnostic to the CI/CD system it is running on. The script requires a variable named `TANZU_REFRESH_TOKEN` which holds a personal VMware Tanzu Network refresh token. To generate such a token see [How to Authenticate](#). The script also uses `curl` and `jq`.

The script downloads artifacts compatible with Tanzu Application Platform version v1.4 and a Linux operating system. Update the script to suit the Tanzu Application Platform version and OS that you are using.

```
#!/bin/bash

# Get access token using personal Tanzu Network refresh token
# See https://network.tanzu.vmware.com/docs/api#how-to-authenticate
ACCESS_TOKEN=$(curl -X POST https://network.tanzu.vmware.com/api/v2/authentication/access_tokens -d '{"refresh_token":"'${TANZU_REFRESH_TOKEN}'"' | jq -r ".access_token")

# Download bundle
# See https://docs.vmware.com/en/VMware-Tanzu-Application-Platform/1.4/tap/GUID-install-tanzu-cli.html#cli-plugin-install
# Update url to download desired version
mkdir -p $HOME/tanzu
curl -L -X GET https://network.tanzu.vmware.com/api/v2/products/tanzu-application-platform/releases/1205491/product_files/1352407/download -H "Authorization: Bearer $ACCESS_TOKEN" --output bundle.tar

# Unpack bundle
export TANZU_CLI_NO_INIT=true
export VERSION=v0.25.0 # Update to desired version
tar -xvf bundle.tar -C $HOME/tanzu
cd $HOME/tanzu

# Install CLI
# Update to use desired OS
```

```
sudo install cli/core/$VERSION/tanzu-core-linux_amd64 /usr/local/bin/tanzu

# Install plugins
tanzu plugin install accelerator
```

## Use the Application Accelerator Visual Studio Code extension

This topic describes how to use the Application Accelerator Visual Studio Code extension to explore and generate projects from the defined accelerators in Tanzu Application Platform (commonly known as TAP) using VS Code.

The Application Accelerator Visual Studio Code extension lets you explore and generate projects from the defined accelerators in Tanzu Application Platform using VS Code.

## Dependencies

- To use the VS Code extension, the extension must access the Tanzu Application Platform GUI URL. For information about how to retrieve the Tanzu Application Platform GUI URL, see [Retrieving the URL for the Tanzu Application Platform GUI](#).
- (Optionally) To use Git repository provisioning during project creation in the VS Code extension, you must enable GitHub repository creation in the Application Accelerator plugin. For more information, see [Create an Application Accelerator Git repository during project creation](#).

## Installation

Use the following steps to install the Application Accelerator Visual Studio extension:

- Sign in to VMware Tanzu Network and download the “Tanzu App Accelerator Extension for Visual Studio Code” file from the product page for [VMware Tanzu Application Platform](#).
- Open VS Code.

### Option 1:

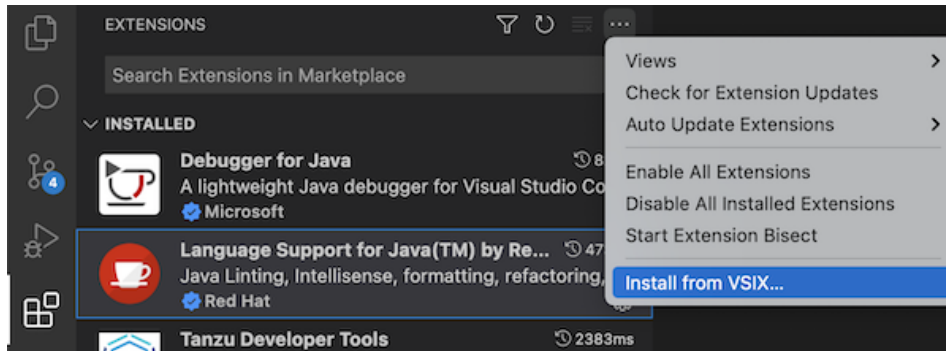
- From the Command Palette (cmd + shift + P), run “Extensions: Install from VSIX...”.
- Select the extension file **tanzu-app-accelerator-`<EXTENSION_VERSION>`.vsix**.



### Option 2:



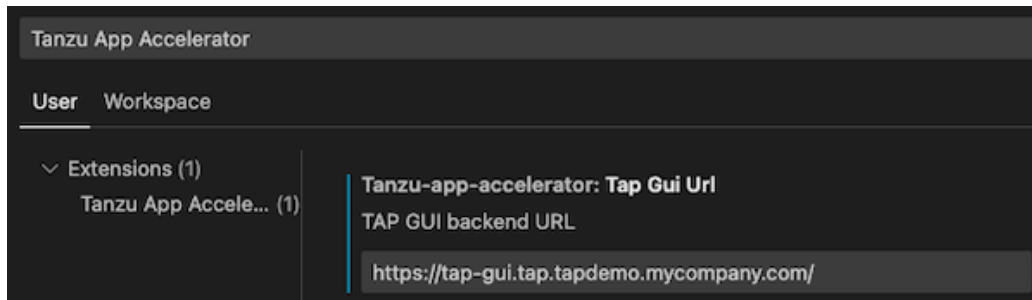
- Select the **Extensions** tab:
- Select **Install from VSIX...** from the overflow menu.



## Configure the extension

Before using the extension, you need follow the next steps:

1. Go to VS Code settings - click **Code > Preferences > Settings > Extensions > Tanzu App Accelerator**.
2. Look for the setting `Tap Gui Url`.
3. Add the `Tanzu Application Platform GUI URL`.

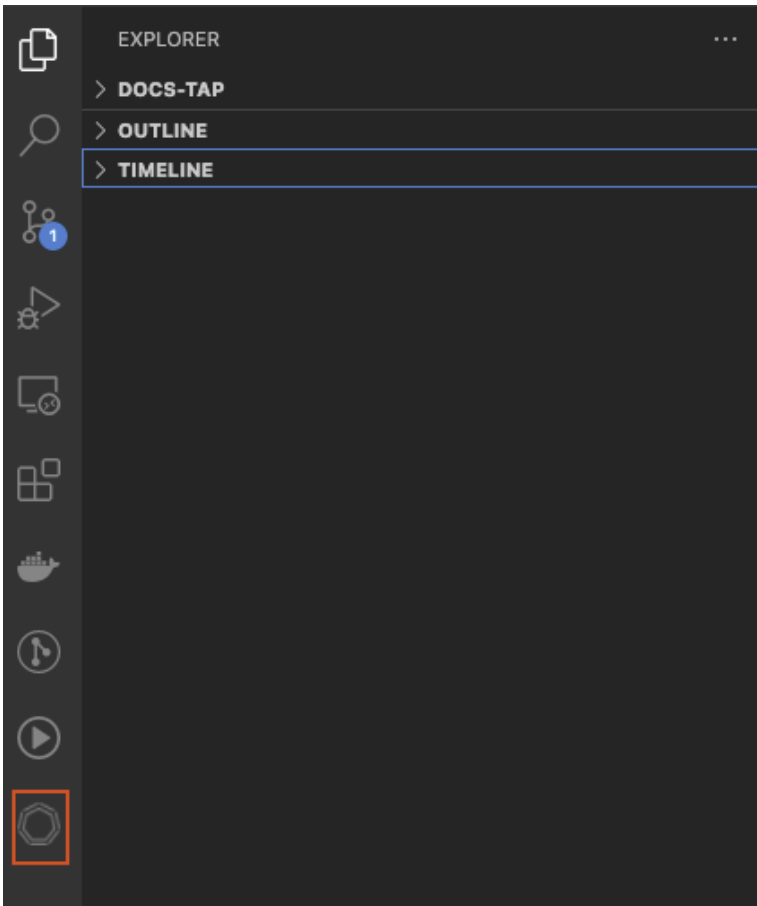


An example URL: `https://tap-gui.myclusterdomain.myorg.com`. If you have access to the Tanzu Application Platform cluster that is running the Tanzu Application Platform GUI, you can run the following command to determine the fully-qualified domain name:

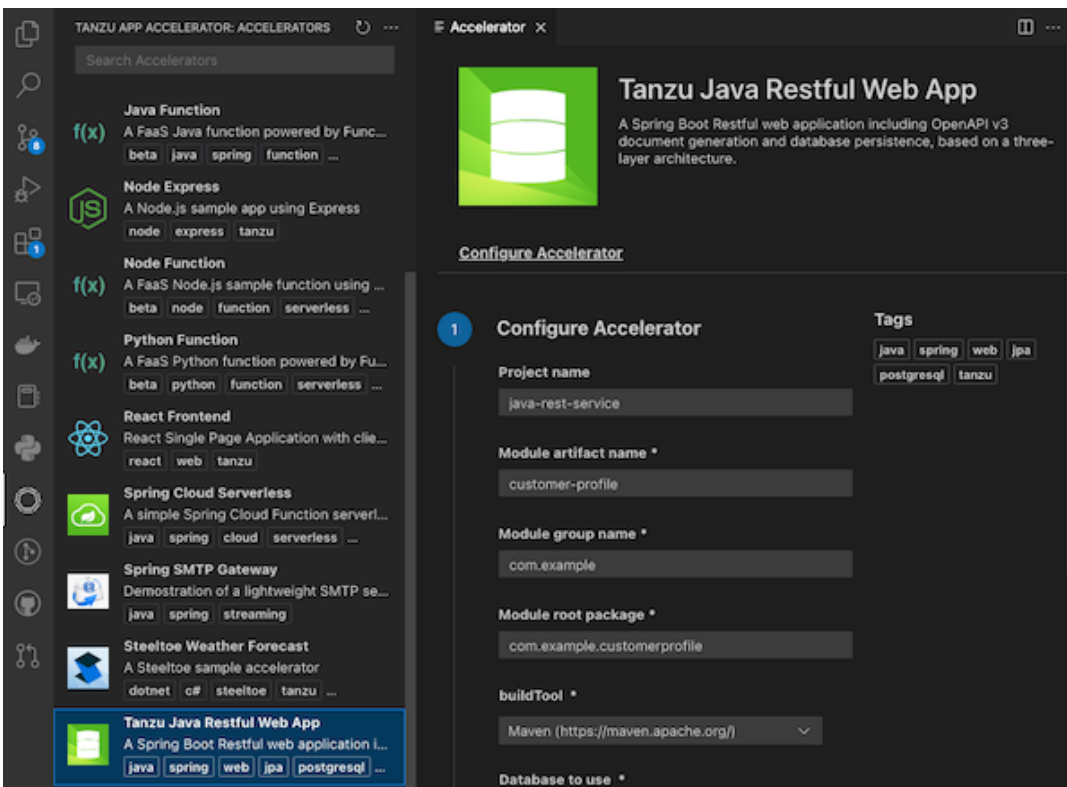
```
kubectl get httpproxy tap-gui -n tap-gui
```

## Using the extension

After adding the `Tap Gui Url` you can explore the defined accelerators accessing the Application Accelerator extension icon:



Choose any of the defined accelerators, fill the options and click the [generate project](#)



## Retrieving the URL for the Tanzu Application Platform GUI



If you have access to the Tanzu Application Platform cluster that is running the Tanzu Application Platform GUI, you can run the following command to determine the fully-qualified domain name:

```
kubectl get httpproxy tap-gui -n tap-gui
```

With an expected response of something similar to:

| NAME             | FQDN                          | TLS SECRET   | STATUS | STATUS DES |
|------------------|-------------------------------|--------------|--------|------------|
| tap-gui<br>Proxy | tap-gui.tap.tapdemo.myorg.com | tap-gui-cert | valid  | Valid HTTP |

## Known Issues

In v0.1.5 of the Application Accelerator extension for VS Code, if an accelerator with [custom types](#) is used *and* the custom type form contains checkboxes, re-prioritizing the custom types entries cause the data to not properly reorder. This is a known issue and is resolved in an upcoming release.

## Application Accelerator best practices

The following topics tells you about best practices for authoring accelerators and fragments.

- [Best practices for using Accelerators](#)  
A collection of best practices for authoring accelerators.
- [Best practices for using Fragments](#)  
A collection of best practices for authoring fragments.

## Best practices for using accelerators

This topic tells you about the benefits, and design considerations for accelerators.

### Benefits of using an accelerator

There are several good reasons to develop accelerators:

- If you're repeatedly using the same application architecture for new applications.
- To enforce standardization of technology stacks and application setups throughout your organization.
- To share best practices around application architecture, application, and test setup.

### Design considerations

Each accelerator must have only one base technology stack, combined with related tooling, and one target architecture. For example, if you use both Spring Boot and C# .NET Core applications in your target environment, you must set up two separate accelerators. Mixing multiple technology stacks and multiple target architectures makes both the directory structure and acceleratory.YAML unreadable.

The scope of your accelerator must align with your different types of deployments. For example, back-end API, front-end UI, business service, and so on.

Choose OpenRewrite-based transformation over ReplaceText-based transformation when possible. OpenRewrite-based transformations understand the semantics of the files they work on, for

example, Maven pom.xml or Java source files. OpenRewrite-based transformations also provide more accurate and robust modifications. As a last resort, ReplaceText supports a regex mode. When used with capturing groups in the replacement string, ReplaceText allows most modifications.

## Housekeeping rules

VMware has found that the following rules keep the set of accelerators clear and findable for end users:

- Use an intuitive name and short description that reflects the accelerators purpose. The word 'accelerator' must not be in the name.
- Use an appropriate and intuitive icon.
- Use tags that reflect language, framework, and type of service. For example, database, messaging, and so on. This helps when searching for an accelerator by tags. Tag names must use lowercase letters, consist of [a-z0-9+#] separated by [-], and not exceed 63 characters.
- Accelerators must expose options to allow configuring an accelerator for different use cases instead of creating multiple similar accelerators.
- Options must be straightforward, the description of each clearly stating the role it plays in the accelerator. Options must have default values when appropriate.
- Options must be short so that they are easy to navigate. Make options conditional on other options as appropriate.
- Free text options that have limitations on their values must ensure these limitations are met by a regular expression-based validation. This validation ensures early feedback on invalid user input.
- Generated application skeletons must have a detailed README file that describes the function and structure of a generated application. It must provide detailed information about how developers can build and deploy a generated application of the accelerator and how to use it.

## Tests

### Application skeleton

An accelerator that generates an application skeleton without a good test suite for the different layers of the application promotes bad behavior. It could result in code running in production without testing.

Tests you could use for the application skeleton:

- An overall application test that bootstraps the application to see if it comes online.
- A test per layer of the application. For example, presentation layer, business layer, and data layer. These tests can be unit tests that leverage stubbing or mocking frameworks.
- An integration test per layer of the application, especially the presentation and data layer. For example, you can provide an integration test with some database interaction by using [test containers](#).

## Best practices for using fragments

This topic tells you about the benefits, and design considerations for fragments.

## Benefits of using Fragment

A fragment is a partial accelerator. It can do the same transformations as an accelerator, but it cannot run on its own. It's always part of the calling (host) accelerator.

Developing a fragment is useful in the following situations:

- When you must update a version of an element of a technology stack in multiple locations. For example, when the Java Development Kit (JDK) version must be updated in the build tool configuration, the buildpack configuration, and in the deployment options.
- To add a consistent cross-cutting concern to a set of accelerators. For example, logging, monitoring, or support for a certain type of deployment or framework.
- To add integration with some technology to a generated application skeleton. For example, certain database support, support for a messaging middleware, or integration with an email provider.

## Design considerations

Developing and maintaining a fragment is complex. The following is a list of design considerations:

- The fragment you develop must work with all possible syntax and format variations. For example, dependency in a `Gradle` build.gradle.kts can have the following forms:
  - `implementation('org.springframework.boot:spring-boot-starter')`
  - `implementation("org.springframework.boot:spring-boot-starter")`
  - `implementation(group = "org.springframework.boot", name= "spring-boot-starter")`
  - `implementation(group = 'org.springframework.boot', name= 'spring-boot-starter')`
  - `implementation(name= "spring-boot-starter", group = "org.springframework.boot")`
- The fragment can be used in multiple accelerator contexts and its behavior must result in a compilable and deployable application skeleton.
- Testing a fragment in isolation is more difficult than testing an accelerator. Testing takes more time because all the combinations must be tested from an accelerator perspective.
- When flexibly reusing fragments in different combinations, each fragment must cover a small, cohesive function. Fragments must follow these two UNIX principles:
  - Small is beautiful.
  - Each fragment does one thing well.
- Keep the files the fragment changes to a minimum. Only change the files that are related to the same technology stack for the same purpose.
- The design of both the accelerator and fragment is limited by the technology stack and the target deployment technology chosen for the accelerator. For example, to create a fragment for standardizing logging, you must create one fragment per base technology stack.

## Housekeeping rules

Fragments are used by accelerator authors. VMware has found that the following guidelines keep fragments understandable and reusable.

- Give fragments an intuitive name and short description that reflects their purpose. Do not include “fragment” in the name.
- Fragments must expose options to allow configuring the output of execution.
- Each fragment must contain a README file explaining the additional functions the fragment adds to a generated application skeleton. List any options expected by this fragment. Also describe how this fragment can be included in a host accelerator. Be sure to state any known limitations or use cases not covered. For example, if the fragment supports Maven and Gradle as build tools but only Groovy DSL of Gradle is supported, the README file must include this information.
- If a fragment must provide additional documentation to end users, it can either be added to a README-X file of the generated application skeleton or append a section to the host’s README.

## Troubleshoot Application Accelerator

This topic provides troubleshooting steps for development, accelerator authorship, and operations issues in Application Accelerator.

### Development issues

#### Failure to generate a new project

`URI is not absolute` error

The `generate` command fails with the following error:

```
% tanzu accelerator generate test --server-url https://accelerator.example.com
Error: there was an error generating the accelerator, the server response was: "URI is
not absolute"

Use:
  tanzu accelerator generate [flags]

Examples:
  tanzu accelerator generate <accelerator-name> --options '{"projectName":"test"}'

Flags:
  -h, --help                help for generate
  --options string          options JSON string
  --options-file string    path to file containing options JSON string
  --output-dir string      directory that the zip file will be written to
  --server-url string       the URL for the Application Accelerator server

Global Flags:
  --context name            name of the kubeconfig context to use (default is current-co
nfig)
  --kubeconfig file        kubeconfig file (default is $HOME/.kube/config)

there was an error generating the accelerator, the server response was: "URI is not ab
solute"

Error: exit status 1

✘ exit status 1
```

This indicates that the accelerator resource requested is not in a `READY` state. Review the instructions in the [When Accelerator ready column is false](#) section or contact your system admin.

## Accelerator authorship issues

### General tips

#### Speed up the reconciliation of the accelerator

Set the `git.interval` to make the accelerator reconcile sooner. The default interval is 10 minutes, which is too long when developing an accelerator.

You can set this when using the YAML manifest:

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: test-accelerator
spec:
  git:
    url: https://github.com/trisberg/test-accelerator
    ref:
      branch: main
    interval: 10s
```

You can also set this when creating the accelerator resource. To do so from the Tanzu CLI, run:

```
tanzu accelerator create test-accelerator --git-repo https://github.com/trisberg/test-accelerator --git-branch main --interval 10s
```

#### Use a source image with local accelerator source directory

You don't have to use a Git repository when developing an accelerator. You can create an accelerator based on content in a local directory using `--local-path` when creating the accelerator resource.

Push the local path content to an OCI image by running:

```
tanzu accelerator create test-accelerator --local-path . --source-image REPO-PREFIX/test-accelerator --interval 10s
```

Where `REPO-PREFIX` is your own repository prefix. Use a repository that the deployed Application Accelerator system can access.

The interval is 10s so that you can push changes to the source-image repository and get faster reconcile time for the accelerator resource. When you have made changes to your accelerator source, push those changes by running:

```
tanzu accelerator push --local-path . --source-image REPO-PREFIX/test-accelerator
```

Where `REPO-PREFIX` is your own repository prefix. Use a repository that is accessible to the deployed Application Accelerator system.

### Expression evaluation errors

Expression evaluation errors include:

- Expression `evaluated to null`, such as:

```
Could not read response from accelerator: java.lang.IllegalArgumentException: Expression '#mytestexp' evaluated to null
```

In most cases, a typo in the variable name causes this error. Compare the expression with the defined options or any variables declared with `let`.

- could not parse SpEL expression, such as:

```
Could not read response from accelerator: Error reading manifest:could not parse SpEL expression at [Source: (InputStreamReader); line: 65, column: 1] (through reference chain: com.vmware.tanzu.accelerator.engine.manifest.Manifest["engine"]->com.vmware.tanzu.accelerator.engine.transform.transforms.Combo["let"]->java.util.ArrayList[0]->com.vmware.tanzu.accelerator.engine.transform.transforms.Let$DerivedSymbol["expression"])
```

In most cases, an error in a `let` expression causes this error. Review the error message and, for more information, see [SpEL samples](#).

- `SpelEvaluationException`, such as:

```
Could not read response from accelerator: org.springframework.expression.spel.SpelEvaluationException: EL1007E: Property or field 'test' cannot be found on null
```

In most cases, an error in a transform expression causes this error. Review the error message and, for more information, see [SpEL samples](#).

## Operations issues

### Check status of accelerator resources

Verify the status of accelerator resources by using `kubectl` or the Tanzu CLI:

- From `kubectl`, run:

```
kubectl get accelerators.accelerator.apps.tanzu.vmware.com -n accelerator-system
```

- From the Tanzu CLI, run:

```
tanzu accelerator list
```

Verify that the `READY` status is `true` for all accelerators.

### When Accelerator ready column is blank

1. View the status of `accelerator-system` by running:

```
kubectl get deployment -n accelerator-system
```

Example output:

| NAME                           | READY | UP-TO-DATE | AVAILABLE | AGE  |
|--------------------------------|-------|------------|-----------|------|
| acc-engine                     | 1/1   | 1          | 1         | 3d5h |
| acc-server                     | 1/1   | 1          | 1         | 2d1h |
| accelerator-controller-manager | 0/1   | 1          | 0         | 3d5h |

2. View the logs for any component with no Pods available by running:

```
kubectl logs deployment/COMPONENT-NAME/ -n accelerator-system -p
```

Where `COMPONENT-NAME` is the component with no pods you retrieved in the previous step.

- If the log has the following error then the Flux CD source-controller is not installed:

```
2021-11-18T20:55:18.963Z ERROR setup problem running manager {"error": "failed to wait for accelerator caches to sync: no matches for kind \"GitRepository\" in version \"source.toolkit.fluxcd.io/v1beta1\""}

```

- o If the log has the following error, the Tanzu Application Platform source-controller is not installed:

```
2021-11-18T20:50:10.557Z ERROR setup problem running manager {"error": "failed to wait for accelerator caches to sync: no matches for kind \"ImageRepository\" in version \"source.apps.tanzu.vmware.com/v1alpha1\""}

```

## When Accelerator ready column is false

View the **REASON** column for non-ready accelerators. Run:

```
kubectl get accelerators.accelerator.apps.tanzu.vmware.com -n accelerator-system
```

**REASON:** `GitRepositoryResolutionFailed`

For example:

```
$ kubectl get accelerators.accelerator.apps.tanzu.vmware.com -n accelerator-system
NAME      READY   REASON                                AGE
more-fun  False   GitRepositoryResolutionFailed         28s

```

1. View the resource status. Run:

```
kubectl get -oyaml accelerators.accelerator.apps.tanzu.vmware.com -n accelerator-system hello-fun
```

2. Read `status.conditions.message` near the end of the output to learn the likely cause of failure. For example:

```
status:
  address:
    url: http://accelerator-engine.accelerator-system.svc.cluster.local/inocations
  artifact:
    message: 'unable to clone 'https://github.com/vmware-tanzu/application-accelerator-samples'',
    error: couldn't find remote ref "refs/heads/test"
    ready: false
    url: ""
  conditions:
  - lastTransitionTime: "2021-11-18T21:05:47Z"
    message: |-
      failed to resolve GitRepository
      unable to clone 'https://github.com/vmware-tanzu/application-accelerator-samples', error: couldn't find remote ref "refs/heads/test"
    reason: GitRepositoryResolutionFailed
    status: "False"
    type: Ready
  description: Test-git
  observedGeneration: 1

```

In this example, `couldn't find remote ref "refs/heads/test"` reveals that the branch or tag specified doesn't exist.

Another common problem is that the Git repository doesn't exist. For example:

```

status:
  address:
    url: http://accelerator-engine.accelerator-system.svc.cluster.local/invo
ations
  artifact:
    message: 'unable to clone 'https://github.com/vmware-tanzu/application-acc
elerator-sampl'',
    error: authentication required'
    ready: false
    url: ""
  conditions:
  - lastTransitionTime: "2021-11-18T21:09:52Z"
    message: |-
      failed to resolve GitRepository
      unable to clone 'https://github.com/vmware-tanzu/application-accelerator-
sampl', error: authentication required
      reason: GitRepositoryResolutionFailed
      status: "False"
    type: Ready
  description: Test-git
  observedGeneration: 1

```

An error message about failed authentication might display because the Git repository doesn't exist. For example:

```

unable to clone 'https://github.com/vmware-tanzu/application-accelerator-samp
l', error: authentication required

```

#### REASON: `GitRepositoryResolutionPending`

For example:

```

$ kubectl get accelerators.accelerator.apps.tanzu.vmware.com -n accelerator-system
NAME          READY   REASON                               AGE
more-fun      False   GitRepositoryResolutionPending       28s

```

1. See the resource status. Run:

```

kubectl get -oyaml accelerators.accelerator.apps.tanzu.vmware.com -n accelera
r-system hello-fun

```

2. Locate `status.conditions` at the end of the output. For example:

```

status:
  address:
    url: http://accelerator-engine.accelerator-system.svc.cluster.local/invo
ations
  artifact:
    message: ""
    ready: false
    url: ""
  conditions:
  - lastTransitionTime: "2021-11-18T20:17:38Z"
    message: GitRepository not yet resolved
    reason: GitRepositoryResolutionPending
    status: "False"
    type: Ready
  description: Test-git
  observedGeneration: 1

```

3. Verify that the Flux system is running and that the `READY` column has `1/1`. Run:



```
kubectl get -n flux-system deployment/source-controller
```

Example output:

| NAME              | READY | UP-TO-DATE | AVAILABLE | AGE  |
|-------------------|-------|------------|-----------|------|
| source-controller | 1/1   | 0          | 0         | 5d4h |

**REASON:** `ImageRepositoryResolutionPending`

For example:

```
$ kubectl get accelerators.accelerator.apps.tanzu.vmware.com -n accelerator-system
NAME          READY   REASON                AGE
more-fun      False   ImageRepositoryResolutionPending  28s
```

1. See the resource status. Run:

```
kubectl get -oyaml accelerators.accelerator.apps.tanzu.vmware.com -n accelerator-system hello-fun
```

2. Locate `status.conditions` at the end of the output. For example:

```
$ kubectl get -oyaml accelerators.accelerator.apps.tanzu.vmware.com -n accelerator-system more-fun

apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"accelerator.apps.tanzu.vmware.com/v1alpha1","kind":"Accelerator","metadata":{"annotations":{},"name":"more-fun","namespace":"accelerator-system"},"spec":{"description":"Test-image","source":{"image":"trisberg/more-fun-source"}}}
  creationTimestamp: "2021-11-18T20:32:36Z"
  generation: 1
  name: more-fun
  namespace: accelerator-system
  resourceVersion: "605401"
  uid: 407b565d-14aa-44fe-ad8d-c9b3c3a7e5ce
spec:
  description: Test-image
  source:
    image: trisberg/more-fun-source
status:
  address:
    url: http://accelerator-engine.accelerator-system.svc.cluster.local/invocations
  artifact:
    message: ""
    ready: false
    url: ""
  conditions:
  - lastTransitionTime: "2021-11-18T20:32:36Z"
    message: ImageRepository not yet resolved
    reason: ImageRepositoryResolutionPending
    status: "False"
    type: Ready
  description: Test-image
  observedGeneration: 1
```

3. Verify that Tanzu Application Platform source-controller system is running and the **READY** column has 1/1. Run:

```
kubectl get -n source-system deployment/source-controller-manager
```

Expected output:

| NAME                      | READY | UP-TO-DATE | AVAILABLE | AGE  |
|---------------------------|-------|------------|-----------|------|
| source-controller-manager | 1/1   | 0          | 0         | 5d5h |

## Overview of Application Live View

Application Live View is a lightweight insights and troubleshooting tool for app developers and app operators that helps you to look inside running applications. It is based on the concept of Spring Boot Actuators.

The application provides information from inside the running processes using endpoints, in this case, HTTP endpoints. Application Live View uses those endpoints to get and interact with the data from apps.

## Value proposition

Application Live View is a diagnostic tool for developers to manage and analyze runtime characteristics of containerized apps. In addition, it provides a Kubernetes-native feel for developers to manage their apps in a Kubernetes environment more effectively.

## Intended audience

This documentation is intended for developers and operators to visualize the actuator information of their running apps on Application Live View for VMware Tanzu. This documentation helps developers to monitor and troubleshoot apps in development, staging, and production environments. It is also intended to help app operators to deploy and administer containerized apps in a Kubernetes environment.

## Supported application platforms

You can extend Application Live View to support multiple app platforms, including, but not limited to, Spring Boot, Spring Cloud Gateway, and Steeltoe.

## Multicloud compatibility

Using Tanzu platform, you can integrate Application Live View to monitor apps running across on-premises, public clouds, and edge. The platform provides a centralized view to manage apps across cloud environments, which accelerates developer productivity and reduces time-to-market.

## Deployment

Use a connector as the mode of deployment for registering apps with the Application Live View running on a Kubernetes cluster. A connector is a component responsible for discovering multiple apps running on a Kubernetes cluster and is installed as a DaemonSet by default.

## Overview of Application Live View

Application Live View is a lightweight insights and troubleshooting tool for app developers and app operators that helps you to look inside running applications. It is based on the concept of Spring Boot Actuators.

The application provides information from inside the running processes using endpoints, in this case, HTTP endpoints. Application Live View uses those endpoints to get and interact with the data from apps.

## Value proposition

Application Live View is a diagnostic tool for developers to manage and analyze runtime characteristics of containerized apps. In addition, it provides a Kubernetes-native feel for developers to manage their apps in a Kubernetes environment more effectively.

## Intended audience

This documentation is intended for developers and operators to visualize the actuator information of their running apps on Application Live View for VMware Tanzu. This documentation helps developers to monitor and troubleshoot apps in development, staging, and production environments. It is also intended to help app operators to deploy and administer containerized apps in a Kubernetes environment.

## Supported application platforms

You can extend Application Live View to support multiple app platforms, including, but not limited to, Spring Boot, Spring Cloud Gateway, and Steeltoe.

## Multicloud compatibility

Using Tanzu platform, you can integrate Application Live View to monitor apps running across on-premises, public clouds, and edge. The platform provides a centralized view to manage apps across cloud environments, which accelerates developer productivity and reduces time-to-market.

## Deployment

Use a connector as the mode of deployment for registering apps with the Application Live View running on a Kubernetes cluster. A connector is a component responsible for discovering multiple apps running on a Kubernetes cluster and is installed as a DaemonSet by default.

## Install Application Live View

This topic tells you how to install Application Live View from the Tanzu Application Platform (commonly known as TAP) package repository.

## Overview

Application Live View includes four packages you must install. The following table lists these packages and shows the Tanzu Application Platform profiles each package is included in.

| Package                                                                                 | Profiles   | Details                                                                                   |
|-----------------------------------------------------------------------------------------|------------|-------------------------------------------------------------------------------------------|
| Application Live View back end<br>( <code>backend.appliveview.tanzu.vmware.com</code> ) | Full, View | Installed with Tanzu Application Platform GUI in the <code>app-live-view</code> namespace |

| Package                                                                         | Profiles             | Details                                                                        |
|---------------------------------------------------------------------------------|----------------------|--------------------------------------------------------------------------------|
| Application Live View connector<br>(connector.appliveview.tanzu.vmware.com)     | Full, Iterate, Run   | Installed as a DaemonSet in the <code>app-live-view-connector</code> namespace |
| Application Live View conventions<br>(conventions.appliveview.tanzu.vmware.com) | Full, Iterate, Build | Installed in the <code>app-live-view-conventions</code> namespace              |

For more information about these packages, see [Application Live View internal architecture](#).



#### Note

Follow the steps in this topic if you do not want to use a profile to install Application Live View. For more information about profiles, see [About Tanzu Application Platform components and profiles](#).

## Prerequisites

Before installing Application Live View, complete all prerequisites to install Tanzu Application Platform. For more information, see [Prerequisites](#).

In addition, install Cartographer Conventions, which is bundled with Supply Chain Choreographer as of the v0.5.3 release. To install, see [Installing Supply Chain Choreographer](#). For more information, see [Cartographer Conventions](#).

## Install Application Live View

You can install Application Live View in single cluster or multicluster environment:

- **Single cluster:** All Application Live View components are deployed in a single cluster. The user can access Application Live View plug-in information of the applications across all the namespaces in the Kubernetes cluster. This is the default mode of Application Live View.
- **Multicluster:** In a multicluster environment, the Application Live View back end component is installed only once in a single cluster and exposes a RSocket registration for the other clusters using Tanzu shared ingress. Each cluster continues to install the connector as a DaemonSet. The connectors are configured to connect to the central instance of the Application Live View back end.

## Install Application Live View back end

To install Application Live View back end:

1. List version information for the package by running:

```
tanzu package available list backend.appliveview.tanzu.vmware.com --namespace 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 1.4.0     2022-12-08T00:00:00Z
```

- (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, `1.4.0`.

For example:

```
$ tanzu package available get backend.appliveview.tanzu.vmware.com/1.4.0 --valu
es-schema --namespace tap-install
```

| KEY                                  | DEFAULT                      | TYPE    | DESCRIPTION                                                                                                                                                |
|--------------------------------------|------------------------------|---------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>ingressDomain</code>           | <code>tap.example.com</code> | string  | Domain to be used by the HTTPProxy ingress object. The "appliveview" subdomain is prepended to the value provided. For example: "appliveview.example.com". |
| <code>ingressEnabled</code>          | <code>false</code>           | boolean | Flag for whether to create an HTTPProxy for ingress.                                                                                                       |
| <code>kubernetes_distribution</code> |                              | string  | Kubernetes distribution that this package is installed on. Accepted values: ['','openshift'].                                                              |
| <code>kubernetes_version</code>      |                              | string  | Optional: The Kubernetes Version. Valid values are '1.24.*', or ''.                                                                                        |
| <code>server.tls.crt</code>          |                              | string  | TLS cert file.                                                                                                                                             |
| <code>server.tls.enabled</code>      | <code>false</code>           | boolean | Flag to enable TLS on backend.                                                                                                                             |
| <code>server.tls.key</code>          |                              | string  | TLS key file.                                                                                                                                              |
| <code>tls.namespace</code>           | <code>&lt;nil&gt;</code>     | string  | The targeted namespace for secret consumption by the HTTPProxy.                                                                                            |
| <code>tls.secretName</code>          | <code>&lt;nil&gt;</code>     | string  | The name of secret for consumption by the HTTPProxy.                                                                                                       |

For more information about values schema options, see the properties listed earlier.

- Create `app-live-view-backend-values.yaml` with the following details:

For a SINGLE-CLUSTER environment, the Application Live View back end is exposed through the Kubernetes cluster service. By default, ingress is disabled for back end.

```
ingressEnabled: false
```

For a multicluster environment, set the flag `ingressEnabled` to true for the Application Live View back end to be exposed on the ingress domain.

```
backend:
  ingressEnabled: true
```



#### Note

If it is a Tanzu Application Platform profile installation and top-level key `shared.ingress_domain` is set in the `tap-values.yaml`, the back end is automatically exposed through the shared ingress.

If you want to override the shared ingress for Application Live View in 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.

The `app-live-view` namespace and the TLS secret for the domain should be created before installing the Tanzu Application Platform packages in the cluster so that the HTTPProxy is updated with the TLS secret. To create a TLS secret, run:

```
kubectl create -n app-live-view secret tls alv-cert --cert=<.cert file> --key=<.key file>
```

To verify the HTTPProxy object with the TLS secret, run:

```
kubectl get httpproxy -A
NAMESPACE          NAME
FQDN                                     TLS SECRET
STATUS  STATUS DESCRIPTION
app-live-view      appliveview
appliveview.192.168.42.55.nip.io
alv-cert  valid  Valid HTTPProxy
```

#### 4. Install the Application Live View back end 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 1.4.0 -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 back end component is deployed in `app-live-view` namespace by default.

- Verify the Application Live View back end package installation by running:

```
tanzu package installed get appliveview -n tap-install
```

For example:

```
tanzu package installed get appliveview -n tap-install
\ Retrieving installation details for appliveview...
NAME:                appliveview
PACKAGE-NAME:        backend.appliveview.tanzu.vmware.com
PACKAGE-VERSION:     1.4.0
STATUS:              Reconcile succeeded
CONDITIONS:          [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`.

## Install Application Live View connector

To install Application Live View connector:

- 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 --namespace tap-install
- Retrieving package versions for connector.appliveview.tanzu.vmware.com...
NAME                                VERSION      RELEASED-AT
connector.appliveview.tanzu.vmware.com 1.4.0       2022-12-08T00:00:00Z
```

- (Optional) Change the default installation settings by running:

```
tanzu package available get connector.appliveview.tanzu.vmware.com/VERSION-NUMBER --values-schema --namespace tap-install
```

Where `VERSION-NUMBER` is the version of the package listed. For example, `1.4.0`.

For example:

```
$ tanzu package available get connector.appliveview.tanzu.vmware.com/1.4.0 --values-schema --namespace tap-install
KEY                                DEFAULT      TYPE      DESCRIPTION
kubernetes_version                 <nil>       string    Optional: The Kubernetes Version. Valid values are '1.24.*', or ''.

backend.sslDeactivated              false       boolean   Flag for whether to disable SSL.
backend.caCertData                  cert-in-pem-format string     CA Cert Data for ingress domain.
backend.host                         <nil>       string    Domain to be used to reach the Application Live View back end. Prepend "appliveview" subdomain to the value if you are using shared ingress. For example: "example.com" becomes "appliveview.example.com".
backend.ingressEnabled              false       boolean   Flag for the connector to connect to ingress on back end.
```

```

  backend.port                <nil>                number    Port to
reach the Application Live View back end.
  connector.namespace_scoped.enabled  false        boolean   Flag fo
r the connector to run in namespace scope.
  connector.namespace_scoped.namespace  default      string    Namespa
ce to deploy connector.
  kubernetes_distribution            string       Kuberne
tes distribution that this package is being installed on. Accepted
values:
['', 'openshift'].

```

For more information about values schema options, see the properties listed earlier.

3. Create `app-live-view-connector-values.yaml` with the following details:

For SINGLE-CLUSTER environment, the Application Live View connector connects to the `cluster-local` Application Live View back end to register the applications.

By default, ingress is disabled for connector.

For a multicluster environment, set the flag `ingressEnabled` to true for the Application Live View connector to connect to the Application Live View back end by using the ingress domain.

```

backend:
  ingressEnabled: true

```

If it is a Tanzu Application Platform profile installation and top-level key `shared.ingress_domain` is set in the `tap-values.yml`, the Application Live View connector and Application Live View back end are configured to communicate through ingress. Then the Application Live View connector uses the `shared.ingress_domain` to reach the back end.

If you want to override the shared ingress for Application Live View in a multicluster environment, use the following values:

```

backend:
  host: appliveview.INGRESS-DOMAIN

```

Where `INGRESS-DOMAIN` is the top level domain the Application Live View back end exposes by using `tanzu-shared-ingress` for the connectors in other clusters to reach the Application Live View back end. Prepend the `appliveview` subdomain to the provided value.

The `backend.sslDeactivated` is set to `false` by default. The CA Cert for the ingress domain can be set in the `backend.caCertData` key for ssl validation. Below is a sample yaml:

```

backend:
  caCertData: |-
    -----BEGIN CERTIFICATE-----
    MIIGMzCCBBUGAwIBAgIJALHHzQjxM6wMMA0GCSqGSIb3DQEBAQUAMGcxCzAJBgNV
    BAGMAk1OMRQwEgYDVQQHDAtNaW5uZWZwb2xpczEPMA0GA1UECgwGVk13YXJlMRMw
    -----END CERTIFICATE-----

```

If TLS is not enabled for the `INGRESS-DOMAIN` in the Application Live View back end, set the `backend.sslDeactivated` to `true`.

```

backend:
  sslDeactivated: true

```



#### Note



The `sslDisabled` key is deprecated and has been renamed to `sslDeactivated`.

You can edit the values to suit your project needs or leave the default values as is.

Using the HTTP proxy either on 80 or 443 based on SSL config exposes the back-end service running on port 7000. The connector connects to the back end on port 80/443 by default. Therefore, you are not required to explicitly configure the `port` field.

4. 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, `1.4.0`.

For example:

```
$ tanzu package install appliveview-connector -p connector.appliveview.tanzu.vmware.com -v 1.4.0 -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 back end. The Application Live View connector component is deployed in `app-live-view-connector` namespace by default.

5. Verify the `Application Live View connector` package installation by running:

```
tanzu package installed get 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:     1.4.0
STATUS:              Reconcile succeeded
CONDITIONS:          [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`.

## Install Application Live View conventions

To install Application Live View conventions:

1. List version information for the package by running:

```
tanzu package available list conventions.appliveview.tanzu.vmware.com --namespace tap-install
```

For example:

```
$ tanzu package available list conventions.appliveview.tanzu.vmware.com --namespace tap-install
- Retrieving package versions for conventions.appliveview.tanzu.vmware.com...
  NAME                                     VERSION      RELEASED-AT
  conventions.appliveview.tanzu.vmware.com  1.4.0        2022-12-08T00:00:00Z
```

2. (Optional) Change the default installation settings by running:

```
tanzu package available get conventions.appliveview.tanzu.vmware.com/VERSION-NUMBER --values-schema --namespace tap-install
```

Where `VERSION-NUMBER` is the version of the package listed. For example, `1.4.0`.

For example:

```
$ tanzu package available get conventions.appliveview.tanzu.vmware.com/1.4.0 --values-schema --namespace tap-install
  KEY                                     DEFAULT      TYPE      DESCRIPTION
  kubernetes_distribution                 string      Kubernetes distribution that this package is installed on. Accepted values: ['','openshift'].
  kubernetes_version                     string      Optional: The Kubernetes Version. Valid values are '1.24.*', or ''.
```

For more information about values schema options, see the properties listed earlier.

3. 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, `1.4.0`.

For example:

```
$ tanzu package install appliveview-conventions -p conventions.appliveview.tanzu.vmware.com -v 1.4.0 -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'
```

4. 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: 1.4.0
STATUS:        Reconcile succeeded
CONDITIONS:    [{"ReconcileSucceeded": true}]
USEFUL-ERROR-MESSAGE:

```

Verify that `STATUS` is `Reconcile succeeded`.

The Application Live View UI plug-in is part of Tanzu Application Platform GUI. To access the Application Live View UI, see [Application Live View in Tanzu Application Platform GUI](#).

## Deprecate the `sslDisabled` key

The `appliveview_connector.backend.sslDisabled` key is deprecated and has been renamed to `appliveview_connector.backend.sslDeactivated`. The `appliveview_connector.backend.sslDisabled` key is marked for removal in Tanzu Application Platform 1.7.0.

## Enabling Spring Boot apps for Application Live View

This topic for developers tells you how to configure a Spring Boot app for observation by Application Live View within Tanzu Application Platform (commonly known as TAP).

### Enable Spring Boot apps

For Application Live View to interact with a Spring Boot app within Tanzu Application Platform, add the `spring-boot-starter-actuator` module dependency.

Add the maven dependency in `pom.xml` as follows:

```

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>

```

To enable Application Live View for Spring Boot apps, Spring Boot conventions automatically sets the Application Live View labels onto the PodSpec. For more information on the labels automatically set by Spring Boot conventions, see [Enable Application Live View for Spring Boot applications](#).

### Enable Spring Cloud Gateway apps

For Application Live View to interact with a Spring Cloud Gateway app within Tanzu Application Platform, add the `spring-boot-starter-actuator` and `spring-cloud-starter-gateway` module dependency.

Add the maven dependencies in `pom.xml` as follows:

```

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-gateway</artifactId>
</dependency>

```

To enable Application Live View on the Spring Cloud Gateway Tanzu Application Platform workload, Spring Boot conventions automatically applies labels on the workload, such as `tanzu.app.live.view.application.flavours: spring-boot_spring-cloud-gateway` and `tanzu.app.live.view: true`, based on the Spring Cloud Gateway image metadata.

Here is an example of creating a workload for a Spring Cloud Gateway Application:

```
tanzu apps workload create tanzu-scg-web-app --git-repo https://github.com/ksankaranar
a-vmw/gs-gateway.git --git-branch main --type web --label app.kubernetes.io/part-of=tan
znu-scg-web-app --yes --annotation autoscaling.knative.dev/min-scale=1
```

## Workload image NOT built with Tanzu Build Service

If your application image is NOT built with Tanzu Build Service, to enable Application Live View on `Spring Boot` Tanzu Application Platform workload, use the following command. For example:

```
tanzu apps workload create boot-app --type web --app boot-app --image <IMAGE NAME> --a
nnotation autoscaling.knative.dev/min-scale=1 --yes --label tanzu.app.live.view=true -
-label tanzu.app.live.view.application.name=boot-app --label tanzu.app.live.view.appli
cation.flavours=spring-boot
```

If your application image is NOT built with Tanzu Build Service, to enable Application Live View on `Spring Cloud Gateway` Tanzu Application Platform workload, use the following command. For example:

```
tanzu apps workload create scg-app --type web --app scg-app --image <IMAGE NAME> --ann
otation autoscaling.knative.dev/min-scale=1 --yes --label tanzu.app.live.view=true --l
abel tanzu.app.live.view.application.name=scg-app --label tanzu.app.live.view applicat
ion.flavours=spring-boot_spring-cloud-gateway
```

If your application image is NOT built with Tanzu Build Service, to enable Application Live View on `Steeltoe` Tanzu Application Platform workload, use the following command. For example:

```
tanzu apps workload create steeltoe-app --type web --app steeltoe-app --image <IMAGE N
AME> --annotation autoscaling.knative.dev/min-scale=1 --yes --label tanzu.app.live.vie
w=true --label tanzu.app.live.view.application.name=steeltoe-app --label tanzu.app.liv
e.view.application.flavours=steeltoe
```

## Enabling Spring Boot apps for Application Live View

This topic for developers tells you how to configure a Spring Boot app for observation by Application Live View within Tanzu Application Platform (commonly known as TAP).

### Enable Spring Boot apps

For Application Live View to interact with a Spring Boot app within Tanzu Application Platform, add the `spring-boot-starter-actuator` module dependency.

Add the maven dependency in `pom.xml` as follows:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

To enable Application Live View for Spring Boot apps, Spring Boot conventions automatically sets the Application Live View labels onto the PodSpec. For more information on the labels

automatically set by Spring Boot conventions, see [Enable Application Live View for Spring Boot applications](#).

## Enable Spring Cloud Gateway apps

For Application Live View to interact with a Spring Cloud Gateway app within Tanzu Application Platform, add the `spring-boot-starter-actuator` and `spring-cloud-starter-gateway` module dependency.

Add the maven dependencies in `pom.xml` as follows:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-gateway</artifactId>
</dependency>
```

To enable Application Live View on the Spring Cloud Gateway Tanzu Application Platform workload, Spring Boot conventions automatically applies labels on the workload, such as `tanzu.app.live.view.application.flavours: spring-boot_spring-cloud-gateway` and `tanzu.app.live.view: true`, based on the Spring Cloud Gateway image metadata.

Here is an example of creating a workload for a Spring Cloud Gateway Application:

```
tanzu apps workload create tanzu-scg-web-app --git-repo https://github.com/ksankaranar
a-vmw/gs-gateway.git --git-branch main --type web --label app.kubernetes.io/part-of=tan
zsu-scg-web-app --yes --annotation autoscaling.knative.dev/min-scale=1
```

## Workload image NOT built with Tanzu Build Service

If your application image is NOT built with Tanzu Build Service, to enable Application Live View on [Spring Boot](#) Tanzu Application Platform workload, use the following command. For example:

```
tanzu apps workload create boot-app --type web --app boot-app --image <IMAGE NAME> --a
nnotation autoscaling.knative.dev/min-scale=1 --yes --label tanzu.app.live.view=true -
-label tanzu.app.live.view.application.name=boot-app --label tanzu.app.live.view.applicat
ion.flavours=spring-boot
```

If your application image is NOT built with Tanzu Build Service, to enable Application Live View on [Spring Cloud Gateway](#) Tanzu Application Platform workload, use the following command. For example:

```
tanzu apps workload create scg-app --type web --app scg-app --image <IMAGE NAME> --ann
otation autoscaling.knative.dev/min-scale=1 --yes --label tanzu.app.live.view=true --l
abel tanzu.app.live.view.application.name=scg-app --label tanzu.app.live.view.applicat
ion.flavours=spring-boot_spring-cloud-gateway
```

If your application image is NOT built with Tanzu Build Service, to enable Application Live View on [Steeltoe](#) Tanzu Application Platform workload, use the following command. For example:

```
tanzu apps workload create steeltoe-app --type web --app steeltoe-app --image <IMAGE N
AME> --annotation autoscaling.knative.dev/min-scale=1 --yes --label tanzu.app.live.vie
w=true --label tanzu.app.live.view.application.name=steeltoe-app --label tanzu.app.liv
e.view.application.flavours=steeltoe
```

## Enable Steeltoe apps for Application Live View

This topic for developers tells you how to extend .NET Core Apps to Steeltoe apps and enable Application Live View on Steeltoe workloads within Tanzu Application Platform (commonly known as TAP).

Application Live View supports Steeltoe .NET apps with .NET core runtime version [v6.0.8](#).

## Extend .NET Core Apps to Steeltoe Apps

A .NET Core application can be extended to a Steeltoe application by adding independent NuGet packages.

To enable the Actuators on a .NET Core App:

1. Add a PackageReference to your `.csproj` file:

```
<PackageReference Include="Steeltoe.Management.EndpointCore" Version="$(SteeltoeVersion)" />
```



### Note

The PackageReference is expected to change to `Steeltoe.Management.Endpoint` from version Steeltoe 4.0 onwards.

2. Call the extension `AddAllActuators` in your `Program.cs` file:

```
builder.WebHost.AddAllActuators();
```

3. (Optional) You can add app-specific configurations, such as the following.

To expose all management actuator endpoints except `env` endpoint, add the following configuration to your `appsettings.json` file:

```
{
  "Management": {
    "Endpoints": {
      "Actuator": {
        "Exposure": {
          "Include": [ "*" ],
          "Exclude": [ "env" ]
        }
      }
    }
  }
}
```

To enable logging, add the following configuration to your `appsettings.json` file:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Steeltoe": "Warning",
      "Sample": "Information"
    }
  }
}
```

To enable heapdump, add the following configuration to your `appsettings.json` file:

```
{
  "Management": {
    "Endpoints": {
      "HeapDump": {
        "HeapDumpType": "Normal"
      }
    }
  }
}
```

## Enable Application Live View on Steeltoe Tanzu Application Platform workload

You can enable Application Live View to interact with a Steeltoe app within Tanzu Application Platform.

To enable Application Live View on the Steeltoe Tanzu Application Platform workload, the Application Live View convention service automatically applies labels on the workload, such as `tanzu.app.live.view.application.flavours: steeltoe` and `tanzu.app.live.view: true`, based on the Steeltoe image metadata.

Here's an example of creating a workload for a Steeltoe Application:

```
tanzu apps workload create steeltoe-app --type web --git-repo https://github.com/vmware-tanzu/application-accelerator-samples --sub-path weatherforecast-steeltoe --git-branch main --annotation autoscaling.knative.dev/min-scale=1 --yes --label app.kubernetes.io/part-of=sample-app
```

If your application image is NOT built with Tanzu Build Service, to enable Application Live View on Steeltoe Tanzu Application Platform workload, use the following command. For example:

```
tanzu apps workload create steeltoe-app --type web --app steeltoe-app --image IMAGE-NAME --annotation autoscaling.knative.dev/min-scale=1 --yes --label tanzu.app.live.view=true --label tanzu.app.live.view.application.name=steeltoe-app --label tanzu.app.live.view.application.flavours=steeltoe
```

Where `IMAGE-NAME` is the name of your application image.



### Note

Thread metrics is available in SteeltoeVersion `3.2.*`. To enable the Threads page in the Application Live View UI, add the following configuration to your `.csproj` file:

```
<PropertyGroup>
  <SteeltoeVersion>3.2.*</SteeltoeVersion>
</PropertyGroup>
```

## Application Live View convention server

This topic provides information about Application Live View convention, which provides a Webhook handler for [Convention Service for VMware Tanzu](#).

## Role of Application Live View convention

Application Live View conventions works in conjunction with core Convention Service. It enhances Tanzu PodIntents with metadata such as labels, annotations, or app properties. This metadata allows Application Live View, specifically the connector, to discover app instances so that Application Live View can access the actuator data from those workloads.



#### Note

Application Live View conventions now supports only Steeltoe applications. Spring Boot conventions supports both Spring Boot and Spring Cloud Gateway applications. For more information about Spring Boot conventions, see [Enable Application Live View with Spring Boot apps](#)

To run Application Live View with Steeltoe apps, the Spring Boot convention recognizes PodIntents and adds the following metadata labels:

- `tanzu.app.live.view: "true"`: Enables the connector to observe application pod.
- `tanzu.app.live.view.application.name: APPLICATION-NAME`: Identifies the app name to be used internally by Application Live View.
- `tanzu.app.live.view.application.actuator.port: ACTUATOR-PORT`: Identifies the port on the pod at which the actuators are available for Application Live View.
- `tanzu.app.live.view.application.flavours: steeltoe`: Exposes the framework flavor of the app.

These metadata labels allow Application Live View to identify pods that are enabled for Application Live View. The metadata labels also tell the Application Live View connector what kind of app it is, and on which port the actuators are accessible for Application Live View.

## Description of metadata labels

If a workload resource explicitly defines a label under `metadata.labels` in the `workload.yaml`, then Convention Service detects the presence of that label and respects its value. When deploying a workload using Tanzu Application Platform, you can override the labels listed in the following table using the `Workload` YAML.

Metadata	Default	Type	Description
<code>tanzu.app.live.view</code>	<code>true</code>	Label	When deploying a workload in Tanzu Application Platform, this label is set to <code>true</code> as default across the supply chain.
<code>tanzu.app.live.view.application.name</code>	<code>steeltoe-app</code>	Label	When deploying a workload in Tanzu Application Platform, this label is set to <code>steeltoe-app</code> if the container image metadata does not contain the app name. Otherwise, the label is set to the app name from container image metadata.
<code>tanzu.app.live.view.application.flavours</code>	<code>steeltoe</code>	Label	When deploying a Spring Boot workload in Tanzu Application Platform, this label is set to <code>steeltoe</code> as default across the supply chain.

## Verify the applied labels and annotations

You can verify the applied labels and annotations by running:

```
kubectl get podintents.conventions.carto.run WORKLOAD-NAME -o yaml
```

Where `WORKLOAD-NAME` is the name of the deployed workload, for example `steeltoe-app`.



## Expected output for Steeltoe workload:

```

apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
  creationTimestamp: "2022-11-14T09:56:53Z"
  generation: 1
  labels:
    app.kubernetes.io/component: intent
    app.kubernetes.io/part-of: sample-app
    apps.tanzu.vmware.com/workload-type: web
    carto.run/cluster-template-name: convention-template
    carto.run/resource-name: config-provider
    carto.run/supply-chain-name: source-to-url
    carto.run/template-kind: ClusterConfigTemplate
    carto.run/workload-name: steeltoe-app
    carto.run/workload-namespace: default
  name: steeltoe-app
  namespace: default
  ownerReferences:
  - apiVersion: carto.run/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: Workload
    name: steeltoe-app
    uid: 97897399-807a-4815-9693-fb06bb4bcled
  resourceVersion: "428904"
  uid: 0c74e045-075c-4af3-beef-b092b951be7f
spec:
  serviceAccountName: default
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/min-scale: "1"
        developer.conventions/target-containers: workload
      labels:
        app.kubernetes.io/component: run
        app.kubernetes.io/part-of: sample-app
        apps.tanzu.vmware.com/workload-type: web
        carto.run/workload-name: steeltoe-app
    spec:
      containers:
      - image: dev.registry.tanzu.vmware.com/app-live-view/test/steeltoe-app-default@sha256:c8ea14d8714ec31ad978085ebff43d15679613a0c12df37812adf22cb47b5232
        name: workload
        resources: {}
        securityContext:
          runAsUser: 1000
        serviceAccountName: default
status:
  conditions:
  - lastTransitionTime: "2022-11-14T09:56:57Z"
    message: ""
    reason: Applied
    status: "True"
    type: ConventionsApplied
  - lastTransitionTime: "2022-11-14T09:56:57Z"
    message: ""
    reason: ConventionsApplied
    status: "True"
    type: Ready
  observedGeneration: 1
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/min-scale: "1"

```

```

conventions.carto.run/applied-conventions: |-
  spring-boot-convention/auto-configure-actuators-check
  spring-boot-convention/app-live-view-appflavour-check
  appliveview-sample/app-live-view-appflavour-check
  appliveview-sample/app-live-view-connector-steeltoe
  appliveview-sample/app-live-view-appflavours-steeltoe
developer.conventions/target-containers: workload
labels:
  app.kubernetes.io/component: run
  app.kubernetes.io/part-of: sample-app
  apps.tanzu.vmware.com/workload-type: web
  carto.run/workload-name: steeltoe-app
  tanzu.app.live.view: "true"
  tanzu.app.live.view.application.flavours: steeltoe
  tanzu.app.live.view.application.name: steeltoe-app
spec:
  containers:
  - image: dev.registry.tanzu.vmware.com/app-live-view/test/steeltoe-app-default@sha256:c8ea14d8714ec31ad978085ebff43d15679613a0c12df37812adf22cb47b5232
    name: workload
    resources: {}
    securityContext:
      runAsUser: 1000
    serviceAccountName: default

```

In your output:

- `status.template.metadata.labels` shows the list of applied labels by Application Live View convention server.
- `status.template.metadata.annotations` shows the list of applied annotations by Application Live View convention server.

## Custom configuration for the connector

This topic for developers tells you how to custom configure an app or workload for Application Live View.

The connector component is responsible for discovering the app and registering it with Application Live View. Labels from the app PodSpec are used to discover the app and configure the connector to access the actuator data of the app.

Usually, Application Live View conventions applies the necessary configuration automatically. To deactivate the convention and configure the app and the workload manually, the list of labels in the following table gives you an overview of the options:

Label Name	Mandatory	Type	Default	Significance
<code>tanzu.app.live.view</code>	true	Boolean	None	Toggle to activate or deactivate pod discovery
<code>tanzu.app.live.view.application.name</code>	true	String	None	Application name
<code>tanzu.app.live.view.application.port</code>	false	Integer	8080	Application port
<code>tanzu.app.live.view.application.path</code>	false	String	/	Application context path
<code>tanzu.app.live.view.application.actuator.port</code>	false	Integer	8080	Application actuator port
<code>tanzu.app.live.view.application.actuator.path</code>	false	String	/actuator	Actuator context path

Label Name	Mandatory	Type	Default	Significance
<code>tanzu.app.live.view.application.protocol</code>	false	http / https	<code>http</code>	Protocol scheme
<code>tanzu.app.live.view.application.actuator.health.port</code>	false	Integer	<code>8080</code>	Health endpoint port
<code>tanzu.app.live.view.application.flavours</code>	false	Comma separated string	<code>spring-boot, spring-cloud-gateway</code>	Application flavors

You can add connector labels in the app `Workload` or override labels that the convention applies, such as `tanzu.app.live.view` and `tanzu.app.live.view.application.name`. If you do not want Application Live View to observe your app, you can override the existing label

`tanzu.app.live.view: "false"`.

## Configure the developer workload in Tanzu Application Platform

The following YAML is an example of a Spring PetClinic workload that overrides the connector label to `tanzu.app.live.view: "false"`:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: spring-petclinic
  namespace: default
  labels:
    tanzu.app.live.view: "false"
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/workload-type: web
  annotations:
    autoscaling.knative.dev/minScale: "1"
spec:
  source:
    git:
      ref:
        branch: main
      url: https://github.com/kdvolder/spring-petclinic
```

## Deploy the workload

To deploy the workload, run:

```
kapp -y deploy -n default -a workloads -f workloads.yaml
```

## Verify the label has propagated through the Supply Chain

To verify the label:

1. Verify that the workload build is successful by ensuring that `SUCCEEDED` is set to `True`:

```
kubectl get builds
NAME                                IMAGE
SUCCEEDED
spring-petclinic-build-1            dev.registry.tanzu.vmware.com/app-live-view/test/s
pring-petclinic-default@sha256:9db2a8a8e77e9215239431fd8afe3f2ecdf09cce8afac565
dad7b5f0c5ac0cdf                    True
```

2. Verify the PodIntent of your workload by ensuring `status.template.metadata.labels` shows the newly added label has propagated through the Supply Chain:

```
kubectl get podintents.conventions.carto.run spring-petclinic -oyaml

status:
  conditions:
  - lastTransitionTime: "2021-12-03T15:14:33Z"
    status: "True"
    type: ConventionsApplied
  - lastTransitionTime: "2021-12-03T15:14:33Z"
    status: "True"
    type: Ready
  observedGeneration: 3
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/minScale: "1"
        boot.spring.io/actuator: http://:8080/actuator
        boot.spring.io/version: 2.5.6
        conventions.carto.run/applied-conventions: |-
          appliveview-sample/app-live-view-connector-boot
          appliveview-sample/app-live-view-appflavours-boot
          appliveview-sample/app-live-view-systemproperties
          spring-boot-convention/spring-boot
          spring-boot-convention/spring-boot-graceful-shutdown
          spring-boot-convention/spring-boot-web
          spring-boot-convention/spring-boot-actuator
          spring-boot-convention/service-intent-mysql
        developer.conventions/target-containers: workload
        kapp.k14s.io/identity: v1;default/carto.run/Workload/spring-petclinic;carto.run/v1alpha1
        kapp.k14s.io/original: '{"apiVersion":"carto.run/v1alpha1","kind":"Workload","metadata":{"annotations":{"autoscaling.knative.dev/minScale":"2"},"labels":{"app.kubernetes.io/part-of":"tanzu-java-web-app","apps.tanzu.vmware.com/workload-type":"web","kapp.k14s.io/app":"1638455805474051000","kapp.k14s.io/association":"v1.5a9384bd7b93ca74ef494c4dec2caa4b","tanzu.app.live.view":"false"},"name":"spring-petclinic","namespace":"default"},"spec":{"source":{"ref":{"branch":"main"},"url":"https://github.com/ksankaranara-vmw/spring-petclinic"}}}'
        kapp.k14s.io/original-diff-md5: 58e0494c51d30eb3494f7c9198986bb9
        services.conventions.carto.run/mysql: mysql-connector-java/8.0.27
      labels:
        app.kubernetes.io/component: run
        app.kubernetes.io/part-of: tanzu-java-web-app
        apps.tanzu.vmware.com/workload-type: web
        carto.run/workload-name: spring-petclinic
        conventions.carto.run/framework: spring-boot
        kapp.k14s.io/app: "1638455805474051000"
        kapp.k14s.io/association: v1.5a9384bd7b93ca74ef494c4dec2caa4b
        services.conventions.carto.run/mysql: workload
        tanzu.app.live.view: "false"
        tanzu.app.live.view.application.flavours: spring-boot
        tanzu.app.live.view.application.name: petclinic
```

3. Verify the ConfigMap was created for the app by ensuring `metadata.labels` shows the newly added label has propagated through the Supply Chain:

```
kubectl describe configmap spring-petclinic

Name:          spring-petclinic
Namespace:    default
Labels:       carto.run/cluster-supply-chain-name=source-to-url
              carto.run/cluster-template-name=config-template
              carto.run/resource-name=app-config
              carto.run/template-kind=ClusterConfigTemplate
```

```

        carto.run/workload-name=spring-petclinic
        carto.run/workload-namespace=default
Annotations:  <none>

Data
====
delivery.yml:
----
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: spring-petclinic
  labels:
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/workload-type: web
    kapp.k14s.io/app: "1638455805474051000"
    kapp.k14s.io/association: v1.5a9384bd7b93ca74ef494c4dec2caa4b
    tanzu.app.live.view: "false"
    app.kubernetes.io/component: run
    carto.run/workload-name: spring-petclinic

```

4. Verify the running Knative application pod by ensuring `labels` shows the newly added label on the Knative application pod:

```

kubectyl get pods -o yaml spring-petclinic-00002-deployment-77dbb85c65-cf7rn | g
rep labels
    kapp.k14s.io/original: '{"apiVersion":"carto.run/v1alpha1","kind":"Workloa
d","metadata":{"annotations":{"autoscaling.knative.dev/minScale":"1"},"labels":
{"app.kubernetes.io/part-of":"tanzu-java-web-app","apps.tanzu.vmware.com/worklo
ad-type":"web","kapp.k14s.io/app":"1638455805474051000","kapp.k14s.io/associati
on":"v1.5a9384bd7b93ca74ef494c4dec2caa4b","tanzu.app.live.view":"false"},"nam
e":"spring-petclinic","namespace":"default"},"spec":{"source":{"git":{"ref":{"b
ranch":"main"},"url":"https://github.com/ksankaranara-vmw/spring-petclini
c"}}}}'

```

You can add or override the connector in the `Workload` of your Knative app.

## Custom configuration for application actuator endpoints

This topic for developers tells you how to configure the Application Live View connector component to access actuator endpoints for custom settings, such as a different base path. By default, the actuator endpoint for an application is exposed on `/actuator`.

The following table describes the actuator configuration scenarios and the associated labels to use, assuming that the app runs on port `8080`:

management.server.base-path	management.server.port	management.endpoints.web.base-path	server.servlet.context.path	Comments	Connector Configuration
None	None	None	None	Actuators endpoints available at localhost:8080/actuator	<code>tanzu.app.live.view.application.actuator.path=actuator,</code> <code>tanzu.app.live.view.application.actuator.port=8080</code>

management.server.base-path	management.server.port	management.endpoints.web.base-path	server.servlet.context.path	Comments	Connector Configuration
/path	8082	/	None	Actuator endpoints available at localhost:8082/path	tanzu.app.live.view.application.actuator.path=path, tanzu.app.live.view.application.actuator.port=8082
/path	8082	/manage/actuator	None	Actuator endpoints available at localhost:8082/path/manage/actuator	tanzu.app.live.view.application.actuator.path=path/manage/actuator, tanzu.app.live.view.application.actuator.port=8082
None	None	/	None	Actuators are deactivated to avoid conflicts	None
None	None	/manage	None	Actuator endpoints available at /manage	tanzu.app.live.view.application.actuator.path=manage, tanzu.app.live.view.application.actuator.port=8080
/path	8082	None	None	Actuator endpoints available at localhost:8082/path/actuator	tanzu.app.live.view.application.actuator.path=path/actuator, tanzu.app.live.view.application.actuator.port=8082
/	8082	None	None	Actuator endpoints available at localhost:8082/actuator	tanzu.app.live.view.application.actuator.path=actuator, tanzu.app.live.view.application.actuator.port=8082

management.server.base-path	management.server.port	management.endpoints.web.base-path	server.servlet.context.path	Comments	Connector Configuration
None	None	None	/api	Actuator endpoints available at localhost:8080/api/actuator	tanzu.app.live.view.application.actuator.path=api/actuator, tanzu.app.live.view.application.actuator.port=8080
/path	8082	None	/api	Actuator endpoints available at localhost:8082/path/actuator	tanzu.app.live.view.application.actuator.path=path/actuator, tanzu.app.live.view.application.actuator.port=8082
/path	8082	/manage	/api	Actuator endpoints available at localhost:8082/path/manage	tanzu.app.live.view.application.actuator.path=path/manage, tanzu.app.live.view.application.actuator.port=8082
/path	None	/manage	/api	Actuator endpoints available at localhost:8080/api/manage	tanzu.app.live.view.application.actuator.path=api/manage, tanzu.app.live.view.application.actuator.port=8080
/path	None	/	/api	Actuators are deactivated to avoid conflicts	None
/path	8082	/	/api	Actuator endpoints available at localhost:8082/path	tanzu.app.live.view.application.actuator.path=path, tanzu.app.live.view.application.actuator.port=8082

management.server.base-path	management.server.port	management.endpoints.web.base-path	server.servlet.context.path	Comments	Connector Configuration
None	None	/manage	/api	Actuator endpoints available at localhost:8080/api/manage	tanzu.app.live.view.application.actuator.path=/api/manage, tanzu.app.live.view.application.actuator.port=8080

## Scaling Knative apps in Tanzu Application Platform

This topic tells you how to use Application Live View when scaling Knative apps or developer workloads in Tanzu Application Platform (commonly known as TAP).

Application Live View is focused on monitoring apps for a `live` window and does not apply to any of the apps that are scaled down to zero. The intended behavior for Knative apps is to keep track of revisions to allow you to rollback easily, but also scale all of the unused revision instances down to zero to keep resource consumption low.

You can configure Knative apps to set `autoscaling.knative.dev/minScale` to `1` so that Application Live View can still observe app instance. This ensures that there is at least one instance of the latest revision, while still scaling down the older instances.

You can configure any app in Tanzu Application Platform using the `Workload` resource. To scale a Knative app, add the annotation `autoscaling.knative.dev/minScale` in the `Workload` and set it to the value you want. For Application Live View to observe an app and have at least one instance of the latest revision, set `autoscaling.knative.dev/minScale = "1"`.

The annotations or labels in the `Workload` get propagated through the Tanzu Application Platform supply chain as follows:

Workload > PodIntent > ConfigMap > Push Config > to repository/registry > git-repository/imagerepository picks the Config from repository/registry > kapp-ctrl deploys and knative runs the config > final pod running on the Kubernetes cluster.

## Configure the developer workload in Tanzu Application Platform

The following YAML is an example `Workload` that adds the annotation `autoscaling.knative.dev/minScale = "1"` to set the minimum scale for the `spring-petclinic` app:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: spring-petclinic
  namespace: default
  labels:
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/workload-type: web
  annotations:
    autoscaling.knative.dev/minScale: "1"
spec:
  source:
    git:
      ref:
```



```
branch: main
url: https://github.com/kdvolder/spring-petclinic
```

## Deploy the workload

To deploy the workload, run:

```
kapp -y deploy -n default -a workloads -f workloads.yaml
```

## Verify the annotation has propagated through the Supply Chain

To verify the annotation:

1. Verify that the workload build is successful by ensuring that `SUCCEEDED` is set to `True`:

```
kubectl get builds
NAME                                IMAGE
SUCCEEDED
spring-petclinic-build-1           dev.registry.tanzu.vmware.com/app-live-view/test/s
pring-petclinic-default@sha256:9db2a8a8e77e9215239431fd8afe3f2ecdf09cce8afac565
dad7b5f0c5ac0cdf                 True
```

2. Verify the PodIntent of your workload by ensuring `status.template.metadata.annotations` shows the newly added annotation has propagated through the Supply Chain:

```
kubectl get podintents.conventions.carto.run spring-petclinic -oyaml

status:
  conditions:
  - lastTransitionTime: "2021-12-03T15:14:33Z"
    status: "True"
    type: ConventionsApplied
  - lastTransitionTime: "2021-12-03T15:14:33Z"
    status: "True"
    type: Ready
  observedGeneration: 3
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/minScale: "1"
```

3. Verify the ConfigMap was created for the app by ensuring `spec.template.metadata.annotations` shows the newly added annotation has propagated through the Supply Chain:

```
kubectl describe configmap spring-petclinic
Name:          spring-petclinic
Namespace:    default
Labels:        carto.run/cluster-supply-chain-name=source-to-url
               carto.run/cluster-template-name=config-template
               carto.run/resource-name=app-config
               carto.run/template-kind=ClusterConfigTemplate
               carto.run/workload-name=spring-petclinic
               carto.run/workload-namespace=default
Annotations:   <none>

Data
====
delivery.yml:
```

```

----
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: spring-petclinic
  labels:
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/workload-type: web
    kapp.k14s.io/app: "1638455805474051000"
    kapp.k14s.io/association: v1.5a9384bd7b93ca74ef494c4dec2caa4b
    tanzu.app.live.view: "false"
    app.kubernetes.io/component: run
    carto.run/workload-name: spring-petclinic
spec:
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/minScale: "1"

```

4. Verify the running Knative application pod by ensuring `annotations` shows the newly added annotation on the Knative application pod:

```

kubectl get pods -o yaml spring-petclinic-00002-deployment-77dbb85c65-cf7rn | g
rep annotations
annotations:
  kapp.k14s.io/original: '{"apiVersion":"carto.run/v1alpha1","kind":"Workload",
"metadata":{"annotations":{"autoscaling.knative.dev/minScale":"1"},"labels":
{"app.kubernetes.io/part-of":"tanzu-java-web-app","apps.tanzu.vmware.com/worklo
ad-type":"web","kapp.k14s.io/app":"1638455805474051000","kapp.k14s.io/associati
on":"v1.5a9384bd7b93ca74ef494c4dec2caa4b","tanzu.app.live.view":"false"},"nam
e":"spring-petclinic","namespace":"default"},"spec":{"source":{"git":{"ref":{"b
ranch":"main"},"url":"https://github.com/ksankaranara-vmw/spring-petclini
c"}}}}'

```

Your Knative app is now set to a minimum scale of one so that Application Live View can observe the instance of the app.

## Application Live View on OpenShift

Application Live View must run with a custom SecurityContextConstraint (SCC) to enable compliance with restricted Kubernetes Pod Security Standards on Openshift. Tanzu Application Platform configures the following SCC for Application Live View back end, Application Live View connector, and Application Live View convention service when you configure the `kubernetes_distribution: openshift` key in the `tap-values.yaml` file.

The following is a `SecurityContextConstraints` specification for Application Live View connector:

```

---
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  name: appliveview-connector-restricted-with-seccomp
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: false
allowPrivilegedContainer: false
allowedCapabilities: null
defaultAddCapabilities: null
fsGroup:
  type: MustRunAs

```

```

priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities:
  - ALL
runAsUser:
  type: MustRunAsNonRoot
seLinuxContext:
  type: MustRunAs
supplementalGroups:
  type: RunAsAny
volumes:
  - configMap
  - downwardAPI
  - emptyDir
  - persistentVolumeClaim
  - projected
  - secret
seccompProfiles:
  - runtime/default
    
```

The preceding `SecurityContextConstraints` specification is applicable to Application Live View back end and Application Live View convention service as well.

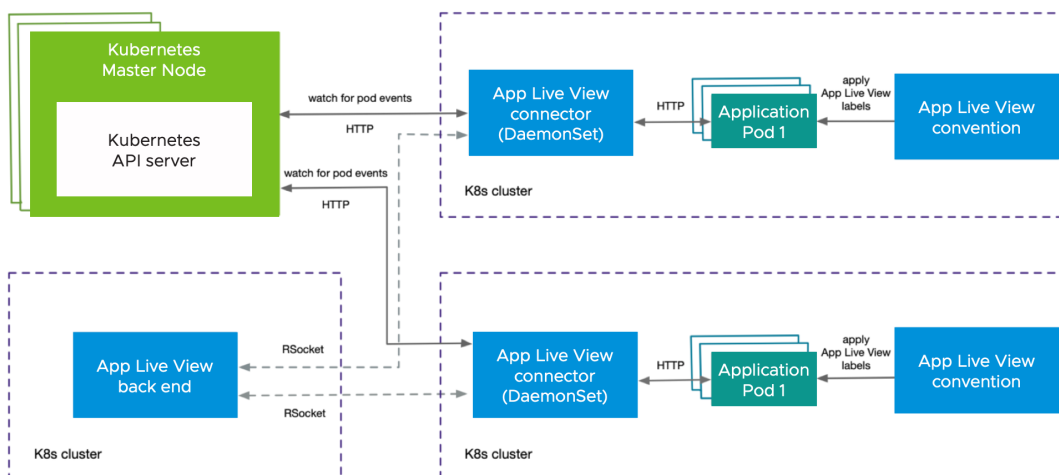
## Support for polyglot apps with Application Live View

Application Live View currently supports Spring Boot, Spring Cloud Gateway, and Steeltoe apps.

- To enable Application Live View on Spring Boot and Spring Cloud Gateway apps, see [Enable Application Live View for Spring Boot apps](#).
- To enable Application Live View on Steeltoe apps, see [Enable Application Live View for Steeltoe apps](#).

## Application Live View internal architecture

This topic describes the architecture of Application Live View and its components. You can deploy this system on a Kubernetes stack and use it to monitor containerized apps on hosted cloud platforms or on-premises.



## Component overview

Application Live View includes the following components as shown in the architecture diagram:

- **Application Live View back end**

Application Live View back end is the central server component that contains a list of registered apps. It is responsible for proxying the request to fetch the actuator information related to the app.

- **Application Live View connector**

Application Live View connector is the component responsible for discovering the app pods running on the Kubernetes cluster and registering the instances to the Application Live View back end for it to be observed. The Application Live View connector is also responsible for proxying the actuator queries to the app pods running in the Kubernetes cluster.

You can deploy Application Live View connector in two modes:

- **Cluster access:** Deploy as a Kubernetes DaemonSet to discover apps across all the namespaces running in a worker node of a Kubernetes cluster. This is the default mode of Application Live View connector.
- **Namespace scoped:** Deploy as a Kubernetes Deployment to discover apps running within a namespace across worker nodes of Kubernetes cluster.

- **Application Live View convention server**

This component provides a webhook handler for the Tanzu convention controller. The webhook handler is registered with Tanzu convention controller. The webhook handler detects supply-chain workloads running a Spring Boot. Such workloads are annotated automatically to enable Application Live View to monitor them. Download and install the Application Live View conventions Webhook component with [Tanzu Application Platform](#).

## Design flow

As illustrated in the diagram, the applications run by the user are registered with Application Live View back end by using Application Live View connector. After the application is registered, the Application Live View back end offers the ability to serve actuator data from that registered application through its REST API. Application Live View back end proxies the call to the connector for querying actuator endpoint information.

Application Live View connector, which is a lean model, uses specific labels to discover apps across cluster or namespace. Application Live View connector serves as the connection between running applications and Application Live View back end. Application Live View connector communicates with the Kubernetes API server requesting events for pod creation and termination, and then filters out the events to find the pod of interest by using labels. Then Application Live View connector registers the filtered app instances with Application Live View back end.

Application Live View back end and Application Live View connector communicate through a bidirectional RSocket channel. Application Live View connector is implemented as a Java/Spring Boot application and runs as a native executable file (Spring Native using GraalVM). Application Live View connector runs as a DaemonSet by default on every node in the cluster.

Application Live View conventions identifies PodIntents for pods that can serve actuator data and annotates the PodSpec with application-specific labels. Those labels are used by the Application Live View connector to identify running pods that can serve actuator data. Application Live View conventions reads the image metadata to determine the application-specific labels applied on the PodSpec.

## Troubleshoot Application Live View

This topic provides information to help you troubleshoot Application Live View.

## App is not visible in Application Live View UI

### Symptom

Your app is not visible in the Application Live View UI.

### Solution

The connector component is responsible for discovering the app and registering it with Application Live View.

To troubleshoot, confirm the following:

1. The app must be a Spring Boot Application.
2. Confirm that an instance of a connector is located in the same namespace as your app.

```
kubectl get pods -n NAMESPACE | grep connector
```

Where `NAMESPACE` is the name of the namespace that your app is located in.

3. Confirm that the actuator endpoints are enabled for your app as follows:

```
management.endpoints.web.exposure.include: "*" 
```

4. Confirm that you have included the following labels within your app deployment YAML file:

```
tanzu.app.live.view="true"
tanzu.app.live.view.application.name="APP-NAME"
```

Where `APP-NAME` is the name of your app.

5. Confirm that the Convention Service workload YAML file does not contain property `management.endpoints.web.exposure.include` overrides.

See also:

- [App is not visible in Application Live View UI with actuator endpoints enabled](#)
- [The UI does not show any information for an app with actuator endpoints exposed at root](#)

## App is not visible in Application Live View UI with actuator endpoints enabled

### Symptom

Your app is not visible in Application Live View UI, but the actuator endpoints are enabled.

### Solution

To troubleshoot:

1. Check the port on which actuator endpoints are configured. By default, they are enabled on the application port. If they are configured on a port different from the application port, set the labels in your app deployment YAML file as follows:

```
tanzu.app.live.view.application.port: "APPLICATION-PORT"
tanzu.app.live.view.application.actuator.port: "ACTUATOR-PORT"
```

Where:

- `APPLICATION-PORT` is the application port.

- `ACTUATOR-PORT` is the actuator port.
2. Check the path on which the app and actuator endpoints are configured. If they are configured on a different paths, set the labels in your app deployment YAML file as follows:

```
tanzu.app.live.view.application.path: "APPLICATION-PATH"
tanzu.app.live.view.application.actuator.path: "ACTUATOR-PATH"
```

Where:

- `APPLICATION-PATH` is the application port.
- `ACTUATOR-PATH` is the actuator port.

## The UI does not show any information for an app with actuator endpoints exposed at root

### Symptom

Your app has actuator endpoints exposed at root and the UI does not show any information.

### Cause

Application Live View cannot display the app details when the app is exposing the actuator endpoint on root (/). This is due to conflict in the actuator context path and app default context path.

## No information shown on the Health page

### Symptom

The app shows up in Application Live View UI, but the **Health** page does not show details of health.

### Solution

The information exposed by the health endpoint depends on the `management.endpoint.health.show-details` property. This must be set to `always` as follows:

```
management.endpoint.health.show-details: "always"
```

## Stale information in Application Live View

### Symptom

You can find your app in the UI, but it is an old instance that no longer exists while the new instance doesn't show up yet.

### Solution

To troubleshoot:

1. View the Application Live View connector pod logs to see if the connector is sending updates to the back end.
2. Delete the connector pod to recreate it by running:

```
kubect1 -n app-live-view-connector delete pods -l=name=application-live-view-co
nconnector
```

## Unable to find CertificateRequests in Application Live View convention

### Symptom

The certificate request is missing for certificate `app-live-view-conventions/appliveview-webhook-cert`.

### Solution

To troubleshoot:

1. Run `kubectl get certificaterequest -A` to see if the certificate request is missing for Application Live View convention.
2. Delete the secret `appliveview-webhook-cert` that corresponds to the certificate in the `app-live-view-conventions` namespace by running:

```
kubectl delete secret appliveview-webhook-cert -n app-live-view-conventions
```

This recreates the certificate request and updates the corresponding certificate.

## No live information for pod with ID

### Symptom

In Tanzu Application Platform GUI, you receive the error `No live information for pod with id`.

### Cause

This might happen because of stale information in Application Live View because it is an old instance that no longer exists while the new instance doesn't show up yet.

### Solution

The workaround is to delete the connector pod so it is re-created by running:

```
kubectl -n app-live-view-connector delete pods -l=name=application-live-view-connector
```

## Cannot override the actuator path in the labels

### Symptom

You are unable to override the actuator path in the labels as part of the workload deployment.

### Cause

The changes to add or override the labels or annotations in the `Workload` are in progress. The changes from the `Workload` must be propagated up through the supply chain for the `PodIntent` to see the new changes.

## Cannot configure SSL in appliveview-connector

### Symptom

This might be because `sslDeactivated` flag in the values YAML file does not accept values without quotes.

### Cause

The `sslDeactivated` Boolean flag is treated as a string in the Kubernetes YAML file.

### Solution

You must specify the value within double quotation marks for the configuration to be picked up.

## Verify the labels in your workload YAML file

To verify that the labels in your workload YAML file are working:

1. Verify the app live view convention webhook is running properly by running:

```
kubectl get pods -n app-live-view | grep webhook
```

2. Verify the conventions controller is running properly by running:

```
kubectl get pods -n conventions-system
```

3. Verify that the conventions are applied properly to the PodSpec by running:

```
kubectl get podintents.conventions.carto.run WORKLOAD-NAME -oyaml
```

Where `WORKLOAD-NAME` is the name of your workload.

If everything works correctly, the status will contain a transformed template that includes the labels added as part of your workload YAML file. For example:

```
status:
conditions:
- lastTransitionTime: "2021-10-26T11:26:35Z"
  status: "True"
  type: ConventionsApplied
- lastTransitionTime: "2021-10-26T11:26:35Z"
  status: "True"
  type: Ready
observedGeneration: 1
template:
  metadata:
    annotations:
      conventions.carto.run/applied-conventions: |-
        appliveview-sample/app-live-view-connector
        appliveview-sample/app-live-view-appflavours
        appliveview-sample/app-live-view-systemproperties
    labels:
      tanzu.app.live.view: "true"
      tanzu.app.live.view.application.flavours: spring-boot
      tanzu.app.live.view.application.name: petclinic
  spec:
    containers:
      - env:
          - name: JAVA_TOOL_OPTIONS
            value: -Dmanagement.endpoint.health.show-details=always -Dmanagement.en
dpoints.web.exposure.include=*
        image: index.docker.io/kdvolder/alv-spring-petclinic:latest@sha256:1aa7bd22
8137471ea38ce36cbf5ffcd629eabeb8ce047f5533b7b9176ff51f98
        name: workload
        resources: {}
```

## Override labels set by the Application Live View convention service

It is not possible to override the labels set by the Application Live View convention service for the workload deployment in Tanzu Application Platform. The labels `tanzu.app.live.view`,



`tanzu.app.live.view.application.flavours` and `tanzu.app.live.view.application.name` cannot be overridden. The default values set by the Application Live View convention server are used.

However, if you want to override `management.endpoints.web.exposure.include` or `management.endpoint.health.show-details`, you can override these environment properties in `application.properties` or `application.yml` in the Spring Boot Application before deploying the workload in Tanzu Application Platform. Environment properties updated in your app take precedence over the default values set by Application Live View convention server.

## Configure labels when `management.endpoints.web.base-path` and `management.server.port` are set

If the custom actuator context path is configured as follows:

```
management.endpoints.web.base-path=/manage
management.server.port=8085
```

Configure the connector as follows:

```
tanzu.app.live.view.application.actuator.path=/manage    (manage is the custom actuator
path set on the application)
tanzu.app.live.view.application.actuator.port=8085      (8085 is the custom management se
rver port set on the application)
```

## Uninstall Application Live View

This topic tells you how to uninstall Application Live View from Tanzu Application Platform (commonly known as TAP).

To uninstall the Application Live View back end, Application Live View connector, and Application Live View convention server, run:

```
tanzu package installed delete appliveview -n tap-install
tanzu package installed delete appliveview-connector -n tap-install
tanzu package installed delete appliveview-conventions -n tap-install
```

## Overview of Application Single Sign-On for VMware Tanzu® 3.0.0

Application Single Sign-On for VMware Tanzu® (AppSSO) provides APIs for curating and consuming a “Single Sign-On as a service” offering on Tanzu Application Platform.

To get started with AppSSO, see [Get started with Application Single Sign-On](#).

With AppSSO, Service Operators can configure and deploy authorization servers. Application Operators can then configure their Workloads with these authorization servers to provide Single Sign-On to their end-users.

AppSSO allows integrating authentication and authorization decisions early in the software development and release life cycle. It provides a seamless transition for workloads from development to production when including Single Sign-On solutions in your software.

It’s easy to get started with AppSSO, deploy an authorization server with static test users, and eventually progress to multiple authorization servers of production-grade scale with token key rotation, multiple upstream identity providers, configured secure storage, and client restrictions.

AppSSO’s authorization server is based off of Spring Authorization Server project. For more information, see [Spring documentation](#).

# Overview of Application Single Sign-On for VMware Tanzu®

## 3.0.0

Application Single Sign-On for VMware Tanzu® (AppSSO) provides APIs for curating and consuming a “Single Sign-On as a service” offering on Tanzu Application Platform.

To get started with AppSSO, see [Get started with Application Single Sign-On](#).

With AppSSO, Service Operators can configure and deploy authorization servers. Application Operators can then configure their Workloads with these authorization servers to provide Single Sign-On to their end-users.

AppSSO allows integrating authentication and authorization decisions early in the software development and release life cycle. It provides a seamless transition for workloads from development to production when including Single Sign-On solutions in your software.

It's easy to get started with AppSSO, deploy an authorization server with static test users, and eventually progress to multiple authorization servers of production-grade scale with token key rotation, multiple upstream identity providers, configured secure storage, and client restrictions.

AppSSO's authorization server is based off of Spring Authorization Server project. For more information, see [Spring documentation](#).

## Get started with Application Single Sign-On

This topic tells you about concepts important to getting started with Application Single Sign-On (commonly called AppSSO).

Use this topic to learn how to:

1. [Set up your first authorization server](#).
2. [Provision a ClientRegistration](#).
3. [Deploy an application](#) that uses the provisioned ClientRegistration to enable SSO.

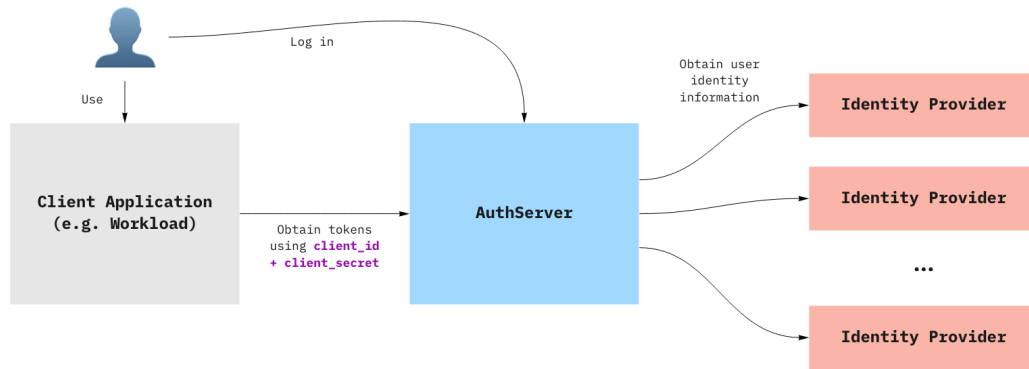
After completing these steps, you can proceed with [securing a Workload](#).

## Prerequisites

You must install AppSSO on your Tanzu Application Platform cluster. For more information, see [Install AppSSO](#).

## Key concepts

At the core of AppSSO is the concept of an Authorization Server, outlined by the [AuthServer custom resource](#). Service Operators create those resources to provision running Authorization Servers, which are [OpenID Connect Providers](#). They issue [ID Tokens](#) to Client applications, which contain identity information about the end user such as email, first name, last name and so on.



When a Client application uses an AuthServer to authenticate an End-User, the typical steps are:

1. The End-User visits the Client application
2. The Client application redirects the End-User to the AuthServer, with an OAuth2 request
3. The End-User logs in with the AuthServer, usually using an external Identity Provider (e.g. Google, Azure AD)
  1. Identity Providers are set up by Service Operators
  2. AuthServers may use various protocols to obtain identity information about the user, such as OpenID Connect, SAML or LDAP, which may involve additional redirects
4. The AuthServer redirects the End-User to the Client application with an authorization code
5. The Client application exchanges with the AuthServer for an `id_token`
  1. The Client application does not know how the identity information was obtained by the AuthServer, it only gets identity information in the form of an ID Token.

ID Tokens are JSON Web Tokens containing standard Claims about the identity of the user (e.g. name, email, etc) and about the token itself (e.g. “expires at”, “audience”, etc.). Here is an example of an `id_token` as issued by an Authorization Server:

```

{
  "iss": "https://appsso.example.com",
  "sub": "213435498y",
  "aud": "my-client",
  "nonce": "fkg0-90_mg",
  "exp": 1656929172,
  "iat": 1656928872,
  "name": "Jane Doe",
  "given_name": "Jane",
  "family_name": "Doe",
  "email": "jane.doe@example.com"
}
  
```

ID Tokens are signed by the `AuthServer`, using `Token Signature Keys`. Client applications may verify their validity using the AuthServer’s public keys.

## Next steps

- [Provision an AuthServer](#)

## Get started with Application Single Sign-On

This topic tells you about concepts important to getting started with Application Single Sign-On (commonly called AppSSO).

Use this topic to learn how to:

1. [Set up your first authorization server.](#)
2. [Provision a ClientRegistration.](#)
3. [Deploy an application](#) that uses the provisioned ClientRegistration to enable SSO.

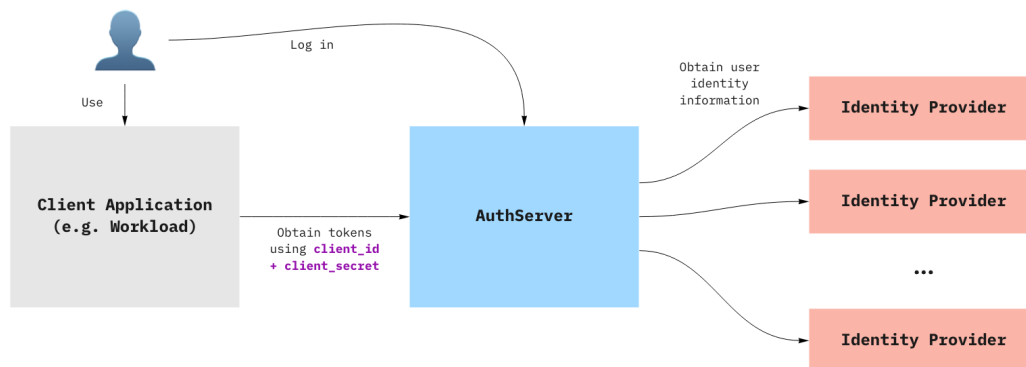
After completing these steps, you can proceed with [securing a Workload](#).

## Prerequisites

You must install AppSSO on your Tanzu Application Platform cluster. For more information, see [Install AppSSO](#).

## Key concepts

At the core of AppSSO is the concept of an Authorization Server, outlined by the [AuthServer custom resource](#). Service Operators create those resources to provision running Authorization Servers, which are [OpenID Connect Providers](#). They issue [ID Tokens](#) to Client applications, which contain identity information about the end user such as email, first name, last name and so on.



When a Client application uses an AuthServer to authenticate an End-User, the typical steps are:

1. The End-User visits the Client application
2. The Client application redirects the End-User to the AuthServer, with an OAuth2 request
3. The End-User logs in with the AuthServer, usually using an external Identity Provider (e.g. Google, Azure AD)
  1. Identity Providers are set up by Service Operators
  2. AuthServers may use various protocols to obtain identity information about the user, such as OpenID Connect, SAML or LDAP, which may involve additional redirects
4. The AuthServer redirects the End-User to the Client application with an authorization code
5. The Client application exchanges with the AuthServer for an `id_token`
  1. The Client application does not know how the identity information was obtained by the AuthServer, it only gets identity information in the form of an ID Token.

[ID Tokens](#) are JSON Web Tokens containing standard Claims about the identity of the user (e.g. name, email, etc) and about the token itself (e.g. "expires at", "audience", etc.). Here is an example of an `id_token` as issued by an Authorization Server:

```
{
  "iss": "https://appsso.example.com",
  "sub": "213435498y",
```

```

"aud": "my-client",
"nonce": "fkg0-90_mg",
"exp": 1656929172,
"iat": 1656928872,
"name": "Jane Doe",
"given_name": "Jane",
"family_name": "Doe",
"email": "jane.doe@example.com"
}

```

ID Tokens are signed by the [AuthServer](#), using [Token Signature Keys](#). Client applications may verify their validity using the AuthServer's public keys.

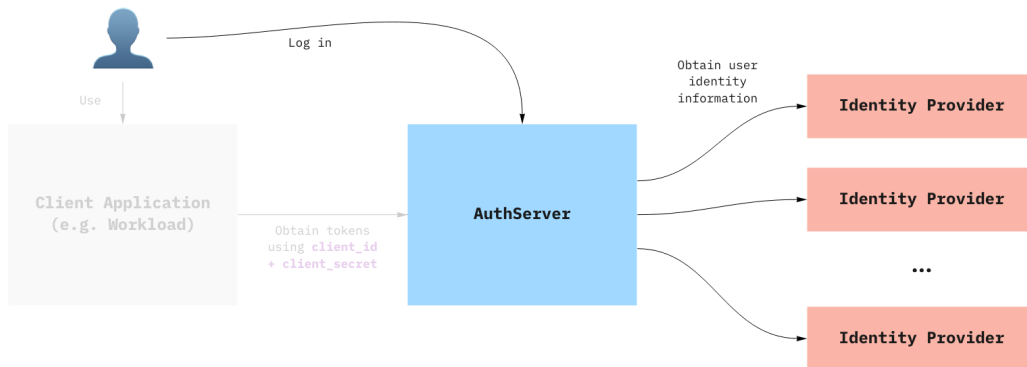
## Next steps

- [Provision an AuthServer](#)

## Provision an AuthServer

This topic tells you how to provision an AuthServer for Application Single Sign-On (commonly called AppSSO). Use this topic to learn how to:

1. Set up your first authorization server in the `default` namespace.
2. Ensure it is running so that users can log in.



## Prerequisites

You must install AppSSO on your Tanzu Application Platform cluster and ensure that your Tanzu Application Platform installation is correctly configured.

AppSSO is installed with the `run`, `iterate`, and `full` profiles, no extra steps required.

To verify AppSSO is installed on your cluster, run:

```
tanzu package installed list -A | grep "sso.apps.tanzu.vmware.com"
```

For more information about the Tanzu Application Platform installation, see [Install Tanzu Application Platform](#).

For more information about the AppSSO installation, see [Install AppSSO](#).

## Provision an AuthServer

Deploy your first Authorization Server along with an `RSAPKey` key for signing tokens.



### Caution

This `AuthServer` example uses an unsafe testing-only identity provider. Never use it in production environments. For more information about identity providers, see [Identity providers](#).

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: my-authserver-example
  namespace: default
  labels:
    name: my-first-auth-server
    env: tutorial
  annotations:
    sso.apps.tanzu.vmware.com/allow-client-namespaces: "default"
    sso.apps.tanzu.vmware.com/allow-unsafe-issuer-uri: ""
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
spec:
  replicas: 1
  tls:
    deactivated: true
  identityProviders:
    - name: "internal"
      internalUnsafe:
        users:
          - username: "user"
            password: "password"
            email: "user@example.com"
            emailVerified: true
            roles:
              - "user"
  tokenSignature:
    signAndVerifyKeyRef:
      name: "authserver-signing-key"
---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: authserver-signing-key
  namespace: default
spec:
  secretTemplate:
    type: Opaque
    stringData:
      key.pem: $(privateKey)
      pub.pem: $(publicKey)
```

You can wait for the `AuthServer` to become ready with:

```
kubectl wait --for=condition=Ready authserver my-authserver-example
```

Alternatively, you can inspect your `AuthServer` like any other resource:

```
kubectl get authservers.sso.apps.tanzu.vmware.com --all-namespaces
```

and you should see:

```
NAMESPACE NAME                REPLICAS ISSUER URI
CLIENTS STATUS
default my-authserver-example 1          http://my-authserver-example.default.<your do
main> 0      Ready
```

As you can see your `AuthServer` gets an issuer URI templated. This is its endpoint. You can find an `AuthServer`'s issuer URI in its status:

```
kubectl get authservers.sso.apps.tanzu.vmware.com my-authserver-example -o jsonpath
='{.status.issuerURI}'
```

Open your `AuthServer`'s issuer URI in your browser. You should see a login page. Log in using username = `user` and password = `password`.

You can review the standard OpenID information of your `AuthServer` by visiting `http://my-authserver-example.default.<your domain>/.well-known/openid-configuration`.



### Important

If the issuer URIs domain is not yours, the AppSSO package installation must be updated. For more information, see [Install Application Single Sign-On](#).

If you are using TKGm or TKGs, which have customizable in-cluster communication CIDR ranges, there is a [known issue](#) regarding AppSSO making requests to external identity providers with `http` rather than `https`.

## The AuthServer spec in detail

Here is a detailed explanation of the `AuthServer` you have applied in the above section. This is intended to give you an overview of the different configuration values that were passed in. It is not intended to describe all the ins-and-outs, but there are links to related docs in each section.

Feel free to skip ahead.

### Metadata

```
metadata:
  labels:
    name: my-first-auth-server
    env: tutorial
  annotations:
    sso.apps.tanzu.vmware.com/allow-client-namespaces: "default"
    sso.apps.tanzu.vmware.com/allow-unsafe-issuer-uri: ""
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
```

The `metadata.labels` uniquely identify the `AuthServer`. They are used as selectors by `ClientRegistrations`, to declare from which authorization server a specific client obtains tokens from.

The `sso.apps.tanzu.vmware.com/allow-client-namespaces` annotation restricts the namespaces in which you can create a `ClientRegistrations` targeting this authorization server. In this case, the authorization server will only pick up client registrations in the `default` namespace.

The `sso.apps.tanzu.vmware.com/allow-unsafe-...` annotations enable “development mode” features, useful for testing. Those should not be used for production-grade authorization servers.

For more information about annotations and labels in `AuthServer` resource, see [Annotation and labels](#).

### TLS & issuer URI

```
spec:
  tls:
```

```
deactivated: true
```

The `tls` field configures whether and how to obtain a certificate for an `AuthServer` to secure its issuer URI. If you deactivate `tls`, the issuer URI uses plain HTTP.



### Caution

Plain HTTP access is for development purposes only and must never be used in production. For more information about the production readiness with TLS, see [Issuer URI & TLS](#).

## Token Signature

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
# ...
spec:
  tokenSignature:
    signAndVerifyKeyRef:
      name: "authserver-signing-key"
---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: authserver-signing-key
  namespace: default
spec:
  secretTemplate:
    type: Opaque
    stringData:
      key.pem: $(privateKey)
      pub.pem: $(publicKey)
```

The token signing key is the private RSA key used to sign ID Tokens, using [JSON Web Signatures](#), and clients use the public key to verify the provenance and integrity of the ID tokens. The public keys used for validating messages are published as [JSON Web Keys](#) at `{.status.issuerURI}/oauth2/jwks`.

The `spec.tokenSignature.signAndVerifyKeyRef.name` references a secret containing PEM-encoded RSA keys, both `key.pem` and `pub.pem`. In this specific example, we are using [Secretgen-Controller](#), a TAP dependency, to generate the key for us.

Learn more about [Token Signatures](#).

## Identity providers

```
spec:
  identityProviders:
    - name: "internal"
      internalUnsafe:
        users:
          - username: "user"
            password: "password"
            email: "user@example.com"
            roles:
              - "user"
```

AppSSO's authorization server delegates login and user management to external identity providers (IDP), such as Google, Azure Active Directory, Okta and so on. See diagram at the top of this topic



for more information.

In this example, we use an `internalUnsafe` identity provider. As the name implies, it is *not* an external IDP, but rather a list of hardcoded user/passwords. As the name also implies, this is not considered safe for production. Here, we declared a user with `username = user`, and `password = password`. For production setups, consider using OpenID Connect IDPs instead.

The `email` and `roles` fields are optional for internal users. However, they will be useful when we want to use SSO with a client application later in this guide.

**Caution**

VMware discourages using the `internalUnsafe` identity provider in production environments.

## Configuring storage

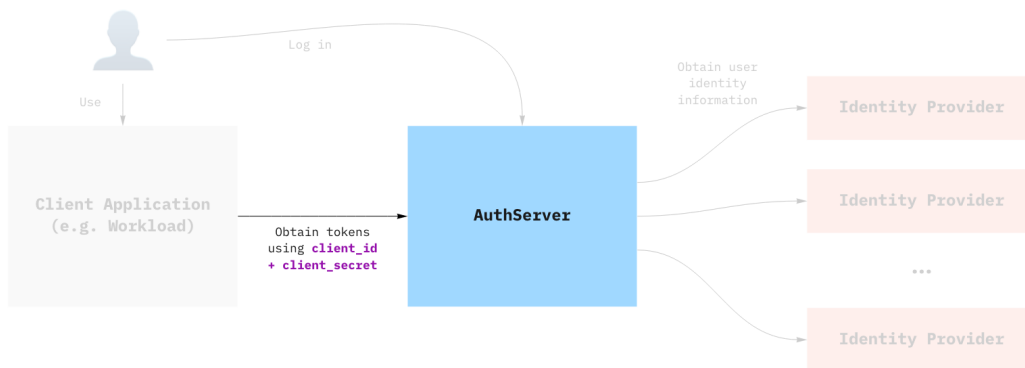
An `AuthServer` issues a Redis instance by default. It can be used for testing, prototyping and other non-production purposes. No additional configuration is required.

To configure your own storage that is ready for production, see [Storage](#).

## Provision a client registration

This topic tells you how to provision a client registration for Application Single Sign-On (commonly called AppSSO). Use this topic to learn how to:

1. Obtain credentials for the Authorization Server that you provisioned in [Provision your first AuthServer](#).
2. Verify that the credentials are valid using client-credentials flow.



## Prerequisites

Complete the steps described in [Get started with Application Single Sign-On](#).

## Creating the ClientRegistration

Assuming you have deployed the `AuthServer` as described previously, you can create and apply the following client registration:

**Note**

AppSSO uses `test-app.example.com` for `ClientRegistration.spec.redirectURIs[0]`. You must customize the URL to match the domain of your Tanzu Application Platform cluster. This is the URL to expose your test application in the next section.

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClientRegistration
metadata:
  name: my-client-registration
  namespace: default
spec:
  authServerSelector:
    matchLabels:
      name: my-first-auth-server
      env: tutorial
  redirectURIs:
    - "http://test-app.example.com/oauth2/callback"
  requireUserConsent: false
  clientAuthenticationMethod: basic
  authorizationGrantTypes:
    - "client_credentials"
    - "authorization_code"
  scopes:
    - name: "openid"
    - name: "email"
    - name: "profile"
    - name: "roles"
    - name: "message.read"
```

The AuthServer should now have this `ClientRegistration` registered. You can verify the status either by looking at the `ClientRegistrations.status` field, or looking at the `AuthServer` itself.

```
# Check the client registration
kubectl get clientregistration my-client-registration -n default -o yaml
# Check the authserver
kubectl get authservers
# NAME                                REPLICAS  ISSUER URI                                CLIENTS  TOKEN KE
YS
# my-authserver-example               1          http://authserver.example.com           1       1
#                                     ^
#                                     the AuthServer now has one client ^
```

AppSSO will create a secret containing the credentials that client applications will use, named after the client registration. The type of the secret is `servicebinding.io/oauth2`. You can obtain the values in the secret by running:

```
kubectl get secret my-client-registration -n default -o json | jq ".data | map_values (@base64d)"
# {
#   "authorization-grant-types": "client_credentials,authorization_code",
#   "client-authentication-method": "basic",
#   "client-id": "default_my-client-registration",
#   "client-secret": "PLACEHOLDER",
#   "issuer-uri": "http://authserver.example.com",
#   "provider": "appsso",
#   "scope": "openid,email,profile,roles,message.read",
#   "type": "oauth2"
# }
```

## Validating that the credentials are working

Before you deploy an app and make use of SSO, you can try the credentials from your machine to try and obtain an `access_token` using the `client_credentials` grant. You need the `client_id` and `client_secret` that were created as part of the client registration.

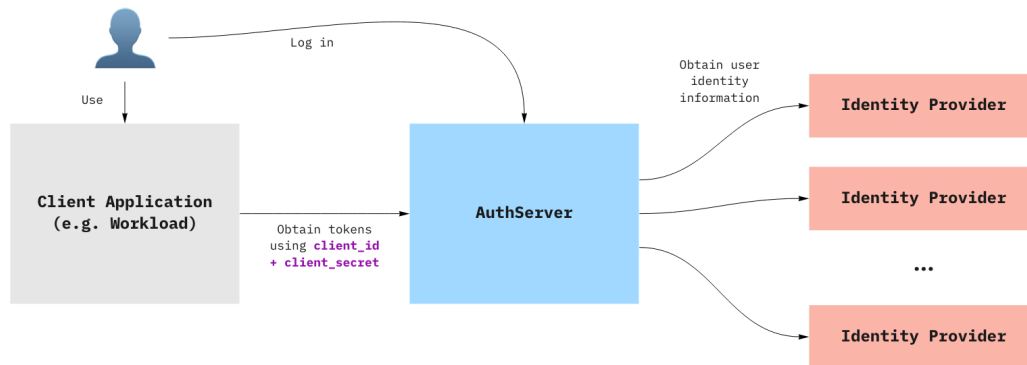
```
CLIENT_ID=$(kubectl get secret my-client-registration -n default -o jsonpath="{.data.client-id}" | base64 -d)
CLIENT_SECRET=$(kubectl get secret my-client-registration -n default -o jsonpath="{.data.client-secret}" | base64 -d)
ISSUER_URI=$(kubectl get secret my-client-registration -n default -o jsonpath="{.data.issuer-uri}" | base64 -d)
curl -XPOST "$ISSUER_URI/oauth2/token?grant_type=client_credentials&scope=message.read" -u "$CLIENT_ID:$CLIENT_SECRET"
```

You can decode the `access_token` using an online service, such as [JWT.io](https://jwt.io).

To learn more about grant types, see [Grant Types](#)

## Deploy an application with Application Single Sign-On

This topic tells you how to deploy a minimal Kubernetes application that is protected by Application Single Sign-On (commonly called AppSSO) by using the credentials that `ClientRegistration` creates.



For more information about how a Client application uses an AuthServer to authenticate an end user, see [AppSSO Overview](#).

## Prerequisites

You must complete the steps described in [Get started with Application Single Sign-On](#). If not, see [Provision a client registration](#).

## Deploy a minimal application

You are going to deploy a two-container pod, as a test application.

**Important**

AppSSO uses `test-app.example.com` for `HTTPProxy.spec.virtualhost.fqdn`. You must customize the URL to match the domain of your Tanzu Application Platform cluster. This URL must match what was set up in `ClientRegistration.spec.redirectURIs[0]` in [Provision a client registration](#)

```
---
apiVersion: apps/v1
kind: Deployment
```

```

metadata:
  name: test-application
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      name: test-application
  template:
    metadata:
      labels:
        name: test-application
    spec:
      containers:
        - image: bitnami/oauth2-proxy:7.3.0
          name: proxy
          ports:
            - containerPort: 4180
              name: proxy-port
              protocol: TCP
          env:
            - name: ISSUER_URI
              valueFrom:
                secretKeyRef:
                  name: my-client-registration
                  key: issuer-uri
            - name: CLIENT_ID
              valueFrom:
                secretKeyRef:
                  name: my-client-registration
                  key: client-id
            - name: CLIENT_SECRET
              valueFrom:
                secretKeyRef:
                  name: my-client-registration
                  key: client-secret
          command: [ "oauth2-proxy" ]
          args:
            - --oidc-issuer-url=$(ISSUER_URI)
            - --client-id=$(CLIENT_ID)
            - --insecure-oidc-skip-issuer-verification=true
            - --client-secret=$(CLIENT_SECRET)
            - --cookie-secret=0000000000000000
            - --cookie-secure=false
            - --http-address=http://:4180
            - --provider=oidc
            - --scope=openid email profile roles
            - --email-domain=*
            - --insecure-oidc-allow-unverified-email=true
            - --oidc-groups-claim=roles
            - --upstream=http://127.0.0.1:8000
            - --redirect-url=http://test-app.example.com/oauth2/callback
            - --skip-provider-button=true
            - --pass-authorization-header=true
            - --prefer-email-to-user=true
        - image: python:3.9
          name: application
          resources:
            limits:
              cpu: 100m
              memory: 100Mi
          command: [ "python" ]
          args:
            - -c
            - |
              from http.server import HTTPServer, BaseHTTPRequestHandler

```

```

import base64
import json

class Handler(BaseHTTPRequestHandler):
    def do_GET(self):
        if self.path == "/token":
            self.token()
            return
        else:
            self.greet()
            return

    def greet(self):
        username = self.headers.get("x-forwarded-user")
        self.send_response(200)
        self.send_header("Content-type", "text/html")
        self.end_headers()
        page = f"""
        <h1>It Works!</h1>
        <p>You are logged in as <b>{username}</b></p>
        """
        self.wfile.write(page.encode("utf-8"))

    def token(self):
        token = self.headers.get("Authorization").split("Bearer ")[-1]
        payload = token.split(".")[1]
        decoded = base64.b64decode(bytes(payload, "utf-8") + b'==').decode("utf-8")

        self.send_response(200)
        self.send_header("Content-type", "application/json")
        self.end_headers()
        self.wfile.write(decoded.encode("utf-8"))

server_address = ('', 8000)
httpd = HTTPServer(server_address, Handler)
httpd.serve_forever()

---
apiVersion: v1
kind: Service
metadata:
  name: test-application
  namespace: default
spec:
  ports:
    - port: 80
      targetPort: 4180
  selector:
    name: test-application

---
apiVersion: projectcontour.io/v1
kind: HTTPProxy
metadata:
  name: test-application
  namespace: default
spec:
  virtualhost:
    fqdn: test-app.example.com
  routes:
    - conditions:
        - prefix: /
      services:
        - name: test-application
          port: 80

```

Now you can navigate to <http://test-app.example.com/>. It may ask you to log into the AuthServer you haven't already. You can also navigate to <http://test-app.example.com/token> if you wish to see the contents of the ID token.

## Deployment manifest

The application was deployed as a two-container pod: one for the app, and one for handling login.

- The main container is called `application`, and runs a bare-bones Python HTTP server, that reads from the `Authorization` header from incoming requests and returns the decoded `id_token`.
- The second container, called `proxy`, is a sidecar container, an “Ambassador”. It receives traffic for the Pod, performs OpenID authentication using `OAuth2 Proxy`, and proxies requests to the `application` with some added headers containing identity information.

Along with this deployment, there is a `Service + HTTPProxy`, to expose the application to the outside world.

## OAuth2-Proxy

The setup of the above `OAuth2 Proxy` is minimal, and is not considered suitable for production use. To configure it for production, please refer to the official documentation.

Note that `OAuth2 Proxy` requires some claims to be present in the `id_token`, notably the `email` claim and the non-standard `groups` claim. The `groups` claim maps to AppSSO's `roles` claim. Therefore, for this proxy to work with AppSSO, users *MUST* have an e-mail defined, and at least one entry in `roles`. If the proxy container logs an error stating `Error redeeming code during OAuth2 callback: could not get claim "groups" [...]`, make sure that the user has `roles` provided in the `identityProvider`.

## Application Single Sign-On for Platform Operators

This topic tells you how to manage the Application Single Sign-On (commonly called AppSSO) package installation and what it installs. Use this topic to learn:

- [Install Application Single Sign-On](#)
- [Configure Application Single Sign-On](#)
- [RBAC for Application Single Sign-On](#)
- [Application Single Sign-On for OpenShift clusters](#)
- [Upgrade Application Single Sign-On](#)
- [Uninstall Application Single Sign-On](#)

## Application Single Sign-On for Platform Operators

This topic tells you how to manage the Application Single Sign-On (commonly called AppSSO) package installation and what it installs. Use this topic to learn:

- [Install Application Single Sign-On](#)
- [Configure Application Single Sign-On](#)
- [RBAC for Application Single Sign-On](#)
- [Application Single Sign-On for OpenShift clusters](#)
- [Upgrade Application Single Sign-On](#)

- [Uninstall Application Single Sign-On](#)

## Install Application Single Sign-On

This topic tells you how to install Application Single Sign-On (commonly called AppSSO) from the Tanzu Application Platform (commonly called TAP) package repository.



### Note

Follow the steps in this topic if you do not want to use a profile to install Application Single Sign-On. For more information about profiles, see [Components and installation profiles](#).

## What's inside

The AppSSO package will install the following resources:

- The `appsso` Namespace with a Deployment of the AppSSO controller and Services for Webhooks
- A `ServiceAccount` with RBAC outlined in detail [here](#)
- [AuthServer and ClientRegistration CRDs](#)

## Prerequisites

Before installing AppSSO, please ensure you have Tanzu Application Platform installed on your Kubernetes cluster.

## Installation

1. Learn more about the AppSSO package:

```
tanzu package available get sso.apps.tanzu.vmware.com --namespace tap-install
```

2. Install the AppSSO package:

```
tanzu package install appsso \
  --namespace tap-install \
  --package-name sso.apps.tanzu.vmware.com \
  --version 3.0.0
```

3. Confirm the package has reconciled successfully:

```
tanzu package installed get appsso --namespace tap-install
```

## Configure Application Single Sign-On

This topic tells you how to configure Application Single Sign-On (commonly called AppSSO).

## TAP values

Most commonly, the AppSSO package installation is configured through TAP's meta package installation. The TAP package has a `shared` top-level configuration key for sharing common configuration between the packages it installs.

AppSSO inherits the `shared.{ingress_domain, ingress_issuer, ca_cert_data, kubernetes_distribution}` configuration values from Tanzu Application Platform. You can configure the AppSSO-specific parameters under `appsso`. AppSSO-specific configuration has precedence over the shared values of Tanzu Application Platform.

For example:

```
#!/ my-tap-values.yaml

shared:
# Shared configuration goes here.

appsso:
# AppSSO-specific configuration goes here.
```

## domain\_name

The AppSSO package has one required configuration value, its `domain_name`. It is used for templating the issuer URI for `AuthServer`. `domain_name` must be the shared ingress domain of your TAP package installation. If your TAP installation is configured with `shared.ingress_domain`, then AppSSO will inherit the correct configuration.



### Note

If omitted, `domain_name` is set to `shared.ingress_domain`.

## domain\_template

You can customize how AppSSO template's issuerURIs with the `domain_template` configuration. This is a Golang `text/template`. The default is `"{{.Name}}.{{.Namespace}}.{{.Domain}}"`.

The domain template will be applied with the given `domain_name` and the `AuthServer`'s name and namespace:

- `{{.Domain}}` will evaluate to the configured `domain_name`
- `{{.Name}}` will evaluate to `AuthServer.metadata.name`
- `{{.Namespace}}` will evaluate to `AuthServer.metadata.namespace`

To be able to use a wild-card certificate, consider `"{{.Name}}-{{.Namespace}}.{{.Domain}}"`.

It is strongly recommended to keep the name and namespace part of the template to avoid domain name collisions.

## default\_authserver\_clusterissuer

You can denote a `cert-manager.io/v1/ClusterIssuer` as a default issuer for `AuthServer.spec.tls.issuerRef` and omit `AuthServer.spec.tls`. When the value of `AuthServer.spec.tls.issuerRef` is the empty string `""`, no default issuer is assumed and `AuthServer.spec.tls` is required.

If you configured `shared.ingress_issuer` and omitted `default_authserver_clusterissuer` while installing Tanzu Application Platform, AppSSO uses the ingress issuer of Tanzu Application Platform and sets `default_authserver_clusterissuer` to `shared.ingress_issuer`.

## ca\_cert\_data



You can configure trust for custom CAs by providing their certificates as a PEM bundle to `ca_cert_data`. As a result, all `AuthServers` trust your custom CAs.

This is useful if you have [identity providers](#) serving certificates from a custom CA and [configuring AuthServer storage](#).

Alternatively, you can [configure trust for a single AuthServer](#).



#### Note

AppSSO-specific `ca_cert_data` is concatenated with `shared.ca_cert_data`. The resulting PEM bundle contains both.

## kubernetes\_distribution

This setting toggles behavior specific to Kubernetes distribution. Currently, the only supported values are `"` and `openshift`.

AppSSO installs *OpenShift*-specific RBAC and resources.



#### Note

If omitted, `kubernetes_distribution` is set to `shared.kubernetes_distribution`.

## Configuration schema

The entire available configuration schema for AppSSO is:

```
#@schema/desc "Optional: Kubernetes platform distribution that this package is being i
nstalled on. Accepted values: ['', 'openshift']"
kubernetes_distribution: ""

#@schema/desc "Domain name for AuthServers"
domain_name: "example.com"

#@schema/desc "Optional: Golang template/text string for constructing AuthServer FQDN
s"
domain_template: "{{.Name}}.{{.Namespace}}.{{.Domain}}"

#@schema/desc "Optional: PEM-encoded certificate data for AuthServers to trust TLS con
nections with a custom CA"
ca_cert_data: ""

#@schema/desc "Optional: Interval at which the controller will re-synchronize applied
resources"
resync_period: "2h"

#@schema/desc "Optional: Number of controller replicas to deploy"
replicas: 1

#@schema/desc "Optional: Resource requirements the controller deployment"
resources:
  requests:
    #@schema/desc "CPU request of the controller"
    cpu: "20m"
    #@schema/desc "Memory request of the controller"
    memory: "100Mi"
  limits:
    #@schema/desc "CPU limit of the controller"
```

```

cpu: "500m"
#@schema/desc "Memory limit of the controller"
memory: "500Mi"

#@schema/desc "Optional: Schema-free extension point for internal, package-private con
figuration (Unsupported! Use at your own risk.)"
#@schema/type any=True
internal: { }

```

## RBAC for Application Single Sign-On

The AppSSO package aggregates the following permissions into TAP's well-known roles:

- app-operator

```

- apiGroups:
  - sso.apps.tanzu.vmware.com
resources:
  - clientregistrations
verbs:
  - "*"

```

- app-editor

```

- apiGroups:
  - sso.apps.tanzu.vmware.com
resources:
  - clientregistrations
verbs:
  - get
  - list
  - watch

```

- app-viewer

```

- apiGroups:
  - sso.apps.tanzu.vmware.com
resources:
  - clientregistrations
verbs:
  - get
  - list
  - watch

```

- service-operator

```

- apiGroups:
  - sso.apps.tanzu.vmware.com
resources:
  - authserver
verbs:
  - "*"

```

To manage the life cycle of AppSSO's [APIs](#), the AppSSO controller's [ServiceAccount](#) has a [ClusterRole](#) with the following permissions:

```

- apiGroups:
  - sso.apps.tanzu.vmware.com
resources:
  - authservers
verbs:
  - get

```

```

- list
- watch
- apiGroups:
  - sso.apps.tanzu.vmware.com
resources:
  - authservers/status
verbs:
  - patch
  - update
- apiGroups:
  - sso.apps.tanzu.vmware.com
resources:
  - authservers/finalizers
verbs:
  - "*"
- apiGroups:
  - sso.apps.tanzu.vmware.com
resources:
  - clientregistrations
verbs:
  - get
  - list
  - watch
- apiGroups:
  - sso.apps.tanzu.vmware.com
resources:
  - clientregistrations/status
verbs:
  - patch
  - update
- apiGroups:
  - sso.apps.tanzu.vmware.com
resources:
  - clientregistrations/finalizers
verbs:
  - "*"
- apiGroups:
  - ""
resources:
  - secrets
  - configmaps
  - services
  - serviceaccounts
verbs:
  - "*"
- apiGroups:
  - apps
resources:
  - deployments
verbs:
  - "*"
- apiGroups:
  - rbac.authorization.k8s.io
resources:
  - roles
  - rolebindings
verbs:
  - "*"
- apiGroups:
  - cert-manager.io
resources:
  - certificates
  - issuers
verbs:
  - "*"
- apiGroups:

```

```

- cert-manager.io
resources:
- clusterissuers
verbs:
- get
- list
- watch
- apiGroups:
  - networking.k8s.io
  resources:
  - ingresses
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - events
  verbs:
  - create
  - update
  - patch
- apiGroups:
  - coordination.k8s.io
  resources:
  - leases
  verbs:
  - create
  - get
  - update

```

AppSSO installs *OpenShift*-specific RBAC and resources.

## Application Single Sign-On for OpenShift clusters

On OpenShift clusters, AppSSO must run with a custom SecurityContextConstraint (SCC) to enable compliance with restricted Kubernetes Pod Security Standards. Tanzu Application Platform configures the following SCC for AppSSO controller and its `AuthServer` managed resources when you configure the `kubernetes_distribution: openshift` key in the `tap-values.yaml` file.

Specification follows:

```

---
kind: SecurityContextConstraints
apiVersion: security.openshift.io/v1
metadata:
  name: appssso-scc
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: false
allowPrivilegedContainer: false
allowedCapabilities: null
defaultAddCapabilities: null
fsGroup:
  type: MustRunAs
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities:
- KILL
- MKNOD
- SETUID
- SETGID

```

```
runAsUser:
  type: MustRunAsNonRoot
seLinuxContext:
  type: MustRunAs
volumes:
- configMap
- downwardAPI
- emptyDir
- persistentVolumeClaim
- projected
- secret
seccompProfiles:
- 'runtime/default'
```

AppSSO controller's `ServiceAccount` is given the following additional permissions, including a `use` permission for AppSSO SCC, so `AuthServer` can use the custom SCC:

```
- apiGroups:
  - security.openshift.io
resources:
  - securitycontextconstraints
verbs:
  - "get"
  - "list"
  - "watch"
```

```
- apiGroups:
  - security.openshift.io
resourceNames:
  - appssso-scc
resources:
  - securitycontextconstraints
verbs:
  - "use"
```

## Upgrade Application Single Sign-On

This topic tells you how to upgrade Application Single Sign-On (commonly called AppSSO) outside of a Tanzu Application Platform profile installation. If you installed Tanzu Application Platform through a profile, see [Upgrade Tanzu Application Platform](#) instead.

For help on migrating your resources in between versions, see the [migration guides](#).

If you installed the `AppSSO` package on its own, and not as part of `TAP`, you can upgrade it individually by running:

```
tanzu package installed update PACKAGE_INSTALLATION_NAME -p sso.apps.tanzu.vmware.com
-v 3.0.0 --values-file PATH_TO_YOUR_VALUES_YAML -n YOUR_INSTALL_NAMESPACE
```



### Note

You can also upgrade AppSSO as part of upgrading Tanzu Application Platform as a whole. See [Upgrading Tanzu Application Platform](#) for more information.

## Migration guides

[v2.0.0 to v3.0.0](#)

VMware recommends that you recreate your `AuthServers` after upgrading your AppSSO to `v3.0.0` with the following changes:

- Migrate field `.spec.tls.disabled` to `.spec.tls.deactivated`.

### `v1.0.0` to `v2.0.0`

VMware recommends that you recreate your `AuthServers` after upgrading your AppSSO to `v2.0.0` with the following changes:

- Migrate from `.spec.issuerURI` to `.spec.tls`:



#### Note

AppSSO templates your issuer URI and enables TLS. When using the newer `.spec.tls`, a custom `Service` and an ingress resource are no longer required.

It is not recommended to continue using `.spec.issuerURI` in AppSSO `v2.0.0`. To use `.spec.issuerURI` in AppSSO `v2.0.0`, you must provide a `Service` and an ingress resource as in AppSSO `v1.0.0`.

1. Configure one of `.spec.tls.{issuerRef, certificateRef, secretRef}`. See [Issuer URI & TLS](#) for more information.
  2. (Optional) Disable TLS with `.spec.tls.disabled`.
  3. Remove `.spec.issuerURI`.
  4. Delete your `AuthServer`-specific `Service` and ingress resources.
  5. Apply your `AuthServer`. You can find its issuer URI in `.status.issuerURI`.
  6. Update the redirect URIs in your upstream identity providers.
- If you use the `internalUnsafe` identity provider to migrate existing users by replacing the bcrypt hash through the plain-text equivalent. You can still use existing bcrypt passwords by prefixing them with `{bcrypt}`:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  # ...
spec:
  identityProviders:
    - name: internal
      internalUnsafe:
        users:
          # v1.0
          - username: test-user-1
            password: $2a$10$201z9o/tHlocFsHFTo0plukh03ApBYe4dRiXcqeyRQH6CNntS8jWK # bcrypt-encoded "password"
          # ...
          # v2.0
          - username: "test-user-1"
            password: "{bcrypt}$2a$10$201z9o/tHlocFsHFTo0plukh03ApBYe4dRiXcqeyRQH6CNntS8jWK" # same bcrypt hash, with {bcrypt} prefix
          - username: "test-user-2"
            password: "password" # plain text
          # ...
```

## Uninstall Application Single Sign-On

This topic tells you how to uninstall Application Single Sign-On (commonly called AppSSO).

Delete the AppSSO package by running:

```
tanzu package installed delete appssso --namespace tap-install
```

To permanently delete and exclude AppSSO package from your Tanzu Application Platform install, edit your Tanzu Application Platform values file by including the following configuration:

```
excluded_packages:
  - sso.apps.tanzu.vmware.com
```

For more information, navigate to [Exclude packages from a Tanzu Application Platform profile](#).

## Application Single Sign-On for Service Operators

The following topics tell you how to configure a fully operational authorization server for Application Single Sign-On (commonly called AppSSO):

- [Annotations and labels](#)
- [Issuer URI and TLS](#)
- [TLS scenario guides](#)
- [CA certificates](#)
- [Configure Workloads to trust a custom CA](#)
- [Identity providers](#)
- [Token signatures](#)
- [Storage](#)
- [AuthServer readiness](#)
- [Scale AuthServer](#)
- [AuthServer audit logs](#)

`AuthServer` represents the request for an OIDC authorization server. It results in the deployment of an authorization server backed by Redis over mutual TLS if no [external storage](#) is explicitly configured.

You can configure the labels with which clients can select an `AuthServer`, the namespaces it allows clients from, its issuer URI, its token signature keys, identity providers and further details for its deployment.

For the full available configuration, `spec` and `status` see [the API reference](#).

## Application Single Sign-On for Service Operators

The following topics tell you how to configure a fully operational authorization server for Application Single Sign-On (commonly called AppSSO):

- [Annotations and labels](#)
- [Issuer URI and TLS](#)
- [TLS scenario guides](#)
- [CA certificates](#)

- [Configure Workloads to trust a custom CA](#)
- [Identity providers](#)
- [Token signatures](#)
- [Storage](#)
- [AuthServer readiness](#)
- [Scale AuthServer](#)
- [AuthServer audit logs](#)

`AuthServer` represents the request for an OIDC authorization server. It results in the deployment of an authorization server backed by Redis over mutual TLS if no [external storage](#) is explicitly configured.

You can configure the labels with which clients can select an `AuthServer`, the namespaces it allows clients from, its issuer URI, its token signature keys, identity providers and further details for its deployment.

For the full available configuration, `spec` and `status` see [the API reference](#).

## Annotations and labels for AppSSO

This topic tells you how to configure annotations and labels for Application Single Sign-On (commonly called AppSSO).

An `AuthServer` is selectable by `ClientRegistration` through labels. The namespace an `AuthServer` allows `ClientRegistrations` from is controlled with an annotation.

## Labels

`ClientRegistrations` select an `AuthServer` with `spec.authServerSelector`. Therefore, an `AuthServer` must have a set of labels that uniquely identifies it amongst all `AuthServer`. A `ClientRegistration` must match only one `AuthServer`. Registration fails if multiple or no `AuthServer` resources are matched.

For example:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  labels:
    env: dev
    ldap: True
    saml: True
# ...
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  labels:
    env: prod
    saml: True
# ...
```

## Allowing client namespaces

`AuthServer` controls which namespace it allows `ClientRegistrations` with the annotation:



```

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  annotations:
    sso.apps.tanzu.vmware.com/allow-client-namespaces: "*"

```

To allow `ClientRegistrations` from all or a restricted set of namespaces this annotation must be set. Its value is a comma-separated list of allowed Namespaces, e.g. `"app-team-red,app-team-green"`, or `"*"` if it should allow clients from all namespaces.



### Caution

If the annotation is missing, no clients are allowed.

## Unsafe configuration

`AuthServer` is designed to enforce secure and production-ready configuration. However, sometimes it is necessary to opt-out of those constraints, e.g. when deploying `AuthServer` on an *iterate* cluster.



### Caution

Allowing **unsafe** is not recommended for production.

## Unsafe identity provider

The `InternalUnsafe` identity provider cannot be used unless explicitly allowed by including the annotation `sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider` as follows:

```

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
spec:
  identityProviders:
    - name: static-users
      internalUnsafe:
        # ...

```

If the annotation is not present and an `InternalUnsafe` identity provider is configured the `AuthServer` will not apply.

## Unsafe issuer URI

It's not possible to use a plain HTTP issuer URI, unless it's explicitly allowed by including the annotation `sso.apps.tanzu.vmware.com/allow-unsafe-issuer-uri` as follows:

```

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-issuer-uri: ""

```

```
spec:
  issuerURI: http://this.is.unsafe
```

If the annotation is not present and a plain HTTP issuer URI is configured, the `AuthServer` does not apply.

## Issuer URI and TLS for AppSSO

This topic tells you how to configure the issuer URI and TLS for Application Single Sign-On (commonly called AppSSO).

### Overview

An `AuthServer` entry point for its clients and their end-users is called *issuer URI*. AppSSO will template the issuer URI and create a TLS-enabled `Ingress` for it. For this purpose, your platform operator configures the domain name and template. Once you created and `AuthServer` you can find the actual URL in `.status.issuerURI`.

You can configure whether and how to obtain a TLS certificate for the issuer URI by using `.spec.tls`. Unless TLS is deactivated, HTTPS is enforced. For example, requests for `http://` are redirected to `https://`. You can observe the TLS configuration in `.status.tls`.

If AppSSO is installed with a [default issuer](#), you can omit `AuthServer.spec.tls` and a TLS certificate is obtained automatically. This is the recommended approach for TLS.

For example:

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: login
  namespace: services
  # ...
spec:
  # ...
status:
  issuerURI: https://login.services.example.com
  tls:
    issuerRef:
      name: my-default-issuer
      kind: ClusterIssuer
      group: cert-manager.io
  # ...
```

This `AuthServer` is reachable at its templated issuer URI `https://login.services.example.com` and serves a TLS certificate obtained from `my-default-issuer`.

Learn how to configure TLS for your `AuthServer`:

- [Configure TLS by using a \(Cluster\)Issuer](#)
- [Configure TLS by using a Certificate](#)
- [Configure TLS by using a Secret](#)
- [Deactivate TLS](#)

There are many use-cases that pertain to TLS use. To find out which scenario applies to you and how to configure it, see [TLS scenario guides](#).

If your `AuthServer` obtains a certificate from a custom CA, you can enable App Operators to trust it. See [Allow Workloads to trust a custom CA AuthServer](#) for more information.

## Configure TLS by using a (Cluster)Issuer

You can obtain a TLS certificate for your `AuthServer` by referencing a `cert-manager.io/v1/Issuer` or `cert-manager.io/v1/ClusterIssuer`. This enables AppSSO to retrieve a `cert-manager.io/v1/Certificate` from the issuer and apply it to the `Ingress` configuration.

The composition of an `AuthServer` and a self-signed `Issuer` looks as follows:

```
---
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: my-selfsigned-issuer
  namespace: services
spec:
  selfSigned: { }

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: login
  namespace: services
  # ...
spec:
  tls:
    issuerRef:
      name: my-selfsigned-issuer
      # 'kind: Issuer' can be omitted. It is the default.
```

The composition of an `AuthServer` and a *self-signed* `ClusterIssuer` for looks as follows:

```
---
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: my-selfsigned-cluster-issuer
spec:
  selfSigned: { }

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: login
  namespace: services
  # ...
spec:
  tls:
    issuerRef:
      name: my-selfsigned-cluster-issuer
      kind: ClusterIssuer
```

Confirm that your `AuthServer` serves a TLS certificate from the specified issuer by visiting its `{.status.issuerURI}`.

For more information about cert-manager and its APIs, see [cert-manager documentation](#).

## Configure TLS by using a Certificate

In order to configure TLS for your `AuthServer` using a `cert-manager.io/v1/Certificate` you must know what its templated issuer URI will be. You can infer it from the AppSSO package's domain template.

The composition of an `AuthServer` and a `Certificate` looks as follows:

```

apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: login
  namespace: services
spec:
  dnsNames:
    - login.services.example.com
  issuerRef:
    name: my-cluster-issuer
    kind: ClusterIssuer
    secretName: login-cert

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: login
  namespace: services
  # ...
spec:
  tls:
    certificateRef:
      name: login

```

Confirm that your `AuthServer` serves the specified `Certificate` by visiting its `{.status.issuerURI}`.

For more information about `cert-manager` and its APIs, see [cert-manager documentation](#).

## Configure TLS by using a Secret

If you don't want to use `cert-manager.io`'s APIs or you have a raw TLS certificate in a TLS `Secret`, you can compose it with your `AuthServer` by referencing it. The certificate must be for the issuer URI that will be templated for the `AuthServer`. You can infer it from the AppSSO package's domain template.

The composition of an `AuthServer` and TLS `Secret` looks as follows:

```

apiVersion: v1
kind: Secret
metadata:
  name: my-tls-cert
  namespace: services
type: kubernetes.io/tls
data:
  tls.key: # ...
  tls.crt: # ...

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: login
  namespace: services
  # ...
spec:
  tls:
    secretRef:
      name: my-tls-cert

```

## Deactivate TLS (unsafe)

If you deactivate TLS autoconfiguration, `AuthServer` only works over plain HTTP. You must deactivate TLS with the `sso.apps.tanzu.vmware.com/allow-unsafe-issuer-uri: ""` annotation.

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: login
  namespace: services
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-issuer-uri: ""
  # ...
spec:
  tls:
    deactivated: true
```



### Caution

Deactivating TLS is unsafe and not recommended for production.

## Allow `Workloads` to trust a custom CA `AuthServer`

If your `AuthServer` obtains a certificate from a custom CA, its consumers do not trust it by default. You can enable App Operators' `Workloads` to trust your `AuthServer` by exporting a `ca-certificates` service binding `Secret` to their `Namespace`.

A composition of `SecretTemplate` and `SecretExport` are a way to achieve this. If your custom CA's TLS `Secret` is present in the namespace `my-certs`, then you can provide a `ca-certificates` service binding `Secret` like so:

```
---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretTemplate
metadata:
  name: ca-cert
  namespace: my-certs
spec:
  inputResources:
    - name: my-custom-ca
      ref:
        apiVersion: v1
        kind: Secret
        name: my-custom-ca
  template:
    data:
      ca.crt: $(.my-custom-ca.data.tls\.crt)
    stringData:
      type: ca-certificates

---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretExport
metadata:
  name: ca-cert
  namespace: my-certs
spec:
  toNamespace: ""
```

This templates a `ca-certificates` service binding `Secret` which `Workload` can claim to trust the custom CA. It does not contain the CA's private key and is generally safe to share.

However, be careful, this example exports to all namespace on the cluster. If this does not comply with your policies, then adjust the target namespaces if required.

For more information about `secretgen-controller` and its APIs, see [secretgen-controller documentation](#) in GitHub.

## TLS scenario guides for AppSSO

This topic tells you how to obtain a TLS certificate in different scenarios for Application Single Sign-On (commonly called AppSSO).

### Overview

`AuthServer` is a piece of security infrastructure. It is imperative to configure TLS for it, so that its issuer URI's scheme is `https://`.

`AuthServer.spec.tls` accommodates different scenarios for obtaining a TLS certificate. Select the scenario that matches your case.

The recommended path is to install AppSSO with a `default issuer`. In that case, you can omit `AuthServer.spec.tls` and a TLS certificate is obtained automatically.

### Prerequisites

Each of the scenarios requires that the AppSSO package is installed and configured. In particular, its `domain_name` must match the ingress domain of your cluster. The presented YAML resources assume `my-tap.example.com` as the ingress domain. Therefore, the AppSSO configuration values look as follows:

```
#! AppSSO values
domain_name: "my-tap.example.com"
```

The default `domain_template: "{{.Name}}.{{.Namespace}}.{{.Domain}}"` works for most scenarios. If a scenario requires a bespoke `domain_template`, it contains the relevant instructions.

After applying each scenario, wait for your `AuthServer` to become ready and then test it by running:

```
kubectl wait --namespace login authserver/sso --for condition=Ready=True --timeout 500s
curl --location "${kubectl get --namespace login authserver sso --output=jsonpath='{.status.issuerURI}'}/.well-known/openid-configuration"
```

Alternatively, visit the `AuthServer` with your browser. You can obtain its issuer URI by running:

```
kubectl get --namespace login authserver sso --output=jsonpath='{.status.issuerURI}'
```



#### Caution

Before applying each scenario, you must configure your AppSSO correctly, and make sure that all certificates and DNS names comply with your setup.

## Using a default issuer

VMware recommend using a [default issuer](#), because this approach separates the responsibilities of platform operators and service operators. In this case, the `Authserver.spec.tls` field is not required.

To verify whether `AppSSO` was installed with a default issuer, run:

```
kctrl package installed get --namespace tap-install --package-install tap --values-file-output tap-values.yaml
```

If a `shared.ingress_issuer` appears in your `tap-values.yaml` file, you have a default issuer.



### Important

Ensure `kctrl` is installed.

```
apiVersion: v1
kind: Namespace
metadata:
  name: login
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: sso
  namespace: login
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
    sso.apps.tanzu.vmware.com/allow-client-namespaces: "*"
    sso.apps.tanzu.vmware.com/documentation: Uses the default issuer for TLS
spec:
  identityProviders:
    - name: test-users
      internalUnsafe:
        users:
          - username: user
            password: password
  tokenSignature:
    signAndVerifyKeyRef:
      name: signing-key
---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: signing-key
  namespace: login
spec:
  secretTemplate:
    type: Opaque
    stringData:
      key.pem: $(privateKey)
      pub.pem: $(publicKey)
```

## Using a `ClusterIssuer`

A `ClusterIssuer` is a cluster-scoped API provided by `cert-manager` from which certificates can be obtained programmatically.

This scenario puts all resources into a single YAML file and uses [Let's Encrypt's](#) production API. You might get the `ClusterIssuer` from your platform operators.

For more information, see [cert-manager documentation](#).



### Caution

Let's Encrypt's production API rate limits apply.

```

---
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-production
spec:
  acme:
    privateKeySecretRef:
      name: letsencrypt-production
    server: https://acme-v02.api.letsencrypt.org/directory
    solvers:
      - http01:
          ingress:
            class: contour

---
apiVersion: v1
kind: Namespace
metadata:
  name: login

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: sso
  namespace: login
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
    sso.apps.tanzu.vmware.com/allow-client-namespaces: "*"
spec:
  #! --- TLS ---
  tls:
    issuerRef:
      name: letsencrypt-production
      kind: ClusterIssuer
  #! -----
  identityProviders:
    - name: test-users
      internalUnsafe:
        users:
          - username: user
            password: password
  tokenSignature:
    signAndVerifyKeyRef:
      name: signing-key

---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: signing-key
  namespace: login
spec:
  secretTemplate:
    type: Opaque
    stringData:

```



```
key.pem: $(privateKey)
pub.pem: $(publicKey)
```

## Using an Issuer

This scenario is identical to [Using a ClusterIssuer](#), except that the Issuer is scoped to a namespace and must be located in the same namespace as the [AuthServer](#).

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: login
---
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: letsencrypt-production
  namespace: login
spec:
  acme:
    privateKeySecretRef:
      name: letsencrypt-production
    server: https://acme-v02.api.letsencrypt.org/directory
    solvers:
      - http01:
          ingress:
            class: contour
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: sso
  namespace: login
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
    sso.apps.tanzu.vmware.com/allow-client-namespaces: "*"
spec:
  #! --- TLS ---
  tls:
    issuerRef:
      name: letsencrypt-production
  #! -----
  identityProviders:
    - name: test-users
      internalUnsafe:
        users:
          - username: user
            password: password
  tokenSignature:
    signAndVerifyKeyRef:
      name: signing-key
---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: signing-key
  namespace: login
spec:
  secretTemplate:
    type: Opaque
```

```
stringData:
  key.pem: $(privateKey)
  pub.pem: $(publicKey)
```

## Using an existing Certificate

A [Certificate](#) is an API provided by [cert-manager](#) that is scoped to a namespace and represents a TLS certificate obtained from a [\(Cluster\) Issuer](#). To create a [Certificate](#), you must know the name and kind of your issuer.

These scenarios use [Let's Encrypt's](#) production API and require that a [ClusterIssuer](#) by the name [letsencrypt-production](#) exists. See [Using a ClusterIssuer](#) for how to set up the issuer.

When using [Certificate](#), its `.spec.dnsNames` must contain the FQDN of the templated issuer URI. The `domain_name` and `domain_template` of your AppSSO package installation must comply with your DNS name.

If you have an existing [Certificate](#) in the same [Namespace](#) where the [AuthServer](#) is installed, use the following AppSSO configuration values:

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: login

---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: sso
  namespace: login
spec:
  dnsNames:
    - "sso.login.my-tap.example.com"
  issuerRef:
    name: letsencrypt-production
    kind: ClusterIssuer
  secretName: sso-cert

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: sso
  namespace: login
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
    sso.apps.tanzu.vmware.com/allow-client-namespaces: "*"
spec:
  #! --- TLS ---
  tls:
    certificateRef:
      name: sso
  #! -----
  identityProviders:
    - name: test-users
      internalUnsafe:
        users:
          - username: user
            password: password
  tokenSignature:
    signAndVerifyKeyRef:
      name: signing-key
```

```

---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: signing-key
  namespace: login
spec:
  secretTemplate:
    type: Opaque
    stringData:
      key.pem: $(privateKey)
      pub.pem: $(publicKey)

```

`secretgen-controller` allows you to export and import `Secrets` across namespaces. When your `Certificate` is located in another namespace, for example, it's controlled by another team, you can import its `Secret` to other namespaces. If you have an existing `Certificate` in another `Namespace`, use the following AppSSO configuration values:

```

---
apiVersion: v1
kind: Namespace
metadata:
  name: tls

---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: sso
  namespace: tls
spec:
  dnsNames:
    - "sso.login.my-tap.example.com"
  issuerRef:
    name: letsencrypt-production
    kind: ClusterIssuer
  secretName: sso-cert

---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretExport
metadata:
  name: sso-cert
  namespace: tls
spec:
  toNamespace: login

---
apiVersion: v1
kind: Namespace
metadata:
  name: login

---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretImport
metadata:
  name: sso-cert
  namespace: login
spec:
  fromNamespace: tls

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1

```

```

kind: AuthServer
metadata:
  name: sso
  namespace: login
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
    sso.apps.tanzu.vmware.com/allow-client-namespaces: "*"
spec:
  #! --- TLS ---
  tls:
    secretRef:
      name: sso-cert
  #! -----
  identityProviders:
    - name: test-users
      internalUnsafe:
        users:
          - username: user
            password: password
  tokenSignature:
    signAndVerifyKeyRef:
      name: signing-key

---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: signing-key
  namespace: login
spec:
  secretTemplate:
    type: Opaque
  stringData:
    key.pem: $(privateKey)
    pub.pem: $(publicKey)

```



### Caution

Be cautious when using `SecretExport` and `SecretImport` to facilitate the transfer across namespaces.

## Using an existing TLS certificate

If you have an existing TLS certificate and private key, for example, if your TLS certificate was created outside the cluster, you can apply it directly.

If you don't have a TLS certificate, there are numerous ways to obtain TLS certificates. One of the simplest methods is to use a tool such as `mkcert`, `step` or `openssl` in GitHub.

If you have an existing TLS certificate in the same `Namespace` where the `AuthServer` is installed, use the following AppSSO configuration values:

```

---
apiVersion: v1
kind: Namespace
metadata:
  name: login

---
apiVersion: v1
kind: Secret
type: kubernetes.io/tls

```

```

metadata:
  name: my-cert
  namespace: login
stringData:
  #! --- ReplaceMe - certificate and private key for "sso.login.my-tap.example.com ---
  tls.crt: |
    -----BEGIN CERTIFICATE-----
    # redacted
    -----END CERTIFICATE-----
  tls.key: |
    -----BEGIN PRIVATE KEY-----
    # redacted
    -----END PRIVATE KEY-----
  #! -----
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: sso
  namespace: login
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
    sso.apps.tanzu.vmware.com/allow-client-namespaces: "*"
spec:
  #! --- TLS ---
  tls:
    secretRef:
      name: my-cert
  #! -----
  identityProviders:
    - name: test-users
      internalUnsafe:
        users:
          - username: user
            password: password
  tokenSignature:
    signAndVerifyKeyRef:
      name: signing-key
---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: signing-key
  namespace: login
spec:
  secretTemplate:
    type: Opaque
    stringData:
      key.pem: $(privateKey)
      pub.pem: $(publicKey)

```



### Important

The TLS certificate `tls.crt` and its corresponding private key `tls.key` must be stored in a secret with these keys.

If you have an existing TLS certificate in another `Namespace`, use the following AppSSO configuration values:

```

---
apiVersion: v1
kind: Namespace
metadata:
  name: tls

---
apiVersion: v1
kind: Secret
type: kubernetes.io/tls
metadata:
  name: my-cert
  namespace: tls
stringData:
  #! --- ReplaceMe - certificate and private key for "sso.login.my-tap.example.com ---
  tls.crt: |
    -----BEGIN CERTIFICATE-----
    # redacted
    -----END CERTIFICATE-----
  tls.key: |
    -----BEGIN PRIVATE KEY-----
    # redacted
    -----END PRIVATE KEY-----
  #! -----
---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretExport
metadata:
  name: my-cert
  namespace: tls
spec:
  toNamespace: login

---
apiVersion: v1
kind: Namespace
metadata:
  name: login

---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretImport
metadata:
  name: my-cert
  namespace: login
spec:
  fromNamespace: tls

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: sso
  namespace: login
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
    sso.apps.tanzu.vmware.com/allow-client-namespaces: "*"
spec:
  #! --- TLS ---
  tls:
    secretRef:
      name: my-cert
  #! -----
  identityProviders:
    - name: test-users

```

```

internalUnsafe:
  users:
    - username: user
      password: password
tokenSignature:
  signAndVerifyKeyRef:
    name: signing-key
---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: signing-key
  namespace: login
spec:
  secretTemplate:
    type: Opaque
    stringData:
      key.pem: $(privateKey)
      pub.pem: $(publicKey)

```



### Important

The TLS certificate `tls.crt` and its corresponding private key `tls.key` must be stored in a secret with these keys.

Be cautious when using `SecretExport` and `SecretImport` to facilitate the transfer across namespaces.

## Using an existing wildcard TLS certificate

To use wildcard certificates for DNS names such as `*.my-tap.example.com`, you must edit the AppSSO's `domain_template` so that the templated issuer URIs for `AuthServer` match the wildcard. For example:

- `sso.login.my-tap.example.com` does not match the wildcard.
- `sso-login.my-tap.example.com` matches the wildcard.

The following AppSSO configuration values accommodates a wildcard certificate for `*.my-tap.example.com`:

```

#! AppSSO values
domain_name: "my-tap.example.com"
domain_template: "{{.Name}}-{{.Namespace}}.{{.Domain}}"
#!
                ^ note the dash

```

The following scenarios require TLS Secrets, but the same concept applies to `Certificate`.



### Important

When using a `(Cluster) Issuer` for `Let's Encrypt`, you cannot request wildcard certificates when it uses the `http01 challenge solver`.

If you have an existing wildcard TLS certificate in the same `Namespace` where the `AuthServer` is installed, use the following AppSSO configuration values:

```

---
apiVersion: v1

```

```

kind: Namespace
metadata:
  name: login

---
apiVersion: v1
kind: Secret
type: kubernetes.io/tls
metadata:
  name: my-wildcard-cert
  namespace: login
stringData:
  #! --- ReplaceMe - certificate and private key for "*.my-tap.example.com ---
  tls.crt: |
    -----BEGIN CERTIFICATE-----
    # redacted
    -----END CERTIFICATE-----
  tls.key: |
    -----BEGIN PRIVATE KEY-----
    # redacted
    -----END PRIVATE KEY-----
  #! -----

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: sso
  namespace: login
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
    sso.apps.tanzu.vmware.com/allow-client-namespaces: "*"
spec:
  #! --- TLS ---
  tls:
    secretRef:
      name: my-wildcard-cert
  #! -----
  identityProviders:
    - name: test-users
      internalUnsafe:
        users:
          - username: user
            password: password
  tokenSignature:
    signAndVerifyKeyRef:
      name: signing-key

---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: signing-key
  namespace: login
spec:
  secretTemplate:
    type: Opaque
    stringData:
      key.pem: $(privateKey)
      pub.pem: $(publicKey)

```



## Note



This scenario is similar to using an existing TLS certificate in the same namespace, except that the certificate is a wildcard.

If you have an existing wildcard TLS certificate in another [Namespace](#), use the following AppSSO configuration values:

```

---
apiVersion: v1
kind: Namespace
metadata:
  name: tls

---
apiVersion: v1
kind: Secret
type: kubernetes.io/tls
metadata:
  name: my-wildcard-cert
  namespace: login
stringData:
  #! --- Certificate and private key for "*.my-tap.example.com ---
  tls.crt: |
    -----BEGIN CERTIFICATE-----
    # redacted
    -----END CERTIFICATE-----
  tls.key: |
    -----BEGIN PRIVATE KEY-----
    # redacted
    -----END PRIVATE KEY-----
  #! -----

---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretExport
metadata:
  name: my-cert
  namespace: tls
spec:
  toNamespace: login

---
apiVersion: v1
kind: Namespace
metadata:
  name: login

---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretImport
metadata:
  name: my-cert
  namespace: login
spec:
  fromNamespace: tls

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: sso
  namespace: login
annotations:
  sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
  sso.apps.tanzu.vmware.com/allow-client-namespaces: "*"

```

```

spec:
  #! --- TLS ---
  tls:
    secretRef:
      name: my-wildcard-cert
  #! -----
  identityProviders:
    - name: test-users
      internalUnsafe:
        users:
          - username: user
            password: password
  tokenSignature:
    signAndVerifyKeyRef:
      name: signing-key

---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: signing-key
  namespace: login
spec:
  secretTemplate:
    type: Opaque
  stringData:
    key.pem: $(privateKey)
    pub.pem: $(publicKey)

```



### Note

This scenario is similar to using an existing TLS certificate in another namespace, except that the certificate is a wildcard.

## CA certificates for AppSSO

This topic tells you how to configure CA certificates for `AuthServer` in Application Single Sign-On (commonly called AppSSO).

An `AuthServer` can trust custom CAs. You can establish either for all `AuthServerS` or for a single `AuthServer`. This is useful when either your `identity provider` or `storage` serves certificates from a custom CA.

In most cases, CA certificates are PEM-encoded and located in a `Secret` referred from `.spec.caCerts[].secretRef.name`. The controller considers all `Secret` entries matching `*(crt|ca-bundle)`. That means you can include multiple CA certificates in a single `Secret` or spread them across multiple `Secrets`.

After being created, an `AuthServer` reports the trusted, total custom CA certificates in its `.status.caCerts` together with the location where it sources them from. This includes the CA certificates that are trusted by all `AuthServers`.

For example:

```

---
apiVersion: v1
kind: Secret
metadata:
  name: my-ca
  namespace: services

```

```

stringData:
  my.ca-bundle: |
    This is My Company's custom CA. It's common name is "My CA".
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: login
  namespace: services
  # ...
spec:
  caCerts:
    - secretRef:
        name: my-ca
    # ...
status:
  caCerts:
    - cert:
        subject: CN=My CA,O=My Company,C=Happyland
        source:
          secretEntry: service/my-ca[my.ca-bundle]
    - cert:
        subject: CN=My other CA,O=My Company,C=Happyland
        source:
          secretEntry: appssso/appssso-controller[controller.yaml]
    # ...

```

CA certificates configured for all `AuthServers` by using the package installation's `ca_cert_data` are sourced from `secretEntry: appssso/appssso-controller[controller.yaml]`. This denotes the AppSSO controller's configuration `Secret`.

## Configure workloads to trust a custom CA

This topic tells you how to configure workloads to trust a custom Certificate Authority (commonly called CA) for Application Single Sign-On (commonly called AppSSO).

### Overview

If your `ClientRegistration` selects an `AuthServer` that serves a certificate from a custom CA, your `Workload` does not trust it by default. This is because the certificate is not issued by a trusted certificate authority from the `Workload`'s perspective.

To establish trust between a `Workload` and an `AuthServer`:

Step	Task	Link
1.	Service Operator exports the custom CA certificate <code>Secret</code> resource from the namespace in which it is issued.	<a href="#">Exporting custom CA certificate Secret</a>
2.	Service Operator imports the custom CA certificate <code>Secret</code> to the namespace in which the <code>Workload</code> is created.	<a href="#">Importing custom CA certificate Secret</a>
3.	Append the deployed <code>Workload</code> as a service resource claim, denoting the custom CA certificate <code>Secret</code> in the workload namespace.	<a href="#">Appending custom CA certificate Secret reference to Workload</a>



#### Important

These steps are mandatory if Tanzu Application Platform is installed with the default self-signed `ClusterIssuer` resource, in which the CA is custom.

## Exporting custom CA certificate Secret

A `ca-certificates` service binding `Secret` allows to configure trust for custom CAs.

For more information about exporting CA certificate Secrets, see [Allow Workloads to trust a custom CA AuthServer](#).

**Example:** Create a `ca-certificates`-type ServiceBinding Secret from template and offer Tanzu Application Platform's default self-signed CA certificate Secret to workloads namespace.

```
---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretTemplate
metadata:
  name: tap-ca-cert
  namespace: cert-manager # The namespace in which your custom CA
Secret resides.
spec:
  inputResources:
    - name: tap-ingress-selfsigned-root-ca
      ref:
        apiVersion: v1 # The custom CA certificate Secret.
        kind: Secret # ^^
        name: tap-ingress-selfsigned-root-ca # ^^
  template:
    data:
      ca.crt: $(.tap-ingress-selfsigned-root-ca.data.tls\.crt)
    stringData:
      type: ca-certificates
---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretExport
metadata:
  name: tap-ca-cert # The name of the SecretTemplate that created the "ca-ce
rtificates" Secret.
  namespace: cert-manager # The namespace in which Tanzu Application Platform's se
lf-signed ClusterIssuer stores its CA cert Secret.
spec:
  toNamespace: my-apps # The namespace in which Workloads are deployed.
```

## Importing custom CA certificate Secret

After the custom CA certificate Secret is exported from its original namespace, you can import it into the workloads' namespace.

**Example:** Accept Tanzu Application Platform's default self-signed CA certificate Secret offer.

```
---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretImport
metadata:
  name: tap-ca-cert
  namespace: my-apps # The namespace in which Workloads are deployed.
spec:
  fromNamespace: cert-manager # The namespace in which your custom CA certificate Se
cret resides.
```

## Appending custom CA certificate Secret reference to Workload

With custom CA certificate available in the workloads' namespace, you can append it to the `Workload` as a service resource claim:

**Example:** Appending custom CA certificate Secret as a resource claim.

```
---
apiVersion: carto.run/v1alpha1
kind: Workload
# ...
spec:
  serviceClaims:
    - name: ca-cert
      ref:
        apiVersion: v1 # The custom CA Secret template that is imported into the wo
rkloads' namespace.
        kind: Secret # ^^
        name: tap-ca-cert # ^^
# ...
```

Alternatively, you can provide the workload with a `--service-ref` parameter for the same effect:

```
--service-ref "ca-cert=v1:Secret:tap-ca-cert"
```

For more information about `secretgen-controller` and its APIs, see [secretgen-controller documentation](#) in GitHub.

## Identity providers for AppSSO

This topic tells you how to configure Application Single Sign-On (commonly called AppSSO) to use external identity providers (commonly called IdPs).

Users can log in by using external identity providers. OpenID Connect and LDAP providers are supported. SAML providers have limited experimental support. An `AuthServer` does not manage users internally. Developers can get started quickly without needing to connect to an IdP by using static hard-coded users, which is for development purposes only.

Identity providers are configured under `spec.identityProviders`, learn more from [the API reference](#).



### Caution

Changes to `spec.identityProviders` does not take effect immediately because the operator will roll out a new deployment of the authorization server.

End-users will be able to log in with these providers when they go to `{spec.issuerURI}` in their browser.

Learn how to configure identity providers for an `AuthServer`:

- [OpenID Connect providers](#)
- [LDAP](#)
- [SAML \(experimental\)](#)
- [Internal, static user](#)

- [Restrictions](#)

## OpenID Connect providers

To set up an OpenID Connect provider, provide the following information for your `AuthServer`:

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
# ...
spec:
  identityProviders:
    - name: my-oidc-provider
      openID:
        issuerURI: https://openid.example.com
        clientID: my-client-abcdef
        clientSecretRef:
          name: my-openid-client-secret
        scopes:
          - "openid"
          - "other-scope"
        authorizationUri: https://example.com/oauth2/authorize
        tokenUri: https://example.com/oauth2/token
        jwksUri: https://example.com/oauth2/jwks
        claimMappings:
          roles: my-oidc-provider-groups
# ...
---
apiVersion: v1
kind: Secret
metadata:
  name: my-openid-client-secret
# ...
stringData:
  clientSecret: very-secr3t
```

Where:

- `openID` is the issuer identifier. You can define as many OpenID providers as you like. If the provider supports OpenID Connect Discovery, the value of `openID` is used to auto-configure the provider by using information from <https://openid.example.com/.well-known/openid-configuration>.
- The value of `issuerURI` must not contain `.well-known/openid-configuration` and must match the value of the `issuer` field. See OpenID Connect documentation at <https://openid.example.com/.well-known/openid-configuration> for more information.



### Note

You can retrieve the values of `issuerURI` and `clientID` when registering a client with the provider, which in most cases, is by using a web UI.

- `scopes` is used in the authorization request. Its value must contain "openid". Other common OpenID values include "profile" and "email". You can also run `curl -s "https://openid.example.com/.well-known/openid-configuration" | jq -r ".issuer"` to retrieve the correct `issuerURI` value.
- The value of `clientSecretRef` must be a `Secret` with the entry `clientSecret`.
- `authorizationUri` (optional) is the URI for performing an authorization request and obtaining an `authorization_code`.

- `tokenUri` (optional) is the URI for performing a token request and obtaining a token.
- `jwtUri` (optional) is the JSON Web Key Set (JWKS) endpoint for obtaining the JSON Web Keys to verify token signatures.
- `claimMappings` (optional) selects which claim in the `id_token` contains the `roles` of the user. `roles` is a non-standard OpenID Connect claim. When `ClientRegistrations` has a `roles` scope, it is used to populate the `roles` claim in the `id_token` issued by the `AuthServer`.
- `my-oidc-provider-groups` claim from the ID token issued by `my-oidc-provider` is mapped into the `roles` claim in tokens issued by AppSSO.

Verify the configuration by visiting the `AuthServer`'s issuer URI in your browser and select `my-oidc-provider`.

## Note for registering a client with the identity provider

The `AuthServer` will set up redirect URIs based on the provider name in the configuration. For example, for a provider with `name: my-provider`, the redirect URI will be `{spec.issuerURI}/login/oauth2/code/my-provider`. The externally accessible user URI for the `AuthServer`, including scheme and port is `spec.issuerURI`. If the `AuthServer` is accessible on `https://appsso.company.example.com:1234/`, the redirect URI registered with the identity provider should be `https://appsso.company.example.com:1234/login/oauth2/code/my-provider`.

## Supported token signing algorithms

AppSSO only supports the `RS256` algorithm for token signature. For more information, see [OpenID Connect documentation](#).

You can find out the signing algorithms your OpenID provider supports by referring to the `id_token_signing_alg_values_supported` response parameter in the [OpenID Connect documentation](#) at `.well-known/openid-configuration`.

For example, you can run:

```
curl -s "ISSUER-URI/.well-known/openid-configuration" | jq ".id_token_signing_alg_values_supported"
```

## LDAP



### Important

You can not configure more than one `ldap` identity provider.

For example:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
# ...
spec:
  identityProviders:
  - name: ldap
    ldap:
      url: "ldaps://example.com:636"
      bind:
        dn: uid=binduser,ou=Users,dc=example,dc=com
        passwordRef:
```

```

      name: ldap-password
    user:
      searchBase: ou=Users,dc=example,dc=com
      searchFilter: cn={0}
    group:
      search:
        filter: member={0}
        base: ou=Users,dc=example,dc=com
        searchSubTree: true
        depth: 10
        roleAttribute: description
  # ...
---
apiVersion: v1
kind: Secret
metadata:
  name: ldap-password
  namespace: default
stringData:
  password: confidential-password-value

```

Where:

- `url` is the URL of the LDAP server. It must be `ldaps` and must contain a port.
- `bind.dn` is the DN to perform the bind.
- `bind.passwordRef` must be a secret with the entry `password`. That entry is the password to perform the bind.
- `user.searchBase` is the branch of tree where the users are located at. Search is performed in nested entries.
- `user.seachFilter` is the filter for LDAP search. It must contain the string `{0}`, which is replaced by the `dn` of the user when performing a search. For example, when logging in with the username `marie`, the filter for LDAP search is `cn=marie`.
- `group` (optional) defaults to unset. It configures how LDAP groups are mapped to user roles in the `id_token` claims. If not set, the user has no roles.
  - `group.roleAttriubte` selects which attribute of the group entry are mapped to a user role. If an attribute has multiple values, the first value is selected.
  - `group.search` (optional) toggles “Active Directory” search and uses recursive search to find groups for a given user.

Verify the configuration by visiting the `AuthServer`'s issuer URI in your browser and log in with the username and password from LDAP.

## ActiveDirectory group search

In ActiveDirectory groups, user entries have a multi-value `memberOf` attribute, which contains the DNs pointing to the groups of a particular user. To enable this search mode, make sure `group.roleAttribute` is set and `group.search` is not set.

For example:

```

apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  # ...
spec:
  identityProviders:
    - name: ldap
      ldap:
        # ...

```



```

user:
  searchBase: OU=Cloud,DC=ad,DC=example,DC=com
  searchFilter: cn={0}
group:
  roleAttribute: sAMAccountName

```

The LDIF definition is as follows:

```

dn: CN=appsso-user,OU=Cloud,DC=ad,DC=example,DC=com
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: user
cn: appsso user
sn: user
givenName: appsso
distinguishedName: CN=appsso user,OU=Cloud,DC=ad,DC=example,DC=com
displayName: appsso user
memberOf: CN=ssogroup,OU=Cloud,DC=ad,DC=example,DC=com
memberOf: CN=developers,OU=Cloud,DC=ad,DC=example,DC=com
sAMAccountName: appssouser
userPrincipalName: appssouser@ad.example.com
objectCategory: CN=Person,CN=Schema,CN=Configuration,DC=ad,DC=example,DC=com
# ...

# Groups
dn: CN=ssogroup,OU=Cloud,DC=ad,DC=example,DC=com
objectClass: top
objectClass: group
cn: ssogroup
member: CN=appsso-user,OU=Cloud,DC=ad,DC=example,DC=com
sAMAccountName: SSO Group
# ...

dn: CN=developers,OU=Cloud,DC=ad,DC=example,DC=com
objectClass: top
objectClass: group
cn: developers
sAMAccountName: Developers
# ...

```

The user `appsso-user` has two values for `memberOf`, pointing to two groups. Given the configuration earlier, `sAMAccountName` is used for the role, so the user has `SSO Group` and `Developers` as roles. The group is not required to have `member` attribute point to the user for the role to be mapped.

## “Classic” group search

In non-ActiveDirectory LDAP, users generally do not have a `memberOf` attribute. Group search is performed by looking up groups in a base branch and filtering based on the groups `member` attribute.

An AuthServer can optionally perform:

- group search in sub-branches.
- nested group search, that is, find a hierarchy of groups, in which a group is a member of another group.

The complete configuration is as follows:

```

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:

```

```
# ...
spec:
  identityProviders:
    - name: ldap
      ldap:
        # ...
        group:
          search:
            base: ou=Users,dc=example,dc=com
            filter: member={0}
            depth: 10
            searchSubTree: true
            roleAttribute: description
# ...
```

Where:

- `search.base` is the base for running an LDAP search for groups.
- `search.filter` is the filter for running an LDAP search for groups. It must contain the string `{0}`, which is replaced by the `dn` of the user when performing group search. For example, `member=cn=marie,ou=Users,dc=example,dc=org`.
- `search.depth` (optional) is the depth at which to perform nested group search. It defaults to unset if left empty.
- `search.searchSubTree` (optional) controls whether to look for groups in sub-trees of the `search.base`. It defaults to unset if left empty.

### Direct group search only

Given the following minimal configuration:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
# ...
spec:
  identityProviders:
    - name: ldap
      ldap:
        # ...
        user:
          searchBase: ou=Users,dc=example,dc=com
          searchFilter: uid={0}
        group:
          search:
            base: ou=Users,dc=example,dc=com
            filter: member={0}
            roleAttribute: description
# ...
```

The LDIF definition is as follows:

```
#####
## Users
#####
## User Marie Curie
## Marie Salomea Skłodowska Curie ; https://en.wikipedia.org/wiki/Marie_Curie
dn: cn=marie,ou=Users,dc=example,dc=org
cn: Marie
sn: Skłodowska Curie
objectClass: inetOrgPerson
```

```
objectClass: posixAccount
objectClass: shadowAccount
uid: marie

#####
## Groups
#####
dn: cn=nobels,ou=Users,dc=example,dc=org
objectClass: groupOfNames
description: Nobel Prizes
member: cn=marie,ou=Users,dc=example,dc=org
```

User `marie` has roles `Nobel Prizes`.

## Groups in sub-trees

AppSSO can perform group search in sub-trees of the base for group search. This is enabled when `group.search.searchSubTree` is explicitly set to `true`. For example:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
# ...
spec:
  identityProviders:
  - name: ldap
    ldap:
      # ...
      group:
        search:
          base: ou=Users,dc=example,dc=com
          filter: member={0}
          searchSubTree: true
          roleAttribute: description
# ...
```

The LDIF definition is as follows:

```
#####
## Users
#####
## User Corazon
## Maria Corazon Sumulong Cojuangco Aquino; https://en.wikipedia.org/wiki/Corazon_Aquino
dn: cn=corazon,ou=Users,dc=example,dc=com
cn: Maria Corazon
sn: Sumulong Cojuangco Aquino
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: shadowAccount
uid: corazon

#####
## Groups
#####
dn: cn=presidents,ou=Users,dc=example,dc=com
objectClass: groupOfNames
description: Presidents
member: cn=corazon,ou=Users,dc=example,dc=com

dn: cn=chief-commanders,ou=LegionHonor,ou=Users,dc=example,dc=com
objectClass: groupOfNames
```

```
description: Chief Commanders
member: cn=corazon,ou=Users,dc=example,dc=com
```

User `corazon` has roles `Presidents` and `Chief Commanders`, which are retrieved from `ou=LegionHonor,ou=Users,dc=example,dc=com`, a subtree of the search base.

### Nested group search

AppSSO can perform nested group search by going up a chain where a user is a member of a group, which is itself a member of a group, and so on. This is enabled by setting `group.search.depth` to greater than 1. `group.search.depth` controls the number of “levels” that AppSSO fetches to get the groups of a user. For example:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
# ...
spec:
  identityProviders:
  - name: ldap
    ldap:
      # ...
      user:
        searchBase: ou=Users,dc=example,dc=com
        searchFilter: uid={0}
      group:
        search:
          base: ou=Users,dc=example,dc=com
          filter: member={0}
          depth: 2
        roleAttribute: description
# ...
```

The LDIF definition is as follows:

```
#####
## Users
#####
## User Corazon
## Maria Corazon Sumulong Cojuangco Aquino; https://en.wikipedia.org/wiki/Corazon_Aquino
dn: cn=corazon,ou=Users,dc=example,dc=com
cn: Maria Corazon
sn: Sumulong Cojuangco Aquino
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: shadowAccount
uid: corazon

#####
## Groups
#####
# Citizen > Politicians > Presidents > corazon (depth = 3)
dn: cn=citizens,ou=Users,dc=example,dc=com
objectClass: groupOfNames
description: Citizens
member: cn=politicians,ou=Users,dc=example,dc=com

# Politicians > Presidents > corazon (depth = 2)
dn: cn=politicians,ou=Users,dc=example,dc=com
objectClass: groupOfNames
```

```

description: Politicians
member: cn=presidents,ou=Users,dc=example,dc=com

# Presidents > corazon (depth = 1, direct group)
dn: cn=presidents,ou=Users,dc=example,dc=com
objectClass: groupOfNames
description: Presidents
member: cn=corazon,ou=Users,dc=example,dc=com

```

User `corazon` has roles `Presidents` and `Politicians`. However, the search stops at depth 2, so they do not have the role `Citizens`, which requires a depth greater or equal to 3.

## SAML (experimental)



### Caution

Support for SAML providers is experimental.

For SAML providers only autoconfiguration through `metadataURI` is supported.

```

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
# ...
spec:
- name: my-saml-provider
  saml:
    metadataURI: https://saml.example.com/sso/saml/metadata # required
    claimMappings: # optional
      # Map SAML attributes into claims in id_tokens issued by AppSSO. The key
      # on the left represents the claim, the value on the right the attribute.
      # For example:
      # The "saml-groups" attribute from the assertion issued by "my-saml-provider"
      # will be mapped into the "roles" claim in id_tokens issued by AppSSO
      roles: saml-groups
      givenName: FirstName
      familyName: LastName
      emailAddress: email

```

## Note for registering a client with the identity provider

The `AuthServer` will set up SSO and metadata URLs based on the provider name in the configuration. For example, for a SAML provider with `name: my-provider`, the SSO URL will be `{spec.issuerURI}/login/saml2/sso/my-provider`. The metadata URL will be `{spec.issuerURI}/saml2/service-provider-metadata/my-provider`. `spec.issuerURI` is the externally accessible issuer URI for an `AuthServer`, including scheme and port. If the `AuthServer` is accessible on `https://appsso.company.example.com:1234/`, the SSO URL registered with the identity provider should be `https://appsso.company.example.com:1234/login/saml2/sso/my-provider`.

## Internal users



### Caution

`InternalUnsafe` is unsafe and not recommended for production.

During development, static users can be useful for testing purposes. You can not configure more than one `internalUnsafe` identity provider.

For example:

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
  # ...
spec:
  identityProviders:
    - name: test-users
      internalUnsafe:
        users:
          - username: ernie
            password: "password" # plain text
            roles:
              - "silly"
          - username: bert
            password: "{bcrypt}$2a$10$201z9o/tHlocFsHFTto0plukh03ApBYe4dRiXcqeyRQH6CnntS8jWK" # bcrypt-hashed "password"
            roles:
              - "grumpy"
        # ...
```

`InternalUnsafe` needs to be explicitly allowed by setting the annotation `sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""`.

The passwords can be plain text or bcrypt-hashed. When bcrypt-hashing passwords they have to be prefixed with `{bcrypt}`. Learn how to bcrypt-hash string below.

Verify the configuration by visiting the `AuthServer`'s issuer URI in your browser and logging in as `ernie/password` or `bert/password`.

## Generating a bcrypt hash from a plain-text password

There are multiple options for generating bcrypt hashes:

1. Use an [online bcrypt generator](#)
2. On Unix platforms, use `htpasswd`. Note, you may need to install it, for example on Ubuntu by running `apt install apache2-utils`

```
htpasswd -bnBC 12 "" your-password-here | tr -d ':\n'
```

## Restrictions

Each identity provider has a declared `name`. The following conditions apply:

- the names must be unique
- the names must not be blank
- the names must follow [Kubernetes' DNS Subdomain Names guidelines](#)
  - contain no more than 253 characters
  - contain only lowercase alphanumeric characters, '-' or '.'
  - start with an alphanumeric character
  - end with an alphanumeric character

- the names may not start with `client` or `unknown`

There can be at most one of each `internalUnsafe` and `ldap`.

## Token signatures for AppSSO

This topic tells you how to configure token signature keys for Application Single Sign-On (commonly called AppSSO).

### Overview

An `AuthServer` must have token signature keys configured to be able to mint tokens.

Learn about token signatures and how to manage keys of an `AuthServer`:

- [Token signature 101](#)
- [Token signature in AppSSO](#)
- [Creating keys](#)
- [Rotating keys](#)
- [Revoking keys](#)

“Token signature key” or just “key” is AppSSO’s wording for a public/private key pair that is tasked with signing and verifying JSON Web Tokens (JWTs). For more information, please refer to the following resources:

- [JSON Web Signature \(JWS\) spec](#)
- [JSON Web Algorithms \(JWA\) spec](#)
- [JSON Web Token \(JWT\) spec](#)

### Token signature 101

Token signature keys are used by an `AuthServer` to sign JSON Web Tokens (JWTs), produce a [JWS Signature](#) and attach it to the [JOSE Header](#) of a JWT. The client application can then verify the JWT signature.

A private key signs a JWT. A public key verifies the signature of a signed JWT.

The sign-and-verify mechanism serves multiple security purposes:

- **Authenticity:** signature verification ensures that the issuer of the JWT is from a source that is advertised.
- **Integrity:** signature verification ensures that the JWT has not been altered in transit or during its issued lifetime. [Integrity is a foundational pillar of the CIA triad concept in Information Security.](#)
- **Non-repudiation:** signature verification ensures that the authorization server that signed the JWT cannot deny that they have signed it after its issuance (granted that the signing key that signed the JWT is available).

AppSSO only supports the `RS256` algorithm for signing tokens. For more information, see [JSON Web Algorithms \(JWA\) documentation](#).

### Token signature of an `AuthServer`

You must configure token signatures for `AuthServer`. An `AuthServer` receives its keys under `spec.tokenSignature`. For example:

```
spec:
  tokenSignature:
    signAndVerifyKeyRef:
      name: sample-token-signing-key
    extraVerifyKeyRefs:
      - name: sample-token-verification-key-1
      - name: sample-token-verification-key-2
```

There can only be **one** token signing key `spec.tokenSignature.signAndVerifyKeyRef` at any given time, and arbitrarily many token verification keys `spec.tokenSignature.extraVerifyKeyRefs`. The token signing key is used to sign and verify actively issued JWTs in circulation, whereas token verification keys are used to verify issued JWTs signatures. Token verification keys are thought to be previous token signing keys but have been rotated into verify only mode as a rotation mechanism measure, and can potentially be slated for eviction at a predetermined time.

The `AuthServer` serves its public keys at `{spec.issuerURI}/oauth2/jwks`. For example:

```
> curl -s authserver-sample.default/oauth2/jwks | jq
{
  "keys": [
    {
      "kty": "RSA",
      "e": "AQAB",
      "kid": "sample-token-signing-key",
      "n": "0iCinir7sWKZE_3QXq4eTub_GU-lvdAKFI9dzDlwX7XZwwSERuzzQQ_Fs7i9djMl5bpv2ma_3Z
B-j2W9pR9Zia3nqBI29AHqx2zmVQ8w-GxPDGRmkBdMOWNwyDQGIRlQnJFpXRoSQ5_vim9gYA56WthkDghrupGU
iB_zqGFYlgnz7sd4lC-thgEkDi9vY68DLIFdsXOQIXFqakyEIo43n_0vg6JRGQW1LU_32Ok6OgA3r6bYcE8VQh
JW3sE1qOSFcP0JrPA3YgmTNUdV6GoCLZeMxDdMDKdDcH5UgERLQe1qMMKw1MCEKamOWgo9eBvcFnWNR0I_MJV6
F14U1WbIcQ"
    },
    {
      "kty": "RSA",
      "e": "AQAB",
      "kid": "sample-token-verification-key-1",
      "n": "wc7uOACU62Yu_zkT9YrI4v-_X3L47nbV1cByi4UTVhg8o0010kiYAPAEoDCEHnDg_54gTWxe3h
DRcOJrd72PkTAaxH8aFdikoyakRVG9NvAPbcfzvI8R8plepUbs1U7TPPDEDARm_fZX6QdVyz0CTSafrz-yktTA
DxJhYPgvFleHq7g7RouB1szTWDCM1haoxKa4960_x9meghNn87z0uF3cAd7TM_k3capYnxNOUT5g1vjJ05Vkl4
JU14R294OpMXPCGcFuvu9auXeBqXyKxxTANLkDdNrgtT0FJHwnh4RGnrNqjYZOw1RvGbzWQ7du97aU2-qgbKkJ
rWYZWcw2bQ"
    },
    {
      "kty": "RSA",
      "e": "AQAB",
      "kid": "sample-token-verification-key-2",
      "n": "qELrLiaD-IVp_nthVn2EsLuShtU9ovyVIPkLVf47AqKogPV2frE_6Sv8k7Zim-SgDXfjLEg-UG
lQrb4KfM_WkaK2Uf6PCapiBnMl1Q5P8qCOWC5LT6XyPYlexCQbMrEsyd89oS0sKxgoc3Qv0XV24jGyIWQyJ7I0
Rub_QEldGM_ds1fbI-1Qt_U6Ll220Ec1D6P1A3MdDrqbur6N7ZemxlKI26-OAdlbNi0u-1FNj3Ss-pfTVi_fd2
hAajRRmc4tmHejQjH36M4F1NSW_gTbb6VX5EerVuDwSCCK0EuGvhcb1hg6kYEoO-qws54AQ0PywBXT5qksCMBm
mzjP6qO4Ow"
    }
  ]
}
```

**Caution** Changes to `spec.tokenSignature.signAndVerifyKeyRef` have immediate effects.

As a *service operator*, you have control over which keys are used for certain purposes. Navigate to the next few sections for more information.

## Creating keys

You can deploy an `AuthServer` without `spec.tokenSignature` but it won't be able to mint tokens. Therefore, keys must be configured to make it fully operational. The following describe how to



create and apply a keys for an `AuthServer`.

An RSA key can be created multiple ways. Below are two recommended approaches – choose one.

## Using secretgen-controller



### Important

This section assumes you have Tanzu Application Platform running on your cluster with `secretgen-controller` installed.

An `RSAKey` CR allows for expedited creation of a Secret resource containing PEM-encoded public and private keys required by an `AuthServer`.

1. Create an `AuthServer` with `RSAKeys` as follows:

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: authserver-sample
  namespace: default
spec:
  tokenSignature:
    signAndVerifyKeyRef:
      name: my-token-signing-key
    extraVerifyKeyRefs:
      - name: my-token-verification-key
  # ...
---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: my-token-signing-key
  namespace: default
spec:
  secretTemplate:
    type: Opaque
    stringData:
      key.pem: $(privateKey)
      pub.pem: $(publicKey)
---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: my-token-verification-key
  namespace: default
spec:
  secretTemplate:
    type: Opaque
    stringData:
      key.pem: $(privateKey)
      pub.pem: $(publicKey)
```

2. Observe the creation of an underlying `Secrets`. The name of the each `Secret` is the same as the `RSAKey` names:

```
# Verify Secret exists
kubectl get secret my-token-signing-key
```

```
# View the base64-encoded keys
kubectl get secret my-token-signing-key -o jsonpath='{.data}'
```

You should be able to see two fields within the Secret resource: `key.pem` (private key) and `pub.pem` (public key).

3. Verify that the `AuthServer` serves its keys

```
curl -s authserver-sample.default/oauth2/jwks | jq
```

If you encounter any issues with this approach, see [Carvel Secretgen Controller documentation](#).

## Using OpenSSL

You can generate an RSA key yourself using OpenSSL. Here are the steps:

1. Generate a PEM-encoded RSA key pair

This guide references [the freely published OpenSSL Cookbook](#) and the approaches mentioned therein around generating a public and private key pair.

```
# Generate an 4096-bit RSA key
openssl genpkey -out privatekey.pem -algorithm RSA -pkeyopt rsa_keygen_bits:4096
# -> privatekey.pem
# The resulting private key output is in the PKCS#8 format

# Next, extract the public key
openssl pkey -in privatekey.pem -pubout -out publickey.pem
# -> publickey.pem
# The resulting public key output is in the PKCS#8 format

# To view details of the private key
openssl pkey -in privatekey.pem -text -noout
```

For OpenSSL key generation examples, see the [OpenSSL documentation](#).



### Important

The minimum key size for an `Authserver` is 2048.

2. Create a secret resource by using the key generated earlier in this procedure:

```
kubectl create secret generic my-key \
--from-file=key.pem=privatekey.pem \
--from-file=pub.pem=publickey.pem \
--namespace default
```

3. Apply your `AuthServer`:

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: authserver-sample
  namespace: default
spec:
  tokenSignature:
    signAndVerifyKeyRef:
      name: my-key
# ...
```

#### 4. Verify that the `AuthServer` serves its keys

```
curl -s authserver-sample.default/oauth2/jwks | jq
```

## Rotating keys

This section describes how to “rotate” token signature keys for an `AuthServer`.

The action of “rotating” means moving the active token signing key into the set of token verification keys, generating a new cryptographic key, and assigning it to be the designated token signing key.

Assuming that you have an `AuthServer` with token signature keys configured, rotate keys as follows:

1. Generate a new token signing key first. See [creating keys](#). Verify that the new `Secret` exists before proceeding to the next step.
2. Edit `AuthServer.spec.tokenSignature`, append the existing `spec.tokenSignature.signAndVerifyKeyRef` to `spec.tokenSignature.extraVerifyKeys` and set your new key as `spec.tokenSignature.signAndVerifyKeyRef`.

For example:

```
# Before
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: authserver-sample
  namespace: default
spec:
  tokenSignature:
    signAndVerifyKeyRef:
      name: old-key
    extraVerifyKeys: []
# ...
```

```
# After
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: authserver-sample
  namespace: default
spec:
  tokenSignature:
    signAndVerifyKeyRef:
      name: new-key
    extraVerifyKeys:
      - name: old-key
# ...
```

Once you apply your changes, key rotation is effective immediately.

Moving the active token signing key to be a token verification key is an *optional* step – check out the [Revoking keys](#) section for more.

## Revoking keys

This section describes how to “revoke” token signature keys for an `AuthServer`.

The action of “revoking” a key means to entirely remove the key from circulation by an `AuthServer`, whether it be a token signing key or a token verification key. This action might be needed if your organization requires a complete key refresh where older keys are never retained. Another scenario might be in the case of an emergency in which a key or a session has been compromised and a complete revocation is warranted.

To revoke an existing key or keys, you may remove any references to the keys in the `spec.tokenSignature` resource. By removing the reference to the key, the system shall no longer acknowledge that the key is used for signing or verifying JWTs.

For example, if you have a token signing key and a few verification keys:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: authserver-sample
  namespace: default
spec:
  tokenSignature:
    signAndVerifyKeyRef:
      name: key-3
    extraVerifyKeys:
      - name: key-2
      - name: key-1
# ...
```

To revoke an existing verification key, remove it from the `extraVerifyKeys` list. In the example above, you can remove “key-2” and “key-1” from the list; JWTs signed with those keys will no longer be verifiable.

To revoke an existing token signing key, remove it from `signAndVerifyKeyRef` field. However, if you remove an existing token signing key without a replacement key, the `AuthServer` will not be able to issue access tokens until a valid token signing key is provided. In the example above, “key-3” would be removed; the system will not be able to sign or verify JWTs.

## References and further reading

- [JSON Web Signature \(JWS\) - rfc7515 \(ietf.org\)](https://tools.ietf.org/html/rfc7515)
- [JSON Web Algorithms \(JWA\) spec](https://tools.ietf.org/html/rfc7518)
- [JSON Web Token \(JWT\) - rfc7519 \(ietf.org\)](https://tools.ietf.org/html/rfc7519)

## Token settings for Application Single Sign-On

This topic tells you how to configure token expiry settings for Application Single Sign-On (commonly called AppSSO).

### Token expiry

AppSSO allows you to optionally configure the token expiry settings in your `AuthServer` resource.

The default token expiry settings are as follows:

Token type	Lifetime
Access token	12 hours
Identity token	12 hours

Token type	Lifetime
Refresh token	720 hours or 30 days

VMware recommends setting a shorter lifetime for access tokens, typically measured in hours, and a longer lifetime for refresh tokens, typically measured in days. Refresh tokens acquire new access tokens, so they have a longer lifespan.

To override the token expiry settings, configure the following in your `AuthServer` resource:

```
kind: AuthServer
# ...
spec:
  token:
    accessToken:
      expiry: "12h"
    idToken:
      expiry: "12h"
    refreshToken:
      expiry: "720h"
```

`expiry` field examples:

Type	Example	Definition
Seconds	10s	10 seconds
Minutes	10m	10 minutes
Hours	10h	10 hours



#### Note

The `expiry` field adheres to the duration constraints of the Go standard time library and does not support durations in units beyond hours, such as days or weeks. For more information, see the [Go documentation](#).

## Constraints

The token expiry constraints are as follows:

- The duration of the `expiry` field cannot be negative or zero.
- The refresh token's expiration time cannot be the same as or shorter than that of the access token.

## Verify token settings

After you set up an Application Single Sign-On `AuthServer`, you can verify that the token received by applications looks as expected. For this purpose, you can create a simple application consuming your `AuthServer`. The following YAML file creates such an application. When you access its URL, it enables you to log in by using your `AuthServer` and displays the token it receives.



#### Caution

- The simple application is not intended for production use. It only serves a tool to help you verify your setup.
- The following YAML file pulls an unvetted public image `bitnami/oauth2-proxy:7.3.0`

- This section does not apply to an air-gapped environment.

If you stored the following YAML in a file named `token-viewer.yaml`, you can apply it to your cluster by running the following command:

```
ytt -f token-viewer.yaml --data-value ingress_domain=YOUR-INGRESS-DOMAIN --data-value-namespace=YOUR-NAMESPACE --data-value-yaml 'authserver_selector=YOUR-AUTHSERVER-SELECTOR' | kubectl apply -f-
```

Where `YOUR-AUTHSERVER-SELECTOR` is the label name and its value. For example: `{"name": "ci"}`.

A full example is as follows:

```
#!
#! Token viewer
#!
#! usage:
#!
#! ytt -f token-viewer.yaml --data-value ingress_domain=example.com --data-value-yaml
#! 'authserver_selector={"name": "ci"}'
#!
#! Then navigate to http://token-viewer.<INGRESS_DOMAIN>
#!

#@ load("@ytt:data", "data")
#@ fqdn = "token-viewer." + data.values.ingress_domain
#@ redirect_uri = "http://" + fqdn + "/oauth2/callback"
#@ namespace = data.values.namespace if "namespace" in data.values else "default"

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClientRegistration
metadata:
  name: my-client-registration
  namespace: #@ namespace
spec:
  authServerSelector:
    matchLabels: #@ data.values.authserver_selector
  redirectURIs:
    - #@ redirect_uri
  requireUserConsent: false
  clientAuthenticationMethod: client_secret_basic
  authorizationGrantTypes:
    - "authorization_code"
  scopes:
    - name: "openid"
    - name: "email"
    - name: "profile"
    - name: "roles"

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: token-viewer
  namespace: #@ namespace
spec:
  replicas: 1
  selector:
    matchLabels:
      name: token-viewer
  template:
    metadata:
      labels:
        name: token-viewer
```

```

spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - image: bitnami/oauth2-proxy:7.3.0
    name: proxy
    securityContext:
      runAsNonRoot: true
      seccompProfile:
        type: RuntimeDefault
      allowPrivilegeEscalation: false
      capabilities:
        drop:
          - ALL
    ports:
      - containerPort: 4180
        name: proxy-port
        protocol: TCP
    env:
      - name: ISSUER_URI
        valueFrom:
          secretKeyRef:
            name: my-client-registration
            key: issuer-uri
      - name: CLIENT_ID
        valueFrom:
          secretKeyRef:
            name: my-client-registration
            key: client-id
      - name: CLIENT_SECRET
        valueFrom:
          secretKeyRef:
            name: my-client-registration
            key: client-secret
    command: [ "oauth2-proxy" ]
    args:
      - --oidc-issuer-url=${ISSUER_URI}
      - --client-id=${CLIENT_ID}
      - --insecure-oidc-skip-issuer-verification=true
      - --client-secret=${CLIENT_SECRET}
      - --cookie-secret=0000000000000000
      - --cookie-secure=false
      - --http-address=http://:4180
      - --provider=oidc
      - --scope=openid email profile roles
      - --email-domain=*
      - --insecure-oidc-allow-unverified-email=true
      - --oidc-groups-claim=roles
      - --upstream=http://127.0.0.1:8000
      - #@ "--redirect-url=" + redirect_uri
      - --ssl-upstream-insecure-skip-verify=true
      - --ssl-insecure-skip-verify=true
      - --skip-provider-button=true
      - --pass-authorization-header=true
      - --prefer-email-to-user=true
  - image: python:3.9
    name: application
    securityContext:
      runAsNonRoot: true
      runAsUser: 1001
      seccompProfile:
        type: RuntimeDefault
      allowPrivilegeEscalation: false
      capabilities:

```

```

    drop:
      - ALL
resources:
  limits:
    cpu: 100m
    memory: 100Mi
command: [ "python" ]
args:
  - -c
  - |
    from http.server import HTTPServer, BaseHTTPRequestHandler
    import base64
    import json

    class Handler(BaseHTTPRequestHandler):
        def do_GET(self):
            if self.path == "/token":
                self.token()
            elif self.path == "/jwt":
                self.jwt()
            else:
                self.greet()

        def greet(self):
            username = self.headers.get("x-forwarded-user")
            self.send_response(200)
            self.send_header("Content-type", "text/html")
            self.end_headers()
            page = f"""
            <h1>It Works!</h1>
            <p>You are logged in as <b>{username}</b></p>
            <p><a href="/jwt">Show me my id_token (JWT format)</a></p>
            <p><a href="/token">Show me my id_token (JSON format)</a></p>
            """
            self.wfile.write(page.encode("utf-8"))

        def token(self):
            token = self.headers.get("Authorization").split("Bearer ")[-1]
            payload = token.split(".")[1].replace("-", "+").replace("_", "/")
            decoded = base64.b64decode(bytes(payload, "utf-8") + b'==').deco
de("utf-8")

            self.send_response(200)
            self.send_header("Content-type", "application/json")
            self.end_headers()
            self.wfile.write(decoded.encode("utf-8"))

        def jwt(self):
            token = self.headers.get("Authorization").split("Bearer ")[-1]
            self.send_response(200)
            self.send_header("Content-type", "text/plain")
            self.end_headers()
            self.wfile.write(token.encode("utf-8"))

    server_address = ('', 8000)
    httpd = HTTPServer(server_address, Handler)
    httpd.serve_forever()

---
apiVersion: v1
kind: Service
metadata:
  name: token-viewer
  namespace: #@ namespace
spec:
  ports:
    - port: 80

```



```

    targetPort: proxy-port
    name: proxy-svc-port
  selector:
    name: token-viewer
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: token-viewer
  namespace: #@ namespace
spec:
  rules:
  - host: #@ fqdn
    http:
      paths:
      - path: "/"
        pathType: Prefix
        backend:
          service:
            name: token-viewer
            port:
              name: proxy-svc-port

```

## Storage for AppSSO

This topic tells you how to configure the storage for Application Single Sign-On (commonly called AppSSO).

### Overview

AppSSOs `AuthServer` handles data pertaining to user's session, identity, access tokens and approved or rejected consents. For production environments, it is critical to provide your own storage source to enable enterprise functions such as data backup and recovery, auditing and long-term persistence according to your organization's data and security policies.

AppSSO currently only supports Redis `v6.0` or above as a storage solution. `v6.0` introduced TLS support to ensure encrypted client-server communication - AppSSO enforces TLS by default.

[Storage provided by default](#) refers to an `AuthServer` resource where `.spec.storage` is not set.

Although data in motion is encrypted by using TLS, data at rest is not encrypted by default through `AuthServer`. Each storage provider is responsible for encrypting their own data. See [data types](#) for more information about storage.

### Securing Data at rest

To be compliant with HIPAA, FISMA, PCI and GDPR, you must encrypt data at rest. Securing the underlying infrastructure that Redis uses is crucial to protect against a potential attack. The National Institute for Standards and Technology – Federal Information Processing Standards (NIST-FIPS) sets the standard for best practice when it comes to data security in the US. Symmetric cryptography can be used to protect data at rest. This means that the same key encrypts and decrypts the data, so there is no need for a different private and public key. The [Advanced Encryption Standard \(AES\)](#) encryption algorithm is an industry standard for securing data at rest. For the highest level security, VMware recommends using a 256-bit key.

### Configuring Redis

To configure Redis as authorization server storage, you must have the following information of your Redis server:

- **Server CA certificate** (optional): the Certificate Authority (CA) certificate to verify Redis TLS connections. It is not required if Redis Server certificate is signed by a public CA.
- **host** (required): the domain name, IP address, or host name of your Redis server.
- **port** (optional): the port number of your Redis server. It default to `6379` and must be a string.
- **username** (optional): the username to authenticate against your Redis server.
- **password** (optional): the password to authenticate against your Redis server.

AppSSO takes the secure-by-default approach and does not establish non-encrypted communication channels. The `AuthServer` resource enters an error state if a non-encrypted connection is attempted.

mTLS is not supported, however Vanilla Redis uses mTLS by default. It can be turned off by setting `tls-auth-clients no`. For more information, see [Redis documentation](#).

The following steps introduce the path to configuring Redis with AppSSO:

1. [Configuring Redis Server CA certificate](#)
2. [Configuring a Redis Secret](#)
3. [Attaching storage to an AuthServer](#)

## Configuring Redis Server CA certificate

If your Redis includes a custom or non-public Server CA certificate, you must instruct AppSSO to trust the CA certificate. This is required for the authorization server to communicate with your Redis over TLS. See [CA certificates](#) for more information about configuring a CA certificate with AppSSO.

## Configuring a Redis Secret

To provide coordinates (the location details) of your Redis server, you must create a `Secret` resource that follows well-known Secret entries conventions. For more information, see [Service Bindings 1.0.0 specification](#).

Example of a properly formatted `Secret` resource:



### Important

The Secret must be created in the same namespace as your `AuthServer`.

```
apiVersion: v1
kind: Secret
metadata:
  name: redis-credentials
  namespace: my-authserver
type: servicebinding.io/redis # required
stringData:
  type: "redis" # required, must equal 'redis'
  ssl: "true" # required, must equal 'true'
  host: "redis01.prod.example.com" # required
  port: "6379" # optional, must be a string, defaults to "6379"
if left empty
```

```
password: "!!veryStrongPassword!!" # optional
username: "redis01-user"           # optional
```

## Attaching storage to an AuthServer

After a Redis Secret resource is applied, you can reference the Secret in `.spec.storage`. An example of an AuthServer with a reference to a Redis Secret is as follows:

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: my-authserver-example
  namespace: my-authserver
spec:
  # ...
  storage:
    redis:
      serviceRef:
        apiVersion: "v1"
        kind: "Secret"
        name: redis-credentials
```

After `AuthServer` is applied, ensure its `Status` is `Ready`.

## Inspecting storage of an AuthServer

You can inspect the status of an `AuthServer`'s storage as follows:

```
kubectl get authserver <authserver-name> \
  --namespace <authserver-namespace> \
  --output jsonpath="{.status.storage.redis}" | jq
```

Expect to see the following output with the actual Redis host and port:

```
{
  "redis": {
    "host": "ci-redis.authservers.svc.cluster.local",
    "port": "6379"
  }
}
```

## Storage provided by default

If no storage is defined, an `AuthServer` provides its own short-lived ephemeral storage solution, Redis. The provided Redis is configured to never flush any data to any volume that might be attached to the pods that operate the authorization server.



### Caution

The default storage configuration is designed for prototyping or testing environments and must not be used in production environments.

To view details for Redis of an `AuthServer`:

```
# Get the Redis image
kubectl get authserver <authserver-name> \
  --namespace <authserver-namespace> \
  --output jsonpath="{.status.deployments.redis}" | jq
```

```
# Get the Redis host and port
kubectl get authserver <authserver-name> \
  --namespace <authserver-namespace> \
  --output jsonpath="{.status.storage.redis}" | jq
```

## Data types

The following data is stored in Redis:

- Client information
  - Authorization grant type
  - Client id
- User session
  - Session token
  - Refresh token
- Identity and access tokens



### Note

This is the data that carries the highest level risk.

- Authentication token including the principal
  - Personally identifying information such as email and name
- Approved or rejected consents
  - A client identifier
  - A reference to the user
  - A list of the Authorities that the user has granted to this client

## Known limitations of storage providers

### Redis Cluster

When your storage is provided by Redis Cluster, additional settings might be required.

The nodes and the maximum number of redirects must be set in your Service Bindings' `Secret`. For example, in addition to the entries in [Configuring a Redis Secret](#), you must provide `cluster` settings as follows:

```
apiVersion: v1
kind: Secret
metadata:
  name: redis-cluster-credentials
  namespace: authservers
type: servicebinding.io/redis
stringData:
  #...
  cluster.max-redirects: 5
  cluster.nodes: 100.90.1.10:6379,100.90.1.11:6379,100.90.1.12:6379
```



### Important

`cluster.nodes` must be a comma-separated list of `<ip>:<port>`.

## AuthServer readiness for AppSSO

This topic tells you how to use `AuthServer.status` as a reliable source to verify an `AuthServer`'s readiness for Application Single Sign-On (commonly called AppSSO).

You can verify your `AuthServer` by ensuring:

- there is at least one token signing key configured.

```
curl -X GET {spec.issuerURI}/oauth2/jwks
```

The response body should yield at least one key in the list. If there are no keys, please [apply a token signing key](#)

- OpenID discovery endpoint is available.

```
curl -X GET {spec.issuerURI}/.well-known/openid-configuration
```

The response body should yield a valid JSON body containing information about the `AuthServer`.

## Client registration check

It is helpful to verify an `AuthServer` by running a test run with a test `ClientRegistration`. It ensures that app developers can register clients with the `AuthServer` successfully.

Follow the steps below to ensure that your installation can:

1. Add a test client.
2. Get an access token.
3. Invalidate/remove the test client.

## Prerequisites

Ensure that you have successfully [applied a token signing key](#) to your `AuthServer` before proceeding.

## Define and apply a test client

Apply a `ClientRegistration` to your cluster in a Namespace that the `AuthServer` should allow clients from:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClientRegistration
metadata:
  name: test-client
  namespace: default
spec:
  authServerSelector:
    matchLabels:
      # appropriate labels for your `AuthServer`
  authorizationGrantTypes:
    - client_credentials
  clientAuthenticationMethod: basic
```

See the [ClientRegistration API reference](#) for more field definitions.

This defines a test `ClientRegistration` with the `client_credentials` OAuth grant type.

Apply the `ClientRegistration`:

```
kubectl apply -f appssso-test-client.yaml
```

Once the `ClientRegistration` is applied, inspect its status and verify it's ready.

## Get an access token

You should be able to get a token with the client credentials grant for example:

```
# Get client id (`base64` command has to be available on the command line)
export APPSSO_TEST_CLIENT_ID=$(kubectl get secret test-client -n default -o jsonpath="{.data['client-id']}" | base64 --decode)

# Get client secret (`base64` command has to be available on the command line)
export APPSSO_TEST_CLIENT_SECRET=$(kubectl get secret test-client -n default -o jsonpath="{.data['client-secret']}" | base64 --decode)

# Attempt to fetch access token
curl \
  --request POST \
  --location "{spec.issuerURI}/oauth2/token" \
  --header "Content-Type: application/x-www-form-urlencoded" \
  --header "Accept: application/json" \
  --data "grant_type=client_credentials" \
  --basic \
  --user $APPSSO_TEST_CLIENT_ID:$APPSSO_TEST_CLIENT_SECRET
```

You should see a response JSON containing populated field `access_token`. If so, the system is working as expected, and client registration check is successful.

Make sure to delete the test `ClientRegistration` once you are done.

## Scale AuthServer for AppSSO

This topic tells you how to scale `AuthServer` for Application Single Sign-On (commonly called AppSSO).

The number of authorization server replicas for an `AuthServer` can be specified under `spec.replicas`.

Furthermore, `AuthServer` implements the `scale` subresource. That means you can scale an `AuthServer` with the existing tooling. For example:

```
kubectl scale authserver authserver-sample --replicas=3
```

The resource of the authorization server and Redis `Deployments` can be configured under `spec.resources` and `spec.redisResources` respectively. See the [API reference](#) for details.

## AuthServer audit logs for AppSSO

This topic tells you how to use `AuthServer` audit logs in Application Single Sign-On (commonly called AppSSO).

## Overview

`AuthServer`s perform the following tasks:

- Handle user authentication
- Issue `id_token` and `access_token`

Each audit event contains the following information:

- `ts` - date/time of the event
- `remoteIpAddress` - the IP of the user-authentication or if not attainable, the IP of the last proxy

## Authentication

`AuthServer` produce the following authentication events:

- `AUTHENTICATION_SUCCESS`
  - **Trigger** successful authentication
  - **Data recorded** Username, Provider ID, Provider Type (INTERNAL, OPENID, ...)
- `AUTHENTICATION_LOGOUT`
  - **Trigger** successful logout
  - **Data recorded** Username, Provider ID, Provider Type (INTERNAL, OPENID, ...)
- `AUTHENTICATION_FAILURE`
  - **Trigger** failed authentication using either `internalUnsafe` or `ldap` identity provider
  - **Data recorded** Username, Provider ID, Provider Type (INTERNAL or LDAP)
- `INVALID_IDENTITY_PROVIDER_CONFIGURATION`
  - **Trigger** some cases of failed authentication with an `openId` or `saml` identity provider
  - **Data recorded** Provider ID, Provider Type, error
  - **Note** usually followed by a human-readable help message, with `"logger": "appsso.help"`

## Token flows

`AuthServer` produce the following `authorization_code` and token events:

- `AUTHORIZATION_CODE_ISSUED`
  - **Trigger** `authorization_code` grant type, successful call to `/oauth2/authorize`
  - **Data recorded** Username, Provider ID, Provider Type, Client ID, Scopes requested, Redirect URI
- `AUTHORIZATION_CODE_REQUEST_REJECTED`
  - **Trigger** `authorization_code` grant type, unsuccessful call to `/oauth2/authorize`, for example invalid Client ID, invalid Redirect URI, ...
  - **Data recorded** Error, Error Code (ex: `invalid_scope`), Client ID, Scopes requested Redirect URI, Username (may be `anonymousUser`), Provider ID and Provider Type if available
- `TOKEN_ISSUED`
  - **Trigger** successful call to `/oauth2/token`
  - **Data recorded** Scopes, Client ID, Grant Type (`authorization_code` or `client_credentials`), Username
- `TOKEN_REQUEST_REJECTED`
  - **Trigger** unsuccessful call to `/oauth2/token`, for example invalid Client Secret

- **Data recorded** Client ID, Scopes requested, Error

## Application Single Sign-On for App Operators

This topic tells you how to secure a sample app with Application Single Sign-On (commonly called AppSSO).

To secure a [Workload](#) with AppSSO you need a [ClientRegistration](#) with these ingredients:

- A unique label selector for the [AuthServer](#) you want to register a client for
- Remaining configuration of your OAuth2 client

Talk to your *Service Operator* to learn which [AuthServers](#) they are running and which labels you should use. Once you have those labels, you can create a [ClientRegistration](#) as follows:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClientRegistration
metadata:
  name: my-client
  namespace: my-team
spec:
  authServerSelector:
    matchLabels: # for example
      env: staging
      ldap: True
      team: my-team
```

Continue with learning how to customize your [ClientRegistration](#) by [securing a Workload with SSO](#).

Learn more about [grant types](#) and find [help for common issues](#)

## Application Single Sign-On for App Operators

This topic tells you how to secure a sample app with Application Single Sign-On (commonly called AppSSO).

To secure a [Workload](#) with AppSSO you need a [ClientRegistration](#) with these ingredients:

- A unique label selector for the [AuthServer](#) you want to register a client for
- Remaining configuration of your OAuth2 client

Talk to your *Service Operator* to learn which [AuthServers](#) they are running and which labels you should use. Once you have those labels, you can create a [ClientRegistration](#) as follows:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClientRegistration
metadata:
  name: my-client
  namespace: my-team
spec:
  authServerSelector:
    matchLabels: # for example
      env: staging
      ldap: True
      team: my-team
```

Continue with learning how to customize your [ClientRegistration](#) by [securing a Workload with SSO](#).



Learn more about [grant types](#) and find [help for common issues](#)

## Register a workload

### Topics

- [Client registration](#)
- [Workloads](#)

### Client registration

Applications or Clients must register with AppSSO to allow users to sign in with single sign on within a Kubernetes cluster. This registration results in the creation of a Kubernetes secret.

To do this, apply a [ClientRegistration](#) to the appropriate Kubernetes cluster.

To confirm that the [ClientRegistration](#) was successfully processed, check the status:

```
kubectl describe clientregistrations.sso.apps.tanzu.vmware.com <client-name>
```

It is also possible, but not recommended, to register clients statically while deploying AppSSO.

VMware recommends registering clients dynamically after deploying AppSSO. When registering a client statically, properties cannot be changed without triggering a rollout of AppSSO.

[Grant Types](#)

### Workloads

This guide will walk you through steps necessary to secure your deployed [Workload](#) with AppSSO.

### Prerequisites

Before attempting to integrate your workload with AppSSO, please ensure that the following items are addressed:

- Tanzu Application Platform (TAP) [v1.4.13](#) or above is available on your cluster.
- AppSSO package [v3.0.0](#) or above is available on your cluster.

### Configuring a Workload with AppSSO

AppSSO and your Workload need to establish a bidirectional relationship: AppSSO is aware of your Workload and your Workload is aware of AppSSO. How does that work?

- To make AppSSO aware of your Workload (i.e. that AppSSO should be responsible for authentication and authorization duties), you have to [create and apply a ClientRegistration resource](#) .
- To make your Workload aware of AppSSO (i.e. that your application shall now rely on AppSSO for authentication and authorization requests), you must [specify a service resource claim](#) which produces the necessary credentials for your Workload to consume.
- (Optional) Ensure [Workload](#) trusts [AuthServer](#). For more information, see [Configure Workloads to trust a custom Certificate Authority \(CA\)](#).



#### Important

You must ensure `Workload` trusts `AuthServer` if you use the default self-signed certificate `ClusterIssuer` while installing Tanzu Application Platform.

The following sections elaborate on both of the concepts in detail.

### Create and apply a `ClientRegistration` resource

Define a `ClientRegistration` resource for your `Workload`. Here is an example:

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClientRegistration
metadata:
  name: my-workload-client-registration
  namespace: my-workload-namespace
spec:
  authServerSelector:
    matchLabels:
      # ask your Service Operator for labels to target an `AuthServer`
  authorizationGrantTypes:
    - client_credentials
    - authorization_code
    - refresh_token
  clientAuthenticationMethod: basic
  requireUserConsent: true
  redirectURIs:
    - "<MY_WORKLOAD_HOSTNAME>/redirect-back-uri"
  scopes:
    - name: openid
```

Once applied successfully, this resource will create the appropriate credentials for your `Workload` to consume. More on this in the next section.

Please refer to the [ClientRegistration custom resource documentation page](#) for additional details on schema and specification of the resource.

### Add a service resource claim to your `Workload`

Once a `ClientRegistration` resource has been defined, you can now create a service resource claim by using Tanzu CLI:

```
tanzu service resource-claim create my-client-claim \
  --namespace my-workload-namespace \
  --resource-api-version sso.apps.tanzu.vmware.com/v1alpha1 \
  --resource-kind ClientRegistration \
  --resource-name my-workload-client-registration \
  --resource-namespace my-workload-namespace
```

Alternatively, you may create the claim as `ResourceClaim` custom resource:

```
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ResourceClaim
metadata:
  name: my-client-claim
  namespace: my-workload-namespace
spec:
  ref:
    apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
    kind: ClientRegistration
    name: my-workload-client-registration
    namespace: my-workload-namespace
```

Observe the status of the service resource claim by running `tanzu service resource-claim list -n my-workload-namespace -o wide`:

NAMESPACE	NAME	READY	REASON	CLAIM REF
my-workload-namespace	my-client-claim	True		services.apps.tanzu.vmware.com/v1alpha1:ResourceClaim:my-client-claim

The created service resource claim is now referable within a Workload:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  labels:
    apps.tanzu.vmware.com/workload-type: web
  name: my-workload
  namespace: my-workload-namespace
spec:
  source:
    git:
      ref:
        branch: main
        url: ssh://git@github.com/my-company/my-workload.git
  serviceClaims:
    - name: my-client-claim
      ref:
        apiVersion: services.apps.tanzu.vmware.com/v1alpha1
        kind: ResourceClaim
        name: my-client-claim
```

Alternatively, you can refer to your `ClientRegistration` when deploying your workload with the `tanzu` CLI. Like so

```
tanzu apps workload create my-workload \
  --service-ref "my-client-claim=services.apps.tanzu.vmware.com/v1alpha1:ResourceClaim:my-client-claim" \
  # ...
```

What this service claim reference binding does under the hood is ensures that your Workload's Pod is mounted with a volume containing the necessary credentials required by your application to become aware of AppSSO.

The credentials provided by the service claim are:

- **Client ID** - the identifier of your Workload that AppSSO is registered with. This is a unique identifier.
- **Client Secret** - secret string value used by AppSSO to verify your client during its interactions. Keep this value secret.
- **Issuer URI** - web address of AppSSO, and the primary location that your Workload will go to when interacting with AppSSO.
- **Authorization Grant Types** - list of desired OAuth 2 grant types that your wants to support.
- **Client Authentication Method** - method in which the client application requests an identity or access token
- **Scopes** - list of desired scopes that your application's users will have access to.

The above credentials are mounted onto your Workload's Pod(s) as individual files at the following locations:

```

/bindings
  /<name-of-service-claim>
    /client-id
    /client-secret
    /issuer-uri
    /authorization-grant-types
    /client-authentication-method
    /scope

```

Taking our example from above, the location of credentials can be found at:

```

/bindings/my-client-claim/{client-id,client-secret,issuer-uri,authorization-grant-types,client-authentication-method,scope}

```

Given these auto-generated values, your Workload is now able to load them at runtime and bind to AppSSO at start-up time. Reading the values from the file system is left to the implementor as to the approach taken.

## Configure grant types

This topic tells you how to configure grant types for Application Single Sign-On (commonly called AppSSO).

Apps use grant types or flows to get an access token on behalf of a user. If not included, the default grant type is `['client_credentials']`. You must include these grant types in the `authorizationGrantTypes` property list in the [Client Registration](#).

To register a client/application, apply the `yaml` with your specifications to your cluster `kubectl apply -f <path-to-your-yaml>`.

## Topics

- [Client Credentials Grant](#)
- [Authorization Code Grant](#)

## Client Credentials Grant Type

This grant type allows an application to get an access token for resources about the client itself, rather than a user.

Dynamic Client Registration (via `ClientRegistration` custom resource):

```

apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClientRegistration
metadata:
  name: <your client name>
spec:
  authorizationGrantTypes:
    - client_credentials
  # ...

```



### Note

Ensure that you are able to retrieve a token through your setup

1. Apply your `ClientRegistration`

```
kubectl apply -f <path-to-the-clientregistration-yaml>
```

2. Verify your `ClientRegistration` was created

```
kubectl get clientregistrations
```

-> you should see a `ClientRegistration` with the name you provided

3. Verify your Secret was created

```
kubectl get secrets
```

-> you should see a Secret with that same name you provided for the `ClientRegistration`

4. Get the client secret and decode it

```
kubectl get secret <your-client-registration-name> -o jsonpath="{.data.client-secret}" | base64 -d
```

5. Get the client id (or get it from your configuration)

```
kubectl get secret <your-client-registration-name> -o jsonpath="{.data.client-id}" | base64 -d
```

6. Request token

```
curl -X POST <AUTH-DOMAIN>/oauth2/token?grant_type=client_credentials -v -u "YOUR_CLIENT_ID:DECODED_CLIENT_SECRET"
```

## Authorization Code Grant Type

This grant type allows clients to exchange this code for access tokens.

Dynamic Client Registration (via `ClientRegistration` custom resource):

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClientRegistration
metadata:
  name: <your client name>
spec:
  authorizationGrantTypes:
    - authorization_code
  scopes:
    - openid
  # ...
```



### Note

Ensure that you are able to retrieve a token through your setup

Ensure there is an Identity Provider configured

1. Get your authserver's label name

```
kubectl get authserver sso4k8s -o jsonpath="{.metadata.labels.name}"
```

2. Apply this sample `ClientRegistration` ([read more about ClientRegistrations](#))

The following is an example `ClientRegistration` that will work in this setup. The required scopes are `openid`, `email`, `profile`, `roles`. The redirect URI here has been set to match that of `oauth2-proxy`.

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClientRegistration
metadata:
  name: oauth2-proxy-client
  namespace: <your-namespace>
spec:
  authServerSelector:
  matchLabels:
    name: <your-authserver-label-name>
  authorizationGrantTypes:
    - client_credentials
    - authorization_code
  requireUserConsent: false
  redirectURIs:
    - http://127.0.0.1:4180/oauth2/callback
  scopes:
    - name: openid
    - name: email
    - name: profile
    - name: roles
```

```
kubectl apply -f <path-to-the-clientregistration-yaml>
```

### 3. Verify your `ClientRegistration` was created

```
kubectl get clientregistrations
```

→ you should see a `ClientRegistration` with the name you provided

### 4. Verify your Secret was created

```
kubectl get secrets
```

→ you should see a Secret with that same name you provided for the `ClientRegistration`

### 5. Get the client secret and decode it

```
CLIENT_SECRET=$(kubectl get secret <your-client-registration-name> -o jsonpath
="{.data.client-secret}" | base64 -d)
```

### 6. Get the client id (or get it from your configuration)

```
CLIENT_ID=$(kubectl get secret <your-client-registration-name> -o jsonpath="{.d
ata.client-id}" | base64 -d)
```

### 7. Get the issuer uri

```
ISSUER_URI=$(kubectl get secret <your-client-registration-name> -o jsonpath="{.
data.issuer-uri}" | base64 -d)
```

### 8. Use the `oauth2-proxy` to spin up a quick trial run of the configured Authserver and run it with docker.

```
docker run -p 4180:4180 --name oauth2-proxy bitnami/oauth2-proxy:latest \
--oidc-issuer-url "$ISSUER_URI" \
--client-id "$CLIENT_ID" \
--insecure-oidc-skip-issuer-verification true \
```

```

--client-secret "$CLIENT_SECRET" \
--cookie-secret "0000000000000000" \
--http-address "http://:4180" \
--provider oidc \
--scope "openid email profile roles" \
--email-domain='*' \
--insecure-oidc-allow-unverified-email true \
--upstream "static://202" \
--oidc-groups-claim "roles" \
--oidc-email-claim "sub" \
--redirect-url "http://127.0.0.1:4180/oauth2/callback"

```



#### Note

Ensure that your issuer URL does not resolve to `127.0.0.1`.

9. Check your browser at `127.0.0.1:4180` to see if your configuration allows you to sign in. You should see a message that says “Authenticated”.

## Secure a workload

This tutorial will walk you through the steps to add an authentication mechanism to a sample Spring Boot application using AppSSO service, running on Tanzu Application Platform (TAP).

## Prerequisites

Before starting the tutorial, please ensure that the following items are addressed:

- **RECOMMENDED** Familiarity with [Workloads and AppSSO](#)
- Tanzu Application Platform (TAP) `v1.2.0` or above is available and fully reconciled in your cluster.
  - Please ensure that you are using one of the following TAP Profiles: `run`, `iterate`, or `full`.
- AppSSO package is available and reconciled successfully on your cluster.
- AppSSO has at least one [identity provider configured](#).
- Access to [AppSSO Starter Java](#) accelerator used in this tutorial.

## Getting started



#### Note

See [Deploying the sample application as a Workload](#) for step-by-step instructions if you are already familiar with the accelerator described in this tutorial.

## Understanding the sample application

In this tutorial, you will be working with a sample Servlet-based Spring Boot application that uses [Spring Security OAuth2 Client library](#).

You can find the [source code for the application here](#). To follow along, be sure to Git clone the repository onto your local environment.

The application, once launched, has two pages:

- a **publicly-accessible home page** (`/home`), available to everyone.
- a **user home page** (`/authenticated/home`), for signed-in users only.

The security configuration for the above is located at

`com.vmware.tanzu.apps.sso.sampleworkload.config.WebSecurityConfig`.

For more information about how apps are configured with Spring Security OAuth2 Client library, see [Spring Boot and OAuth2 documentation](#).

By default, there is no application properties file in our sample application and this is by design: even the simplest application can be deployed with AppSSO, you can even go to [start.spring.io](#) and download a Spring Boot app with Spring Security OAuth2 Client library, and you are good to go! There is yet another reason for the absence of any properties files: a demonstration of [Spring Cloud Bindings](#) in action, which removes the need for any OAuth related properties. Spring Cloud Bindings will be introduced later in this tutorial.

### The sample application's `ClientRegistration`

A critical piece of integration with AppSSO is to create a `ClientRegistration` custom resource definition. A `ClientRegistration` is a way for AppSSO to learn about the sample application. In the sample application, you can find the definition file named `client.yaml`, at the root of the source directory.

The `ClientRegistration` resource definition contains a few critical pieces in its specification:

- `authorizationGrantTypes` is set to a list of one: `authorization_code`. Authorization Code grant type is required for [OpenID Connect authentication](#) which we will be using in this tutorial.
- `redirectURIs` is set to a list of two URIs: a remote URI and a local URI (i.e. `127.0.0.1`). The remote URI will be the full URL to which AppSSO will redirect the user back upon successful authentication. The local URI is only meant for debugging purposes and can be ignored unless desired. The suffix of both URIs is important for Spring Security - it adheres to the [default redirect URI template](#).
- `scopes` is set to a list of one scope, the `openid` scope. The `openid` scope is [required by OpenID Connect specification](#) in order to issue identity tokens which designate a user as 'signed in'.

For more information about `ClientRegistration` custom resource, see [ClientRegistration CRD](#).

The `client.yaml` file is using [ytt templating](#) conventions. If you have the [Tanzu Cluster Essentials](#) installed, you should already have `ytt` available on your command line. Later in the tutorial, we will generate a final output `ClientRegistration` declaration that will look similar to the below:

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClientRegistration
metadata:
  name: appssso-starter-java
  namespace: workloads
spec:
  authServerSelector:
    matchLabels:
      # ask your Service Operator for labels to target an `AuthServer`
  clientAuthenticationMethod: basic
  authorizationGrantTypes:
    - authorization_code
  redirectURIs:
    - http://<app-url>/login/oauth2/code/<claim-name>
  scopes:
    - name: openid
```



## Understanding Workloads

To deploy the sample application onto a TAP cluster, we must first craft it as a `Workload` resource (a `Cartographer CRD`). A `Workload` resource can be thought of as a manifest for a process you want to execute on the cluster, and in this context, the type of workload is `web` - a web application. TAP clusters provide the capability to apply `Workload` resources out of the box within the proper profiles, as described in the [prerequisites](#) section.

To deploy a workload, it is best to work in a separate workload-specific namespace. Once created, there are [required TAP configurations that need to be applied](#) before a `Workload` in a specific namespace can be deployed properly.

## Deploying the sample application as a Workload

To tie it all together and deploy the sample application, the following are the steps involved.

### Create workload namespace

Create a workload namespace called `workloads`:

```
kubectl create namespace workloads
```

### Apply required TAP workload configurations

Within the `workloads` namespace, apply [TAP required developer namespaces](#) as described.

### Apply the `ClientRegistration`

Apply the `client.yaml` definition file (described [above](#))



#### Caution

Make sure to set `auth_server_name` field to the value of the label name on the `AuthServer` custom resource. This might differ from the name of the `AuthServer` custom resource.

```
ytt \
  --file client.yaml \
  --data-value namespace=workloads \
  --data-value workload_name=appsso-starter-java \
  --data-value domain=127.0.0.1.nip.io \
  --data-value auth_server_name="" \
  --data-value claim_name=appsso-starter-java | \
  kubectl apply -f-
```

A bit more detail on the above YTT data values:

- **namespace** - the namespace in which the workload will run.
- **workload\_name** - the distinct name of the instance of the accelerator being deployed.
- **domain** - the domain name under which the workload will be deployed. The workload instance will use a subdomain to distinguish itself from other workloads. If working locally, `127.0.0.1.nip.io` is the easiest approach to get a working DNS route on a local cluster.
- **auth\_server\_name** - the value of the label “name” on the `AuthServer` resource that you installed and want to use with your workload. This may differ from the name of the `AuthServer` custom resource.

- **claim\_name** - the service resource claim name being assigned for this workload, this is the binding between the workload and AppSSO. You may choose any reasonably descriptive name for this, it will be used in the next step.

This command has generated a `ClientRegistration` definition and applied it to the cluster. To check the status of the client registration, run:

```
kubectl get clientregistration appssso-starter-java --namespace workloads
```

You should see the `ClientRegistration` entry listed.

## Create a ClientRegistration service resource claim for the workload

Using Tanzu Services plugin CLI, create a service resource claim for the workload:



### Caution

Name of the claim must be the same as the value of `claim_name` from the previous step.

Resource name must be the same name as the workload name.

```
tanzu service claim create appssso-starter-java \
  --namespace workloads \
  --resource-namespace workloads \
  --resource-name appssso-starter-java \
  --resource-kind ClientRegistration \
  --resource-api-version "sso.apps.tanzu.vmware.com/v1alpha1"
```

Once applied, you may check the status of the claim like so:

```
tanzu service claim list --namespace workloads
```

You should see `appssso-starter-java` claim with `Ready` status as `True`.

## (Optional) Ensure Workload trusts AuthServer

You must ensure `Workload` trusts `AuthServer` if you use the default self-signed certificate `ClusterIssuer` while installing Tanzu Application Platform.

For more information, see [Configure Workloads to trust a custom Certificate Authority \(CA\)](#).

## Deploy the workload

The Tanzu CLI command to create a workload for the sample application should look like the following:

```
tanzu apps workload create appssso-starter-java \
  --namespace workloads \
  --type web \
  --label app.kubernetes.io/part-of=appssso-starter-java \
  --service-ref "appssso-starter-java=services.apps.tanzu.vmware.com/v1alpha1:ResourceClaim:appssso-starter-java" \
  --git-repo "https://github.com/vmware-tanzu/application-accelerator-samples" \
  --sub-path "appssso-starter-java" \
  --git-branch main \
  --live-update \
  --yes
```

The above command creates a web `Workload` named ‘appsso-starter-java’ in the `workloads` namespace. The sample applications’ source code repository is defined in the `git-repo` and `git-branch` parameters. The original client yaml definition contains the reference to a service claim which enables the `Workload`’s Pods to have the necessary AppSSO-generated credentials available as a Service Binding. Learn more about how this works [here](#).

It takes some minutes for the workload to become available as a URL.

To query the latest status of the `Workload`, run:

```
tanzu apps workload get appsso-starter-java --namespace workloads
```



### Caution

You may see the status of the workload at first:

**message:** waiting to read value [.status.latestImage] from resource [image.kpack.io/appsso-starter-java] in namespace [workloads]

**reason:** `MissingValueAtPath`

**status:** `Unknown`

This is not an error, this is normal operation of a pending workload. Watch the status for changes.

Follow the `Workload` logs:

```
tanzu apps workload tail appsso-starter-java --namespace workloads
```

After the status of the workload reaches the `Ready` state, you can navigate to the URL provided, which looks similar to:

```
http://appsso-starter-java.workloads.127.0.0.1.nip.io
```

Navigate to the URL in your favorite browser, and observe a large login button tailored for logging with AppSSO.

Once you have explored the accelerator and its operation, head on to the next section for uninstall instructions.

## Cleaning up

You may delete the running accelerator by running the following:

Delete the sample application workload

```
tanzu apps workload delete appsso-starter-java --namespace workloads
```

Delete the service resource claim for the `ClientRegistration`

```
tanzu service claim delete appsso-starter-java --namespace workloads
```

Disconnect the accelerator from AppSSO

```
kubectl delete clientregistration appsso-starter-java --namespace workloads
```

## ClientRegistration API for AppSSO

In Application Single Sign-On (commonly called AppSSO), `ClientRegistration` is the request for client credentials for an `AuthServer`.

It implements the `Service Bindings`' `ProvisionedService`. The credentials are returned as a `Service Bindings Secret`.

A `ClientRegistration` needs to uniquely identify an `AuthServer` via `spec.authServerSelector`. If it matches none, too many or a disallowed `AuthServer` it won't get credentials. The other fields are for the configuration of the client on the `AuthServer`.

## Spec

```

apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClientRegistration
metadata:
  name: ""
  namespace: ""
spec:
  authServerSelector: # required
    matchLabels: { }
  redirectURIs: # required
    - ""
  scopes: # optional
    - name: ""
      description: ""
  authorizationGrantTypes: # optional
    - client_credentials
    - authorization_code
    - refresh_token
  clientAuthenticationMethod: basic # or "post", optional
  requireUserConsent: false # optional
status:
  authServerRef:
    apiVersion: ""
    issuerURI: ""
    kind: ""
    name: ""
    namespace: ""
  binding:
    name: ""
  clientID: ""
  clientSecretHelp: ""
  conditions:
    - lastTransitionTime: ""
      message: ""
      reason: ""
      status: "True" # or "False"
      type: ""
  observedGeneration: 0

```

Alternatively, you can interactively discover the spec with:

```
kubectl explain clientregistrations.sso.apps.tanzu.vmware.com
```

## Status & conditions

The `.status` subresource helps you to learn about your client credentials, the matched `AuthServer` and to troubleshoot issues.

`.status.authServerRef` identifies the successfully matched `AuthServer` and its issuer URI.

`.status.binding.name` is the name of the Service Bindings `Secret` which contains the client credentials.

`.status.conditions` documents each step in the reconciliation:

- `Valid`: Is the spec valid?
- `AuthServerResolved`: Has the targeted `AuthServer` been resolved?
- `ClientSecretResolved`: Has the client secret been resolved?
- `ServiceBindingSecretApplied`: Has the Service Bindings `Secret` with the client credentials been applied?
- `AuthServerConfigured`: Has the resolved `AuthServer` been configured with the client?
- `Ready`: whether all the previous conditions are “True”

The super condition `Ready` denotes a fully successful reconciliation of a given `ClientRegistration`.

If everything goes well you will see something like this:

```
status:
  authServerRef:
    apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
    issuerURI: http://authserver-sample.default
    kind: AuthServer
    name: authserver-sample
    namespace: default
  binding:
    name: clientregistration-sample
    clientID: default_clientregistration-sample
    clientSecretHelp: 'Find your clientSecret: ''kubectl get secret clientregistration-s
sample --namespace default'''
  conditions:
  - lastTransitionTime: "2022-05-13T07:56:41Z"
    message: ""
    reason: Updated
    status: "True"
    type: AuthServerConfigured
  - lastTransitionTime: "2022-05-13T07:56:40Z"
    message: ""
    reason: Resolved
    status: "True"
    type: AuthServerResolved
  - lastTransitionTime: "2022-05-13T07:56:40Z"
    message: ""
    reason: ResolvedFromBindingSecret
    status: "True"
    type: ClientSecretResolved
  - lastTransitionTime: "2022-05-13T07:56:41Z"
    message: ""
    reason: Ready
    status: "True"
    type: Ready
  - lastTransitionTime: "2022-05-13T07:56:40Z"
    message: ""
    reason: Applied
    status: "True"
    type: ServiceBindingSecretApplied
  - lastTransitionTime: "2022-05-13T07:56:40Z"
    message: ""
    reason: Valid
    status: "True"
    type: Valid
  observedGeneration: 1
```

## Example

```

apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClientRegistration
metadata:
  name: my-client-registration
  namespace: app-team
spec:
  authServerSelector:
    matchLabels:
      for: app-team
      ldap: "true"
  redirectURIs:
    - "https://127.0.0.1:8080/authorized"
    - "https://my-application.com/authorized"
  requireUserConsent: false
  clientAuthenticationMethod: basic
  authorizationGrantTypes:
    - "client_credentials"
    - "refresh_token"
  scopes:
    - name: "openid"
      description: "To indicate that the application intends to use OIDC to verify the
user's identity"
    - name: "email"
      description: "The user's email"
    - name: "profile"
      description: "The user's profile information"

```

The client is being registered with the authorization server with the given specs. The resulting client credentials are available in a `Secret` that's owned by the `ClientRegistration`.

```

apiVersion: v1
kind: Secret
type: servicebinding.io/oauth2
metadata:
  name: my-client-registration
  namespace: app-team
data: # fields below are base64-decoded for display purposes only
  type: oauth2
  provider: appsso
  client-id: default_my-client-registration
  client-secret: c2VjcmV0 # auto-generated
  issuer-uri: https://appsso.example.com
  client-authentication-method: basic
  scope: openid,email,profile
  authorization-grant-types: client_credentials,refresh_token

```

## Configuring public clients

A *public client* is a client application that does not require credentials to obtain tokens, such as single-page apps (SPAs). Public clients rely on PKCE (Proof Key for Code Exchange) Authorization Code flow extension.

When configuring a `ClientRegistration` for a public client, ensure that you set your Client Authentication Method to `none` and that your public client supports *Authorization Code with PKCE*. With PKCE, the client does not authenticate (Client Authentication Method is none), but rather presents a code challenge, and subsequent code verifier in order to establish trust with the authorization server.

To set Client Authentication Method to `none`, ensure your `ClientRegistration` resource defines the following:

```
.spec.clientAuthenticationMethod: none
```

Public clients that support Authorization Code with PKCE flow ensure that:

- On every OAuth `authorize` request, parameters `code_challenge` and `code_challenge_method` (default: `S256`) are provided.
- On every OAuth `token` request, parameter `code_verifier` is provided. Public clients do not provide a Client Secret as they are not tailored to retain any secrets in public view.

## ClientRegistration API for AppSSO

In Application Single Sign-On (commonly called AppSSO), `ClientRegistration` is the request for client credentials for an `AuthServer`.

It implements the `Service Bindings`' `ProvisionedService`. The credentials are returned as a `Service Bindings Secret`.

A `ClientRegistration` needs to uniquely identify an `AuthServer` via `spec.authServerSelector`. If it matches none, too many or a disallowed `AuthServer` it won't get credentials. The other fields are for the configuration of the client on the `AuthServer`.

## Spec

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClientRegistration
metadata:
  name: ""
  namespace: ""
spec:
  authServerSelector: # required
    matchLabels: { }
  redirectURIs: # required
  - ""
  scopes: # optional
  - name: ""
    description: ""
  authorizationGrantTypes: # optional
  - client_credentials
  - authorization_code
  - refresh_token
  clientAuthenticationMethod: basic # or "post", optional
  requireUserConsent: false # optional
status:
  authServerRef:
    apiVersion: ""
    issuerURI: ""
    kind: ""
    name: ""
    namespace: ""
  binding:
    name: ""
  clientID: ""
  clientSecretHelp: ""
  conditions:
  - lastTransitionTime: ""
    message: ""
    reason: ""
    status: "True" # or "False"
```

```
type: ""
observedGeneration: 0
```

Alternatively, you can interactively discover the spec with:

```
kubectl explain clientregistrations.sso.apps.tanzu.vmware.com
```

## Status & conditions

The `.status` subresource helps you to learn about your client credentials, the matched `AuthServer` and to troubleshoot issues.

`.status.authServerRef` identifies the successfully matched `AuthServer` and its issuer URI.

`.status.binding.name` is the name of the Service Bindings `Secret` which contains the client credentials.

`.status.conditions` documents each step in the reconciliation:

- `Valid`: Is the spec valid?
- `AuthServerResolved`: Has the targeted `AuthServer` been resolved?
- `ClientSecretResolved`: Has the client secret been resolved?
- `ServiceBindingSecretApplied`: Has the Service Bindings `Secret` with the client credentials been applied?
- `AuthServerConfigured`: Has the resolved `AuthServer` been configured with the client?
- `Ready`: whether all the previous conditions are “True”

The super condition `Ready` denotes a fully successful reconciliation of a given `ClientRegistration`.

If everything goes well you will see something like this:

```
status:
  authServerRef:
    apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
    issuerURI: http://authserver-sample.default
    kind: AuthServer
    name: authserver-sample
    namespace: default
  binding:
    name: clientregistration-sample
    clientID: default_clientregistration-sample
    clientSecretHelp: 'Find your clientSecret: 'kubect1 get secret clientregistration-s
ample --namespace default'''
  conditions:
  - lastTransitionTime: "2022-05-13T07:56:41Z"
    message: ""
    reason: Updated
    status: "True"
    type: AuthServerConfigured
  - lastTransitionTime: "2022-05-13T07:56:40Z"
    message: ""
    reason: Resolved
    status: "True"
    type: AuthServerResolved
  - lastTransitionTime: "2022-05-13T07:56:40Z"
    message: ""
    reason: ResolvedFromBindingSecret
    status: "True"
    type: ClientSecretResolved
  - lastTransitionTime: "2022-05-13T07:56:41Z"
    message: ""
```



```

reason: Ready
status: "True"
type: Ready
- lastTransitionTime: "2022-05-13T07:56:40Z"
  message: ""
  reason: Applied
  status: "True"
  type: ServiceBindingSecretApplied
- lastTransitionTime: "2022-05-13T07:56:40Z"
  message: ""
  reason: Valid
  status: "True"
  type: Valid
observedGeneration: 1

```

## Example

```

apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClientRegistration
metadata:
  name: my-client-registration
  namespace: app-team
spec:
  authServerSelector:
    matchLabels:
      for: app-team
      ldap: "true"
  redirectURIs:
    - "https://127.0.0.1:8080/authorized"
    - "https://my-application.com/authorized"
  requireUserConsent: false
  clientAuthenticationMethod: basic
  authorizationGrantTypes:
    - "client_credentials"
    - "refresh_token"
  scopes:
    - name: "openid"
      description: "To indicate that the application intends to use OIDC to verify the
user's identity"
    - name: "email"
      description: "The user's email"
    - name: "profile"
      description: "The user's profile information"

```

The client is being registered with the authorization server with the given specs. The resulting client credentials are available in a [Secret](#) that's owned by the [ClientRegistration](#).

```

apiVersion: v1
kind: Secret
type: servicebinding.io/oauth2
metadata:
  name: my-client-registration
  namespace: app-team
data: # fields below are base64-decoded for display purposes only
  type: oauth2
  provider: appssso
  client-id: default_my-client-registration
  client-secret: c2VjcmV0 # auto-generated
  issuer-uri: https://appssso.example.com
  client-authentication-method: basic
  scope: openid,email,profile
  authorization-grant-types: client_credentials,refresh_token

```

## Configuring public clients

A *public client* is a client application that does not require credentials to obtain tokens, such as single-page apps (SPAs). Public clients rely on PKCE (Proof Key for Code Exchange) Authorization Code flow extension.

When configuring a `ClientRegistration` for a public client, ensure that you set your Client Authentication Method to `none` and that your public client supports *Authorization Code with PKCE*. With PKCE, the client does not authenticate (Client Authentication Method is none), but rather presents a code challenge, and subsequent code verifier in order to establish trust with the authorization server.

To set Client Authentication Method to `none`, ensure your `ClientRegistration` resource defines the following:

```
.spec.clientAuthenticationMethod: none
```

Public clients that support Authorization Code with PKCE flow ensure that:

- On every OAuth `authorize` request, parameters `code_challenge` and `code_challenge_method` (default: `S256`) are provided.
- On every OAuth `token` request, parameter `code_verifier` is provided. Public clients do not provide a Client Secret as they are not tailored to retain any secrets in public view.

## AuthServer API for AppSSO

In Application Single Sign-On (commonly called AppSSO), `AuthServer` represents the request for an OIDC authorization server. It causes the deployment of an authorization server backed by Redis over mutual TLS if no storage is defined.

An `AuthServer` should have labels which allow to uniquely match it amongst others.

`ClientRegistration` selects an `AuthServer` by label selector and needs a unique match to be successful.

To allow `ClientRegistrations` from all or a restricted set of Namespaces, the annotation `sso.apps.tanzu.vmware.com/allow-client-namespaces` must be set. Its value is a comma-separated list of allowed Namespaces, e.g. `"app-team-red, app-team-green"`, or `"*"` if it should allow clients from all namespaces. If the annotation is missing, no clients are allowed.

The issuer URI, which is the point of entry for clients and end-users, is constructed through the package's `domain_template`. You can view the issuer URI by running `kubectl get authserver -n authservers`.

See [Issuer URI & TLS](#) for more information.

Token signature keys are configured by using `spec.tokenSignature`. This is a required field. See [Token signatures](#) for more context.

Identity providers are configured under `spec.identityProviders`. If there are none, end-users won't be able to log in.

The deployment can be further customized by configuring replicas, resources, http server and logging properties.

An `AuthServer` reconciles into the following resources in its namespace:

```
AuthServer/my-authserver
├─Certificate/my-authserver-redis-client           # if no storage is defined
├─Certificate/my-authserver-redis-server          # if no storage is defined
└─Certificate/my-authserver-root
```

```

|--ConfigMap/my-authserver-ca-cert
|--Deployment/my-authserver-auth-server
|--Deployment/my-authserver-redis # if no storage is defined
|--Issuer/my-authserver-bootstrap
|--Issuer/my-authserver-root
|--Role/my-authserver-auth-server
|--RoleBinding/my-authserver-auth-server
|--Secret/my-authserver-auth-server-clients
|--Secret/my-authserver-auth-server-keys
|--Secret/my-authserver-auth-server-properties
|--Secret/my-authserver-redis-service-binding # if no storage is defined
|--Secret/my-authserver-redis-client-cert-keystore-password # if no storage is defined
|--Secret/my-authserver-registry-credentials
|--Service/my-authserver-redis # if no storage is defined
|--ServiceAccount/my-authserver-auth-server

```

## Spec

```

apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: ""
  namespace: ""
  labels: { } # required, must uniquely identify this AuthServer
  annotations:
    sso.apps.tanzu.vmware.com/allow-client-namespaces: "" # required, must be "*" or a
    comma-separated list of allowed client namespaces
    sso.apps.tanzu.vmware.com/allow-unsafe-issuer-uri: "" # optional
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: "" # optional
spec:
  # .tls is optional if a default issuer is set
  tls:
    # must be one and only one of issuerRef, certificateRef or secretRef, unless deact
    ivated
    issuerRef:
      name: ""
      kind: ""
      group: cert-manager.io
    certificateRef:
      name: ""
    secretRef:
      name: ""
    deactivated: false # If true, requires annotation `sso.apps.tanzu.vmware.com/allow
    -unsafe-issuer-uri: ""`.
    disabled: false # Deprecated, use 'deactivated' instead. If true, requires annotat
    ion `sso.apps.tanzu.vmware.com/allow-unsafe-issuer-uri: ""`.
    tokenSignature: # required
    signAndVerifyKeyRef:
      name: "" # Must be a secret that contains an RSA private key with a minimum leng
      th of 2048 bits.
    extraVerifyKeyRefs:
      - name: "" # Must be a secret that contains an RSA private key with a minimum le
      ngth of 2048 bits.
    storage: # optional
      redis: # required if 'storage' is defined
      serviceRef: # Reference to a provisioned service within the same namespace as th
      is AuthServer. Only supports Secret reference.
        apiVersion: "v1"
        kind: "Secret"
        name: ""
      caCerts: # optional
      - secretRef: # Reference to Secret resource within the same namespace as this Auth
      Server.
        name: ""

```

```

identityProviders: # optional
  # each must be one and only one of internalUnsafe, ldap, openID or saml
  - name: "" # must be unique
    internalUnsafe: # requires annotation `sso.apps.tanzu.vmware.com/allow-unsafe-id
entity-provider: ""`
  users:
    - username: ""
      password: ""
      givenName: ""
      familyName: ""
      email: ""
      emailVerified: false
      roles:
        - ""
  - name: "" # must be unique
    ldap:
      server:
        scheme: ""
        host: ""
        port: 0
        base: ""
      bind:
        dn: ""
        passwordRef:
          name: ldap-password
      user:
        searchFilter: ""
        searchBase: ""
      group:
        searchFilter: ""
        searchBase: ""
        searchSubTree: false
        searchDepth: 0
        roleAttribute: ""
  - name: "" # must be unique
    openID:
      issuerURI: ""
      clientID: ""
      clientSecretRef:
        name: ""
      scopes:
        - ""
  - name: "" # must be unique
    saml:
      metadataURI: ""
      claimMappings: { }
replicas: 1 # optional, default 2
logging: "" # optional, must be valid YAML
server: "" # optional, must be valid YAML
resources: # optional, default {requests: {cpu: "256m", memory: "300Mi"}, limits: {c
pu: "2", memory: "768Mi"}}
  requests:
    cpu: ""
    mem: ""
  limits:
    cpu: ""
    mem: ""
  redisResources: # optional, default {requests: {cpu: "50m", memory: "100Mi"}, limit
s: {cpu: "100m", memory: "256Mi"}}
  requests:
    cpu: ""
    mem: ""
  limits:
    cpu: ""
    mem: ""
status:

```

```

observedGeneration: 0
issuerURI: ""
clientRegistrationCount: 1
tokenSignatureKeyCount: 0
deployments:
  authServer:
    configHash: ""
    image: ""
    replicas: 0
  redis: # leave empty if storage is configured by the service operator
    image: ""
storage:
  redis:
    host: "" # the hostname of the configured Redis
    port: "" # the port of the configured Redis
tls:
  deactivated: false
  issuerRef:
    name: ""
    kind: ""
    group: cert-manager.io
conditions:
- lastTransitionTime:
  message: ""
  reason: ""
  status: "True" # or "False"
  type: ""

```

Alternatively, you can interactively discover the spec with:

```
kubectl explain authservers.sso.apps.tanzu.vmware.com
```

## Status & conditions

The `.status` subresource helps you to learn the `AuthServer`'s readiness, resulting deployments, attached clients and to troubleshoot issues.

`.status.issuerURI` is the templated issuer URI. This is the entry point for any traffic.

`.status.tls` is the actual TLS configuration.

`.status.tokenSignatureKeyCount` is the number of currently configured token signature keys.

`.status.clientRegistrationCount` is the number of currently registered clients.

`.status.deployments.authServer` describes the current authorization server deployment by listing the running image, its replicas, the hash of the current configuration and the generation which has last resulted in a restart.

`.status.deployments.redis` describes the current provided Redis deployment by listing its running image. This field is nil if storage is defined explicitly by using `.spec.storage`.

`.status.storage.redis` describes the configured Redis storage such as host name and port number.

`.status.conditions` documents each step in the reconciliation:

- `Valid`: Is the spec valid?
- `ImagePullSecretApplied`: Has the image pull secret been applied?
- `SignAndVerifyKeyResolved`: Has the single sign-and-verify key been resolved?
- `ExtraVerifyKeysResolved`: Have the single extra verify keys been resolved?
- `IdentityProvidersResolved`: Has all identity provider configuration been resolved?

- **ConfigResolved**: Has the complete configuration for the authorization server been resolved?
- **AuthServerConfigured**: Has the complete configuration for the authorization server been applied?
- **IssuerURIReady**: Is the authorization server yet responding to `{.status.issuerURI}/.well-known/openid-configuration?`
- **Ready**: whether all the previous conditions are "True"

The super condition **Ready** denotes a fully successful reconciliation of a given **ClientRegistration**.

If everything goes well you will see something like this:

```

issuerURI: "https://..."
observedGeneration: 1
tokenSignatureKeyCount: 0
clientRegistrationCount: 0
caCerts:
  - cert:
      subject: ""
      source:
        secretEntry: ""
deployments:
  authServer:
    LastParentGenerationWithRestart: 1
    configHash: "11216479096262796218"
    image: "..."
    replicas: 1
  redis: # leave empty if external storage is defined
    image: "..."
storage:
  redis:
    host: "" # the host name of the configured Redis
    port: "" # the port of the configured Redis
tls:
  deactivated: false
  # One of issuerRef, certificateRef or secretRef is set if TLS is enabled
  issuerRef:
    name: ""
    kind: ""
    group: ""
  certificateRef:
    name: ""
  secretRef:
    name: ""
conditions:
  - lastTransitionTime: "2022-08-24T09:58:10Z"
    message: ""
    reason: KeysConfigSecretUpdated
    status: "True"
    type: AuthServerConfigured
  - lastTransitionTime: "2022-08-24T09:58:10Z"
    message: ""
    reason: Resolved
    status: "True"
    type: ConfigResolved
  - lastTransitionTime: "2022-08-24T09:58:10Z"
    message: ""
    reason: ExtraVerifyKeysResolved
    status: "True"
    type: ExtraVerifyKeysResolved
  - lastTransitionTime: "2022-08-24T09:58:10Z"
    message: ""
    reason: Resolved
    status: "True"

```

```

    type: IdentityProvidersResolved
  - lastTransitionTime: "2022-08-24T09:58:10Z"
    message: ""
    reason: ImagePullSecretApplied
    status: "True"
    type: ImagePullSecretApplied
  - lastTransitionTime: "2022-08-24T09:58:28Z"
    message: ""
    reason: Ready
    status: "True"
    type: IssuerURIReady
  - lastTransitionTime: "2022-08-24T09:58:28Z"
    message: ""
    reason: Ready
    status: "True"
    type: Ready
  - lastTransitionTime: "2022-08-24T09:58:10Z"
    message: ""
    reason: SignAndVerifyKeyResolved
    status: "True"
    type: SignAndVerifyKeyResolved
  - lastTransitionTime: "2022-08-24T09:58:10Z"
    message: ""
    reason: Valid
    status: "True"
    type: Valid

```

## RBAC

The `ServiceAccount` of the authorization server has a `Role` with the following permissions:

```

- apiGroups:
  - ""
  resources:
  - secrets
  verbs:
  - get
  - list
  - watch
  resourceNameNames:
  - { name }-auth-server-keys
  - { name }-auth-server-clients

```

## Example

This example requests an authorization server with two token signature keys and two identity providers.



### Note

The label used for matching to `ClientRegistrations` must be unique across namespaces.

```

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: authserver-sample
  namespace: default
  labels:

```

```

  identifier: authserver-identifier
  sample: "true"
  annotations:
    sso.apps.tanzu.vmware.com/allow-client-namespaces: "*"
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
spec:
  replicas: 1
  tls:
    issuerRef:
      name: my-cluster-issuer
      kind: ClusterIssuer
  tokenSignature:
    signAndVerifyKeyRef:
      name: sample-token-signing-key
    extraVerifyKeyRefs:
      - name: sample-token-verification-key
  identityProviders:
    - name: internal
      internalUnsafe:
        users:
          - username: user
            password: password
            roles:
              - message.write
    - name: okta
      openID:
        issuerURI: https://dev-xxxxxx.okta.com
        clientID: xxxxxxxxxxxxxxxx
        clientSecretRef:
          name: okta-client-secret
        authorizationUri: https://dev-xxxxxx.okta.com/oauth2/v1/authorize
        tokenUri: https://dev-xxxxxx.okta.com/oauth2/v1/token
        jwksUri: https://dev-xxxxxx.okta.com/oauth2/v1/keys
        scopes:
          - openid
        claimMappings:
          roles: my_custom_okta_roles_claim
---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: sample-token-signing-key
  namespace: default
spec:
  secretTemplate:
    type: Opaque
  stringData:
    key.pem: $(privateKey)
    pub.pem: $(publicKey)
---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: sample-token-verification-key
  namespace: default
spec:
  secretTemplate:
    type: Opaque
  stringData:
    key.pem: $(privateKey)
    pub.pem: $(publicKey)
---
apiVersion: v1

```





When attempting to sign in, you see `This commonly happens due to an incorrect [client_secret]`. It might be because the client secret of an identity provider is misconfigured.

## Solution:

Validate the `spec.OpenId.clientSecretRef`.

## Misconfigured redirect URI

### Problem:

You see `Error: [invalid_request] OAuth 2.0 Parameter: redirect_uri` when signing in.

### Solution:

The `redirectUri` of this `ClientRegistration` must refer to the URI of the registered Workload. It does not refer to the URI of the AuthServer.

## Unsupported `id_token_signed_response_alg` with openid identityProviders

### Problem:

When trying to log in with an OpenID Connect `identityProvider`, you are unable to sign in and you see the following error in the logs:

```
[invalid_id_token] An error occurred while attempting to decode the Jwt: Signed JWT rejected: Another algorithm expected, or no matching key(s) found.
```

### Solution:

Verify the `identityProvider`'s discovery endpoint at `ISSUER-URI/.well-known/openid-configuration` where `ISSUER-URI` is the value set at `spec.identityProviders.openid.issuerURI`.

The value of `id_token_signing_alg_values_supported` must include `RS256`. If it is not in the list, your identity configuration might not support AppSSO.

If `RS256` is present, expect to see a `jwks_uri` key in the discovery endpoint. If you visit the URL stored in this key, it must return at least one RSA key. Otherwise, your identity provider might be misconfigured.

Refer to your identity provider's documentation to enable `RS256` token signing.

## Misconfigured identity provider clientSecret

### Problem:

- When attempting to sign in, you see `client.samples.localhost.identity.team redirected you too many times`. It might be because the client secret of an identity provider is misconfigured.
- If you have access to the authserver logs, verify if there is an entry with the text `"error": "[invalid_client] Client authentication failed: client_secret"`.

### Solution:

- Validate the secret referenced by the `clientSecretRef` for this particular identity provider in your `authserver.spec`.

## Missing scopes

### Problem:

When attempting to fetch data after signing in to your application by using AppSSO, you see `[invalid_scope] OAuth 2.0 Parameter: scope`.

### Solution:

Add the required scopes into your `ClientRegistration` yaml under `spec.scopes`.

Changes to the secret do not propagate to the `ClientRegistration`. If you recreated the `Secret` that contains the `clientSecret`, you must re-deploy the `ClientRegistration`.

## Misconfigured `sub` claim

### Problem:

The `sub` claims in `id_tokens` and `access_tokens` follow the `<providerId>_<userId>` pattern. The previous `<providerId>/<userId>` pattern might cause bugs in URLs without proper URL-encoding in client applications.

### Solution:

If your client application stores `sub` claims, you must update the `sub` claims to match the new pattern `<providerId>_<userId>`.

## Known Issues

Application Single Sign-On (commonly called AppSSO) has the following known issues.

### Unregistration by deletion

You can only deregister an existing, ready `ClientRegistration` from its selected `AuthServer` by deleting it. Breaking the match between the two resources by updating either the labels of the `AuthServer` or the label selector on the `ClientRegistration` does not deregister the client from the authorization server.

### Limited number of `ClientRegistrations` per `AuthServer`

The number of `ClientRegistration` for an `AuthServer` is limited at **~2,000**. This is a soft limitation, and if you are attempting to apply more `ClientRegistration` resources than the limit, we cannot guarantee those clients applied past the limit to be in working order. This is subject to change in future product versions.

### LetsEncrypt: domain name for Issuer URI limited to 64 characters maximum

If using LetsEncrypt to issue TLS certificates for an `AuthServer`, the domain name for the Issuer URI (excluding the `http{s}` prefix) cannot exceed 64 characters in length. If exceeded, you may receive

a LetsEncrypt-specific error during Certificate issuance process. This limitation may be observed when your base domain and subdomain joined together exceed the maximum limit.

**Workaround** - if your default Issuer URI is too long, utilize the `domain_template` field in AppSSO values yaml to potentially shorten the domain.

For example, you may forgo the namespace in the Issuer URI like so:

```
domain_template: "{{.Name}}.{{.Domain}}"
```



#### Caution

By leaving out the namespace in your domain template, application routes might conflict if there are multiple `AuthServers` with the same name but in different namespaces.

## Redirect URIs change to http instead of https

### Description

AppSSO makes requests to external identity providers with `http` rather than `https`.

The external identity provider (IDP) informs the user that there is an issue with the `redirect_uri` upon a redirect from the AppSSO auth server to the IDP. The payload of the request to the IDP has a `redirect_uri` of AppSSO Issuer URI that has `http` protocol prefix, while the configuration on the external IDP side has it registered as `https` protocol prefixed.

The underlying issue is that the default Classless Inter-Domain Routing (CIDR) for pod-to-pod traffic is not a default internal network trusted by AppSSO.

### Solution

Add these CIDR ranges to the `AuthServer.spec` (this is a sample range):

```
server: |
  tomcat:
    remoteip:
      internal-proxies: "100\.9[6-9]\.\d{1,3}\.\d{1,3}|\
        100\.1[01]\d\.\d{1,3}\.\d{1,3}|\
        100\.12[0-7]\.\d{1,3}\.\d{1,3}"
```

## AuthServer only supports `response_type=code`

For public clients, the `AuthServer` only supports the Authorization Code Flow: `response_type=code`, because public clients such as single page apps cannot safely store sensitive information such as client secrets.

## Overview of Convention Service for VMware Tanzu

The [Cartographer Conventions](#) component must be installed to add conventions to your pod. The v0.7.x version of the convention controller is a passive system that translates the CRDs to the [new group](#).



#### Caution

This component is deprecated in favor of [Cartographer Conventions](#).

## Sample conventions

There are several out-of-the-box conventions provided with a full profile installation of Tanzu Application Platform or individual component installation of the following packages.

```

> kubectl get pkgi -n tap-install | grep conventions
  appliveview-conventions      conventions.appliveview.tanzu.vmware.com      1.3.0-build.1
  Reconcile succeeded          2m5s
  developer-conventions        developer-conventions.tanzu.vmware.com        0.7.0
  Reconcile succeeded          2m5s
  spring-boot-conventions      spring-boot-conventions.tanzu.vmware.com      0.4.1
  Reconcile succeeded          2m5s

> kubectl get clusterpodconventions
Warning: conventions.apps.tanzu.vmware.com/v1alpha1 ClusterPodConvention is deprecated; use conventions.carto.run/v1alpha1 ClusterPodConvention instead
NAME                                READY   REASON   AGE
appliveview-sample                  True    InSync   4m51s
developer-conventions               True    InSync   4m50s
spring-boot-convention              True    InSync   4m51s

```

The webhook configuration for each convention is as follows:

- Conventions for [AppLiveView](#)

```

...
# webhook configuration
apiVersion: conventions.apps.tanzu.vmware.com/v1alpha1
kind: ClusterPodConvention
...
spec:
  priority: Late
  webhook:
    clientConfig:
      service:
        name: appliveview-webhook
        namespace: app-live-view-conventions

```

```

> kubectl get deployment.apps/appliveview-webhook -n app-live-view-conventions
NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
appliveview-webhook                 1/1     1             1           8m45s

```

- Developer conventions

```

...
# webhook configuration
spec:
  webhook:
    clientConfig:
      service:
        name: webhook
        namespace: developer-conventions

```

```

> kubectl get deployment.apps/webhook -n developer-conventions
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
webhook 1/1     1             1           10m

```

- Spring boot conventions

```

...
# webhook configuration
spec:
  webhook:
    clientConfig:
      service:
        name: spring-boot-webhook
        namespace: spring-boot-convention

```

```

> kubectl get deployment.apps/spring-boot-webhook -n spring-boot-convention
NAME                                READY    UP-TO-DATE    AVAILABLE    AGE
spring-boot-webhook                 1/1      1              1            12m

```

## Overview of Cartographer Conventions

This topic describes an overview of Cartographer Conventions and how you can use it with Tanzu Application Platform.

### Overview



#### Note

This component is replacing the [convention controller](#).

Cartographer Conventions provides a means for operators to express their knowledge about how applications should run on Kubernetes as a convention. Cartographer Conventions applies these opinions to fleets of developer workloads as they are deployed to the platform, saving operator and developer time.

The service is composed of two components:

- **The convention controller:** The convention controller provides the metadata to the convention server and executes the updates to Pod Template Spec as per the convention server's requests.
- **The convention server:** The convention server receives and evaluates metadata associated with a workload and requests updates to the Pod Template Spec associated with that workload. You can have one or more convention servers for a single convention controller instance. Cartographer Conventions supports defining and applying conventions for Pods.

### About applying conventions

The convention server uses criteria defined in the convention to discover whether the configuration of a workload must change. The server receives the OCI metadata from the convention controller. If the metadata meets the criteria defined by the convention server, the conventions are applied. It is possible for a convention to apply to all workloads regardless of metadata.

### Applying conventions by using image metadata

You can define conventions to target workloads by using properties of their OCI metadata.

Conventions can use this information to only apply changes to the configuration of workloads when they match specific criteria. Such as, Spring Boot or .Net apps, or Spring Boot v2.3+. Targeted conventions can ensure uniformity across specific workload types deployed on the cluster.

You can use all the metadata details of an image when evaluating workloads. To see the metadata details, use the Docker CLI command `docker image inspect IMAGE`.



#### Note

Depending on how the image was built, metadata might not be available to reliably identify the image type and match the criteria for a convention server. Images built with Cloud Native Buildpacks reliably include rich descriptive metadata. Images built by some other process can not include the same metadata.

## Applying conventions without using image metadata

Conventions can be defined to apply to workloads without targeting build service metadata. Examples of possible uses of this type of convention include appending a logging or metrics sidecar, adding environment variables, or adding cached volumes. Such conventions are a great way to ensure infrastructure uniformity across workloads deployed on the cluster while reducing developer toil.



#### Important

Adding a sidecar alone does not make the log or metrics collection work. This requires having collector agents deployed and accessible from the Kubernetes cluster, and configuring required access by using role-based access control (RBAC) policy.

## Overview of Cartographer Conventions

This topic describes an overview of Cartographer Conventions and how you can use it with Tanzu Application Platform.

### Overview



#### Note

This component is replacing the [convention controller](#).

Cartographer Conventions provides a means for operators to express their knowledge about how applications should run on Kubernetes as a convention. Cartographer Conventions applies these opinions to fleets of developer workloads as they are deployed to the platform, saving operator and developer time.

The service is composed of two components:

- **The convention controller:** The convention controller provides the metadata to the convention server and executes the updates to Pod Template Spec as per the convention server's requests.
- **The convention server:** The convention server receives and evaluates metadata associated with a workload and requests updates to the Pod Template Spec associated with that workload. You can have one or more convention servers for a single convention controller instance. Cartographer Conventions supports defining and applying conventions for Pods.

## About applying conventions

The convention server uses criteria defined in the convention to discover whether the configuration of a workload must change. The server receives the OCI metadata from the convention controller. If the metadata meets the criteria defined by the convention server, the conventions are applied. It is possible for a convention to apply to all workloads regardless of metadata.

### Applying conventions by using image metadata

You can define conventions to target workloads by using properties of their OCI metadata.

Conventions can use this information to only apply changes to the configuration of workloads when they match specific criteria. Such as, Spring Boot or .Net apps, or Spring Boot v2.3+. Targeted conventions can ensure uniformity across specific workload types deployed on the cluster.

You can use all the metadata details of an image when evaluating workloads. To see the metadata details, use the Docker CLI command `docker image inspect IMAGE`.



#### Note

Depending on how the image was built, metadata might not be available to reliably identify the image type and match the criteria for a convention server. Images built with Cloud Native Buildpacks reliably include rich descriptive metadata. Images built by some other process can not include the same metadata.

### Applying conventions without using image metadata

Conventions can be defined to apply to workloads without targeting build service metadata. Examples of possible uses of this type of convention include appending a logging or metrics sidecar, adding environment variables, or adding cached volumes. Such conventions are a great way to ensure infrastructure uniformity across workloads deployed on the cluster while reducing developer toil.



#### Important

Adding a sidecar alone does not make the log or metrics collection work. This requires having collector agents deployed and accessible from the Kubernetes cluster, and configuring required access by using role-based access control (RBAC) policy.

## Install Cartographer Conventions

Cartographer Conventions is bundled with [Supply Chain Choreographer](#) as of the v0.4.0 release. See [Installing Supply Chain Choreographer](#).

## Create conventions with Cartographer Conventions

This topic describes how you can create and deploy custom conventions to the Tanzu Application Platform by using Cartographer Conventions.

## Introduction



Tanzu Application Platform helps developers transform their code into containerized workloads with a URL. The Supply Chain Choreographer for Tanzu manages this transformation. For more information, see [Supply Chain Choreographer](#).

[Cartographer Conventions](#) is a key component of the supply chain compositions the choreographer calls into action. Cartographer Conventions enables people in operational roles to efficiently apply their expertise. They can specify the runtime best practices, policies, and conventions of their organization to workloads as they are created on the platform. The power of this component becomes evident when the conventions of an organization are applied consistently, at scale, and without hindering the velocity of application developers.

Opinions and policies vary from organization to organization. Cartographer Convention supports the creation of custom conventions to meet the unique operational needs and requirements of an organization.

Before jumping into the details of creating a custom convention, you can view two distinct components of Cartographer Conventions:

- [Convention controller](#)
- [Convention server](#)

## Convention server

The convention server is the component that applies a convention already defined on the server. For a go-lang example of creating a convention server to add springboot conventions, see [spring-convention-server](#) in Github. The resource that facilitates structuring the request body of the request and response from the server is the [PodConventionContext](#).

The [PodConventionContext](#) is a [webhooks.conventions.carto.run/v1alpha1](#) type that defines the structure used to communicate internally by the webhook convention server. It **does not exist** on the Kubernetes API Server.

[PodConventionContext](#) is a wrapper for two types:

- [PodConventionContextSpec](#) which acts as a wrapper for a [PodTemplateSpec](#) and a list of [ImageConfigs](#) provided in the request body of the server.
- [PodConventionContextStatus](#) which is a status type used to represent the current status of the context retrieved by the request.

For information about an example [PodConventionContext](#), see [PodConventionContext](#) in GitHub. For information about a Convention server and the structure of these types, see [OpenAPI Spec](#) in GitHub.

### How the convention server works

Each convention server can host one or more conventions. The application of each convention by a convention server are controlled conditionally. The conditional criteria governing the application of a convention is customizable and are based on the evaluation of a custom Kubernetes resource called [PodIntent](#). [PodIntent](#) is the vehicle by which Cartographer Conventions as a whole delivers its value.

A [PodIntent](#) is created, or updated if already existing, when a workload is run by using a Tanzu Application Platform supply chain. The custom resource includes both the [PodTemplateSpec](#) and the OCI image metadata associated with a workload. See the [Kubernetes documentation](#). The conditional criteria for a convention are based on any property or value found in the [PodTemplateSpec](#) or the Open Containers Initiative (OCI) image metadata available in the [PodIntent](#).

If a convention's criteria are met, the convention server enriches the `PodTemplateSpec` in the `PodIntent`. The convention server also updates the `status` section of the `PodIntent` with the name of the convention that's been applied. So if needed, you can figure out after the fact which conventions were applied to the workload.

To provide flexibility in how conventions are organized, you can deploy multiple convention servers. Each server can contain a convention or set of conventions focused on a specific class of runtime modifications, on a specific language framework, and so on. How the conventions are organized, grouped, and deployed is up to you and the needs of your organization.

Convention servers deployed to the cluster does not take action unless triggered to do so by the second component of Cartographer Conventions, the [Convention controller](#).

## Convention controller

The convention controller is the orchestrator of one or many convention servers deployed to the cluster. There are resources available on the `conventions.carto.run/v1alpha1` API that allow the controller to carry out its functions. These resources include:

- [ClusterPodConvention](#)
  - `ClusterPodConvention` is a resource type that allows the conventions author to register a webhook server with the controller using its `spec.webhook` field.

```
...
spec:
  selectorTarget: PodTemplateSpec # optional field with options, defaults
to PodTemplateSpec
  selectors: # optional, defaults to match all workloads
  - <metav1.LabelSelector>
  webhook:
    certificate:
      name: sample-cert
      namespace: sample-conventions
    clientConfig:
      <admissionregistrationv1.WebhookClientConfig>
```

- [PodIntent](#)
  - The `PodIntent` is a `conventions.carto.run/v1alpha1` resource type that is continuously reconciled and applies decorations to a workload `PodTemplateSpec` exposing the enriched `PodTemplateSpec` on its status. Whenever the status of the `PodIntent` is updated, no side effects are caused on the cluster.

As key types defined on the `conventions.carto.run` API, the `ClusterPodConvention` and `PodIntent` resources are both present on the Kubernetes API Server and are queried using `clusterpodconventions.conventions.carto.run` for the former and `podintents.conventions.carto.run` for the later.

### How the convention controller works

When the Supply Chain Choreographer creates or updates a `PodIntent` for a workload, the convention controller retrieves the OCI image metadata from the repository containing the workload's images and sets it in the `PodIntent`.

The convention controller then uses a webhook architecture to pass the `PodIntent` to each convention server deployed to the cluster. The controller orchestrates the processing of the `PodIntent` by the convention servers sequentially, based on the `priority` value that's set on the convention server. For more information, see [ClusterPodConvention](#).

After all convention servers are finished processing a PodIntent for a workload, the convention controller updates the PodIntent with the latest version of the PodTemplateSpec and sets `PodIntent.status.conditions[].status=True` where `PodIntent.status.conditions[].type=Ready`. This status change signals the Supply Chain Choreographer that Cartographer Conventions is finished with its work. The status change also executes whatever steps are waiting in the supply chain.

## Getting started

With this high-level understanding of Cartographer Conventions components, you can create and deploy a custom convention.



### Note

This topic covers developing conventions using [GOLANG](#), but this is done using other languages by following the specifications.

## Prerequisites

The following prerequisites must be met before a convention is developed and deployed:

- The Kubernetes command line interface tool (kubectl) CLI is installed. For more information, see the [Kubernetes documentation](#).
- Tanzu Application Platform prerequisites are installed. For more information, see [Prerequisites](#)
- Tanzu Application Platform components are installed. For more information, see the [Installing the Tanzu CLI](#).
- The default supply chain is installed. Download Supply Chain Security Tools for VMware Tanzu from [Tanzu Network](#).
- Your kubeconfig context is set to the Tanzu Application Platform-enabled cluster:

```
kubectl config use-context CONTEXT_NAME
```

- You use Github to install the ko CLI. See the [google/ko](#) GitHub repository. These instructions use `ko` to build an image. If there is an existing image or build process, `ko` is optional.)

## Define convention criteria

The `server.go` file contains the configuration for the server and the logic the server applies when a workload matches the defined criteria. For example, adding a Prometheus sidecar to web applications, or adding a `workload-type=spring-boot` label to any workload that has metadata, indicating it is a Spring Boot app.



### Important

For this example, the package `model` defines [resource types](#).

1. The example `server.go` configures the `ConventionHandler` to ingest the webhook requests from the convention controller. See [PodConventionContext](#). Here the handler must only deal with the existing `PodTemplateSpec` and `ImageConfig`.

```

...
import (
    corev1 "k8s.io/api/core/v1"
)
...
func ConventionHandler(template *corev1.PodTemplateSpec, images []model.ImageCo
nfig) ([]string, error) {
    // Create custom conventions
}
...

```

Where:

- `template` is the predefined `PodTemplateSpec` that the convention edits. For more information about `PodTemplateSpec`, see the [Kubernetes documentation](#).
- `images` are the `ImageConfig` used as reference to make decisions in the conventions. In this example, the type was created within the `model` package.

2. The example `server.go` also configures the convention server to listen for requests:

```

...
import (
    "context"
    "fmt"
    "log"
    "net/http"
    "os"
    ...
)
...
func main() {
    ctx := context.Background()
    port := os.Getenv("PORT")
    if port == "" {
        port = "9000"
    }
    http.HandleFunc("/", webhook.ServerHandler(convention.ConventionHandler))
    log.Fatal(webhook.NewConventionServer(ctx, fmt.Sprintf(":%s", port)))
}
...

```

Where:

- `PORT` is a possible environment variable, for this example, defined in the [Deployment](#).
- `ServerHandler` is the *handler* function called when any request comes to the server.
- `NewConventionServer` is the function in charge of configuring and creating the *http webhook* server.
- `port` is the calculated port of the server to listen for requests. It must match the [Deployment](#) if the `PORT` variable is not defined in it.
- The `path` or pattern (default to `/`) is the convention server's default path. If it is changed, it must be changed in the [ClusterPodConvention](#).



#### Note

The *Server Handler*, `func ConventionHandler(...)`, and the configure or start web server, `func NewConventionServer(...)`, is defined in the convention controller in the `webhook` package, but you can use a custom one.

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
        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 TLS 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 defines the criteria for applying conventions. See [PodTemplateSpec](#) in the Kubernetes documentation. The following are a few examples.

### Matching criteria by labels or annotations

The `conventions.carto.run/v1alpha1` API allows convention authors to use the `selectorTarget` field which complements the `ClusterPodConvention` matchers to specify whether to consider labels on either one of the following available options:

- PodTemplateSpec

```

...
template:
  metadata:
    labels:
      awesome-label: awesome-value
    annotations:
      awesome-annotation: awesome-value
...

```

- PodIntent

```

...
kind: PodIntent
metadata:

```

```

name: test-pod
labels:
  environment: production
  ...

```

The `selectorTarget` field is configured on the `ClusterPodConvention` as follows:

```

...
spec:
  selectorTarget: PodIntent # optional, defaults to PodTemplateSpec
  selectors: # optional, defaults to match all workloads
  - <metav1.LabelSelector>
  webhook:
    certificate:
      name: sample-cert
      namespace: sample-conventions
    clientConfig:
      <admissionregistrationv1.WebhookClientConfig>

```

If you do not provide a value for this optional field while using the `conventions.carto.run/v1alpha1` API, the default value is set to `PodTemplateSpec` without the conventions author explicitly doing so. The `selectorTarget` field is not available in the `conventions.apps.tanzu.vmware.com/v1alpha1` API and labels specified in the `PodTemplateSpec` are considered if a matcher is defined in a `ClusterPodConvention` while referencing this deprecated API.

## Matching criteria by environment variables

When using environment variables to define whether the convention is applicable, it must be present in the `PodTemplateSpec`, `spec`, `containers`, and `env` to validate the value.

- `PodTemplateSpec`

```

...
template:
  spec:
    containers:
      - name: awesome-container
        env:
...

```

- `Handler`

```

package convention
...
func conventionHandler(template *corev1.PodTemplateSpec, images []model.ImageCo
nfig) ([]string, error) {
    if len(template.Spec.Containers[0].Env) == 0 {
        template.Spec.Containers[0].Env = append(template.Spec.Containers[0].En
v, corev1.EnvVar{
            Name: "MY_AWESOME_VAR",
            Value: "MY_AWESOME_VALUE",
        })
        return []string{"awesome-envs-convention"}, nil
    }
    return []string{}, nil
...
}

```

## Matching criteria by image metadata

For each image contained within the `PodTemplateSpec`, the convention controller fetches the OCI image metadata and known [bill of materials \(BOMs\)](#), providing it to the convention server as

**ImageConfig**. This metadata is introspected to make decisions about how to configure the PodTemplateSpec.

## Configure and install the convention server

The `server.yaml` defines the Kubernetes components that enable the convention server in the cluster. The next definitions are within the file.

1. A `namespace` is created for the convention server components and has the required objects to run the server. It's used in the `ClusterPodConvention` section to indicate to the controller where the server is.

```
...
---
apiVersion: v1
kind: Namespace
metadata:
  name: awesome-convention
---
...
```

2. (Optional) A certificate manager `Issuer` is created to issue the certificate needed for TLS communication.

```
...
---
# The following manifests contain a self-signed issuer CR and a certificate CR.
# More document can be found at https://docs.cert-manager.io
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: awesome-selfsigned-issuer
  namespace: awesome-convention
spec:
  selfSigned: {}
---
...
```

3. (Optional) A self-signed `Certificate` is created.

```
...
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: awesome-webhook-cert
  namespace: awesome-convention
spec:
  subject:
    organizations:
      - vmware
    organizationalUnits:
      - tanzu
  commonName: awesome-webhook.awesome-convention.svc
  dnsNames:
    - awesome-webhook.awesome-convention.svc
    - awesome-webhook.awesome-convention.svc.cluster.local
  issuerRef:
    kind: Issuer
    name: awesome-selfsigned-issuer
  secretName: awesome-webhook-cert
  revisionHistoryLimit: 10
```



```

---
...

```

4. A Kubernetes `Deployment` is created to run the webhook from. The `Service` uses the container port defined by the `Deployment` to expose the server.

```

...
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: awesome-webhook
  namespace: awesome-convention
spec:
  replicas: 1
  selector:
    matchLabels:
      app: awesome-webhook
  template:
    metadata:
      labels:
        app: awesome-webhook
    spec:
      containers:
        - name: webhook
          # Set the prebuilt image of the convention or use ko to build an image
          # from code.
          # see https://github.com/google/ko
          image: ko://awesome-repo/awesome-user/awesome-convention
          env:
            - name: PORT
              value: "8443"
          ports:
            - containerPort: 8443
              name: webhook
          livenessProbe:
            httpGet:
              scheme: HTTPS
              port: webhook
              path: /healthz
          readinessProbe:
            httpGet:
              scheme: HTTPS
              port: webhook
              path: /healthz
          volumeMounts:
            - name: certs
              mountPath: /config/certs
              readOnly: true
          volumes:
            - name: certs
              secret:
                defaultMode: 420
                secretName: awesome-webhook-cert
---
...

```

5. A Kubernetes `Service` to expose the convention deployment is created. For this example, the exposed port is the default `443`. If you change the port, the `ClusterPodConvention` must be updated.

```

...
---
apiVersion: v1

```

```

kind: Service
metadata:
  name: awesome-webhook
  namespace: awesome-convention
  labels:
    app: awesome-webhook
spec:
  selector:
    app: awesome-webhook
  ports:
    - protocol: TCP
      port: 443
      targetPort: webhook
---
...

```

- The [ClusterPodConvention](#) adds the convention to the cluster to make it available for the convention controller:



### Important

The `annotations` block is only needed if you use a self-signed certificate. See the [cert-manager documentation](#).

```

...
---
apiVersion: conventions.carto.run/v1alpha1
kind: ClusterPodConvention
metadata:
  name: awesome-convention
  annotations:
    conventions.carto.run/inject-ca-from: "awesome-convention/awesome-webhook-c
ert"
spec:
  webhook:
    clientConfig:
      service:
        name: awesome-webhook
        namespace: awesome-convention
        # path: "/" # default
        # port: 443 # default

```

## Deploy a convention server

To deploy a convention server:

- Build and install the convention.
  - To build and deploy the convention, use the [ko tool](#) on GitHub. It compiles your Go code into a Docker image and pushes it to the registry `KO_DOCKER_REGISTRY`.

```
ko apply -f dist/server.yaml
```

- If a different tool builds the image, the configuration is also applied by using either `kubectl` or `kapp`, setting the correct image in the [Deployment](#) descriptor.

`kubectl`

```
kubectl apply -f server.yaml
```

kapp

```
kapp deploy -y -a awesome-convention -f server.yaml
```

2. Verify the convention server. To verify the status of the convention server, confirm the running convention pods:

- o If the server is running, `kubectl get all -n awesome-convention` returns output such as:

```
NAME                                     READY   STATUS    RESTARTS   A
GE
pod/awesome-webhook-1234567890-12345    1/1     Running   0           8
h

NAME                                     TYPE          CLUSTER-IP   EXTERNAL-IP   POR
T(S)   AGE
service/awesome-webhook                 ClusterIP     10.56.12.49  <none>        44
3/TCP   28h

NAME                                     READY   UP-TO-DATE   AVAILABLE   AG
E
deployment.apps/awesome-webhook         1/1     1             1           28
h

NAME                                     DESIRED   CURRENT   READ
Y   AGE
replicaset.apps/awesome-webhook-1234563213  0         0         0
23h
replicaset.apps/awesome-webhook-5b79d5cb59  0         0         0
28h
replicaset.apps/awesome-webhook-5bf557c9f8  1         1         1
20h
replicaset.apps/awesome-webhook-77c647c987  0         0         0
23h
replicaset.apps/awesome-webhook-79d9c6f74c  0         0         0
23h
replicaset.apps/awesome-webhook-7d9d667b8d  0         0         0
9h
replicaset.apps/awesome-webhook-8668664d75  0         0         0
23h
replicaset.apps/awesome-webhook-9b6957476   0         0         0
24h
```

- o To verify that the conventions are applied, ensure that the `PodIntent` of a workload that matches the convention criteria:

```
kubectl -o yaml get podintents.conventions.apps.tanzu.vmware.co awesome-a
pp
```

```
apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
  creationTimestamp: "2021-10-07T13:30:00Z"
  generation: 1
  labels:
    app.kubernetes.io/component: intent
    carto.run/cluster-supply-chain-name: awesome-supply-chain
    carto.run/cluster-template-name: convention-template
    carto.run/component-name: config-provider
    carto.run/template-kind: ClusterConfigTemplate
    carto.run/workload-name: awesome-app
    carto.run/workload-namespace: default
```

```

name: awesome-app
namespace: default
ownerReferences:
- apiVersion: carto.run/v1alpha1
  blockOwnerDeletion: true
  controller: true
  kind: Workload
  name: awesome-app
  uid: "*****"
resourceVersion: "*****"
uid: "*****"
spec:
imagePullSecrets:
- name: registry-credentials
  serviceAccountName: default
template:
  metadata:
    annotations:
      developer.conventions/target-containers: workload
    labels:
      app.kubernetes.io/component: run
      app.kubernetes.io/part-of: awesome-app
      carto.run/workload-name: awesome-app
  spec:
    containers:
      - image: awesome-repo.com/awesome-project/awesome-app@sha256:****
        name: workload
        resources: {}
        securityContext:
          runAsUser: 1000
status:
  conditions:
  - lastTransitionTime: "2021-10-07T13:30:00Z"
    status: "True"
    type: ConventionsApplied
  - lastTransitionTime: "2021-10-07T13:30:00Z"
    status: "True"
    type: Ready
observedGeneration: 1
template:
  metadata:
    annotations:
      awesome-annotation: awesome-value
      conventions.carto.run/applied-conventions: |-
        awesome-label-convention
        awesome-annotation-convention
        awesome-envs-convention
        awesome-image-convention
        developer.conventions/target-containers: workload
    labels:
      awesome-label: awesome-value
      app.kubernetes.io/component: run
      app.kubernetes.io/part-of: awesome-app
      carto.run/workload-name: awesome-app
      conventions.carto.run/framework: go
  spec:
    containers:
      - env:
        - name: MY_AWESOME_VAR
          value: "MY_AWESOME_VALUE"
        image: awesome-repo.com/awesome-project/awesome-app@sha256:*****
        name: workload
        ports:
          - containerPort: 8080
            protocol: TCP

```

```
resources: {}
securityContext:
  runAsUser: 1000
```

## Next Steps

Keep Exploring:

- Try to use different matching criteria for the conventions or enhance the supply chain with multiple conventions.

## Troubleshoot Cartographer Conventions

This topic describes how you can troubleshoot Cartographer Conventions.

### No server in the cluster

#### Symptoms

- When a `PodIntent` is submitted, no `convention` is applied.

#### Cause

When there are no `convention servers` (`ClusterPodConvention`) deployed in the cluster or none of the existing convention servers applied any conventions, the `PodIntent` is not being mutated.

#### Solution

Deploy a `convention server` (`ClusterPodConvention`) in the cluster.

### Server with wrong certificates configured

#### Symptoms

- When a `PodIntent` is submitted, the `conventions` are not applied.
- The `convention-controller logs` reports an error `failed to get CABundle` as follows:

```
{
  "level": "error",
  "ts": 1638222343.6839523,
  "logger": "controllers.PodIntent.PodIntent.ResolveConventions",
  "msg": "failed to get CABundle",
  "ClusterPodConvention": "base-convention",
  "error": "unable to find valid certificaterequests for certificate \"convention-template/webhook-certificate\"",
  "stacktrace": "reflect.Value.Call\n\treflect/value.go:339\ngithub.com/vmware-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\nsigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).Reconcile\n\tgithub.com/vmware-labs/reconciler-runtime@v0.10.3/pkg/internal/controller/controller.go:114\nsigs.k8s.io/controller-runtime/pkg/internal/controller.(*Con
```

```
troller).reconcileHandler\n\tsigs.k8s.io/controller-runtime@v0.10.3/pkg/interna
l/controller/controller.go:311\nsigs.k8s.io/controller-runtime/pkg/internal/con
troller.(*Controller).processNextWorkItem\n\tsigs.k8s.io/controller-runtime@v0.
10.3/pkg/internal/controller/controller.go:266\nsigs.k8s.io/controller-runtime/
pkg/internal/controller.(*Controller).Start.func2.2\n\tsigs.k8s.io/controller-r
untime@v0.10.3/pkg/internal/controller/controller.go:227"
```

## Cause

`convention server` (`ClusterPodConvention`) is configured with wrong certificates. The `convention-controller` cannot figure out the CA Bundle to perform the request to the server.

## Solution

Ensure that the `convention server` (`ClusterPodConvention`) is configured with the correct certificates. To do so, verify the value of annotation `conventions.carto.run/inject-ca-from` which must be set to the used Certificate.



### Important

Do not set annotation `conventions.carto.run/inject-ca-from` if no certificate is used.

## Server fails when processing a request

### Symptoms

- When a `PodIntent` is submitted, the `convention` is not applied.
- The `convention-controller` logs reports `failed to apply convention` error like this.

```
{ "level": "error", "ts": 1638205387.8813763, "logger": "controllers.PodIntent.PodInt
ent.ApplyConventions", "msg": "failed to apply convention", "Convention": { "Nam
e": "base-convention", "Selectors": null, "Priority": "Normal", "ClientConfig": { "serv
ice": { "namespace": "convention-template", "name": "webhook", "port": 443 }, "caBundl
e": "..."} }, "error": "Post \"https://webhook.convention-template.svc:443/?timeout
=30s\": EOF", "stacktrace": "reflect.Value.call\n\treflect/value.go:543\nreflect.
Value.Call\n\treflect/value.go:339\ngithub.com/vmware-labs/reconciler-runtime/r
econcilers.(*SyncReconciler).sync\n\tgithub.com/vmware-labs/reconciler-runtime@
v0.3.0/reconcilers/reconcilers.go:287\ngithub.com/vmware-labs/reconciler-runtim
e/reconcilers.(*SyncReconciler).Reconcile\n\tgithub.com/vmware-labs/reconciler-
runtime@v0.3.0/reconcilers/reconcilers.go:276\ngithub.com/vmware-labs/reconcil
e-r-runtime/reconcilers.Sequence.Reconcile\n\tgithub.com/vmware-labs/reconciler-r
untime@v0.3.0/reconcilers/reconcilers.go:815\ngithub.com/vmware-labs/reconciler
-runtime/reconcilers.(*ParentReconciler).reconcile\n\tgithub.com/vmware-labs/re
conciler-runtime@v0.3.0/reconcilers/reconcilers.go:146\ngithub.com/vmware-labs/
reconciler-runtime/reconcilers.(*ParentReconciler).Reconcile\n\tgithub.com/vmwa
re-labs/reconciler-runtime@v0.3.0/reconcilers/reconcilers.go:120\nsigs.k8s.io/c
ontroller-runtime/pkg/internal/controller.(*Controller).Reconcile\n\tsigs.k8s.i
o/controller-runtime@v0.10.0/pkg/internal/controller/controller.go:114\nsigs.k8
s.io/controller-runtime/pkg/internal/controller.(*Controller).reconcileHandler
\n\tsigs.k8s.io/controller-runtime@v0.10.0/pkg/internal/controller/controller.g
o:311\nsigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).pro
cessNextWorkItem\n\tsigs.k8s.io/controller-runtime@v0.10.0/pkg/internal/control
ler/controller.go:266\nsigs.k8s.io/controller-runtime/pkg/internal/controller.
(*Controller).Start.func2.2\n\tsigs.k8s.io/controller-runtime@v0.10.0/pkg/inter
nal/controller/controller.go:227" }
```

- When a `PodIntent` status message is updated with `failed to apply convention from source base-convention: Post "https://webhook.convention-template.svc:443/?`

```
timeout=30s": EOF.
```

## Cause

An unmanaged error occurs in the `convention server` when processing a request.

## Solution

1. Check the `convention server` logs to identify the cause of the error:
  1. Use the following command to retrieve the `convention server` logs:

```
kubectl -n convention-template logs deployment/webhook
```

Where:

- The `convention server` was deployed as a `Deployment`
  - `webhook` is the name of the `convention server Deployment`.
  - `convention-template` is the namespace where the `convention server` is deployed.
2. Identify the error and deploy a fixed version of `convention server`.
    - Be aware that the new deployment is not applied to the existing `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": { "Nam
e": "base-convention", "Selectors": null, "Priority": "Normal", "ClientConfig": { "serv
ice": { "namespace": "convention-template", "name": "webhook", "port": 443 }, "caBundl
e": "..."}, "error": "Post \"https://webhook.convention-template.svc:443/?timeout
=30s\": dial tcp 10.56.13.206:443: connect: connection refused", "stacktrace": "r
eflect.Value.call\n\treflect/value.go:543\nreflect.Value.Call\n\treflect/value.
go:339\ngithub.com/vmware-labs/reconciler-runtime/reconcilers.(*SyncReconcile
r).sync\n\tgithub.com/vmware-labs/reconciler-runtime@v0.3.0/reconcilers/reconci
lers.go:287\ngithub.com/vmware-labs/reconciler-runtime/reconcilers.(*SyncReconc
iler).Reconcile\n\tgithub.com/vmware-labs/reconciler-runtime@v0.3.0/reconciler
s/reconcilers.go:276\ngithub.com/vmware-labs/reconciler-runtime/reconcilers.Seq
uence.Reconcile\n\tgithub.com/vmware-labs/reconciler-runtime@v0.3.0/reconciler
s/reconcilers.go:815\ngithub.com/vmware-labs/reconciler-runtime/reconcilers.(*P
arentReconciler).reconcile\n\tgithub.com/vmware-labs/reconciler-runtime@v0.3.0/
reconcilers/reconcilers.go:146\ngithub.com/vmware-labs/reconciler-runtime/recon
cilers.(*ParentReconciler).Reconcile\n\tgithub.com/vmware-labs/reconciler-runti
me@v0.3.0/reconcilers/reconcilers.go:120\nsigs.k8s.io/controller-runtime/pkg/in
ternal/controller.(*Controller).Reconcile\n\tgithub.com/vmware-labs/reconciler-runti
me@v0.10.0/pkg/internal/controller/controller.go:114\nsigs.k8s.io/controller-runtime/
pkg/internal/controller.(*Controller).reconcileHandler\n\tgithub.com/vmware-labs/reconciler-runti
me@v0.10.0/pkg/internal/controller/controller.go:311\nsigs.k8s.io/controlle
r-runtime/pkg/internal/controller.(*Controller).processNextWorkItem\n\tgithub.
com/vmware-labs/reconciler-runtime@v0.10.0/pkg/internal/controller/controller.go:266\nsi
gs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).Start.func2.
```

```
2\n\tSIGS.k8s.io/controller-runtime@v0.10.0/pkg/internal/controller/controller.go:227"}

```

- The `convention server` fails to start due to `server gave HTTP response to HTTPS client`:
  - When checking the `convention server` events by running the following command:

```
kubectl -n convention-template describe pod webhook-594d75d69b-4w4s8
```

Where:

- The `convention server` was deployed as a `Deployment`
- `webhook-594d75d69b-4w4s8` is the name of the `convention server` Pod.
- `convention-template` is the namespace where the `convention server` is deployed.

For example:

```
Name:          webhook-594d75d69b-4w4s8
Namespace:     convention-template
...
Containers:
  webhook:
    ...
Events:
Type     Reason          Age          From          Message
----     -
Normal   Scheduled       14m         default-scheduler   Successfully 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   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.carto.run/inject-ca-from` as a pointer to the deployed `Certificate` resource.



## Self-signed certificate authority (CA) not propagated to the Convention Service

### Symptoms

The self-signed certificate authority (CA) for a registry is not propagated to the Convention Service.

### Cause

When you provide the self-signed certificate authority (CA) for a registry through `convention-controller.ca_cert_data`, it cannot be propagated to the Convention Service.

### Solution

Define the CA by using the available `.shared.ca_cert_data` top-level key to supply the CA to the Convention Service.

## No imagePullSecrets configured

### Symptoms

When a PodIntent is submitted:

- No convention is applied.
- You see an `unauthorized to access repository OR fetching metadata for Images failed` error when you inspect the workload.

### Cause

The errors are seen when a `workload` is created in a developer namespace where `imagePullSecrets` are not defined on the `default` serviceAccount or on the preferred serviceAccount.

### Solution

Add the `imagePullSecrets` name to the default serviceAccount or the preferred serviceAccount.

For example:

```
kind: ServiceAccount
metadata:
  name: default
  namespace: my-workload-namespace
imagePullSecrets:
  - name: registry-credentials # ensure this secret is defined
secrets:
  - name: registry-credentials
```

## Convention Service Resources for Cartographer Conventions

This reference topic describes the convention service resources you can use with Cartographer Conventions.

### Overview

There are several [resources](#) involved in the application of conventions to workloads.

## API Structure

The `PodConventionContext` API object in the `webhooks.conventions.carto.run` API group is the structure used for both request and response from the convention server.

## Template Status

The enriched `PodTemplateSpec` is reflected at `.status.template`. For more information about `PodTemplateSpec`, see the [Kubernetes documentation](#).

## Chaining Multiple Conventions

You can define multiple `ClusterPodConventions` and apply them to different types of workloads. You can also apply multiple conventions to a single workload.

The `PodIntent` reconciler lists all `ClusterPodConvention` resources and applies them serially. To ensure the consistency of enriched `PodTemplateSpec`, the list of `ClusterPodConventions` is sorted alphabetically by name before applying conventions. You can use strategic naming to control the order in which the conventions are applied.

After the conventions are applied, the `Ready` status condition on the `PodIntent` resource is used to indicate whether it is applied successfully. A list of all applied conventions is stored under the annotation `conventions.carto.run/applied-conventions`.

## Collecting Logs from the Controller

The convention controller is a Kubernetes operator and can be deployed in a cluster with other components. If you have trouble, you can retrieve and examine the logs from the controller to help identify issues.

To retrieve Pod logs from the `conventions-controller-manager` running in the `conventions-system` namespace:

```
kubectl -n conventions-system logs -l control-plane=controller-manager
```

For example:

```
...
{"level":"info","ts":1637073467.3334172,"logger":"controllers.PodIntent.PodIntent.ApplyConventions","msg":"applied convention","diff":"  interface{}(\n- \tts\"&PodTemplateSpec{ObjectMeta:{          0 0001-01-01 00:00:00 +0000 UTC <nil> <nil> map[app.kubernetes.io/component:run app.kubernetes.io/part-of:spring-petclinic-app-db carto.run/workload-name:spring-petclinic-app-db] map[developer.conventions/target-container\"...,\n+ \tvl.PodTemplateSpec{\n+ \t\tObjectMeta: v1.ObjectMeta{\n+ \t\t\tLabels: map[string]string{\n+ \t\t\t\t\t\"app.kubernetes.io/component\": \"run\", \n+ \t\t\t\t\t\"app.kubernetes.io/part-of\": \"spring-petclinic-app-db\", \n+ \t\t\t\t\t\"carto.run/workload-name\": \"spring-petclinic-app-db\", \n+ \t\t\t\t\t\"tanzu.app.live.view\": \"true\", \n+ \t\t\t\t\t\"developer.conventions/target-containers\": \"workload\"}, \n+ \t\t\t}, \n+ \t\t\tSpec: v1.PodSpec{Containers: [v1.Container{...}], ServiceAccountName: \"default\"}, \n+ \t}, \n )\n\", \"convention\": \"apliveview-sample\"}
...
```

## References

- [ImageConfig](#)

- [PodConventionContextSpec](#)
- [PodConventionContextStatus](#)
- [PodConventionContext](#)
- [ClusterPodConvention](#)
- [PodIntent](#)
- [BOM](#)

## Convention Service Resources for Cartographer Conventions

This reference topic describes the convention service resources you can use with Cartographer Conventions.

### Overview

There are several [resources](#) involved in the application of conventions to workloads.

### API Structure

The [PodConventionContext](#) API object in the `webhooks.conventions.carto.run` API group is the structure used for both request and response from the convention server.

### Template Status

The enriched [PodTemplateSpec](#) is reflected at `.status.template`. For more information about [PodTemplateSpec](#), see the [Kubernetes documentation](#).

## Chaining Multiple Conventions

You can define multiple [ClusterPodConventions](#) and apply them to different types of workloads. You can also apply multiple conventions to a single workload.

The [PodIntent](#) reconciler lists all [ClusterPodConvention](#) resources and applies them serially. To ensure the consistency of enriched [PodTemplateSpec](#), the list of [ClusterPodConventions](#) is sorted alphabetically by name before applying conventions. You can use strategic naming to control the order in which the conventions are applied.

After the conventions are applied, the [Ready](#) status condition on the [PodIntent](#) resource is used to indicate whether it is applied successfully. A list of all applied conventions is stored under the annotation `conventions.carto.run/applied-conventions`.

## Collecting Logs from the Controller

The convention controller is a Kubernetes operator and can be deployed in a cluster with other components. If you have trouble, you can retrieve and examine the logs from the controller to help identify issues.

To retrieve Pod logs from the `conventions-controller-manager` running in the `conventions-system` namespace:

```
kubectl -n conventions-system logs -l control-plane=controller-manager
```

For example:

```

...
{"level":"info","ts":1637073467.3334172,"logger":"controllers.PodIntent.PodIntent.ApplyConventions","msg":"applied convention","diff":"  interface{}(\n-  \ts\"&PodTemplateSpec{ObjectMeta:{      0 0001-01-01 00:00:00 +0000 UTC <nil> <nil> map[app.kubernetes.io/component:run app.kubernetes.io/part-of:spring-petclinic-app-db carto.run/workload-name:spring-petclinic-app-db] map[developer.conventions/target-container\"...\n+  \tv1.PodTemplateSpec{\n+  \t\tObjectMeta: v1.ObjectMeta{\n+  \t\t\tLabels: map[string]string{\n+  \t\t\t\t\t\"app.kubernetes.io/component\": \"run\", \n+  \t\t\t\t\t\"app.kubernetes.io/part-of\": \"spring-petclinic-app-db\", \n+  \t\t\t\t\t\"carto.run/workload-name\": \"spring-petclinic-app-db\", \n+  \t\t\t\t\t\"tanzu.app.live.view\": \"true\", \n+  \t\t\t\t\t...\n+  \t\t\t}, \n+  \t\t\tAnnotations: map[string]string{\"developer.conventions/target-containers\": \"workload\"}, \n+  \t\t\t}, \n+  \t\t\tSpec: v1.PodSpec{Containers: []v1.Container{...}, ServiceAccountName: \"default\"}, \n+  \t}, \n+  )\n\", \"convention\":\"appliveview-sample\"}
...

```

## References

- [ImageConfig](#)
- [PodConventionContextSpec](#)
- [PodConventionContextStatus](#)
- [PodConventionContext](#)
- [ClusterPodConvention](#)
- [PodIntent](#)
- [BOM](#)

## ImageConfig for Cartographer Conventions

This reference topic describes the ImageConfig object you can use with Cartographer Conventions.

### Overview

The image configuration object holds the name of the image, the [BOM](#), and the [OCI image configuration](#) with image metadata from the repository.

[OCI image configuration](#) contains the metadata from the image repository.

The [BOM](#) represents the content of the image and may be zero or more per image.

```

{
  "name": "oci-image-name",
  "boms": [{
    "name": "bom-name",
    "raw": "`a byte array`"
  }],
  "config": {
    {
      "created": "2015-10-31T22:22:56.015925234Z",
      "author": "Alyssa P. Hacker <alypspdev@example.com>",
      "architecture": "amd64",
      "os": "linux",
      "config": {
        "User": "alice",
        "ExposedPorts": {
          "8080/tcp": {}
        },
      },
      "Env": [

```

```

        "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
        "FOO=oci_is_a",
        "BAR=well_written_spec"
    ],
    "Entrypoint": [
        "/bin/my-app-binary"
    ],
    "Cmd": [
        "--foreground",
        "--config",
        "/etc/my-app.d/default.cfg"
    ],
    "Volumes": {
        "/var/job-result-data": {},
        "/var/log/my-app-logs": {}
    },
    "WorkingDir": "/home/alice",
    "Labels": {
        "com.example.project.git.url": "https://example.com/project.git",
        "com.example.project.git.commit": "45a939b2999782a3f005621a8d0f29aa387
e1d6b"
    }
},
"rootfs": {
"diff_ids": [
    "sha256:c6f988f4874bb0add23a778f753c65efe992244e148a1d2ec2a8b664fb66bbd1",
    "sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef"
],
"type": "layers"
},
"history": [
{
    "created": "2015-10-31T22:22:54.690851953Z",
    "created_by": "/bin/sh -c #(nop) ADD file:a3bc1e842b69636f9df5256c49c5374f
b4eef1e281fe3f282c65fb853ee171c5 in /"
},
{
    "created": "2015-10-31T22:22:55.613815829Z",
    "created_by": "/bin/sh -c #(nop) CMD [\"sh\"]",
    "empty_layer": true
},
{
    "created": "2015-10-31T22:22:56.329850019Z",
    "created_by": "/bin/sh -c apk add curl"
}
]
}
}
}

```

## PodConventionContextSpec for Cartographer Conventions

This reference topic describes the `PodConventionContextSpec` you can use with Cartographer Conventions.

### Overview

The Pod convention context specification is a wrapper of the `PodTemplateSpec` and the `ImageConfig` provided in the request body of the server. It represents the original `PodTemplateSpec`. For more information on `PodTemplateSpec`, see the [Kubernetes documentation](#).

```

{
  "template": {

```

```

"metadata": {
  ...
},
"spec": {
  ...
}
},
"imageConfig": {
  ...
  "name": "oci-image-name",
  "config": {
    ...
  }
}
}

```

## PodConventionContextStatus for Cartographer Conventions

This reference topic describes the `PodConventionContextStatus` status type that you can use with Cartographer Conventions.

### Overview

The Pod convention context status type is used to represent the current status of the context retrieved by the request. It holds the applied conventions by the server and the modified version of the `PodTemplateSpec`. For more information about `PodTemplateSpec`, see the [Kubernetes documentation](#).

The field `.template` is populated with the enriched `PodTemplateSpec`. The field `.appliedConventions` is populated with the names of any applied conventions.

```

{
  "template": {
    "metadata": {
      ...
    },
    "spec": {
      ...
    }
  },
  "appliedConventions": [
    "convention-1",
    "convention-2",
    "convention-4"
  ]
}

```

yaml version:

```

---
apiVersion: webhooks.conventions.carto.run/v1alpha1
kind: PodConventionContext
metadata:
  name: sample # the name of the ClusterPodConvention
spec: # the request
  imageConfig:
  template:
    <corev1.PodTemplateSpec>
status: # the response
  appliedConventions: # list of names of conventions applied

```

```

- my-convention
template:
spec:
  containers:
  - name : workload
    image: helloworld-go-mod

```

## PodConventionContext for Cartographer Conventions

This reference topic describes the `PodConventionContext` that you can use with Cartographer Conventions.

### Overview

The Pod convention context is the body of the webhook request and response. The specification is provided by the convention controller and the status is set by the convention server.

The context is a wrapper of the individual object description in an API (`TypeMeta`), the persistent metadata of a resource (`ObjectMeta`), the `PodConventionContextSpec` and the `PodConventionContextStatus`.

### PodConventionContext Objects

In the `PodConventionContext` API resource:

- Object path `.spec.template` field defines the `PodTemplateSpec` to be enriched by conventions. For more information about `PodTemplateSpec`, see the [Kubernetes documentation](#).
- Object path `.spec.imageConfig[]` field defines `ImageConfig`. Each entry of it is populated with the name of the image (`.spec.imageConfig[].image`) and its OCI metadata (`.spec.imageConfig[].config`). These entries are generated for each image referenced in `PodTemplateSpec` (`.spec.template`).

The following is an example of a `PodConventionContext` resource request received by the convention server. This resource is generated for a [Go language-based application image](#) in GitHub. It is built with Cloud Native Paketo Buildpacks that use Go mod for dependency management.

```

---
apiVersion: webhooks.conventions.carto.run/v1alpha1
kind: PodConventionContext
metadata:
  name: sample # the name of the ClusterPodConvention
spec: # the request
  imageConfig: # one entry per image referenced by the PodTemplateSpec
  - image: sample/go-based-image
    bom:
      - 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":"7fef8ba6a0786143efc666b0bbfbfbab02afeef522b4e09833c5b550d7
`...
    template:
      spec:
        containers:
          - name : workload
            image: helloworld-go-mod

```

## PodConventionContext Structure

This section introduces more information about the image configuration in [PodConventionContext](#). The convention-controller passes this information for each image in good faith. The controller is not the source of the metadata, and there is no guarantee that the information is correct.

The `config` field in the image configuration passes through the [OCI Image metadata in GitHub](#) loaded from the registry for the image.

The `boms` field in the image configuration passes through the [BOMs](#) of the image. Conventions might parse the BOMs they want to inspect. There is no guarantee that an image contains a BOM or that the BOM is in a certain format.

## ClusterPodConvention for Cartographer Conventions

This reference topic describes the [ClusterPodConvention](#) that you can use with Cartographer Conventions.

### Overview

[ClusterPodConvention](#) defines a way to connect to convention servers. It provides a way to apply a set of conventions to a [PodTemplateSpec](#) and the artifact metadata. A convention will typically focus on a particular application framework, but may be cross cutting. Applied conventions must be pure functions.

### Define conventions

Webhook servers are the only way to define conventions.

```

apiVersion: conventions.carto.run/v1alpha1
kind: ClusterPodConvention
metadata:
  name: base-convention
  annotations:
    conventions.carto.run/inject-ca-from: "convention-template/webhook-cert"
spec:
  selectorTarget: PodTemplateSpec # optional, defaults to PodTemplateSpec; field optio
ns include PodTemplateSpec|PodIntent
  webhook:
    clientConfig:
      service:
        name: webhook
        namespace: convention-template

```

## PodIntent for Cartographer Conventions



This reference topic describes `PodIntent` that you can use with Cartographer Conventions.

## Overview

The conditional criteria governing the application of a convention is customizable and is based on the evaluation of a custom Kubernetes resource called `PodIntent`.

`PodIntent` applies conventions to a workload. A `PodIntent` is created, or updated, when a workload is run by using a Tanzu Application Platform supply chain.

The `.spec.template`'s `PodTemplateSpec` is enriched by the conventions and exposed as the `.status.templates` `PodTemplateSpec`. A log of which sources and conventions are applied is captured with the `conventions.carto.run/applied-conventions` annotation on the `PodTemplateSpec`.

```
apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
  name: sample
spec:
  template:
    spec:
      containers:
      - name: workload
        image: ubuntu
```

## BOM for Cartographer Conventions

This reference topic describes the `BOM` structure you can use with Cartographer Conventions.

### Overview

The `BOM` is a type/structure wrapping a Software Bill of Materials (SBOM) describing the software components and their dependencies.

### Structure

The structure of the `BOM` is defined as follows:

```
{
  "name": "BOM-NAME",
  "raw": "BYTE-ARRAY"
}
```

Where:

- `BOM-NAME` is the prefix `cnb-sbom:`, followed by the location of the BOM definition in the layer for a cloud native buildpack (CNB) SBOM. For example: `cnb-sbom:/layers/sbom/launch/paketo-buildpacks_executable-jar/sbom.cdx.json`. For a non-CN B SBOM, the value of `name` might change.
- `BYTE-ARRAY`: The content of the BOM. The content may be in any format or encoding. Consult the name to infer how the content is structured.

The convention controller forwards BOMs to the convention servers that it can discover from known sources, including:

- [CNB-SBOM](#)

## Overview of cert-manager

cert-manager adds certificates and certificate issuers as resource types to Kubernetes clusters. It also helps you to obtain, renew, and use those certificates. For more information about cert-manager, see [cert-manager documentation](#).

The cert-manager package allows you to, optionally, configure a number of `ClusterIssuer`. When you install Tanzu Application Platform by using profiles, a self-signed `ClusterIssuer` is included by default.

As of `cert-manager.tanzu.vmware.com/2.0.0`, versioning departs from the upstream, open-source project's version. The contained cert-manager version is reflected in `Package.spec.includedSoftware`. You can identify the version of cert-manager as follows:

```
kubectl get package -n tap-install cert-manager.tanzu.vmware.com.2.0.0 -ojsonpath='{.spec.includedSoftware}' | jq
[
  {
    "description": "X.509 certificate management for Kubernetes and OpenShift",
    "displayName": "cert-manager",
    "version": "1.9.1"
  }
]
```

## Overview of cert-manager

cert-manager adds certificates and certificate issuers as resource types to Kubernetes clusters. It also helps you to obtain, renew, and use those certificates. For more information about cert-manager, see [cert-manager documentation](#).

The cert-manager package allows you to, optionally, configure a number of `ClusterIssuer`. When you install Tanzu Application Platform by using profiles, a self-signed `ClusterIssuer` is included by default.

As of `cert-manager.tanzu.vmware.com/2.0.0`, versioning departs from the upstream, open-source project's version. The contained cert-manager version is reflected in `Package.spec.includedSoftware`. You can identify the version of cert-manager as follows:

```
kubectl get package -n tap-install cert-manager.tanzu.vmware.com.2.0.0 -ojsonpath='{.spec.includedSoftware}' | jq
[
  {
    "description": "X.509 certificate management for Kubernetes and OpenShift",
    "displayName": "cert-manager",
    "version": "1.9.1"
  }
]
```

## Install cert-manager

This topic tells you how to install cert-manager from the Tanzu Application Platform (commonly known as TAP) package repository.



### Note

Follow the steps in this topic if you do not want to use a profile to install cert-manager. For more information about profiles, see [Components and installation profiles](#).

The cert-manager package installs cert-manager and, optionally, a number of `ClusterIssuer`. To install cert-manager with a self-signed `ClusterIssuer` from the Tanzu Application Platform package repository:

1. List version information for the package by running:

```
tanzu package available list cert-manager.tanzu.vmware.com -n tap-install
```

For example:

```
$ tanzu package available list cert-manager.tanzu.vmware.com -n tap-install
/ Retrieving package versions for cert-manager.tanzu.vmware.com...
NAME                                VERSION    RELEASED-AT
cert-manager.tanzu.vmware.com      2.0.0     ...
```

2. Discover available configuration for the package by running:

```
tanzu package available get cert-manager.tanzu.vmware.com/2.0.0 --namespace tap
-install --values-schema
```

For example:

```
$ tanzu package available get cert-manager.tanzu.vmware.com/2.0.0 --namespace t
ap-install --values-schema

KEY                                DEFAULT  TYPE    DESCRIPTION
certManager.pspNames              []       array   PodSecurityPolicy names which cert-manag
er is allowed to use
issuers                            []       array   The ClusterIssuers to install - default:
[]
namespace                          string   string  Cert-manager's namespace - also used as
its cluster resource namespace
https://cert-manager.io/v1.9-docs/faq/cluster-resource/
```

3. Create a file named `cert-manager-rbac.yaml` by using the following sample:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: cert-manager-tap-install-cluster-admin-role
rules:
- apiGroups:
  - '*'
  resources:
  - '*'
  verbs:
  - '*'
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: cert-manager-tap-install-cluster-admin-role-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cert-manager-tap-install-cluster-admin-role
subjects:
```

```

- kind: ServiceAccount
  name: cert-manager-tap-install-sa
  namespace: tap-install
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: cert-manager-tap-install-sa
  namespace: tap-install

```

Apply the configuration:

```
kubectl apply -f cert-manager-rbac.yaml
```

4. Create a file named `cert-manager-install.yaml` by using the following sample:

```

---
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: cert-manager
  namespace: tap-install
spec:
  serviceAccountName: cert-manager-tap-install-sa
  packageRef:
    refName: cert-manager.tanzu.vmware.com
    versionSelection:
      constraints: "VERSION-NUMBER"
      prereleases: {}
  values:
    - secretRef:
        name: cert-manager-values
---
apiVersion: v1
kind: Secret
metadata:
  name: cert-manager-values
  namespace: tap-install
stringData:
  values.yaml: |
    issuers:
      - name: tap-ingress-selfsigned
        self_signed: {}

```

Where:

- o `VERSION-NUMBER` is the version of the package listed earlier.
- o Secret `cert-manager-values` contains your configuration of the cert-manager package.

Apply the configuration:

```
kubectl apply -f cert-manager-install.yaml
```

5. Verify the package installation:

```
tanzu package installed get cert-manager -n tap-install
```

For example:

```

$ tanzu package installed get cert-manager -n tap-install
/ Retrieving installation details for cert-manager...

```

```

NAME: cert-manager
PACKAGE-NAME: cert-manager.tanzu.vmware.com
PACKAGE-VERSION: 2.0.0
STATUS: Reconcile succeeded
CONDITIONS: [{ReconcileSucceeded True}]
USEFUL-ERROR-MESSAGE:

```

Verify that `STATUS` is `Reconcile succeeded`

6. Verify that cert-manager is up and running:

```
kubectl get deployment -n cert-manager
```

For example:

```

$ kubectl get deployment -n cert-manager
NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
cert-manager            1/1     1             1           5m
cert-manager-cainjector 1/1     1             1           5m
cert-manager-webhook    1/1     1             1           5m

```

7. Verify that the self-signed `ClusterIssuer` is present:

```
kubectl get clusterissuer
```

For example:

```

$ kubectl get clusterissuer
NAME                    READY   AGE
tap-ingress-selfsigned  True    5m
tap-ingress-selfsigned-bootstrap  True    5m
...

```

## Overview of 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](#).

## Overview of Cloud Native Runtimes

Cloud Native Runtimes for Tanzu is a serverless application runtime for Kubernetes that is based on Knative and runs on a single Kubernetes cluster.

To learn more about Cloud Native Runtimes, see [Cloud Native Runtimes for VMware Tanzu](#).

## Install Cloud Native Runtimes

This topic describes how you can install Cloud Native Runtimes from the Tanzu Application Platform package repository.



### Note

Follow the steps in this topic if you do not want to use a profile to install Cloud Native Runtimes. For more information about profiles, see [Components and installation profiles](#).

## Prerequisites

Before installing Cloud Native Runtimes:

- Complete all prerequisites to install Tanzu Application Platform. See [Prerequisites](#).
- Ensure that Contour v1.22.0 or later is installed. Tanzu Application Platform includes a correctly versioned package of Contour if you do not have it installed already.

## 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 2.1.0    2022-12-08 16:00:00 -0800 PST
```

2. (Optional) Make changes to the default installation settings:

1. Gather values schema.

```
tanzu package available get cnrs.tanzu.vmware.com/2.1.0 --values-schema -n tap-install
```

For example:

```
$ tanzu package available get cnrs.tanzu.vmware.com/2.1.0 --values-schema -n tap-install
| Retrieving package details for cnrs.tanzu.vmware.com/2.1.0...
KEY                                DEFAULT                                TYPE
DESCRIPTION
https_redirection                  true                                   boole
an CNRs ingress will send a 301 redirect for all http connections, asking the clients to use HTTPS
ingress.internal.namespace         tanzu-system-ingress                 strin
g Required. Specify a namespace where an existing Contour is installed on your cluster. CNR will use this Contour instance for internal services.
ingress.external.namespace         tanzu-system-ingress                 strin
g Required. Specify a namespace where an existing Contour is installed on your cluster. CNR will use this Contour instance for external services.
ingress_issuer                     string
Cluster issuer to be used in CNRs. To use this property the domain_name or domain_config must be set. Under the hood, when this property is set auto-tls is Enabled.
namespace_selector                 string
Specifies a LabelSelector which determines which namespaces should have a wildcard certificate provisioned. Set this property only if the Cluster issuer is type DNS-01 challenge.
domain_config                       <nil>                                  <nil>
Optional. Overrides the Knative Serving "config-domain" ConfigMap, allowing you to map Knative Services to specific domains. Must be valid YAML and conform to the "config-domain" specification.
domain_template                     {{.Name}}.{{.Namespace}}.{{.Domain}} string
Optional. Specifies the goolang text template string to use when const
```

```

ructing the DNS name for a Knative Service.
  lite.enable                false                <nil>
Optional. Set to "true" to enable lite mode. Reduces CPU and Memory resource
requests for all cnrs Deployments, Daemonsets, and StatefulSets by half. Not
recommended for production.
  pdb.enable                 true                <nil>
Optional. Set to true to enable a PodDisruptionBudget for the Knative Serving
activator and webhook deployments.
  default_tls_secret        string
Optional. Specify a fallback TLS Certificate for use by Knative Services if
autoTLS is disabled. Will set default external scheme for Knative Service
URLs to "https". Requires either "domain_name" or "domain_config" to be set.
  kubernetes_version       0.0.0         <nil>
Optional. Version of K8s infrastructure being used. Supported Values: valid
Kubernetes major.minor.patch versions
  ca_cert_data             string
Optional. PEM Encoded certificate data to trust TLS connections with a private
CA.
  provider                  <nil>         <nil>
Deprecated. Instead, use "lite.enable" and "pdb.enable" options combined.
Supported Values: local
  domain_name              string
Optional. Default domain name for Knative Services.
  kubernetes_distribution  <nil>         <nil>
Optional. Type of K8s infrastructure being used. Supported Values: openshift

```

2. Create a `cnr-values.yaml` by using the following sample as a guide:

```

---
domain_name: example.com
ingress:
external:
  namespace: tanzu-system-ingress
internal:
  namespace: tanzu-system-ingress

```



#### 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 `lite.enable: true` option. This option reduces resources requests for Cloud Native Runtimes deployments.

Cloud Native Runtimes uses the existing Contour installation in the `tanzu-system-ingress` namespace by default for external and internal access.

If your environment has Contour installed already, and it is not the Tanzu Application Platform provided Contour, you can configure Cloud Native Runtimes to use it. See [Installing Cloud Native Runtimes for Tanzu with an Existing Contour Installation](#) in the Cloud Native Runtimes documentation.

3. Install the package by running:

```

tanzu package install cloud-native-runtimes -p cnrs.tanzu.vmware.com -v 2.1.0 -n
tap-install -f cnr-values.yaml --poll-timeout 30m

```

For example:

```
$ tanzu package install cloud-native-runtimes -p cnrs.tanzu.vmware.com -v 2.1.0
-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 earlier 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:     2.1.0
STATUS:              Reconcile succeeded
CONDITIONS:          [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`

5. Configure a namespace to use Cloud Native Runtimes:



### Important

This step covers configuring a namespace to run Knative services. If you rely on a SupplyChain to deploy Knative services into your cluster, skip this step because namespace configuration is covered in [Set up developer namespaces to use your installed packages](#). Otherwise, you must follow these 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 these commands to create an empty secret and annotate it as a target of the secretgen controller:

```
kubectl create secret generic pull-secret --from-literal=.dockerconfigjso
n={} --type=kubernetes.io/dockerconfigjson
kubectl annotate secret pull-secret secretgen.carvel.dev/image-pull-secre
t=""
```



2. After you create a `pull-secret` secret in the same namespace as the service account, run the following command to add the secret to the service account:

```
kubectl patch serviceaccount default -p '{"imagePullSecrets": [{"name": "pull-secret"}]}'
```

3. Verify that a service account is correctly configured by running:

```
kubectl describe serviceaccount default
```

For example:

```
kubectl describe sa default
Name:                default
Namespace:           default
Labels:              <none>
Annotations:         <none>
Image pull secrets:  pull-secret
Mountable secrets:   default-token-xh6p4
Tokens:              default-token-xh6p4
Events:              <none>
```



#### Note

The service account has access to the `pull-secret` image pull secret.

## Overview of Contour

Contour is an ingress controller for Kubernetes that supports dynamic configuration updates and multi-team ingress delegation. It provides the control plane for the Envoy edge and service proxy. For more information about Contour, see [Contour documentation](#).

## Overview of Contour

Contour is an ingress controller for Kubernetes that supports dynamic configuration updates and multi-team ingress delegation. It provides the control plane for the Envoy edge and service proxy. For more information about Contour, see [Contour documentation](#).

## Install Contour

This topic tells you how to install Contour from the Tanzu Application Platform (commonly known as TAP) package repository.



#### Note

Follow the steps in this topic if you do not want to use a profile to install Contour. For more information about profiles, see [Components and installation profiles](#).

To install Contour from the Tanzu Application Platform package repository without a profile:

1. [Install cert-manager](#).
2. List version information for the package by running:

```
tanzu package available list contour.tanzu.vmware.com -n tap-install
```

For example:

```
$ tanzu package available list contour.tanzu.vmware.com -n tap-install
- Retrieving package versions for contour.tanzu.vmware.com...
NAME                               VERSION      RELEASED-AT
contour.tanzu.vmware.com 1.22.0+tap.8 2022-12-05 19:00:00 -0500 EST
```

3. Create a file named `contour-rbac.yaml` by using the following sample and apply the configuration:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: contour-tap-install-cluster-admin-role
rules:
- apiGroups:
  - '*'
  resources:
  - '*'
  verbs:
  - '*'
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: contour-tap-install-cluster-admin-role-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: contour-tap-install-cluster-admin-role
subjects:
- kind: ServiceAccount
  name: contour-tap-install-sa
  namespace: tap-install
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: contour-tap-install-sa
  namespace: tap-install
```

4. Apply the configuration by running:

```
kubectl apply -f contour-rbac.yaml
```

5. Create a file named `contour-install.yaml` by using the following sample and apply the configuration:



#### Note

The following configuration installs the Contour package with default options. To make changes to the default installation settings, go to the next step.

```
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: contour
```

```

namespace: tap-install
spec:
  serviceAccountName: contour-tap-install-sa
  packageRef:
    refName: contour.tanzu.vmware.com
    versionSelection:
      constraints: "VERSION-NUMBER"
      prereleases: {}

```

Where `VERSION-NUMBER` is the version of the package listed earlier.

## 6. (Optional) Make changes to the default installation settings:

### 1. Gather values schema by running:

```
tanzu package available get contour.tanzu.vmware.com/1.22.0+tap.8 --value
s-schema -n tap-install
```

For example:

```

$ tanzu package available get contour.tanzu.vmware.com/1.22.0+tap.8 --val
ues-schema -n tap-install

  KEY                                DEFAULT                                TYPE    DES
  CRIPITION
  kubernetes_version                 0.0.0                                string  Opt
  ional. Version of K8s infrastructure being used. Supported Values: valid
  Kubernetes major.minor.patch versions
  namespace                          tanzu-system-ingress                 string  The
  namespace in which to deploy Contour and Envoy.
  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>                                <nil>   The
  YAML contents of the Contour config file. See https://projectcontour.io/d
  ocs/v1.22.0/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.terminationGracePeriodSeconds 300                                  integer The
  termination grace period, in seconds, for the Envoy pods.
  envoy.workload.replicas              2                                    integer The
  number of Envoy replicas to deploy when 'type' is set to 'Deployment'. If
  not specified, it will default to '2'.
  envoy.workload.type                  DaemonSet                             string  The
  type of Kubernetes workload Envoy is deployed as. Options are 'Deploymen
  t' or 'DaemonSet'. If not specified, will default to 'DaemonSet'.
  envoy.hostNetwork                    false                                 boolean Whe
  ther to enable host networking for the Envoy pods.
  envoy.hostPorts.enable               true                                  boolean Whe
  ther to enable host ports. If false, http & 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.nodePorts.https        0                                    integer The
  node port number to expose Envoy's HTTPS listener on. If not specified, a
  node port will be auto-assigned by Kubernetes.

```

```

  envoy.service.nodePorts.http          0          integer  The
  node port number to expose Envoy's HTTP listener on. If not specified, a
  node port will be auto-assigned by Kubernetes.
  envoy.service.type                    string     The
  type of Kubernetes service to provision for Envoy. If not specified, will
  default to 'NodePort' for docker and vsphere and 'LoadBalancer' for other
  s.
  envoy.service.annotations             <nil>     <nil>     Ann
  otations to set on the Envoy service.
  envoy.service.aws.LBType              classic   string     AWS
  loadbalancer type.
  envoy.service.externalTrafficPolicy   string     The
  external traffic policy for the Envoy service. If type is 'ClusterIP', th
  is field is ignored. Otherwise, defaults to 'Cluster' for vsphere and 'Lo
  cal' for others.
  envoy.service.loadBalancerIP         string     The
  desired load balancer IP. If type is not 'LoadBalancer', this field is ig
  nored. It is up to the cloud provider whether to honor this request. If n
  ot specified, then load balancer IP will be assigned by the cloud provide
  r.
  infrastructure_provider               vsphere   string     The
  underlying infrastructure provider. Valid values are `vsphere`, `aws` and
  `azure`.
  kubernetes_distribution               string     Kub
  ernetes distribution that this package is being installed on. Supported v
  alues: ['', 'openshift']

```

2. Create a `contour-install.yaml` file by using the following sample as a guide:



#### Note

This sample is for installation in an AWS public cloud with `LoadBalancer` services.

```

apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: contour
  namespace: tap-install
spec:
  serviceAccountName: contour-tap-install-sa
  packageRef:
    refName: contour.tanzu.vmware.com
    versionSelection:
      constraints: 1.22.0+tap.8
      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.

For local clusters, you can configure `contour.evnoy.service.type` to be `NodePort`. If your local cluster is set up with extra port mappings on the nodes, you might also need configure `envoy.service.nodePorts.http` and `envoy.service.nodePorts.https` to match the port mappings from your local machine into one of the nodes of your local cluster. This pattern is seen when using the [Learning Center on Kind](#).

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.

7. Install the package by running:

```
kubectl apply -f contour-install.yaml
```

8. Verify the package install by running:

```
tanzu package installed get contour -n tap-install
```

For example:

```
$ tanzu package installed get contour -n tap-install
/ Retrieving installation details for contour...
NAME:                contour
PACKAGE-NAME:        contour.tanzu.vmware.com
PACKAGE-VERSION:     1.22.0+tap.8
STATUS:              Reconcile succeeded
CONDITIONS:          [{"ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`

9. Verify the installation by running:

```
kubectl get po -n tanzu-system-ingress
```

For example:

```
$ kubectl get po -n tanzu-system-ingress
NAME                                READY   STATUS    RESTARTS   AGE
contour-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.

## Configure Cipher Suites and TLS version in Contour

This topic tells you how to configure TLS options for Contour in Tanzu Application Platform (commonly known as TAP).

Contour provides configuration options for TLS version and Cipher Suites. Rather than directly exposed through a top level key in the package, they fall into the category of advanced Contour configurations by using the `contour.configFileContents` key. For more information about these configuration options, see [Contour documentation](#).

To configure TLS options for Contour in Tanzu Application Platform, edit the `contour` section of your TAP values file as follows:

```
contour:
  # ... there maybe some configuration already here
  contour:
    configFileContents:
      tls:
        minimum-protocol-version: "1.2"
        cipher-suites:
          - '[ECDHE-ECDSA-AES128-GCM-SHA256|ECDHE-ECDSA-CHACHA20-POLY1305]'
          - '[ECDHE-RSA-AES128-GCM-SHA256|ECDHE-RSA-CHACHA20-POLY1305]'
          - 'ECDHE-ECDSA-AES256-GCM-SHA384'
          - 'ECDHE-RSA-AES256-GCM-SHA384'
```

Expect to see the following Cipher Suites and TLS version data in the Contour configmap:

```
$ kubectl -n tanzu-system-ingress get configmap contour -oyaml
apiVersion: v1
data:
  contour.yaml: |
    tls:
      minimum-protocol-version: "1.2"
      cipher-suites:
        - '[ECDHE-ECDSA-AES128-GCM-SHA256|ECDHE-ECDSA-CHACHA20-POLY1305]'
        - '[ECDHE-RSA-AES128-GCM-SHA256|ECDHE-RSA-CHACHA20-POLY1305]'
        - ECDHE-ECDSA-AES256-GCM-SHA384
        - ECDHE-RSA-AES256-GCM-SHA384
kind: ConfigMap
metadata:
  ...
```



### Important

To update the configmap, you must configure it through Tanzu Application Platform values file. If you change it directly in the configmap, kapp-controller reverts all the changes you made.

## Configure Contour

This topic tells you how to configure Contour to best suit your cluster.

By default, Contour installs with the Controllers as a [Deployment](#) and the Envoy as a [DaemonSet](#). In most cases, this is sufficient. However, VMware recommends running Envoy as a [Deployment](#) in the following scenarios:

- [Smaller Clusters](#)
- [Larger Clusters](#)

## Smaller Clusters

On most clusters, a `DaemonSet` works without any issues. However, if you limit resources per node and the nodes are heavily used, deploying Envoy as a `DaemonSet` might consume unnecessary resources on every node. In this case, VMware recommends using `Deployment` with a fixed number of replicas.

## Larger Clusters

On larger clusters, running Envoy as a `DaemonSet` might be inefficient. The more Envoy's in the cluster, the more resources the Contour controller needs to keep them updated. If the Contour controllers use lots of resources, VMware recommends running Envoy as a `Deployment`.

## Configuring Envoy as a Deployment

To configure Envoy as a `Deployment`, update your Contour values file as follows:

```
envoy:
  workload:
    type: Deployment
    replicas: N
```

If you use a Tanzu Application Platform values file, you can add these configurations to the `contour` section.

## Overview of Eventing

Eventing in Tanzu Application Platform (commonly known as TAP) is a collection of APIs based on [Knative Eventing](#) that allows the use of an event-driven architecture with your applications.

## Overview of Eventing

Eventing in Tanzu Application Platform (commonly known as TAP) is a collection of APIs based on [Knative Eventing](#) that allows the use of an event-driven architecture with your applications.

## Install Eventing

This topic tells you how to install the Eventing package from the Tanzu Application Platform (commonly known as TAP) package repository.



### Note

Follow the steps in this topic if you do not want to use a profile to install Eventing. For more information about profiles, see [Components and installation profiles](#).

## Prerequisites

Before installing Eventing:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see [Prerequisites](#).

## Install

To install Eventing:

1. List version information for the package by running:

```
tanzu package available list eventing.tanzu.vmware.com --namespace tap-install
```

For example:

```
$ tanzu package available list eventing.tanzu.vmware.com --namespace tap-install
- Retrieving package versions for eventing.tanzu.vmware.com...
NAME                                VERSION  RELEASED-AT
eventing.tanzu.vmware.com           2.0.1   2022-10-11T00:00:00Z
```

2. (Optional) Make changes to the default installation settings:

1. Gather values schema.

```
tanzu package available get eventing.tanzu.vmware.com/2.0.1 --values-schema -n tap-install
```

For example:

```
$ tanzu package available get eventing.tanzu.vmware.com/2.0.1 --values-schema -n tap-install
| Retrieving package details for eventing.tanzu.vmware.com/2.0.1...
KEY          DEFAULT  TYPE      DESCRIPTION
lite.enable  false    boolean   Optional: Not recommended for production. Set to "true" to reduce CPU and Memory resource requests for all Eventing Deployments, Daemonsets, and Statefulsets by half. On by default when "provider" is set to "local".
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 Eventing on a local Kubernetes cluster provider.
```

2. Create a `eventing-values.yaml` by using the following sample `eventing-values.yaml` as a guide:

```
---
lite:
  enable: true
```



#### Note

For most installations, you can leave the `eventing-values.yaml` empty, and use the default values.

If you run on a single-node cluster, such as kind or minikube, set the `lite.enable:` property to `true`. This option reduces resources requests for Eventing deployments.

3. Install the package by running:

```
tanzu package install eventing -p eventing.tanzu.vmware.com -v 2.0.1 -n tap-install -f eventing-values.yaml --poll-timeout 30m
```

For example:

```
$ tanzu package install eventing -p eventing.tanzu.vmware.com -v 2.0.1 -n tap-install -f eventing-values.yaml --poll-timeout 30m
```



```

- Installing package 'eventing.tanzu.vmware.com'
| Getting package metadata for 'eventing.tanzu.vmware.com'
| Creating service account 'eventing-tap-install-sa'
| Creating cluster admin role 'eventing-tap-install-cluster-role'
| Creating cluster role binding 'eventing-tap-install-cluster-rolebinding'
| Creating secret 'eventing-tap-install-values'
| Creating package resource
| Waiting for 'PackageInstall' reconciliation for 'eventing'
| 'PackageInstall' resource install status: Reconciling

Added installed package 'eventing'

```

Use an empty file for `eventing-values.yaml` to enable default installation configuration. Otherwise, see the previous step to set installation configuration values.

4. Verify the package install by running:

```
tanzu package installed get eventing -n tap-install
```

For example:

```

tanzu package installed get eventing -n tap-install
| Retrieving installation details for eventing...
NAME:                eventing
PACKAGE-NAME:        eventing.tanzu.vmware.com
PACKAGE-VERSION:     2.0.1
STATUS:              Reconcile succeeded
CONDITIONS:          [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:

```

Verify that `STATUS` is `Reconcile succeeded`.

## Namespace Provisioner

Namespace Provisioner provides a secure, automated way for you to provision namespaces with the resources and namespace-level privileges required for your workloads to function as intended in Tanzu Application Platform (commonly known as TAP).

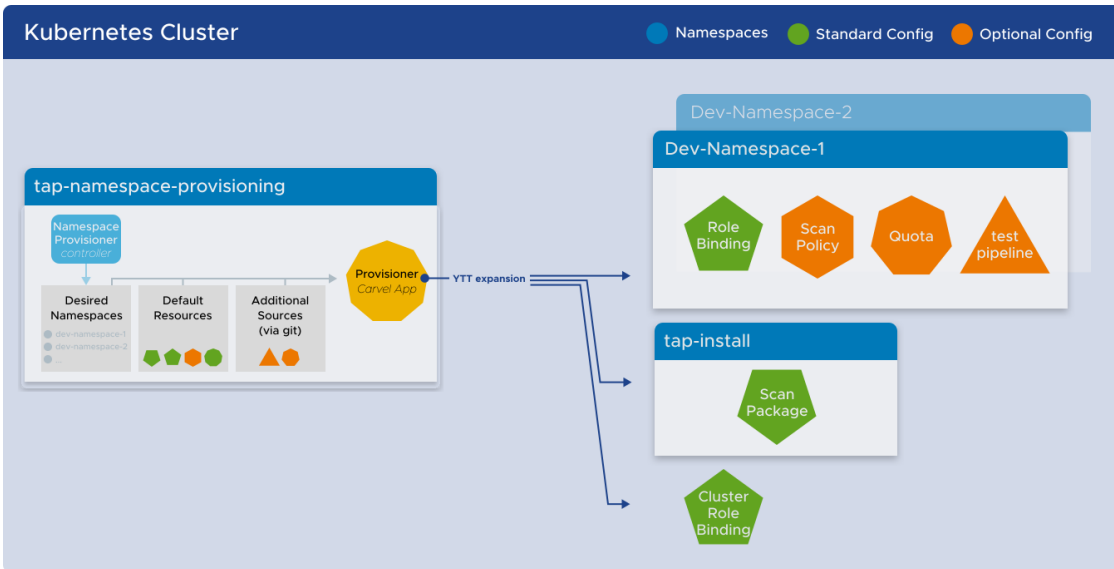
### Description

Namespace Provisioner enables platform operators to add additional customized namespace-scoped resources using GitOps to meet their organization's requirements and provides continuous reconciliation using the kapp-controller to maintain the desired state of the namespace-scoped resources.

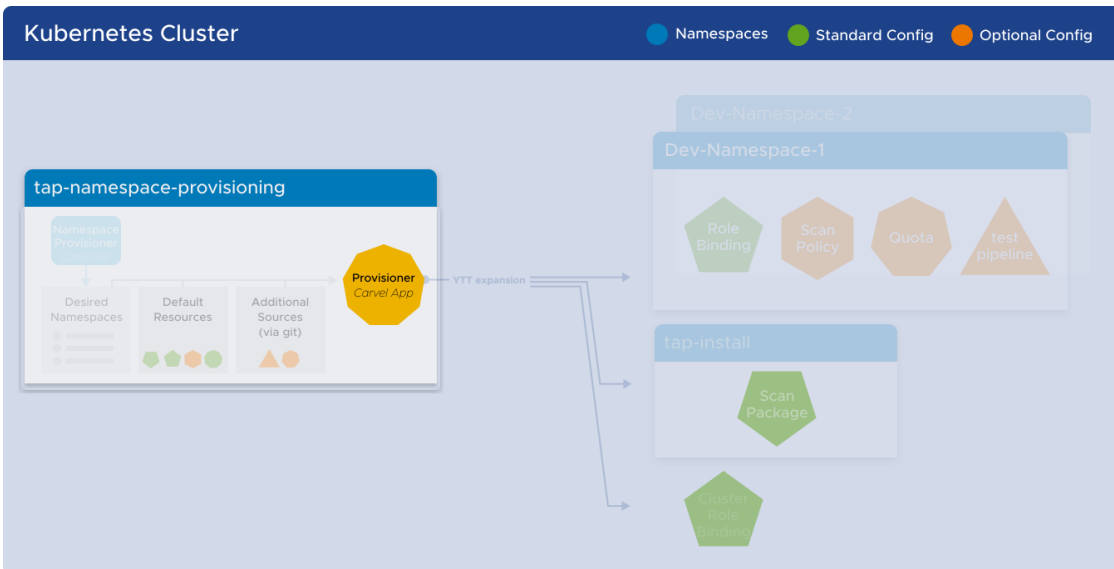
Namespace Provisioner enables operators that are new to Kubernetes to automate the provisioning of multiple developer namespaces in a shared cluster. For organizations that have already adopted Kubernetes, Namespace Provisioner is also compatible with existing Kubernetes tooling.

### Component Overview

The following diagram shows the components that are installed as part of the Namespace Provisioner package and how they work together to automate resource creation in developer namespaces:

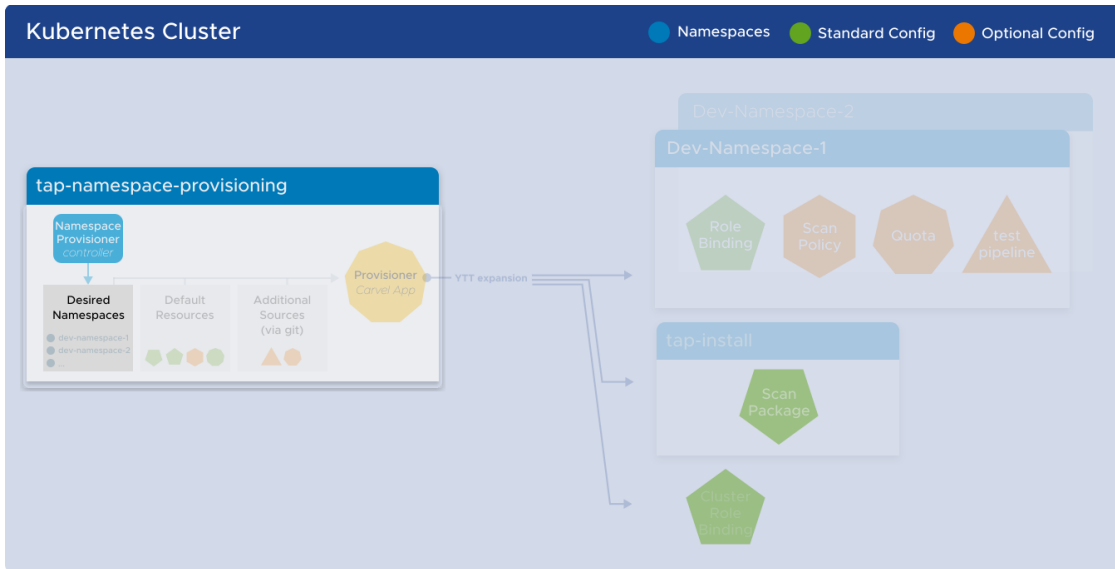


## Provisioner Carvel Application



Namespace Provisioner consists of a Carvel application named provisioner that is installed in the `tap-namespace-provisioning` namespace. The provisioner application uses ytt to templatize a set of resources into installations in multiple namespaces. The provisioner application references a ConfigMap and a secret.

## Desired Namespaces ConfigMap



The `desired-namespaces` ConfigMap in the `tap-namespace-provisioning` namespace provides a declarative way to indicate which namespaces should be populated with resources. The ConfigMap consists of a list of namespace objects, with a required `name` parameter, and optional additional parameters which are used as `data.values` for customizing defined resources. Manage the ConfigMap directly with [GitOps](#), or the [Namespace Provisioner Controller](#).

### Example

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: desired-namespaces
  namespace: tap-namespace-provisioning
  annotations:
    kapp.k14s.io/create-strategy: fallback-on-update
    namespace-provisioner.apps.tanzu.vmware.com/no-overwrite: "" #! This annotation tells the provisioner app to not override this configMap as this is your desired state.
data:
  namespaces.yaml: |
    #@data/values
    ---
    namespaces:
    - name: dev-ns1
      # additional parameters about dev-ns1 in the object...
    - name: dev-ns2
      # additional parameters about dev-ns2 in the object...
```

## Namespace Provisioner Controller

If you prefer to have the `desired-namespaces` ConfigMap automatically managed by a controller on the cluster, use the Namespace Provisioner controller. It is installed as part of Namespace Provisioner. It watches namespaces in the cluster and updates the `desired-namespaces` ConfigMap in the `tap-namespace-provisioning` namespace with a list of namespaces that match the `namespace_selector` label selector in `tap-values.yaml`. The default label selector is configurable in `tap-values.yaml`.

## Default Resources Secret

The `default-resources` secret is templated by `tap-values.yaml` to contain the appropriate resources for the given profile, set of supply chains installed, and other similar values. For the full

list of resources that are created for different profiles, see [Default resources mapping](#).

## Expansion Template ConfigMap

The expansion-template ConfigMap contains the ytt logic to expand the resources defined in the following locations into each of the namespaces listed in the [desired-namespaces ConfigMap](#):

- Default-resources secret
- `additional_sources` in the `namespace-provisioner` config in the `tap-values.yaml` file

## Install Namespace Provisioner

This topic tells you how to install Namespace Provisioner using a profile, and how to customize a standard installation Namespace Provisioner in Tanzu Application Manager (commonly known as TAP).

Namespace Provisioner is packaged and distributed using the Carvel set of tools. The Namespace Provisioner Carvel package is published to the Tanzu Application Platform package repository and two installation approaches are supported:

- [Install Namespace Provisioner](#)
  - [Install using a Profile](#)
  - [Customized Installation](#)

## Install using a Profile

To install Namespace Provisioner as part of a wider Tanzu Application Platform profile based installation, see [Installing Tanzu Application Platform](#).

The Namespace Provisioner package is installed as part of the [standard installation profiles](#). The default set of resources provisioned in a namespace is based on a combination of the Tanzu Application Platform installation profile employed and the supply chain that is installed on the cluster.

For a list of what resources are created for different profile and supply chain combinations, see [default resource mapping table](#).

## Customized Installation

Run:

```
tanzu package available get namespace-provisioner.apps.tanzu.vmware.com/0.1.2 --values
-schema -n tap-install
```

The following values are configurable:

**controller:** Whether to install the [controller](#) that is part of the package. - Set to `true` (Default) to manage the [desired-namespaces ConfigMap](#) automatically using a [controller](#) on the cluster. - Set to `false` to populate the [desired-namespaces ConfigMap](#) using an external mechanism such as GitOps, see [Control the desired-namespaces ConfigMap via GitOps](#).

**aws\_iam\_role\_arn:** If the installation is on AWS with EKS, use the selected IAM Role for Kubernetes Service Accounts.

**additional\_sources:** Add additional sources which contain Platform Operator templated resources to be set on the provisioned namespaces using GitOps in addition to the default-resources that are shipped with Tanzu Application Platform. - See the `fetch` section of the [kapp App](#) specification

section for the format. Only the Git type fetch is supported. - See [Extending the default provisioned resources](#)

- **namespace\_selector**: The label selector used by the [controller](#) to determine which namespaces should be added to the [desired-namespaces ConfigMap](#). See [label selector](#) in the Kubernetes documentation.

Example snippet of `tap-values.yaml`:

```
...
namespace_provisioner:
  controller: true
  namespace_selector:
    matchExpressions:
      - key: apps.tanzu.vmware.com/tap-ns
        operator: Exists
  aws_iam_role_arn: "arn:aws:iam::123456789012:role/EKSIAMRole"
  additional_sources:
    # Patches the OOTB scan policy with a different rego data
    - git:
        ref: tap-1.4-np
        subPath: namespace-provisioner-gitops-examples/default-resources-overrides/overlays
      url: https://github.com/vmware-tanzu/application-accelerator-samples.git
      path: _ytt_lib/customize
    # Add a custom workload service account and a bunch of git secrets
    - git:
        ref: tap-1.4-np
        subPath: namespace-provisioner-gitops-examples/custom-resources/workload-sa
      url: https://github.com/vmware-tanzu/application-accelerator-samples.git
      path: _ytt_lib/workload-sa
  ...
```

## Provision namespace resources

This topic tells you how to use Namespace Provisioner to provision namespace-scoped resources in Tanzu Application Platform (commonly known as TAP).

There are two approaches to provisioning namespace-scoped resources supported:

[Using Namespace Provisioner Controller](#) is recommended for Tanzu Application Platform clusters that:

- include [Out of the Box Supply Chain Basic](#)
- require only the default namespace-scoped resources to be provisioned

[Using GitOps](#) is required for Tanzu Application Platform clusters that meet any of the following:

- include [Out of the Box Supply Chain - Testing and Scanning](#)
- require customization or extension of the default namespace-scoped resources that are provisioned
- prefer to control which namespaces get provisioned with GitOps

## Using Namespace Provisioner Controller

Ensure that the following prerequisites are met before provisioning namespace-scoped resources using Namespace Provisioner Controller.

### Prerequisites

- The Namespace Provisioner package is installed and reconciled.
- The `controller tap value key` is set to `true` (Default is `true`).
- The `registry-credentials` secret referenced by the Tanzu Build Service is added to tap-install and exported to all namespaces. If you don't want to export this secret to all namespaces for any reason, you must complete an additional step to create this secret in each namespace you want to provision.
  - Example secret creation, exported to all namespaces

```
tanzu secret registry add tbs-registry-credentials --server REGISTRY-SERVER --username REGISTRY-USERNAME --password REGISTRY-PASSWORD --export-to-all-namespaces --yes --namespace tap-install
```

- Example secret creation for a specific namespace

```
tanzu secret registry add tbs-registry-credentials --server REGISTRY-SERVER --username REGISTRY-USERNAME --password REGISTRY-PASSWORD --yes --namespace YOUR-NEW-DEVELOPER-NAMESPACE
```

## Provision a new developer namespace

Complete the following steps to provision a new developer namespace:

1. Create a namespace using `kubectl` or any other means

```
kubectl create namespace YOUR-NEW-DEVELOPER-NAMESPACE
```

2. Label your new developer namespace with the label selector `apps.tanzu.vmware.com/tap-ns=""`

```
kubectl label namespaces YOUR-NEW-DEVELOPER-NAMESPACE apps.tanzu.vmware.com/tap-ns=""
```

- This label tells the controller to add this namespace to the `desired-namespaces` ConfigMap.
- The label's value can be anything, including `""`.
- If required, you can change the default label selector by configuring the `namespace_selector` property/value in `tap-values` for namespace provisioner.

3. (Optional) This step is only required if the `registry-credentials` secret that was created during Tanzu Application Platform Installation **was not** exported to all namespaces (see the [Prerequisites](#) section above for details).

- Add the `registry-credentials` secret referenced by the Tanzu Build Service to the new namespace and patch the service account that will be used by the workload to refer to this new secret.

```
tanzu secret registry add registry-credentials --server REGISTRY-SERVER --username REGISTRY-USERNAME --password REGISTRY-PASSWORD --yes --namespace YOUR-NEW-DEVELOPER-NAMESPACE
```

4. Run the following command to verify the correct resources were created in the namespace:

```
kubectl get secrets,serviceaccount,rolebinding,pods,workload,configmap -n YOUR-NEW-DEVELOPER-NAMESPACE
```

- To see the list of resources that are provisioned in your namespace based on the installation profile and supply chain values configured in your `tap-values.yaml` file, see [Default resources mapping](#).

## Using GitOps

This section describes how to use GitOps to manage the list of namespaces in the [desired-namespaces ConfigMap](#) instead of the built-in controller.

**WARNING** If there is a namespace in your GitOps repository [desired-namespaces ConfigMap](#) list that does not exist on the cluster, the [provisioner application](#) fails to reconcile and cannot create resources. Creation of the namespaces is out of the scope for the Namespace Provisioner package.

## Prerequisites

The prerequisites for using GitOps are the same as those specified in the [controller prerequisites](#) above except for the `controller` tap value key's value as follows:

- The `controller` tap value key is set to `false` (Default is `true`)

For more information about provisioning namespaces with GitOps, see [Control the desired-namespaces ConfigMap with GitOps](#).

## Customize Namespace Provisioner

This topic describes advanced use cases associated with Namespace Provisioner in Tanzu Application Platform (commonly known as TAP).

## Data values templating guide

Customize your custom resources with data values from:

1. The `tap-values.yaml` file
2. The `desired-namespaces` ConfigMap.

Namespace Provisioner inherits all of the configuration in **both** the [desired-namespaces ConfigMap](#) and the tap config under the key `tap_values` making it available to use as ytt `data.values` when [extending the resources via GitOps](#).

For example, if the [desired-namespaces ConfigMap](#) has a namespace `dev-ns1` with an additional `language: java` parameter, the `data.values` config that is available for templating custom resources is as follows:

```
# data.values map that can be used for templating custom resources
tap_values:
  ...
  supply_chain: testing_scanning
  profile: full
  ootb_delivery_basic:
    service_account: default
  ootb_supply_chain_basic:
    service_account: default
  ootb_supply_chain_testing_scanning:
    scanning:
      image:
        policy: image-scan-policy
      source:
        policy: scan-policy
    service_account: default
```

```

ootb_supply_chain_testing:
  service_account: default

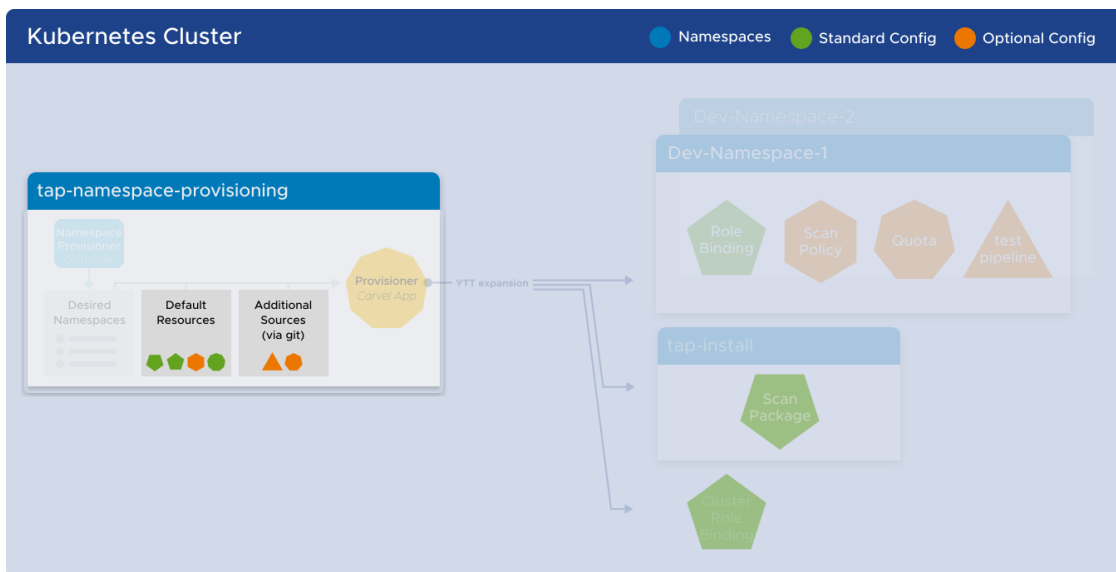
# Everything below this comes from desired-namespaces ConfigMap
name: dev-ns1
# additional parameters about dev-ns1 from desired-namespaces ConfigMap
language: java

```

You can use this config while creating custom resources to extend the default provisioned resources.

Here's a sample of a templated Tekton pipeline.

## GitOps Customizations



## Extending the default provisioned resources

To customize and extend the default configuration for the Namespace Provisioner that is templated in the `default-resources` Secret, add `additional sources` in the `tap-values.yaml` configuration file.

For example, to adjust quota allocation or to create other namespace resources. For details of the list of resources that are templated in the `default-resources` Secret, see [Default Resource Mapping](#).

The following are examples of additional sources:

- This additional source points to an example of a [workload service account yaml file](#). After importing this source, Namespace Provisioner creates the following resources in all namespaces listed in the `desired-namespaces` ConfigMap.
- This additional source points to examples of [ytt templated testing and scanpolicy](#). After importing this source, Namespace Provisioner creates a `scan-policy` and a `developer-defined-tekton-pipeline-java` in all namespaces in the `desired-namespaces` ConfigMap with the default setup in [Install OOTB Supply Chain with Testing and Scanning](#) documentation.
- This additional source points to an example of a [ytt templated scanpolicy yaml file](#). After importing this source, Namespace Provisioner creates a `snyk-scan-policy` in all namespaces in the `desired-namespaces` ConfigMap that has an additional parameter `scanpolicy: snyk`.
- This additional source points to [examples of ytt templated tekton pipelines](#). After importing this source, Namespace Provisioner creates a `developer-defined-tekton-pipeline-python*`



and **developer-defined-tekton-pipeline-angular** for namespaces in the [desired-namespaces ConfigMap](#) that has an additional parameter **language: python** and **language: angular** respectively.

The following example provides a snippet from `tap-values.yaml` with custom configuration for `additional_sources`. Each of the user-generated `namespace_provisioner.additional_sources[].path` values must be unique, and each path must begin with “`_ytt_lib/`” to be identified as a ytt library.

```
namespace_provisioner:
  additional_sources:
    # Add a custom workload service account and some secrets
    - git:
        ref: tap-1.4-np
        subPath: namespace-provisioner-gitops-examples/custom-resources/workload-sa
        url: https://github.com/vmware-tanzu/application-accelerator-samples.git
        path: _ytt_lib/workload-sa
    # Add templated Grype scan policy and java Tekton pipeline
    - git:
        ref: tap-1.4-np
        subPath: namespace-provisioner-gitops-examples/custom-resources/testing-scanning-supplychain
        url: https://github.com/vmware-tanzu/application-accelerator-samples.git
        path: _ytt_lib/testingscanning
    # Add templated snyk scan policy
    - git:
        ref: tap-1.4-np
        subPath: namespace-provisioner-gitops-examples/custom-resources/scanpolicies
        url: https://github.com/vmware-tanzu/application-accelerator-samples.git
        path: _ytt_lib/scanpolicies
    # Add templated tekton pipelines for angular, colang and python based on data.values
    - git:
        ref: tap-1.4-np
        subPath: namespace-provisioner-gitops-examples/custom-resources/tekton-pipelines
        url: https://github.com/vmware-tanzu/application-accelerator-samples.git
        path: _ytt_lib/tektonpipelines
```

## Add the resources required by the Out of the Box Testing and Scanning Supply Chain

Follow these instructions to install the Java scan policy and Tekton pipeline resources required by the OOTB Testing and Scanning Supply Chain.

1. Add or update `tap-values.yaml` with the following `namespace_provisioner.additional_resources` configuration (The ytt templated testing and scanpolicy files that are mounted are [here](#)).

```
namespace_provisioner:
  additional_sources:
    # Add templated java scan policy and tekton pipeline
    - git:
        ref: tap-1.4-np
        subPath: namespace-provisioner-gitops-examples/custom-resources/testing-scanning-supplychain
        url: https://github.com/vmware-tanzu/application-accelerator-samples.git
        path: _ytt_lib/testingscanning # this user-generated path must always begin with "_ytt_lib/"
```

2. Apply your updated `tap-values.yaml` to the target cluster

```
tanzu package installed update tap -f tap-values.yaml --namespace tap-install
```

- o After the tap-values changes are applied and the `namespace_provisioner.additional_resources` are imported, Namespace Provisioner creates the defined `scan-policy` and `developer-defined-tekton-pipeline-java` in all namespaces defined in the `desired-namespaces ConfigMap`.

## Customizing the default resources that get provisioned

Customize the Out-Of-The-box `default-resources` using GitOps with some specific characteristics:

- Use the `ytt overlay` feature for GitOps customization, set in the `tap-values.yaml` under `additional_sources`.
- Mount additional Git resource in the path `_ytt_lib/customize`, otherwise the customization is not applied. Namespace Provisioner mounts all the additional sources as a `ytt library` so it can expand the manifests in the additional sources for all managed namespaces using the logic in the expansion template.
- The GitOps repository directory must have a file with an extension `lib.yaml` to be recognized as a ytt library with members to be exported.
- The library file in the GitOps repository directory must have a function called `customize` with the overlays to be applied to the resources, it can contain one or more overlays.

The sample file `sa-secrets.lib.yaml` shows how to completely override the `secrets` and `imagePullSecrets` sections of the default ServiceAccount to add custom created secrets by using other additional resources.

Sample tap-values change to pull this ytt customization overlay:

```
namespace_provisioner:
  additional_sources:
    # Patches the OOTB default service account to add different secrets
    - git:
        ref: tap-1.4-np
        subPath: namespace-provisioner-gitops-examples/default-resources-overrides/overl
ays
        url: https://github.com/vmware-tanzu/application-accelerator-samples.git
        path: _ytt_lib/customize # this path must always be exactly "_ytt_lib/customize"
    # Adds the secrets referenced in the overlay
    - git:
        ref: tap-1.4-np
        subPath: namespace-provisioner-gitops-examples/custom-resources/workload-sa
        url: https://github.com/vmware-tanzu/application-accelerator-samples.git
        path: _ytt_lib/workload-sa # this user-generated path must always begin with "_y
tt_lib/"
```

Sample customization (`.lib.yaml`) file for overriding the `secrets` and `imagePullSecrets` of the default ServiceAccount - [Link to the Sample file](#)

```
#@ load("@ytt:overlay", "overlay")
#@ def customize():

#@overlay/match by=overlay.subset({"apiVersion": "v1", "kind": "ServiceAccount", "metad
ata":{"name":"default"}}), expects="0+"
---
secrets:
  - name: gitlab-workload-token
  - name: github-workload-token
  - name: registries-credentials
imagePullSecrets:
  - name: gitlab-workload-token
```

```
- name: github-workload-token
- name: registries-credentials
#@ end
```

## Control the Namespace Provisioner reconcile behavior for specific resources

There are certain OOTB [default-resources](#) like the [ServiceAccount](#) that are annotated with a special annotation [namespace-provisioner.apps.tanzu.vmware.com/no-overwrite](#).

Any changes to the resources that have the [.no-overwrite](#) annotation are not overwritten by the [provisioner application](#) that controls resource provision. To restore the default state of those resources, you can delete them and the [provisioner application](#) re-creates them in their initial default state.

The [provisioner application](#) has a synchronization interval of 10 minutes. To manually force the reconciliation of the resources, for example, to delete a resource so that it can be re-created into its default initial state, use the [Carvel kctrl](#) CLI to “kick” the [provisioner application](#) reconciliation.

Run the following command to initiate the “kick”:

```
kctrl app kick --app provisioner -n tap-namespace-provisioning -y
```

## Control the [desired-namespaces](#) ConfigMap with GitOps

You can maintain the [desired-namespaces](#) ConfigMap in your Git repository instead of using the [controller](#). You can use the GitOps tool of your choice to override the [desired-namespaces ConfigMap](#) in the [tap-namespace-provisioning](#) namespace.

### Prerequisites

Ensure that the following prerequisites are met:

- The Namespace Provisioner package is installed.
- [controller](#) must be set to “false”. If the [controller](#) is set to “true”, it overwrites the declarative desired state configured in your GitOps repository.
- The registry-credentials secret referred by the Tanzu Build Service is added to [tap-install.yaml](#) and exported to all namespaces. If you don’t want to export this secret to all namespaces for any reason, you must complete an additional step to create this secret in the namespace.

Use the following snippet as a reference for the [desired-namespaces ConfigMap](#) that you can put on your Git repository. [Desired-namespaces.yaml](#) ([Link to sample repo file](#))

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: desired-namespaces
  namespace: tap-namespace-provisioning
  annotations:
    kapp.k14s.io/create-strategy: fallback-on-update
    namespace-provisioner.apps.tanzu.vmware.com/no-overwrite: "" #! This annotation tells the provisioner app to not override this configMap as this is your desired state.
data:
  namespaces.yaml: |
    #@data/values
    ---
    namespaces:
```

```

- name: python-backend-app
  language: python
  scanpolicy: snyk
- name: angular-fe-app
  language: angular
- name: golang-opts
  language: golang

```

The recommended approach is to maintain a list of namespace objects in your GitOps repository and use the GitOps tool of your choice to create namespaces in the cluster and the provisioner application populates it with the appropriate resources.

The following command uses Kubectl to override this [desired-namespaces](#) ConfigMap manually. The ConfigMap can be overridden with your tool of choice.

```

kubectl apply -f https://raw.githubusercontent.com/vmware-tanzu/application-accelerator-samples/tap-1.4-np/namespace-provisioner-gitops-examples/desired-namespaces/gitops-managed-desired-namespaces.yaml

```

When this change is applied, the [provisioner application](#) starts the reconcile process and provisions the resources on the given namespaces.

**WARNING:** If there is a namespace in your GitOps repository [desired-namespaces](#) ConfigMap list that does not exist on the cluster, the [provisioner](#) application fails to reconcile and cannot create resources. Creating namespaces is out of scope for the Namespace Provisioner package.

## Troubleshoot Namespace Provisioner

This topic provides information to help troubleshoot Namespace Provisioner.

### Controller logs

To get the logs when using the [controller](#) to manage the [desired-namespaces](#) ConfigMap

```

kubectl -n tap-namespace-provisioning logs deployments/controller-manager

```

Use `-f` to follow the log output

### Provisioner application error

After the Namespace Provisioner is installed in the Tanzu Application Platform cluster, the main resource to check is the [provisioner application](#) in the `tap-namespace-provisioning` namespace.

```

kubectl -n tap-namespace-provisioning get app/provisioner --template={{.status.usefulErrorMessage}}

```

For information about why certain resources are not reconciled automatically, and as a result, might need some manual configuration, see [Control the Namespace Provisioner reconcile behavior for specific resources](#).



#### Note

Any error with the [provisioner application](#) is reported in the Carvel Package Install as a high level message.

```
kubectl -n tap-install get packageinstalls.packaging.carvel.dev/namespace-provisioner
--template={{.status.usefulErrorMessage}}
```

## Common errors

You might encounter one of the following errors:

### Namespace selector malformed

When using the [controller](#) to manage the [desired-namespaces ConfigMap](#) and customizing the [namespace\\_selector](#) from [tap\\_values.yaml](#), the match expression must be compliant with the [Kubernetes label selector](#). If it is not compliant when labeling a namespace, the Namespace Provisioner doesn't create any object in the [desired-namespaces ConfigMap](#) and the controller outputs a [log](#) message.

For example, if the configured [namespace\\_selector](#) is

```
namespace_provisioner:
  namespace_selector:
    matchExpressions:
      - key: apps.tanzu.vmware.com/tap-ns
        operator: exists
```

This is not compliant as the operator must be `Exist` instead of `exists`, then when labeling the namespace `ns2` with `apps.tanzu.vmware.com/tap-ns`, the [controller](#) produces an error message similar to the following, (followed by some reconciliation messages)

```
{"level":"error","ts":"2022-12-14T15:41:44.639402794Z","logger":".0.1.NamespaceSelectorReconciler","msg":"unable to sync","controller":"namespace","controllerGroup":"","controllerKind":"Namespace","Namespace":{"name":"ns2"},"namespace":"","name":"ns2","reconcileID":"26395d34-418b-446d-9b5e-a4a73cc657ed","resourceType":"/v1, Kind=Namespace","error":"\"exists\" is not a valid pod selector operator","stacktrace":"..."}
```



#### Note

The [provisioner application](#) won't show an error as the controller cannot update the [desired-namespaces ConfigMap](#).

### Carvel-YTT error in additional\_sources

When working with `ytt`, it is easy to mistakenly miswrite the template, as a result, the Namespace Provisioner fails when the [additional\\_sources](#) is provided with errors. To check the problem in the [provisioner application](#), see [Provisioner application error](#).

For example, the following file is used as [additional\\_sources](#)

```
## load("@ytt:data", "data")
---
apiVersion: v1
kind: Secret
metadata:
  name: gitlab-workload-token
  annotations:
    tekton.dev/git-0: https://git.company.com
type: kubernetes.io/basic-auth
stringData:
  #! data.values does not have any key of gl-secret-user nor gl-secret-pass
  username: #@ data.values.gl-secret-user
```

```

password: #@ data.values.gl-secret-pass
---
apiVersion: v1
kind: Secret
metadata:
  name: github-workload-token
  annotations:
    tekton.dev/git-0: https://github.aetna.com
type: kubernetes.io/basic-auth
stringData:
  #! data.values does not have any key of gh-secret-user nor gh-secret-pass
  username: #@ data.values.gh-secret-user
  password: #@ data.values.gh-secret-pass
---

```

Where the used `data.values` does not exist, and after adding it as an `additional_source` in the `witherror` library, the `provisioner` application shows an error as follows:

```

$ kubectl -n tap-namespace-provisioning get app/provisioner --template={{.status.usefulErrorMessage}}

ytt: Error:
- library.eval: Evaluating library 'witherror':
  in <tolevel>
    template.yaml:155 | #@           instances = overlay.apply(instance.eval(), customize())

    reason:
    - struct has no .gl_secret_user field or method
      in <tolevel>
        _ytt_lib/witherror/secrets.yaml:12 | username: #@ data.values.gl_secret_u
ser

```

This shows any error coming from the Carvel-YTT template resolution.

Another common error is defining resources several times (like adding a resource that is created by default instead of overlaying it), this is reported in the `provisioner` application also.

## Unable to delete namespace

When a namespace is provisioned and listed in the `desired-namespaces ConfigMap` (with `controller` or `GitOps`), delete it using the `kubectl delete namespace` command.

It must be *cleared* of workload before being deleted. When the provisioned namespace is deleted and there is a workload already created in the namespace, the `namespace` will likely remain in the `Terminating` state because some resources can not be deleted.

One of the causes of this behavior is that the workload creates a Carvel Kapp App that references the `ServiceAccount` in the namespace. `Kubectl` does not adhere to Carvel `kapp-controller` delete order and the `ServiceAccount` is deleted before the workload Carvel App. As a result, the Carvel App blocks the namespace termination while waiting for the `ServiceAccount` to exist with a `finalizer (finalizers.kapp-ctrl.k14s.io/delete)` message.

**Solution:** Remove the Kapp App `finalizer` in the Kapp App

- Another possible cause is when you use the `controller` to manage the `desired-namespaces ConfigMap` and it fails to remove the custom `finalizer` added to the namespace (`namespace-provisioner.apps.tanzu.vmware.com/finalizer`)

**Solution:** Remove the `finalizer` in the `namespace`

## Namespace Provisioner Reference Guide

This topic tells you about the known limitations and default resource mapping for the Namespace Provisioner component in Tanzu Application Platform (commonly known as TAP).

## Known Limitations

- If there is a namespace in your GitOp's repository [desired-namespaces](#) ConfigMap that does not exist on the cluster, the `provisioner` application fails to reconcile and cannot create resources.
- The creation of the namespaces is out of scope for Namespace Provisioner.
- Removing the Namespace Provisioner package removes all the components created by it.
- Before uninstalling the Namespace Provisioner, you must:
  - If you are using the `controller` to manage [desired-namespaces](#) ConfigMap, un-label all the namespaces provisioned by Namespace Provisioner
  - If you are using GitOps to manage [desired-namespaces](#) ConfigMap, set the list of namespaces to an empty list.
- The namespace selector label to provision resources cannot be applied to the developer namespace which is configured at deployment time under the Grype package values as it causes the provisioner application to fail due to ownership issues.

## Default resources mapping

Namespace Provisioner is installed as part of the [standard installation profiles](#). The default set of resources provisioned in a namespace is based on a combination of the installation profile and the supply chain that is installed on the cluster. The following table shows the list of resources that are templated in the `default-resources` Secret for an installation profile and supply chain value combination:

Namespace	Kind	Name	supply_chain	Install Profile	Reconcile
tap-install	PackageInstal	grype-scanner-(ns)	testing_scanning	full, build	Yes
tap-install	Secret	grype-scanner-(ns)	testing_scanning	full, build	Yes
Developer Namespace	Secret	registries-credentials	n/a	full, iterate, build, run	Yes
Developer Namespace	ServiceAccount	From: ootb_supply_chain_{supply_chain}.service_account (default: "default")	n/a	full, iterate, build, run	No
Developer Namespace	ServiceAccount	From: ootb_delivery_basic.service_account (default: "default")	n/a	full, iterate, run	No
Developer Namespace	RoleBinding	default-permit-deliverable	n/a	full, iterate, run	Yes
Developer Namespace	RoleBinding	default-permit-workload	n/a	full, iterate, build	Yes



### Note

For Install OOTB Supply Chain with Testing and Scanning, see [Extending the default provisioned resources](#).

## Overview of Spring Boot conventions

This topic tells you about the Spring Boot convention server.

The Spring Boot convention server is a bundle of small conventions applied to any Spring Boot application that is submitted to the supply chain in which the convention controller is configured.

Run the `docker inspect` command to make the Spring Boot convention server look inside the image. Example command:

```
$ docker inspect springio/petclinic
```

Example output:

```
[
  {
    "Id": "sha256:...",
    "RepoTags": [
      "springio/petclinic:latest"
    ],
    "RepoDigests": [
      "springio/petclinic@sha256:..."
    ],
    "Parent": "",
    "Container": "",
    ...
    "ContainerConfig": {
      "Hostname": "",
      "Domainname": "",
      "User": "",
      ...
      "Labels": null
    },
    "DockerVersion": "",
    "Author": "",
    "Config": {
      ...
    }
  }
]
```

The convention server searches inside the image for `Config -> Labels -> io.buildpacks.build.metadata` to find the `bom` file. It looks inside the `bom` file for metadata to evaluate whether the convention is to be applied.

For the list of conventions, see [Conventions](#).

## Overview of Spring Boot conventions

This topic tells you about the Spring Boot convention server.

The Spring Boot convention server is a bundle of small conventions applied to any Spring Boot application that is submitted to the supply chain in which the convention controller is configured.

Run the `docker inspect` command to make the Spring Boot convention server look inside the image. Example command:

```
$ docker inspect springio/petclinic
```

Example output:

```
[
  {
    "Id": "sha256:...",
```



```

    "RepoTags": [
      "springio/petclinic:latest"
    ],
    "RepoDigests": [
      "springio/petclinic@sha256:..."
    ],
    "Parent": "",
    "Container": "",
    ...
    "ContainerConfig": {
      "Hostname": "",
      "Domainname": "",
      "User": "",
      ...
      "Labels": null
    },
    "DockerVersion": "",
    "Author": "",
    "Config": {
      ...
    }
  ]
}

```

The convention server searches inside the image for `Config -> Labels -> io.buildpacks.build.metadata` to find the `bom` file. It looks inside the `bom` file for metadata to evaluate whether the convention is to be applied.

For the list of conventions, see [Conventions](#).

## Install Spring Boot conventions

This topic tells you how to install Spring Boot conventions from the Tanzu Application Platform package repository.



### Note

Follow the steps in this topic if you do not want to use a profile to install Spring Boot conventions. For more information about profiles, see [Components and installation profiles](#).

## Prerequisites

Before installing Spring Boot conventions:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see [Prerequisites](#).
- Install [Supply Chain Choreographer](#).

## Install Spring Boot conventions

To install Spring Boot conventions:

1. Get the exact name and version information for the Spring Boot conventions package to install by running:

```
tanzu package available list spring-boot-conventions.tanzu.vmware.com --namespace tap-install
```

For example:

```
$ tanzu package available list spring-boot-conventions.tanzu.vmware.com --namespace tap-install
/ Retrieving package versions for spring-boot-conventions.tanzu.vmware.com...
NAME                                VERSION                                RELEASED-AT
...
spring-boot-conventions.tanzu.vmware.com  1.4.0                                2022-12-08T00:00:00Z
...
```

- (Optional) Change the default installation settings by running:

```
tanzu package available get spring-boot-conventions.tanzu.vmware.com/VERSION-NUMBER \
--values-schema --namespace tap-install
```

Where **VERSION-NUMBER** is the version of the package listed. For example: **1.4.13**.

For example:

```
$ tanzu package available get spring-boot-conventions.tanzu.vmware.com/1.4.0 --values-schema --namespace tap-install
KEY                                DEFAULT                                TYPE                                DESCRIPTION
autoConfigureActuators             false                                 boolean                            Enable or disable the automatic configuration of actuators on the TAP platform level
kubernetes_distribution             string                               Kubernetes distribution that this package is being installed on. Accepted
values:
kubernetes_version                 string                               Optional: The Kubernetes Version. Valid values are '1.24.*', or ''
```

- Install the package by running:

```
tanzu package install spring-boot-conventions \
--package-name spring-boot-conventions.tanzu.vmware.com \
--version 1.4.13 \
--namespace tap-install
```

- Verify you installed the package by running:

```
tanzu package installed get spring-boot-conventions --namespace tap-install
```

For example:

```
tanzu package installed get spring-boot-conventions -n tap-install
| Retrieving installation details for spring-boot-conventions...
NAME:                                spring-boot-conventions
PACKAGE-NAME:                        spring-boot-conventions.tanzu.vmware.com
PACKAGE-VERSION:                    1.4.13
STATUS:                              Reconcile succeeded
CONDITIONS:                          [{ReconcileSucceeded True }]
USEFUL-ERROR-MESSAGE:
```

Verify that **STATUS** is **Reconcile succeeded**

## Configure and access Spring Boot actuators in Tanzu Application Platform

This topic tells you how the Spring Boot conventions in Tanzu Application Platform configure Spring Boot actuators automatically. With this feature, users can activate or deactivate the automatic configuration of actuators on Tanzu Application Platform and on individual workloads.

## Workload-level configuration

Developers can add a label to their workloads to activate or deactivate the automatic configuration of actuators. By default, all existing and future accelerator projects are configured to activate automatic configuration on the workload level.

To activate or deactivate the automatic configuration of actuators at the workload level, follow these steps:

1. To activate automatic configuration of actuators, set the following label to `true` in your workload YAML:

```
apps.tanzu.vmware.com/auto-configure-actuators: "true"
```

If the preceding label is set to `true`, the Spring Boot actuator convention sets the following actuator configuration:

- The `JAVA_TOOL_OPTIONS` property is set as `-Dmanagement.server.port="8081"`.
- The `JAVA_TOOL_OPTIONS` property is set as `-Dmanagement.endpoints.web.base-path="/actuator"`.
- Annotation on the PodIntent is set as `boot.spring.io/actuator: http://:8081/actuator`.

In addition to these settings, Application Live View is activated with the following actuator configuration:

- Label on the PodIntent is set as `tanzu.app.live.view.application.actuator: actuator`.
- Label on the PodIntent is set as `tanzu.app.live.view.application.actuator.port: 8081`.

2. To deactivate automatic configuration of actuators, set the following label to `false` in your workload YAML:

```
apps.tanzu.vmware.com/auto-configure-actuators: "false"
```

If the preceding label is set to `false`, the Spring Boot actuator convention does not set any `JAVA_TOOL_OPTIONS` and does not set the annotation `boot.spring.io/actuator`.

Application Live View is activated and configured with default values for Spring Boot web applications, assuming that some actuators are activated on the default port. On activating Application Live View, the following actuator settings are set:

- The `JAVA_TOOL_OPTIONS` property is set as `-Dserver.port="8080"`.
- Label on the PodIntent is set as `tanzu.app.live.view.application.actuator: actuator`.
- Label on the PodIntent is set as `tanzu.app.live.view.application.actuator.port: 8080`.

The Application Live View GUI renders the pages with accessible information based on whether the actuator endpoints are accessible for an application.

By default, as an additional security measure, Spring Boot conventions does not expose all the actuator data over HTTP by exposing all the actuator endpoints. In addition, the information exposed by the health endpoint is not set to `always` by default.

If the automatic configuration of actuators is set to `true` either at the workload level or platform level, the Spring Boot convention then sets the runtime environment properties `management.endpoints.web.exposure.include="*" and management.endpoint.health.show-details=true on to the PodSpec to expose all the actuator endpoints and detailed health`

information. You do not need to add these properties manually in `application.properties` or `application.yml`.

## Platform-level configuration

In contrast to activating or deactivating the automatic configuration of actuators on the level of individual workloads, you can set a global setting for the platform instead. This setting is taken into account **ONLY** when there is no specific `auto-configure-actuators` setting on the individual workload.

To activate or deactivate the automatic configuration of actuators at a global level, follow these steps:

1. When you install Spring Boot conventions, you can provide an entry in the `values.yaml` file to activate automatic configuration. For example:

```
springboot_conventions:
  autoConfigureActuators: true
```

2. To deactivate the automatic configuration, you can provide the following entry:

```
springboot_conventions:
  autoConfigureActuators: false
```



### Note

The default values for both platform level and workload level configuration is false.

To run Application Live View with Spring Boot apps, the Spring Boot convention recognizes PodIntents and adds the following metadata labels:

- `tanzu.app.live.view: "true"`: Activates the connector to observe application pod
- `tanzu.app.live.view.application.name: APPLICATION-NAME`: Identifies the app name to be used internally by Application Live View
- `tanzu.app.live.view.application.actuator.port: ACTUATOR-PORT`: Identifies the port on the pod at which the actuators are available for Application Live View
- `tanzu.app.live.view.application.flavours: spring-boot`: Exposes the framework flavor of the app

To run Application Live View with Spring Cloud Gateway apps, Spring Boot conventions recognizes PodIntents and adds the following metadata labels:

- `tanzu.app.live.view: "true"`: Activates the connector to observe application pod
- `tanzu.app.live.view.application.name: APPLICATION-NAME`: Identifies the app name to be used internally by Application Live View
- `tanzu.app.live.view.application.actuator.port: ACTUATOR-PORT`: Identifies the port on the pod at which the actuators are available for Application Live View
- `tanzu.app.live.view.application.flavours: spring-boot, spring-cloud-gateway`: Exposes the framework flavors of the app

## Enable Application Live View for Spring Boot applications

To run Application Live View for Spring Boot apps, Spring Boot conventions recognizes PodIntents and automatically adds the following metadata labels:

- `tanzu.app.live.view: "true"`: Enables the connector to observe application pod
- `tanzu.app.live.view.application.name: APPLICATION-NAME`: Identifies the app name to be used internally by Application Live View
- `tanzu.app.live.view.application.actuator.port: ACTUATOR-PORT`: Identifies the port on the pod at which the actuators are available for Application Live View
- `tanzu.app.live.view.application.flavours: spring-boot`: Exposes the framework flavor of the app

To run Application Live View for Spring Cloud Gateway apps, Spring Boot conventions recognizes PodIntents and adds the following metadata labels:

- `tanzu.app.live.view: "true"`: Enables the connector to observe application pod
- `tanzu.app.live.view.application.name: APPLICATION-NAME`: Identifies the app name to be used internally by Application Live View
- `tanzu.app.live.view.application.actuator.port: ACTUATOR-PORT`: Identifies the port on the pod at which the actuators are available for Application Live View
- `tanzu.app.live.view.application.flavours: spring-boot, spring-cloud-gateway`: Exposes the framework flavors of the app

These metadata labels allow Application Live View to identify pods that are enabled for Application Live View. The metadata labels also tell the Application Live View connector what kind of app it is and on which port the actuators are accessible for Application Live View. For more information, see [Configuring and accessing Spring Boot actuators in Tanzu Application Platform](#).

## Verify the applied labels and annotations

To verify the applied labels and annotations, run:

```
kubectl get podintents.conventions.carto.run WORKLOAD-NAME -o yaml
```

Where `WORKLOAD-NAME` is the name of the deployed workload. For example: `tanzu-java-web-app`.

Expected output of Spring Boot workload:

```
apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
  creationTimestamp: "2022-11-14T10:07:55Z"
  generation: 1
  labels:
    app.kubernetes.io/component: intent
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/auto-configure-actuators: "true"
    apps.tanzu.vmware.com/workload-type: web
    carto.run/cluster-template-name: convention-template
    carto.run/resource-name: config-provider
    carto.run/supply-chain-name: source-to-url
    carto.run/template-kind: ClusterConfigTemplate
    carto.run/workload-name: tanzu-java-web-app
    carto.run/workload-namespace: default
name: tanzu-java-web-app
namespace: default
ownerReferences:
- apiVersion: carto.run/v1alpha1
  blockOwnerDeletion: true
```

```

controller: true
kind: Workload
name: tanzu-java-web-app
uid: dfd3c0c2-9d1f-4231-9390-3e16f23bb62d
resourceVersion: "444497"
uid: 224de2aa-307a-48e3-a826-2c474c435bb2
spec:
  serviceAccountName: default
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/min-scale: "1"
        developer.conventions/target-containers: workload
      labels:
        app.kubernetes.io/component: run
        app.kubernetes.io/part-of: tanzu-java-web-app
        apps.tanzu.vmware.com/auto-configure-actuators: "true"
        apps.tanzu.vmware.com/workload-type: web
        carto.run/workload-name: tanzu-java-web-app
    spec:
      containers:
        - image: dev.registry.tanzu.vmware.com/app-live-view/test/tanzu-java-web-app-def
          name: workload
          resources: {}
          securityContext:
            runAsUser: 1000
          serviceAccountName: default
status:
  conditions:
  - lastTransitionTime: "2022-11-14T10:07:59Z"
    message: ""
    reason: Applied
    status: "True"
    type: ConventionsApplied
  - lastTransitionTime: "2022-11-14T10:07:59Z"
    message: ""
    reason: ConventionsApplied
    status: "True"
    type: Ready
observedGeneration: 1
template:
  metadata:
    annotations:
      autoscaling.knative.dev/min-scale: "1"
      boot.spring.io/actuator: http://:8081/actuator
      boot.spring.io/version: 2.7.3
      conventions.carto.run/applied-conventions: |-
        spring-boot-convention/auto-configure-actuators-check
        spring-boot-convention/spring-boot
        spring-boot-convention/spring-boot-graceful-shutdown
        spring-boot-convention/spring-boot-web
        spring-boot-convention/spring-boot-actuator
        spring-boot-convention/spring-boot-actuator-probes
        spring-boot-convention/app-live-view-appflavour-check
        spring-boot-convention/app-live-view-connector-boot
        spring-boot-convention/app-live-view-appflavours-boot
        appliveview-sample/app-live-view-appflavour-check
      developer.conventions/target-containers: workload
    labels:
      app.kubernetes.io/component: run
      app.kubernetes.io/part-of: tanzu-java-web-app
      apps.tanzu.vmware.com/auto-configure-actuators: "true"
      apps.tanzu.vmware.com/workload-type: web
      carto.run/workload-name: tanzu-java-web-app
      conventions.carto.run/framework: spring-boot

```

```

tanzu.app.live.view: "true"
tanzu.app.live.view.application.actuator.path: actuator
tanzu.app.live.view.application.actuator.port: "8081"
tanzu.app.live.view.application.flavours: spring-boot
tanzu.app.live.view.application.name: tanzu-java-web-app
spec:
  containers:
  - env:
    - name: JAVA_TOOL_OPTIONS
      value: -Dmanagement.endpoint.health.probes.add-additional-paths="true" -Dmanagement.endpoint.health.show-details="always"
        -Dmanagement.endpoints.web.base-path="/actuator" -Dmanagement.endpoints.web.exposure.include="*"
        -Dmanagement.health.probes.enabled="true" -Dmanagement.server.port="8081"
        -Dserver.port="8080" -Dserver.shutdown.grace-period="24s"
      image: dev.registry.tanzu.vmware.com/app-live-view/test/tanzu-java-web-app-default@sha256:444686bb8bfbaba5552676140619b00f43c8f85b6823b87676c0ccdcdead65ac
      livenessProbe:
        httpGet:
          path: /livez
          port: 8080
          scheme: HTTP
      name: workload
      ports:
      - containerPort: 8080
        protocol: TCP
      readinessProbe:
        httpGet:
          path: /readyz
          port: 8080
          scheme: HTTP
      resources: {}
      securityContext:
        runAsUser: 1000
      serviceAccountName: default

```

#### Expected output of Spring Cloud Gateway workload:

```

apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
  creationTimestamp: "2022-11-14T10:29:51Z"
  generation: 1
  labels:
    app.kubernetes.io/component: intent
    app.kubernetes.io/part-of: tanzu-scg-web-app
    apps.tanzu.vmware.com/auto-configure-actuators: "true"
    apps.tanzu.vmware.com/workload-type: web
    carto.run/cluster-template-name: convention-template
    carto.run/resource-name: config-provider
    carto.run/supply-chain-name: source-to-url
    carto.run/template-kind: ClusterConfigTemplate
    carto.run/workload-name: tanzu-scg-web-app
    carto.run/workload-namespace: default
  name: tanzu-scg-web-app
  namespace: default
  ownerReferences:
  - apiVersion: carto.run/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: Workload
    name: tanzu-scg-web-app
    uid: 5d8cdc5b-0236-471d-8c1e-335e659f1ae6
  resourceVersion: "475756"
  uid: d086f02c-6ff0-47f8-8dee-4da8748d8adc
spec:

```

```

serviceAccountName: default
template:
  metadata:
    annotations:
      autoscaling.knative.dev/min-scale: "1"
      developer.conventions/target-containers: workload
    labels:
      app.kubernetes.io/component: run
      app.kubernetes.io/part-of: tanzu-scg-web-app
      apps.tanzu.vmware.com/auto-configure-actuators: "true"
      apps.tanzu.vmware.com/workload-type: web
      carto.run/workload-name: tanzu-scg-web-app
  spec:
    containers:
      - image: dev.registry.tanzu.vmware.com/app-live-view/test/tanzu-scg-web-app-default@sha256:7656f4ca56b7d0d6376b374643d6ac09c8cdcdcbcc13d065f9224651b12724d0b
        name: workload
        resources: {}
        securityContext:
          runAsUser: 1000
        serviceAccountName: default
status:
  conditions:
  - lastTransitionTime: "2022-11-14T10:29:58Z"
    message: ""
    reason: Applied
    status: "True"
    type: ConventionsApplied
  - lastTransitionTime: "2022-11-14T10:29:58Z"
    message: ""
    reason: ConventionsApplied
    status: "True"
    type: Ready
observedGeneration: 1
template:
  metadata:
    annotations:
      autoscaling.knative.dev/min-scale: "1"
      boot.spring.io/actuator: http://:8081/actuator
      boot.spring.io/version: 2.6.3
      conventions.carto.run/applied-conventions: |-
        spring-boot-convention/auto-configure-actuators-check
        spring-boot-convention/spring-boot
        spring-boot-convention/spring-boot-web
        spring-boot-convention/spring-boot-actuator
        spring-boot-convention/spring-boot-actuator-probes
        spring-boot-convention/app-live-view-appflavour-check
        spring-boot-convention/app-live-view-connector-boot
        spring-boot-convention/app-live-view-appflavours-boot
        spring-boot-convention/app-live-view-connector-scg
        spring-boot-convention/app-live-view-appflavours-scg
        appliveview-sample/app-live-view-appflavour-check
      developer.conventions/target-containers: workload
    labels:
      app.kubernetes.io/component: run
      app.kubernetes.io/part-of: tanzu-scg-web-app
      apps.tanzu.vmware.com/auto-configure-actuators: "true"
      apps.tanzu.vmware.com/workload-type: web
      carto.run/workload-name: tanzu-scg-web-app
      conventions.carto.run/framework: spring-boot
      tanzu.app.live.view: "true"
      tanzu.app.live.view.application.actuator.path: actuator
      tanzu.app.live.view.application.actuator.port: "8081"
      tanzu.app.live.view.application.flavours: spring-boot_spring-cloud-gateway
      tanzu.app.live.view.application.name: tanzu-scg-web-app
  spec:

```



```

containers:
  - env:
    - name: JAVA_TOOL_OPTIONS
      value: -Dmanagement.endpoint.health.probes.add-additional-paths="true" -Dmanagement.endpoint.health.show-details="always"
        -Dmanagement.endpoints.web.base-path="/actuator" -Dmanagement.endpoints.web.exposure.include="*"
        -Dmanagement.health.probes.enabled="true" -Dmanagement.server.port="8081"
        -Dserver.port="8080"
      image: dev.registry.tanzu.vmware.com/app-live-view/test/tanzu-scg-web-app-default@sha256:7656f4ca56b7d0d6376b374643d6ac09c8cdcdcc13d065f9224651b12724d0b
      livenessProbe:
        httpGet:
          path: /livez
          port: 8080
          scheme: HTTP
      name: workload
      ports:
        - containerPort: 8080
          protocol: TCP
      readinessProbe:
        httpGet:
          path: /readyz
          port: 8080
          scheme: HTTP
      resources: {}
      securityContext:
        runAsUser: 1000
      serviceAccountName: default

```

## List of Spring Boot conventions

This topic tells you about what the conventions do and how to apply them.

When submitting the following pod [Pod Intent](#) on each convention, the output can change depending on the applied convention.

Before any Spring Boot conventions are applied, the pod intent looks similar to this YAML:

```

apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
  name: spring-sample
spec:
  template:
    spec:
      containers:
        - name: workload
          image: springio/petclinic

```

Most of the Spring Boot conventions either edit or add properties to the environment variable [JAVA\\_TOOL\\_OPTIONS](#). You can override those conventions by providing the [JAVA\\_TOOL\\_OPTIONS](#) value you want through the Tanzu CLI or [workload.yaml](#).

When a [JAVA\\_TOOL\\_OPTIONS](#) property already exists for a workload, the convention uses the existing value rather than the value that the convention applies by default. The property value that you provide is used for the pod specification mutation.

## Set a [JAVA\\_TOOL\\_OPTIONS](#) property for a workload

Do one of the following actions to set [JAVA\\_TOOL\\_OPTIONS](#) property and values:

### Use the Tanzu CLI apps plug-in

When creating or updating a workload, set a `JAVA_TOOL_OPTIONS` property using the `--env` flag by running:

```
tanzu apps workload create APP-NAME --env JAVA_TOOL_OPTIONS="-DPROPERTY-NAME=VALUE"
```

For example, to set the management port to `8080` rather than the `spring-boot-actuator-convention` default port `8081`, run:

```
tanzu apps workload create APP-NAME --env JAVA_TOOL_OPTIONS="-Dmanagement.server.port=8080"
```

### Use workload.yaml

Follow these steps:

1. Provide one or more values for the `JAVA_TOOL_OPTIONS` property in the `workload.yaml`.

For example:

```
apiVersion: carto.run/v1alpha1
kind: Workload
...
spec:
env:
- name: JAVA_TOOL_OPTIONS
  value: -Dmanagement.server.port=8082
source:
...

```

2. Apply the `workload.yaml` file by running the command:

```
tanzu apps workload create -f workload.yaml
```

## Spring Boot convention

If the `spring-boot` dependency is in the metadata within the `SBOM` file under `dependencies`, the Spring Boot convention is applied to the `PodTemplateSpec` object.

The Spring Boot convention adds a label (`conventions.carto.run/framework: spring-boot`) to the `PodTemplateSpec` that describes the framework associated with the workload, and adds an annotation (`boot.spring.io/version: VERSION-NO`) that describes the Spring Boot version of the dependency.

The label and annotation are added for informational purposes only.

Example of `PodIntent` after applying the convention:

```
apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
  annotations:
    kubectrl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"conventions.carto.run/v1alpha1","kind":"PodIntent","metadata":{"an
notations":{},"name":"spring-sample","namespace":"default"},"spec":{"template":{"spe
c":{"containers":[{"image":"springio/petclinic","name":"workload"}]}}}
...
status:
  conditions:
```

```

- lastTransitionTime: "... " # This status indicates that all worked as expected
  status: "True"
  type: ConventionsApplied
- lastTransitionTime: "... "
  status: "True"
  type: Ready
observedGeneration: 1
template:
  metadata:
    annotations:
      boot.spring.io/version: 2.3.3.RELEASE
      conventions.carto.run/applied-conventions: |-
        spring-boot-convention/spring-boot
    labels:
      conventions.carto.run/framework: spring-boot
  spec:
    containers:
      - image: index.docker.io/springio/petclinic@sha256:...
        name: workload
        resources: {}

```

## Spring boot graceful shut down convention

If any of the following dependencies are in the metadata within the SBOM file under `dependencies`, the Spring Boot graceful shut down convention is applied to the `PodTemplateSpec` object:

- `spring-boot-starter-tomcat`
- `spring-boot-starter-jetty`
- `spring-boot-starter-reactor-netty`
- `spring-boot-starter-undertow`
- `tomcat-embed-core`

The Graceful Shutdown convention `spring-boot-graceful-shutdown` adds a property in the environment variable `JAVA_TOOL_OPTIONS` with the key `server.shutdown.grace-period`. The key value is calculated to be 80% of the value set in `.target.Spec.TerminationGracePeriodSeconds`. The default value for `.target.Spec.TerminationGracePeriodSeconds` is 30 seconds.

Example of PodIntent after applying the convention:

```

apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"conventions.carto.run/v1alpha1","kind":"PodIntent","metadata":{"annotations":{},"name":"spring-sample","namespace":"default"},"spec":{"template":{"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.carto.run/applied-conventions: |-
    spring-boot-convention/spring-boot
    spring-boot-convention/spring-boot-graceful-shutdown
labels:
  conventions.carto.run/framework: spring-boot
spec:
  containers:
  - env:
    - name: JAVA_TOOL_OPTIONS
      value: -Dserver.shutdown.grace-period="24s"
    image: index.docker.io/springio/petclinic@sha256:...
    name: workload
    resources: {}

```

## Spring Boot web convention

If any of the following dependencies are in the metadata within the SBOM file under `dependencies`, the Spring Boot web convention is applied to the `PodTemplateSpec` object:

- `spring-boot`
- `spring-boot-web`

The web convention `spring-boot-web` obtains the `server.port` property from the `JAVA_TOOL_OPTIONS` environment variable and sets it as a port in the `PodTemplateSpec`. If `JAVA_TOOL_OPTIONS` environment variable does not contain a `server.port` property or value, the convention adds the property and sets the value to `8080`, which is the Spring Boot default.

Example of PodIntent after applying the convention:

```

apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"conventions.carto.run/v1alpha1","kind":"PodIntent","metadata":{"a
nnotations":{},"name":"spring-sample","namespace":"default"},"spec":{"template":{"spe
c":{"containers":[{"image":"springio/petclinic","name":"workload"}]}}}
...
status:
  conditions:
  - lastTransitionTime: "..." # This status indicates that all worked as expected
    status: "True"
    type: ConventionsApplied
  - lastTransitionTime: "..."
    status: "True"
    type: Ready
  observedGeneration: 1
template:
  metadata:
    annotations:
      boot.spring.io/version: 2.3.3.RELEASE
      conventions.carto.run/applied-conventions: |-
        spring-boot-convention/spring-boot
        spring-boot-convention/spring-boot-web
    labels:
      conventions.carto.run/framework: spring-boot
  spec:
    containers:
    - env:

```

```

- name: JAVA_TOOL_OPTIONS
  value: -Dserver.port="8080"
image: index.docker.io/springio/petclinic@sha256:...
name: workload
ports:
- containerPort: 8080
  protocol: TCP
resources: {}

```

## Spring Boot Actuator convention

If the `spring-boot-actuator` dependency is in the metadata within the `SBOM` file under `dependencies`, the Spring Boot actuator convention is applied to the `PodTemplateSpec` object.

The Spring Boot Actuator convention does the following actions:

If the workload-level or platform-level automatic configuration of actuators is enabled:

1. Sets the management port in the `JAVA_TOOL_OPTIONS` environment variable to `8081`.
2. Sets the base path in the `JAVA_TOOL_OPTIONS` environment variable to `/actuator`.
3. Adds an annotation, `boot.spring.io/actuator`, to where the actuator is accessed.

The management port is set to port `8081` for security reasons. Although you can prevent public access to the actuator endpoints that are exposed on the management port when it is set to the default `8080`, the threat of exposure through internal access remains. The best practice for security is to set the management port to something other than `8080`.

However, if a management port number value is provided using the `-Dmanagement.server.port` property in `JAVA_TOOL_OPTIONS`, the Spring Boot actuator convention uses that value rather than its default `8081` as the management port.

You can access the management context of a Spring Boot application by creating a service pointing to port `8081` and base path `/actuator`.



### Important

To override the management port setting applied by this convention, see [How to set a JAVA\\_TOOL\\_OPTIONS property for a workload](#) earlier in this topic. Any alternative methods for setting the management port are overwritten. For example, if you configure the management port using `application.properties/yml` or `config server`, the Spring Boot Actuator convention overrides your configuration.

If the workload-level or platform-level automatic configuration of actuators is deactivated, the Spring Boot actuator convention does not set any `JAVA_TOOLS_OPTIONS` and does not set the annotation `boot.spring.io/actuator`.

Example of `PodIntent` after applying the convention:

```

apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
  annotations:
    kubect1.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"conventions.carto.run/v1alpha1","kind":"PodIntent","metadata":{"an
notations":{},"name":"spring-sample","namespace":"default"},"spec":{"template":{"spe
c":{"containers":[{"image":"springio/petclinic","name":"workload"}]}}}
...

```

```

status:
  conditions:
  - lastTransitionTime: "..." # This status indicates that all worked as expected
    status: "True"
    type: ConventionsApplied
  - lastTransitionTime: "..."
    status: "True"
    type: Ready
  observedGeneration: 1
  template:
    metadata:
      annotations:
        boot.spring.io/actuator: http://:8081/actuator
        boot.spring.io/version: 2.3.3.RELEASE
        conventions.carto.run/applied-conventions: |-
          spring-boot-convention/spring-boot
          spring-boot-convention/spring-boot-web
          spring-boot-convention/spring-boot-actuator
      labels:
        conventions.carto.run/framework: spring-boot
    spec:
      containers:
      - env:
        - name: JAVA_TOOL_OPTIONS
          value: Dmanagement.endpoints.web.base-path="/actuator" -Dmanagement.server.port="8081" -Dserver.port="8080"
        image: index.docker.io/springio/petclinic@sha256:...
        name: workload
        ports:
        - containerPort: 8080
          protocol: TCP
        resources: {}

```

## Spring Boot Actuator Probes convention

The Spring Boot Actuator Probes convention is applied only if all of the following conditions are met:

- The `spring-boot-actuator` dependency exists and is `>= 2.6`
- The `JAVA_TOOL_OPTIONS` environment variable does not include the following properties or, if either of the properties is included, it is set to a value of `true`:
  - `-Dmanagement.health.probes.enabled`
  - `-Dmanagement.endpoint.health.probes.add-additional-paths`

The Spring Boot Actuator Probes convention does the following actions:

1. Uses the main server port, which is the `server.port` value on `JAVA_TOOL_OPTIONS`, to set the liveness and readiness probes. For more information see the [Kubernetes documentation](#)
2. Adds the following properties and values to the `JAVA_TOOL_OPTIONS` environment variable:
  - `-Dmanagement.health.probes.enabled="true"`
  - `-Dmanagement.endpoint.health.probes.add-additional-paths="true"`

When this convention is applied, the probes are exposed as follows:

- Liveness probe: `/livez`
- Readiness probe: `/readyz`

Example of PodIntent after applying the convention:

```

apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
  annotations:
    kubectrl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"conventions.carto.run/v1alpha1","kind":"PodIntent","metadata":{"a
nnotations":{},"name":"spring-sample","namespace":"default"},"spec":{"template":{"spe
c":{"containers":[{"image":"springio/petclinic","name":"workload"}]}}}
...
status:
  conditions:
  - lastTransitionTime: "... " # This status indicates that all worked as expected
    status: "True"
    type: ConventionsApplied
  - lastTransitionTime: "... "
    status: "True"
    type: Ready
  observedGeneration: 1
  template:
    metadata:
      annotations:
        boot.spring.io/actuator: http://:8080/actuator
        boot.spring.io/version: 2.6.0
        conventions.carto.run/applied-conventions: |-
          spring-boot-convention/spring-boot
          spring-boot-convention/spring-boot-web
          spring-boot-convention/spring-boot-actuator
      labels:
        conventions.carto.run/framework: spring-boot
    spec:
      containers:
      - env:
        - name: JAVA_TOOL_OPTIONS
          value: -Dmanagement.endpoint.health.probes.add-additional-paths="true" -Dman
agement.endpoints.web.base-path="/actuator" -Dmanagement.health.probes.enabled="true"
-Dmanagement.server.port="8081" -Dserver.port="8080"
        image: index.docker.io/springio/petclinic@sha256:...
        name: workload
        livenessProbe:
          httpGet:
            path: /livez
            port: 8080
            scheme: HTTP
        ports:
        - containerPort: 8080
          protocol: TCP
        readinessProbe:
          httpGet:
            path: /readyz
            port: 8080
            scheme: HTTP
      resources: {}

```

## Service intent conventions

The Service intent conventions do not change the behavior of the final deployment, but you can use them as added information to process in the supply chain. For example, when an app requires to be bound to database service. This convention adds an annotation and a label to the `PodTemplateSpec` for each detected dependency. It also adds an annotation and a label to the `conventions.carto.run/applied-conventions`.

The list of the supported intents are:

### MySQL

- **Name:** `service-intent-mysql`
- **Label:** `services.conventions.apps.tanzu.vmware.com/mysql`
- **Dependencies:** `mysql-connector-java`, `r2dbc-mysql`

### PostgreSQL

- **Name:** `service-intent-postgres`
- **Label:** `services.conventions.apps.tanzu.vmware.com/postgres`
- **Dependencies:** `postgresql`, `r2dbc-postgresql`

### MongoDB

- **Name:** `service-intent-mongodb`
- **Label:** `services.conventions.apps.tanzu.vmware.com/mongodb`
- **Dependencies:** `mongodb-driver-core`

### RabbitMQ

- **Name:** `service-intent-rabbitmq`
- **Label:** `services.conventions.apps.tanzu.vmware.com/rabbitmq`
- **Dependencies:** `amqp-client`

### Redis

- **Name:** `service-intent-redis`
- **Label:** `services.conventions.apps.tanzu.vmware.com/redis`
- **Dependencies:** `jedis`

### Kafka

- **Name:** `service-intent-kafka`
- **Label:** `services.conventions.apps.tanzu.vmware.com/kafka`
- **Dependencies:** `kafka-clients`

### Kafka-streams

- **Name:** `service-intent-kafka-streams`
- **Label:** `services.conventions.apps.tanzu.vmware.com/kafka-streams`
- **Dependencies:** `kafka-streams`

## Example

When you apply the `Pod Intent` and the image contains a dependency, for example, of MySQL, then the output of the convention is:

```
apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"conventions.carto.run/v1alpha1","kind":"PodIntent","metadata":
{"annotations":{},"name":"spring-sample","namespace":"default"},"spec":{"template":{"s
pec":{"containers":[{"image":"springio/petclinic","name":"workload"}]}}}
      creationTimestamp: "..."
```



```

generation: 1
name: spring-sample
namespace: default
resourceVersion: "...
uid: ...
spec:
  serviceAccountName: default
  template:
    metadata: {}
    spec:
      containers:
      - image: springio/petclinic
        name: workload
        resources: {}
status:
  conditions:
  - lastTransitionTime: "... # This status indicates that all worked as expected
    status: "True"
    type: ConventionsApplied
  - lastTransitionTime: "...
    status: "True"
    type: Ready
observedGeneration: 1
template:
  metadata:
    annotations:
      boot.spring.io/actuator: http://:8080/actuator
      boot.spring.io/version: 2.3.3.RELEASE
      conventions.carto.run/applied-conventions: |-
        spring-boot-convention/spring-boot
        spring-boot-convention/spring-boot-web
        spring-boot-convention/spring-boot-actuator
        spring-boot-convention/service-intent-mysql
      services.conventions.apps.tanzu.vmware.com/mysql: mysql-connector-java/8.0.2
1
  labels:
    conventions.apps.tanzu.vmware.com/framework: spring-boot
    services.conventions.apps.tanzu.vmware.com/mysql: workload
  spec:
    containers:
    - env:
      - name: JAVA_TOOL_OPTIONS
        value: Dmanagement.endpoints.web.base-path="/actuator" -Dmanagement.serv
r.port="8081" -Dserver.port="8080"
      image: index.docker.io/springio/petclinic@sha256:...
      name: workload
      ports:
      - containerPort: 8080
        protocol: TCP
      resources: {}

```

## Troubleshoot Spring Boot conventions

This topic tells you how to troubleshoot Spring Boot conventions.

### Collect logs

If you have trouble, you can retrieve and examine logs from the Spring Boot convention server as follows:

1. The Spring Boot convention server creates a namespace to contain all of the associated resources. By default the namespace is `spring-boot-convention`. To inspect the logs, run:

```
kubectl logs -l app=spring-boot-webhook -n spring-boot-convention
```

For example:

```
$ kubectl logs -l app=spring-boot-webhook -n spring-boot-convention

{"level":"info","timestamp":"2021-11-11T16:00:26.597Z","caller":"spring-boot-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-web","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.

## Overview of Service Bindings

This topic tells you about using Service Bindings in Tanzu Application Platform (commonly know as TAP).

### Supported service binding specifications

Service Bindings for Kubernetes is an open-source product. For more information, see the [Service Binding for Kubernetes readme](#) and the [Service Binding for Kubernetes community website](#).

This implementation provides support for:

- [Provisioned Service](#)
- [Workload Projection](#)
- [Service Binding](#)
- [Direct Secret Reference](#)
- [Role-Based Access Control \(RBAC\)](#)

The following are not supported:

- [Workload Resource Mapping](#)
- Extensions including:
  - [Binding Secret Generation Strategies](#)

## Overview of Service Bindings

This topic tells you about using Service Bindings in Tanzu Application Platform (commonly know as TAP).

### Supported service binding specifications

Service Bindings for Kubernetes is an open-source product. For more information, see the [Service Binding for Kubernetes readme](#) and the [Service Binding for Kubernetes community website](#).

This implementation provides support for:

- [Provisioned Service](#)
- [Workload Projection](#)
- [Service Binding](#)
- [Direct Secret Reference](#)
- [Role-Based Access Control \(RBAC\)](#)

The following are not supported:

- [Workload Resource Mapping](#)
- Extensions including:
  - [Binding Secret Generation Strategies](#)

## Install Service Bindings

This topic tells you how to install Service Bindings from the Tanzu Application Platform (commonly known as TAP) package repository.



### Note

Follow the steps in this topic if you do not want to use a profile to install Service Bindings. For more information about profiles, see [Components and installation profiles](#).

## Prerequisites

Before installing Service Bindings:

- Complete all prerequisites to install Tanzu Application Platform (commonly known as TAP). For more information, see [Prerequisites](#).

## Install Service Bindings

Use the following procedure to install Service Bindings:

1. List version information for the package by running:

```
tanzu package available list service-bindings.labs.vmware.com --namespace tap-  
install
```

For example:

```
$ tanzu package available list service-bindings.labs.vmware.com --namespace tap  
-install  
- Retrieving package versions for service-bindings.labs.vmware.com...  
NAME                                VERSION  RELEASED-AT  
service-bindings.labs.vmware.com    0.5.0    2021-09-15T00:00:00Z
```

2. Install the package by running:

```
tanzu package install service-bindings -p service-bindings.labs.vmware.com -v
```

```
0.5.0 -n tap-install
```

Example output:

```
/ Installing package 'service-bindings.labs.vmware.com'
| Getting namespace 'tap-install'
- Getting package metadata for 'service-bindings.labs.vmware.com'
| Creating service account 'service-bindings-tap-install-sa'
| Creating cluster admin role 'service-bindings-tap-install-cluster-role'
| Creating cluster role binding 'service-bindings-tap-install-cluster-rolebinding'
\ Creating package resource
| Package install status: Reconciling

Added installed package 'service-bindings' in namespace 'tap-install'
```

3. Verify the package install by running:

```
tanzu package installed get service-bindings -n tap-install
```

Example output:

```
- Retrieving installation details for service-bindings...
NAME:                service-bindings
PACKAGE-NAME:        service-bindings.labs.vmware.com
PACKAGE-VERSION:     0.5.0
STATUS:              Reconcile succeeded
CONDITIONS:          [{"ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

4. Run the following command:

```
kubectl get pods -n service-bindings
```

For example:

```
$ kubectl get pods -n service-bindings
NAME                                READY   STATUS    RESTARTS   AGE
manager-6d85ffffbcd-j4qvs          1/1     Running   0           22s
```

Verify that `STATUS` is `Running`

## Troubleshoot Service Bindings

This topic tells you how to troubleshoot Service Bindings in Tanzu Application Platform (commonly known as TAP).

### Collect logs

To help identify issues when troubleshooting, you can retrieve and examine logs from the service binding manager.

To retrieve pod logs from the `manager` running in the `service-bindings` namespace, run:

```
kubectl -n service-bindings logs -l role=manager
```

For example:

```
$ kubectl -n service-bindings logs -l role=manager
```

```

2021/11/05 15:25:28 Registering 3 clients
2021/11/05 15:25:28 Registering 3 informer factories
2021/11/05 15:25:28 Registering 7 informers
2021/11/05 15:25:28 Registering 8 controllers
{"severity":"INFO","timestamp":"2021-11-05T15:25:28.483823208Z","caller":"logging/nfi
g.go:116","message":"Successfully created the logger."}
{"severity":"INFO","timestamp":"2021-11-05T15:25:28.48392361Z","caller":"logging/confi
g.go:117","message":"Logging level set to: info"}
{"severity":"INFO","timestamp":"2021-11-05T15:25:28.483999911Z","caller":"logging/conf
ig.go:79","message":"Fetch GitHub commit ID from kodata failed","error":"open /var/ru
n/ko/HEAD: no such file or directory"}
{"severity":"INFO","timestamp":"2021-11-05T15:25:28.484035711Z","logger":"webhook","ca
ller":"profiling/server.go:64","message":"Profiling enabled: false"}
{"severity":"INFO","timestamp":"2021-11-05T15:25:28.522884909Z","logger":"webhook","ca
ller":"leaderelection/context.go:46","message":"Running with Standard leader electio
n"}
{"severity":"INFO","timestamp":"2021-11-05T15:25:28.523358615Z","logger":"webhook","ca
ller":"provisionedservice/controller.go:31","message":"Setting up event handlers."}
...
{"severity":"ERROR","timestamp":"2021-11-17T12:30:24.557178813Z","logger":"webhook","c
aller":"controller/controller.go:548","message":"Reconcile error","duration":"276.504µ
s","error":"deployments.apps \"spring-petclinic\" not found","stacktrace":"knative.de
v/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg/v0.0-20210331065221-952fdd90db
b0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkIte
m\n\tknative.dev/pkg/v0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\n
knative.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg/v0.0-20210331
065221-952fdd90dbb0/controller/controller.go:468"}
{"severity":"ERROR","timestamp":"2021-11-17T12:47:04.558217679Z","logger":"webhook","c
aller":"controller/controller.go:548","message":"Reconcile error","duration":"249.103µ
s","error":"deployments.apps \"spring-petclinic\" not found","stacktrace":"knative.de
v/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg/v0.0-20210331065221-952fdd90db
b0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkIte
m\n\tknative.dev/pkg/v0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\n
knative.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg/v0.0-20210331
065221-952fdd90dbb0/controller/controller.go:468"}
{"severity":"ERROR","timestamp":"2021-11-17T13:03:44.558683121Z","logger":"webhook","c
aller":"controller/controller.go:548","message":"Reconcile error","duration":"177.403µ
s","error":"deployments.apps \"spring-petclinic\" not found","stacktrace":"knative.de
v/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg/v0.0-20210331065221-952fdd90db
b0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkIte
m\n\tknative.dev/pkg/v0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\n
knative.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg/v0.0-20210331
065221-952fdd90dbb0/controller/controller.go:468"}
{"severity":"ERROR","timestamp":"2021-11-17T13:20:24.559192644Z","logger":"webhook","c
aller":"controller/controller.go:548","message":"Reconcile error","duration":"223.203µ
s","error":"deployments.apps \"spring-petclinic\" not found","stacktrace":"knative.de
v/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg/v0.0-20210331065221-952fdd90db
b0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkIte
m\n\tknative.dev/pkg/v0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\n
knative.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg/v0.0-20210331
065221-952fdd90dbb0/controller/controller.go:468"}
{"severity":"ERROR","timestamp":"2021-11-17T13:37:04.559648412Z","logger":"webhook","c
aller":"controller/controller.go:548","message":"Reconcile error","duration":"173.003µ
s","error":"deployments.apps \"spring-petclinic\" not found","stacktrace":"knative.de
v/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg/v0.0-20210331065221-952fdd90db
b0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkIte
m\n\tknative.dev/pkg/v0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\n
knative.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg/v0.0-20210331
065221-952fdd90dbb0/controller/controller.go:468"}
{"severity":"ERROR","timestamp":"2021-11-17T13:53:44.56010516Z","logger":"webhook","ca
ller":"controller/controller.go:548","message":"Reconcile error","duration":"182.402µ
s","error":"deployments.apps \"spring-petclinic\" not found","stacktrace":"knative.de
v/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg/v0.0-20210331065221-952fdd90db
b0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkIte
m\n\tknative.dev/pkg/v0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\n
knative.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg/v0.0-20210331

```

```

065221-952fdd90dbb0/controller/controller.go:468"}
{"severity":"ERROR","timestamp":"2021-11-17T14:10:24.560536033Z","logger":"webhook","c
aller":"controller/controller.go:548","message":"Reconcile error","duration":"155.603µ
s","error":"deployments.apps \"spring-petclinic\" not found","stacktrace":"knative.de
v/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg/v0.0.0-20210331065221-952fdd90db
b0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkIte
m\n\tknative.dev/pkg/v0.0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\n
knative.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg/v0.0.0-20210331
065221-952fdd90dbb0/controller/controller.go:468"}
{"severity":"ERROR","timestamp":"2021-11-17T14:27:04.560960243Z","logger":"webhook","c
aller":"controller/controller.go:548","message":"Reconcile error","duration":"171.002µ
s","error":"deployments.apps \"spring-petclinic\" not found","stacktrace":"knative.de
v/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg/v0.0.0-20210331065221-952fdd90db
b0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkIte
m\n\tknative.dev/pkg/v0.0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\n
knative.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg/v0.0.0-20210331
065221-952fdd90dbb0/controller/controller.go:468"}
{"severity":"ERROR","timestamp":"2021-11-17T14:43:44.56142548Z","logger":"webhook","c
aller":"controller/controller.go:548","message":"Reconcile error","duration":"179.203µ
s","error":"deployments.apps \"spring-petclinic\" not found","stacktrace":"knative.de
v/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg/v0.0.0-20210331065221-952fdd90db
b0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkIte
m\n\tknative.dev/pkg/v0.0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\n
knative.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg/v0.0.0-20210331
065221-952fdd90dbb0/controller/controller.go:468"}
{"severity":"ERROR","timestamp":"2021-11-17T15:00:24.561881861Z","logger":"webhook","c
aller":"controller/controller.go:548","message":"Reconcile error","duration":"167.902µ
s","error":"deployments.apps \"spring-petclinic\" not found","stacktrace":"knative.de
v/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg/v0.0.0-20210331065221-952fdd90db
b0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkIte
m\n\tknative.dev/pkg/v0.0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\n
knative.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg/v0.0.0-20210331
065221-952fdd90dbb0/controller/controller.go:468"}

```

## Service Bindings resource specification

This topic tells you about the Service Bindings resource specification in Tanzu Application Platform (commonly known as TAP).

The [ServiceBinding](#) resource shape and behavior is defined by the following specification:

```

apiVersion: servicebinding.io/v1alpha3
kind: ServiceBinding
metadata:
  name: account-db
spec:
  service:
    apiVersion: mysql.example/v1alpha1
    kind: MySQL
    name: account-db
  workload:
    apiVersion: apps/v1
    kind: Deployment
    name: account-service

```

## Overview of Services Toolkit

Services Toolkit comprises the following Kubernetes-native components, which support the management, lifecycle, discoverability, and connectivity of service resources on Kubernetes, such as databases, message queues, DNS records, and more:

- Service offering

- Service API projection
- Service resource replication
- Service resource claims

To learn more about Services Toolkit, see the [Services Toolkit documentation](#)

## Overview of Services Toolkit

Services Toolkit comprises the following Kubernetes-native components, which support the management, lifecycle, discoverability, and connectivity of service resources on Kubernetes, such as databases, message queues, DNS records, and more:

- Service offering
- Service API projection
- Service resource replication
- Service resource claims

To learn more about Services Toolkit, see the [Services Toolkit documentation](#)

## Install Services Toolkit

This document describes how to install Services Toolkit from the Tanzu Application Platform package repository.



### Note

Follow the steps in this topic if you do not want to use a profile to install Services Toolkit. For more information about profiles, see [Components and installation profiles](#).

## Prerequisites

Before installing Services Toolkit:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see [Prerequisites](#).
- Install cert-manager. For more information, see [Install cert-manager](#).

## Install Services Toolkit

To install Services Toolkit:

1. See what versions of Services Toolkit are available to install by running:

```
tanzu package available list -n tap-install services-toolkit.tanzu.vmware.com
```

For example:

```
$ tanzu package available list -n tap-install services-toolkit.tanzu.vmware.com
- Retrieving package versions for services-toolkit.tanzu.vmware.com...
  NAME                               VERSION          RELEASED-AT
  services-toolkit.tanzu.vmware.com  0.9.0           2022-09-08T00:00:00Z
```

2. Install Services Toolkit by running:

```
tanzu package install services-toolkit -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, `0.9.0`.

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:     0.9.0
STATUS:              Reconcile succeeded
CONDITIONS:          [{"ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

## Overview of Flux CD Source Controller

Flux CD Source Controller provides you with APIs for acquiring resources such as Git, Helm repositories and S3 buckets on the cluster. For more information, see [Flux CD Source Controller documentation](#).

The source-controller implements the [source.toolkit.fluxcd.io](#) API in GitHub and is a core component of the [GitOps toolkit](#).

## Overview of Flux CD Source Controller

Flux CD Source Controller provides you with APIs for acquiring resources such as Git, Helm repositories and S3 buckets on the cluster. For more information, see [Flux CD Source Controller documentation](#).

The source-controller implements the [source.toolkit.fluxcd.io](#) API in GitHub and is a core component of the [GitOps toolkit](#).

## Install Flux CD Source Controller

This topic tells you how to install Flux CD Source Controller from the Tanzu Application Platform (commonly known as TAP) package repository.



### Note

Follow the steps in this topic if you do not want to use a profile to install Flux CD Source Controller. For more information about profiles, see [Components and installation profiles](#).

## Prerequisites

Before installing Flux CD Source Controller:



- Complete all prerequisites to install Tanzu Application Platform. For more information, see [Prerequisites](#).
- Install cert-manager on the cluster. For more information, see [Install cert-manager](#).

## Configuration

The Flux CD Source Controller package has no configuration values.

## Installation

To install Flux CD Source Controller from the Tanzu Application Platform package repository:

1. List version information for the package by running:

```
tanzu package available list fluxcd.source.controller.tanzu.vmware.com -n tap-install
```

For example:

```
$ tanzu package available list fluxcd.source.controller.tanzu.vmware.com -n tap-install
\ Retrieving package versions for fluxcd.source.controller.tanzu.vmware.com...
NAME                                VERSION  RELEASED-AT
fluxcd.source.controller.tanzu.vmware.com 0.16.0  2021-10-27 19:00:00 -0500 -05
```

2. Install the package by running:

```
tanzu package install fluxcd-source-controller -p fluxcd.source.controller.tanzu.vmware.com -v VERSION-NUMBER -n tap-install
```

Where:

- `VERSION-NUMBER` is the version of the package listed in step 1.

For example:

```
tanzu package install fluxcd-source-controller -p fluxcd.source.controller.tanzu.vmware.com -v 0.16.0 -n tap-install
\ Installing package 'fluxcd.source.controller.tanzu.vmware.com'
| Getting package metadata for 'fluxcd.source.controller.tanzu.vmware.com'
| Creating service account 'fluxcd-source-controller-tap-install-sa'
| Creating cluster admin role 'fluxcd-source-controller-tap-install-cluster-role'
| Creating cluster role binding 'fluxcd-source-controller-tap-install-cluster-rolebinding'
| Creating package resource
- Waiting for 'PackageInstall' reconciliation for 'fluxcd-source-controller'
| 'PackageInstall' resource install status: Reconciling

Added installed package 'fluxcd-source-controller'
```

This package creates a new namespace called `flux-system`. This namespace hosts all the elements of fluxcd.

3. Verify the package install by running:

```
tanzu package installed get fluxcd-source-controller -n tap-install
```

For example:

```
tanzu package installed get fluxcd-source-controller -n tap-install
\ Retrieving installation details for fluxcd-source-controller...
NAME:                fluxcd-source-controller
PACKAGE-NAME:        fluxcd.source.controller.tanzu.vmware.com
PACKAGE-VERSION:     0.16.0
STATUS:              Reconcile succeeded
CONDITIONS:          [{"ReconcileSucceeded True  "}]
USEFUL-ERROR-MESSAGE:
```

Verify that **STATUS** is **Reconcile succeeded**.

```
kubectl get pods -n flux-system
```

For example:

```
$ kubectl get pods -n flux-system
NAME                                READY   STATUS    RESTARTS   AGE
source-controller-69859f545d-1l8fj  1/1     Running   0           3m38s
```

Verify that **STATUS** is **Running**.

## Try fluxcd-source-controller

1. Verify the main components of **fluxcd-source-controller** were installed by running:

```
kubectl get all -n flux-system
```

Expect to see the following outputs or similar:

```
NAME                                READY   STATUS    RESTARTS   AGE
pod/source-controller-7684c85659-2zfxb  1/1     Running   0           40m

NAME                                TYPE          CLUSTER-IP      EXTERNAL-IP    PORT(S)
AGE
service/source-controller             ClusterIP     10.108.138.74   <none>         80/TCP
40m

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/source-controller     1/1     1             1           40m

NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/source-controller-7684c85659  1         1         1       40m
```

2. Verify all the CRD were installed by running:

```
kubectl get crds -n flux-system | grep ".fluxcd.io"
buckets.source.toolkit.fluxcd.io      2022-03-07T19:20:14Z
gitrepositories.source.toolkit.fluxcd.io 2022-03-07T19:20:14Z
helmcharts.source.toolkit.fluxcd.io    2022-03-07T19:20:14Z
helmrepositories.source.toolkit.fluxcd.io 2022-03-07T19:20:14Z
```



### Note

You will communicate with **fluxcd-source-controller** through its CRDs.

3. Follow these steps to consume a **GitRepository** object:

1. Create the following **gitrepository-sample.yaml** file:

```

apiVersion: source.toolkit.fluxcd.io/v1beta1
kind: GitRepository
metadata:
  name: gitrepository-sample
spec:
  interval: 1m
  url: https://github.com/vmware-tanzu/application-accelerator-samples
  ref:
    branch: main

```

2. Apply the created conf:

```

kubectl apply -f gitrepository-sample.yaml
gitrepository.source.toolkit.fluxcd.io/gitrepository-sample created

```

3. Verify the git-repository was fetched correctly:

```

kubectl get GitRepository
NAME                                URL
READY   STATUS
AGE
gitrepository-sample               https://github.com/vmware-tanzu/application-accele
rator-samples   True   Fetched revision: main/132f4e719209eb10b9485302f8
593fc0e680f4fc   4s

```

For more examples, see the samples directory on [fluxcd/source-controller/samples](#) in GitHub.

## Documentation

For documentation specific to fluxcd-source-controller, see the main repository [fluxcd/source-controller](#) in GitHub.

## Overview of Source Controller

Tanzu Source Controller provides a standard interface for artifact acquisition and extends the function of [Flux CD Source Controller](#).

Tanzu Source Controller supports the following two resource types:

- `ImageRepository`
- `MavenArtifact`

An `ImageRepository` resource can resolve the source from the contents of an image in an image registry. This enables app developers to create and update workloads from local source code or a code repository.

A `MavenArtifact` resource can resolve a binary artifact from a Maven repository. This functionality enables the supply chain to support artifacts produced externally.



### Note

Fetching `RELEASE` version from GitHub packages is not currently supported. The `metadata.xml` in GitHub packages does not have the `release` tag that contains the released version number. For more information, see [Maven-metadata.xml is corrupted on upload to registry](#) on GitHub.

## Overview of Source Controller

Tanzu Source Controller provides a standard interface for artifact acquisition and extends the function of [Flux CD Source Controller](#).

Tanzu Source Controller supports the following two resource types:

- `ImageRepository`
- `MavenArtifact`

An `ImageRepository` resource can resolve the source from the contents of an image in an image registry. This enables app developers to create and update workloads from local source code or a code repository.

A `MavenArtifact` resource can resolve a binary artifact from a Maven repository. This functionality enables the supply chain to support artifacts produced externally.



### Note

Fetching `RELEASE` version from GitHub packages is not currently supported. The `metadata.xml` in GitHub packages does not have the `release` tag that contains the released version number. For more information, see [Maven-metadata.xml is corrupted on upload to registry](#) on GitHub.

## Install Source Controller

This topic tells you how to install Source Controller from the Tanzu Application Platform (commonly known as TAP) package repository.



### Note

Follow the steps in this topic if you do not want to use a profile to install Source Controller. For more information about profiles, see [Components and installation profiles](#).

## Prerequisites

Before installing Source Controller:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see [Prerequisites](#).
- Install cert-manager on the cluster. For more information, see [Install cert-manager](#).

## Install

To install Source Controller:

1. List version information for the package by running:

```
tanzu package available list controller.source.apps.tanzu.vmware.com --namespace tap-install
```

For example:

```
$ tanzu package available list controller.source.apps.tanzu.vmware.com --namespace tap-install
- Retrieving package versions for controller.source.apps.tanzu.vmware.com...
  NAME                                VERSION  RELEASED-AT
  controller.source.apps.tanzu.vmware.com 0.3.1    2022-01-23 19:00:00 -0500 -05
  controller.source.apps.tanzu.vmware.com 0.3.2    2022-02-21 19:00:00 -0500 -05
  controller.source.apps.tanzu.vmware.com 0.3.3    2022-03-03 19:00:00 -0500 -05
  controller.source.apps.tanzu.vmware.com 0.4.1    2022-06-09 19:00:00 -0500 -05
```

## 2. (Optional) Gather the values schema:

```
tanzu package available get controller.source.apps.tanzu.vmware.com/VERSION-NUMBER --values-schema --namespace tap-install
```

Where `VERSION-NUMBER` is the version of the package listed in step 1 above.

For example:

```
tanzu package available get controller.source.apps.tanzu.vmware.com/0.4.1 --values-schema --namespace tap-install

Retrieving package details for controller.source.apps.tanzu.vmware.com/0.4.1...
KEY                DEFAULT  TYPE      DESCRIPTION
aws_iam_role_arn   string   Optional: The AWS IAM Role ARN to attach to the Source Controller service account
ca_cert_data       string   Optional: PEM Encoded certificate data for image registries with private CA.
```

## 3. (Optional) Create a file named `source-controller-values.yaml` to override the default installation settings. You can configure the following fields:

- `ca_cert_data`:** Enables Source Controller to connect to image registries that use self-signed or private certificate authorities. If a certificate error `x509: certificate signed by unknown authority` occurs, use this option to trust additional certificate authorities.

To provide a custom certificate, add the PEM-encoded CA certificate data to `source-controller-values.yaml`. For example:

```
ca_cert_data: |
  -----BEGIN CERTIFICATE-----
  MIICpTCCAYUCBgkqhkiG9w0BBQ0wMzAbBgkqhkiG9w0BBQwwDgQIYg9x6gkCAggA
  ...
  9T1A7A4FFpQqbhAuAVH6KQ8WMZIrVxJSQ03c91KVkI62wQ==
  -----END CERTIFICATE-----
```

- `aws_iam_role_arn`:** Annotates the Source Controller service with an AWS Identity and Access Management (IAM) role. This allows Source Controller to pull images from Amazon Elastic Container Registry (ECR).

To add the AWS IAM role Amazon Resource Name (ARN) to the Source Controller service, add the ARN to `source-controller-values.yaml`. For example:

```
aws_iam_role_arn: "eks.amazonaws.com/role-arn: arn:aws:iam::112233445566:role/source-controller-manager"
```

## 4. Install the package by running:

```
tanzu package install source-controller -p controller.source.apps.tanzu.vmware.com -v VERSION-NUMBER -n tap-install -f VALUES-FILE
```

Where:

- `VERSION-NUMBER` is the version of the package listed in step 1 above.
- `VALUES-FILE` is the path to the file created in step 3.

For example:

```
tanzu package install source-controller -p controller.source.apps.tanzu.vmware.com -v 0.4.1 -n tap-install -f source-controller-values.yaml
\ Installing package 'controller.source.apps.tanzu.vmware.com'
| Getting package metadata for 'controller.source.apps.tanzu.vmware.com'
| Creating service account 'source-controller-default-sa'
| Creating cluster admin role 'source-controller-default-cluster-role'
| Creating cluster role binding 'source-controller-default-cluster-rolebinding'
| Creating secret 'source-controller-default-values'
| Creating package resource
- Waiting for 'PackageInstall' reconciliation for 'source-controller'
- 'PackageInstall' resource install status: Reconciling

Added installed package 'source-controller'
```

#### 5. Verify the package installation by running:

```
tanzu package installed get source-controller -n tap-install
```

For example:

```
tanzu package installed get source-controller -n tap-install
- Retrieving installation details for source-controller...
NAME:                source-controller
PACKAGE-NAME:        controller.source.apps.tanzu.vmware.com
PACKAGE-VERSION:     0.4.1
STATUS:              Reconcile succeeded
CONDITIONS:          [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`:

```
kubectl get pods -n source-system
```

For example:

```
$ kubectl get pods -n source-system
NAME                                READY   STATUS    RESTARTS   AGE
source-controller-manager-f68dc7bb6-41rn6  1/1     Running  0          100s
```

Verify that `STATUS` is `Running`.

## Troubleshoot Source Controller

This topic gives you guidance about how to troubleshoot issues with Source Controller.

### Collecting Logs from Source Controller Manager

To retrieve Pod logs from the `controller-manager`, run the following command in the `source-system` namespace:

```
kubectl logs -n source-system -l control-plane=controller-manager
```

For example:

```
kubectl logs -n source-system -l control-plane=controller-manager
2021-11-18T17:59:43.152Z      INFO      controller.imagerepository      Starting Event
Source {"reconciler group": "source.apps.tanzu.vmware.com", "reconciler kind": "Image
Repository", "source": "kind source: /, Kind="}
2021-11-18T17:59:43.152Z      INFO      controller.metarepository      Starting Event
Source {"reconciler group": "source.apps.tanzu.vmware.com", "reconciler kind": "MetaR
epository", "source": "kind source: /, Kind="}
2021-11-18T17:59:43.152Z      INFO      controller.metarepository      Starting Event
Source {"reconciler group": "source.apps.tanzu.vmware.com", "reconciler kind": "MetaR
epository", "source": "kind source: /, Kind="}
2021-11-18T17:59:43.152Z      INFO      controller.metarepository      Starting Event
Source {"reconciler group": "source.apps.tanzu.vmware.com", "reconciler kind": "MetaR
epository", "source": "kind source: /, Kind="}
2021-11-18T17:59:43.152Z      INFO      controller.metarepository      Starting Contr
oller {"reconciler group": "source.apps.tanzu.vmware.com", "reconciler kind": "MetaR
epository"}
2021-11-18T17:59:43.152Z      INFO      controller.imagerepository      Starting Event
Source {"reconciler group": "source.apps.tanzu.vmware.com", "reconciler kind": "Image
Repository", "source": "kind source: /, Kind="}
2021-11-18T17:59:43.152Z      INFO      controller.imagerepository      Starting Event
Source {"reconciler group": "source.apps.tanzu.vmware.com", "reconciler kind": "Image
Repository", "source": "kind source: /, Kind="}
2021-11-18T17:59:43.152Z      INFO      controller.imagerepository      Starting Contr
oller {"reconciler group": "source.apps.tanzu.vmware.com", "reconciler kind": "Image
Repository"}
2021-11-18T17:59:43.389Z      INFO      controller.metarepository      Starting worke
rs {"reconciler group": "source.apps.tanzu.vmware.com", "reconciler kind": "MetaR
epository", "worker count": 1}
2021-11-18T17:59:43.391Z      INFO      controller.imagerepository      Starting worke
rs {"reconciler group": "source.apps.tanzu.vmware.com", "reconciler kind": "Image
Repository", "worker count": 1}
```

## Source Controller reference

This topic provides reference documentation for Source Controller.

## ImageRepository

```
---
apiVersion: source.apps.tanzu.vmware.com/v1alpha1
kind: ImageRepository
spec:
  image: registry.example/image/repository:tag
  # optional fields
  interval: 5m
  imagePullSecrets: []
  serviceAccountName: default
```

`ImageRepository` resolves source code defined in an Open Container Initiative (OCI) image repository, exposing the resulting source artifact at a URL defined by `.status.artifact.url`.

The interval determines how often to check tagged images for changes. Setting this value too high will result in delays in discovering new sources, while setting it too low may trigger a registry's rate limits.

Repository credentials can be defined as image pull secrets. You can reference them either directly from the resources at `.spec.imagePullSecrets` or attach them to a service account referenced at

`.spec.serviceAccountName`. The default service account name "default" is used if not otherwise specified. The default credential helpers for the registry are also used, for example, pulling from Google Container Registry (GCR) on a Google Kubernetes Engine (GKE) cluster.

## MavenArtifact

```
---
apiVersion: source.apps.tanzu.vmware.com/v1alpha1
kind: MavenArtifact
metadata:
  name: mavenartifact-sample
spec:
  artifact:
    groupId: org.springframework.boot
    artifactId: spring-boot
    version: "2.7.0"
  repository:
    url: https://repo1.maven.org/maven2
    interval: 5m0s
    timeout: 1m0s
```

`MavenArtifact` resolves artifact from a Maven repository, exposing the resulting artifact at a URL defined by `.status.artifact.url`.

The `interval` determines how often to check artifact for changes. Setting this value too high results in delays in discovering new sources, while setting it too low may trigger a repository's rate limits.

Repository credentials may be defined as secrets referenced from the resources at `.spec.repository.secretRef`. Secrets referenced by `spec.repository.secretRef` is parsed as follows:

```
---
apiVersion: v1
kind: Secret
metadata:
  name: auth-secret
type: Opaque
data:
  username: <BASE64>
  password: <BASE64>
  caFile: <BASE64> // PEM Encoded certificate data for Custom CA
  certFile: <BASE64> // PEM-encoded client certificate
  keyFile: <BASE64> // Private Key
```

Maven supports a broad set of `version` syntax. Source Controller supports a strict subset of Maven's version syntax in order to ensure compatibility and avoid user confusion. The subset of supported syntax may grow over time, but will never expand past the syntax defined directly by Maven. This behavior means that we can use `mvn` as a reference implementation for artifact resolution.

Version support implemented in the following order:

1. Pinned version - an exact match of a version in `maven-metadata.xml` (`versioning/versions/version`).
2. `RELEASE` - metaversion defined in `maven-metadata.xml` (`versioning/release`).
3. `*-SNAPSHOT` - the newest artifact for a snapshot version.
4. `LATEST` - metaversion defined in `maven-metadata.xml` (`versioning/latest`).



- Version ranges - <https://maven.apache.org/enforcer/enforcer-rules/versionRanges.html>. Support is planned for a future release.



#### Note

Pinned versions should be immutable, all other versions are dynamic and can change at any time. The `.spec.interval` defines how frequently to check for updated artifacts.

## Overview of Developer Conventions

Developer Conventions is a set of conventions that enable your workloads to support live-update and debug operations in Tanzu Application Platform (commonly known as TAP).

### Prerequisites

- Tanzu CLI Apps plug-in
- Tanzu Dev Tools for VSCode IDE extension.

### Features

#### Enabling Live Updates

Developer Conventions modifies your workload to enable live updates in either of the following situations:

- You deploy a workload by using the Tanzu CLI Apps plug-in and include the flag `--live-update=true`. For more information about how to deploy a workload with the CLI, see [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:

- Looks for the `apps.tanzu.vmware.com/live-update=true` annotation on a PodTemplateSpec associated with a workload.
- Verifies that the image to which conventions are applied contains a process that can be live updated.
- 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](#)

## Overview of Developer Conventions

Developer Conventions is a set of conventions that enable your workloads to support live-update and debug operations in Tanzu Application Platform (commonly known as TAP).

## Prerequisites

- [Tanzu CLI Apps plug-in](#)
- [Tanzu Dev Tools for VSCode IDE extension](#).

## Features

### Enabling Live Updates

Developer Conventions modifies your workload to enable live updates in either of the following situations:

- You deploy a workload by using the Tanzu CLI Apps plug-in and include the flag `--live-update=true`. For more information about how to deploy a workload with the CLI, see [Tanzu apps workload apply](#).
- You deploy a workload by using the [Tanzu: Live Update Start](#) option through the Tanzu Developer Tools for VS Code extension. For more information about live updating with the extension, see [Overview of Tanzu Developer Tools for Visual Studio Code](#).

When either of the preceding actions take place, the convention behaves as follows:

1. Looks for the `apps.tanzu.vmware.com/live-update=true` annotation on a PodTemplateSpec associated with a workload.
2. Verifies that the image to which conventions are applied contains a process that can be live updated.
3. Adds annotations to the PodTemplateSpec to modify the Knative properties `minScale` & `maxScale` such that the minimum and maximum number of pods is 1. This ensures the eventual running pod is not scaled down to 0 during a live update session.

After these changes are made, you can use the Tanzu Dev Tools extension or the Tilt CLI to make live update changes to source code directly on the cluster.

## Enabling debugging

Developer Conventions modifies your workload to enable debugging in either of the following situations:

- You deploy a workload by using the Tanzu CLI Apps plug-in and include the flag `--debug=true`. For more information about how to deploy a workload with the CLI, see [Tanzu apps workload apply](#).
- You deploy a workload by using the `Tanzu Java Debug Start` option through the Tanzu Developer Tools for VS Code extension. For more information about debugging with the extension, see [Overview of Tanzu Developer Tools for Visual Studio Code](#).

When either of the preceding actions take place, the convention behaves as follows:

1. It looks for the `apps.tanzu.vmware.com/debug=true` annotation on a PodTemplateSpec associated with a workload.
2. It checks for the `debug-8` or `debug-9` labels on the image configuration's bill of materials (BOM).
3. It sets the TimeoutSeconds of the Liveness, Readiness, and Startup probes to 600 if currently set to a lower number.
4. It adds annotations to the PodTemplateSpec to modify the Knative properties `minScale` & `maxScale` such that the minimum and maximum number of pods is 1. This ensures the eventual running pod won't be scaled down to 0 during a debug session.

After these changes are made, you can use the Tanzu Dev Tools extension or other CLI-based debuggers to debug your workload directly on the cluster.



### Note

Currently, Developer Conventions only supports debug operations for Java applications.

## Next steps

- [Install Developer Conventions](#)

## Install Developer Conventions

This document tells you how to install Developer Conventions from the Tanzu Application Platform (commonly known as TAP) package repository.



### Note

Follow the steps in this topic if you do not want to use a profile to install Developer Conventions. For more information about profiles, see [About Tanzu Application Platform components and profiles](#).

## Prerequisites

Before installing Developer Conventions:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see [Prerequisites](#).
- Install [Supply Chain Choreographer](#).

## Install

To install Developer Conventions:

1. Get the exact name and version information for the Developer Conventions package to be installed by running:

```
tanzu package available list developer-conventions.tanzu.vmware.com --namespace tap-install
```

For example:

```
$ tanzu package available list developer-conventions.tanzu.vmware.com --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 [Uninstall Tanzu Application Platform packages](#). The package name for developer conventions is `developer-conventions`.

## Run Developer Conventions on an OpenShift cluster

This topic tells you about considerations for running Developer Conventions on OpenShift.

On OpenShift clusters, Developer Conventions must run with a custom SecurityContextConstraint (SCC) to enable compliance with restricted Kubernetes pod security standards. Tanzu Application Platform configures the following SCC for the Developer Convention's webhook when you configure the `kubernetes_distribution: openshift` key in the `tap-values.yaml` file.

Specification follows:

```
---
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  name: developer-conventions-scc
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: false
allowPrivilegedContainer: false
defaultAddCapabilities: null
fsGroup:
  type: RunAsAny
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities: null
runAsUser:
  type: MustRunAsNonRoot
seLinuxContext:
  type: MustRunAs
supplementalGroups:
  type: RunAsAny
volumes:
  - secret
seccompProfiles: []
groups:
  - system:serviceaccounts:developer-conventions
```

## Overview of Learning Center for Tanzu Application Platform

Learning Center provides a platform for creating and self-hosting workshops. It allows you to create workshops from markdown files that are displayed to the learner in a terminal shell environment with an instructional wizard UI.

The UI can embed slide content, an integrated development environment (IDE), a web console for accessing the Kubernetes cluster, and other custom web applications.

Although Learning Center requires Kubernetes to run, and is used to teach users about Kubernetes, you can use it to host training for other purposes as well. For example, you can use it to help train users in web-based applications, use of databases, or programming languages, where the user has no interest or need for Kubernetes.

## Use cases

Use case scenarios that Learning Center supports include:

- Supervised workshops. For example, a workshop run at a conference, at a customer site, or online. The workshop has a set time period and you know the maximum number of users to expect. After the training is complete, the Kubernetes cluster created for the workshop is destroyed.
- Temporary learning portal. This is for when you must provide access to a small set of workshops for a short duration for hands on demos at a conference vendor booth. Users select which topic they want to learn about and do that workshop. The workshop instance is created on demand. When they have finished the workshop, that workshop instance is destroyed to free up resources. After the conference has finished, the Kubernetes cluster is destroyed.
- Permanent learning portal. Similar to the temporary learning portal, but runs on an extended basis as a public website where anyone can come and learn at any time.
- Personal training or demos. This is where anyone who wants to run a workshop on their own Kubernetes cluster to learn that topic, or where a product demo was packaged up as a workshop and they want to use it to demonstrate the product to a customer. You can destroy the workshop environment when complete, but there is no need for the cluster to be destroyed.

When running workshops, wherever possible a shared Kubernetes cluster reduces the amount of setup required. This works for developer-focused workshops, because it is usually not necessary to provide elevated access to the Kubernetes cluster, and you can use role-based access controls (RBAC) to prevent users from interfering with each other. You can also set quotas so users are restricted as to how much resources they can use.

When you run workshops that deal with cluster operations, for which users need cluster admin access, create a separate cluster for each user. Learning Center doesn't deal with provisioning clusters, only with deploying a workshop environment in a cluster after it exists.

## Use case requirements

In implementing to the preceding scenarios, the primary requirements related to creation of workshop content, and what you can do at runtime, are as follows:

- You must store everything for the workshop in a Git repository, with no dependency on using a special web application or service to create a workshop.
- Use GitHub as a means to distribute workshop content. Alternatively, you can distribute the workshop as a container image. The latter is necessary if special tools must be installed for use in a workshop.
- Provide instructions to the user to complete the workshop as Markdown or AsciiDoc files.
- You can annotate instructions as executable commands so that when clicked in the workshop dashboard, they execute for the user in the appropriate terminal to avoid

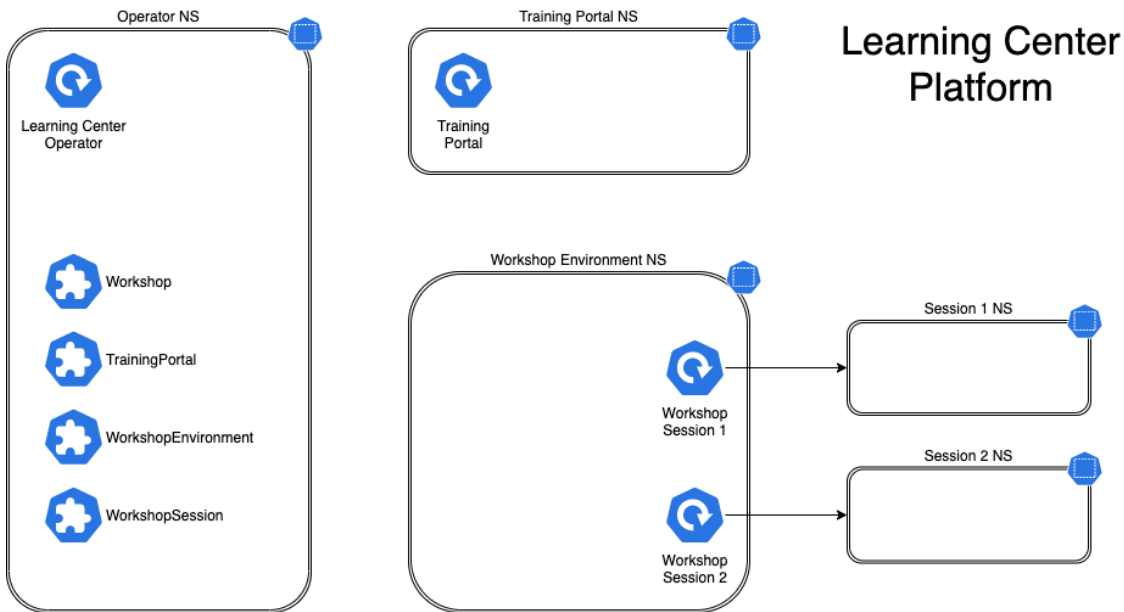
mistakes when commands are entered manually.

- You can annotate text as copyable so when clicked in the workshop dashboard, it is copied into the browser paste buffer ready for pasting into the terminal or other web application.
- Provide each user access to one or more namespaces in the Kubernetes cluster unique to their session. For Kubernetes based workshops, this is where applications are deployed as part of the workshop.
- You can create additional Kubernetes resources specific to a workshop session in advance of the session. This enables the deployment of applications for each user session.
- You can deploy additional Kubernetes resources common to all workshop sessions when the workshop environment is first created. This enables deployment of applications shared by all users.
- Apply resource quotas on each workshop session to control how much resources users can consume.
- Apply role-based access control (RBAC) on each workshop session to control what users can do.
- Provide access to an editor (IDE) in the workshop dashboard in the web browser for users to edit files during the workshop.
- Provide access to a web-based console for accessing the Kubernetes cluster. Use of the Kubernetes dashboard or Octant is supported.
- Ability to integrate additional web-based applications into the workshop dashboard specific to the topic of the workshop.
- Ability for the workshop dashboard to display slides used by an instructor in support of the workshop.

## Platform architectural overview

The Learning Center relies on a Kubernetes Operator to perform the bulk of the work. The actions of the operator are controlled by using a set of custom resources specific to the Learning Center.

There are multiple ways of using the custom resources to deploy workshops. The primary way is to create a training portal, which in turn then triggers the setup of one or more workshop environments, one for each distinct workshop. When users access the training portal and select the workshop they want to do, the training portal allocates to that user a workshop session (creating one if necessary) against the appropriate workshop environment, and the user is redirected to that workshop session instance.



You can associate each workshop session with one or more Kubernetes namespaces specifically for use during that session. Role based access control (RBAC) applied to the unique Kubernetes service account for that session ensures that the user can only access the namespaces and other resources that they are allowed to for that workshop.

In this scenario, the custom resource types that come into play are:

- **Workshop** - Provides the definition of a workshop. Preloaded by an admin into the cluster, it defines where the workshop content is hosted, or the location of a container image which bundles the workshop content and any additional tools required for the workshop. The definition also lists additional resources that must be created which are to be shared between all workshop sessions, or for each session, with details of resources quotas and access roles required by the workshop.
- **TrainingPortal** - Created by an admin in the cluster to trigger the deployment of a training portal. The training portal can provide access to one or more distinct workshops defined by a **Workshop** resource. The training portal provides a web based interface for registering for workshops and accessing them. It also provides a REST API for requesting access to workshops, allowing custom front ends to be created which integrate with separate identity providers and which provide an alternate means for browsing and accessing workshops.
- **WorkshopEnvironment** - Used by the training portal to trigger the creation of a workshop environment for a workshop. This causes the operator to set up a namespace for the workshop into which shared resources are deployed, and where the workshop sessions are run.
- **WorkshopSession** - Used by the training portal to trigger the creation of a workshop session against a specific workshop environment. This causes the operator to set up any namespaces specific to the workshop session and pre-create additional resources required for a workshop session. Workshop sessions can either be created up front in reserve, to be handed out when requested, or created on demand.

## Next steps

Learn more about:

- [Workshops](#)
- [Get started with Learning Center](#)



- [Installing Learning Center](#)
- [Local install guides](#)
- [Air-gapped environment requirements](#)

## Overview of Learning Center for Tanzu Application Platform

Learning Center provides a platform for creating and self-hosting workshops. It allows you to create workshops from markdown files that are displayed to the learner in a terminal shell environment with an instructional wizard UI.

The UI can embed slide content, an integrated development environment (IDE), a web console for accessing the Kubernetes cluster, and other custom web applications.

Although Learning Center requires Kubernetes to run, and is used to teach users about Kubernetes, you can use it to host training for other purposes as well. For example, you can use it to help train users in web-based applications, use of databases, or programming languages, where the user has no interest or need for Kubernetes.

### Use cases

Use case scenarios that Learning Center supports include:

- Supervised workshops. For example, a workshop run at a conference, at a customer site, or online. The workshop has a set time period and you know the maximum number of users to expect. After the training is complete, the Kubernetes cluster created for the workshop is destroyed.
- Temporary learning portal. This is for when you must provide access to a small set of workshops for a short duration for hands on demos at a conference vendor booth. Users select which topic they want to learn about and do that workshop. The workshop instance is created on demand. When they have finished the workshop, that workshop instance is destroyed to free up resources. After the conference has finished, the Kubernetes cluster is destroyed.
- Permanent learning portal. Similar to the temporary learning portal, but runs on an extended basis as a public website where anyone can come and learn at any time.
- Personal training or demos. This is where anyone who wants to run a workshop on their own Kubernetes cluster to learn that topic, or where a product demo was packaged up as a workshop and they want to use it to demonstrate the product to a customer. You can destroy the workshop environment when complete, but there is no need for the cluster to be destroyed.

When running workshops, wherever possible a shared Kubernetes cluster reduces the amount of setup required. This works for developer-focused workshops, because it is usually not necessary to provide elevated access to the Kubernetes cluster, and you can use role-based access controls (RBAC) to prevent users from interfering with each other. You can also set quotas so users are restricted as to how much resources they can use.

When you run workshops that deal with cluster operations, for which users need cluster admin access, create a separate cluster for each user. Learning Center doesn't deal with provisioning clusters, only with deploying a workshop environment in a cluster after it exists.

### Use case requirements

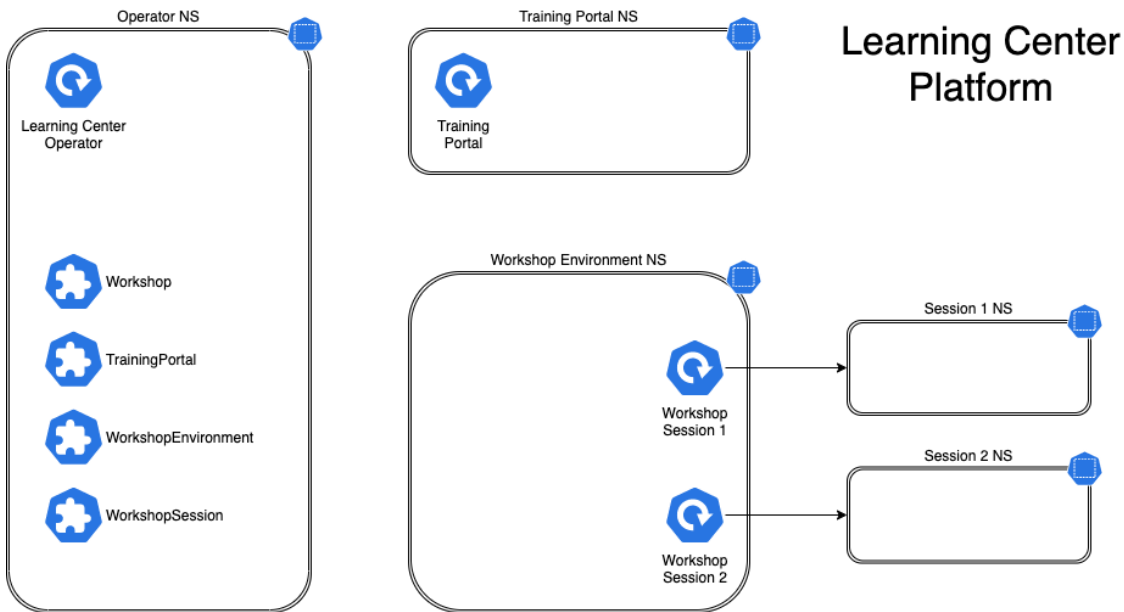
In implementing to the preceding scenarios, the primary requirements related to creation of workshop content, and what you can do at runtime, are as follows:

- You must store everything for the workshop in a Git repository, with no dependency on using a special web application or service to create a workshop.
- Use GitHub as a means to distribute workshop content. Alternatively, you can distribute the workshop as a container image. The latter is necessary if special tools must be installed for use in a workshop.
- Provide instructions to the user to complete the workshop as Markdown or AsciiDoc files.
- You can annotate instructions as executable commands so that when clicked in the workshop dashboard, they execute for the user in the appropriate terminal to avoid mistakes when commands are entered manually.
- You can annotate text as copyable so when clicked in the workshop dashboard, it is copied into the browser paste buffer ready for pasting into the terminal or other web application.
- Provide each user access to one or more namespaces in the Kubernetes cluster unique to their session. For Kubernetes based workshops, this is where applications are deployed as part of the workshop.
- You can create additional Kubernetes resources specific to a workshop session in advance of the session. This enables the deployment of applications for each user session.
- You can deploy additional Kubernetes resources common to all workshop sessions when the workshop environment is first created. This enables deployment of applications shared by all users.
- Apply resource quotas on each workshop session to control how much resources users can consume.
- Apply role-based access control (RBAC) on each workshop session to control what users can do.
- Provide access to an editor (IDE) in the workshop dashboard in the web browser for users to edit files during the workshop.
- Provide access to a web-based console for accessing the Kubernetes cluster. Use of the Kubernetes dashboard or Octant is supported.
- Ability to integrate additional web-based applications into the workshop dashboard specific to the topic of the workshop.
- Ability for the workshop dashboard to display slides used by an instructor in support of the workshop.

## Platform architectural overview

The Learning Center relies on a Kubernetes Operator to perform the bulk of the work. The actions of the operator are controlled by using a set of custom resources specific to the Learning Center.

There are multiple ways of using the custom resources to deploy workshops. The primary way is to create a training portal, which in turn then triggers the setup of one or more workshop environments, one for each distinct workshop. When users access the training portal and select the workshop they want to do, the training portal allocates to that user a workshop session (creating one if necessary) against the appropriate workshop environment, and the user is redirected to that workshop session instance.



You can associate each workshop session with one or more Kubernetes namespaces specifically for use during that session. Role based access control (RBAC) applied to the unique Kubernetes service account for that session ensures that the user can only access the namespaces and other resources that they are allowed to for that workshop.

In this scenario, the custom resource types that come into play are:

- **Workshop** - Provides the definition of a workshop. Preloaded by an admin into the cluster, it defines where the workshop content is hosted, or the location of a container image which bundles the workshop content and any additional tools required for the workshop. The definition also lists additional resources that must be created which are to be shared between all workshop sessions, or for each session, with details of resources quotas and access roles required by the workshop.
- **TrainingPortal** - Created by an admin in the cluster to trigger the deployment of a training portal. The training portal can provide access to one or more distinct workshops defined by a **Workshop** resource. The training portal provides a web based interface for registering for workshops and accessing them. It also provides a REST API for requesting access to workshops, allowing custom front ends to be created which integrate with separate identity providers and which provide an alternate means for browsing and accessing workshops.
- **WorkshopEnvironment** - Used by the training portal to trigger the creation of a workshop environment for a workshop. This causes the operator to set up a namespace for the workshop into which shared resources are deployed, and where the workshop sessions are run.
- **WorkshopSession** - Used by the training portal to trigger the creation of a workshop session against a specific workshop environment. This causes the operator to set up any namespaces specific to the workshop session and pre-create additional resources required for a workshop session. Workshop sessions can either be created up front in reserve, to be handed out when requested, or created on demand.

## Next steps

Learn more about:

- [Workshops](#)
- [Get started with Learning Center](#)

- [Installing Learning Center](#)
- [Local install guides](#)
- [Air-gapped environment requirements](#)

## Install Learning Center

This topic describes how to install Learning Center from the Tanzu Application Platform (commonly known as TAP) package repository.



### Note

Follow the steps in this topic if you do not want to use a profile to install Learning Center. For more information about profiles, see [Components and installation profiles](#).

To install Tanzu Learning Center, see the following sections.

For general information about Learning Center, see [Learning Center](#). For information about deploying Learning Center operator, see [Install and configure the Learning Center operator](#).

## Prerequisites

Before installing Learning Center:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see [Prerequisites](#).
- The cluster must have an ingress router configured. If you have installed Learning Center through a profile, it already deploys a Contour ingress controller.
- The operator, when deploying instances of the workshop environments, must be able to expose them through an external URL for access. For the custom domain you are using, DNS must have been configured with a wildcard domain to forward all requests for sub-domains of the custom domain to the ingress router of the Kubernetes cluster.
- By default, the workshop portal and workshop sessions are accessible over HTTP connections. If you wish to use secure HTTPS connections, you must have access to a wildcard SSL certificate for the domain under which you want to host the workshops. You cannot use a self-signed certificate.
- Any ingress routes created use the default ingress class if you have multiple ingress class types available and you must override which is used.

## Install Learning Center

To install Learning Center:

1. List version information for the package by running:

```
tanzu package available list learningcenter.tanzu.vmware.com --namespace tap-in
stall
```

Example output:

NAME	VERSION	RELEASED-AT
learningcenter.tanzu.vmware.com	0.1.0	2021-12-01 08:18:48 -0500 EDT

- (Optional) See all the configurable parameters on this package by running:

**Remember to change the O.x.x version**

```
tanzu package available get learningcenter.tanzu.vmware.com/0.x.x --values-sche
ma --namespace tap-install
```

- Create a config file named `learning-center-config.yaml`.
- To override the `shared.ingress_domain` in the values file of Tanzu Application Platform, add the parameter `ingressDomain` to `learning-center-config.yaml`. For example:

```
ingressDomain: YOUR-INGRESS-DOMAIN
```

Where `YOUR-INGRESS-DOMAIN` is the domain name for your Kubernetes cluster.

When deploying workshop environment instances, the operator must be able to expose the instances through an external URL. You need this access to discover the domain name that can be used as a suffix to host names for instances.

For the custom domain you are using, DNS must have been configured with a wildcard domain to forward all requests for sub-domains of the custom domain to the ingress router of the Kubernetes cluster.

If you are running Kubernetes on your local machine using a system such as `minikube` and you don't have a custom domain name that maps to the IP address for the cluster, you can use a `nip.io` address. For example, if `minikube ip` returns `192.168.64.1`, you can use the `192.168.64.1.nip.io` domain. You cannot use an address of form `127.0.0.1.nip.io` or `subdomain.localhost`. This causes a failure. Internal services needing to connect to each other connect to themselves instead, because the address resolves to the host loopback address of `127.0.0.1`.

- Add the `ingressSecret` to `learning-center-config.yaml`, as in this example:

```
ingressSecret:
certificate: |
  -----BEGIN CERTIFICATE-----
  MIIFLTCCBBWgAwIBAgASAYS/V2NCTG9uXa9aAiYt7WJ3MA0GCSqGSIb3DQEBCwUA
  ...
  dHa6Ly9yMy5vamxlbmNyLm9yZzAiBggrBgEFBQawAoYWaHR0cDoaL3IzLmkubGVu
  -----END CERTIFICATE-----
privateKey: |
  -----BEGIN PRIVATE KEY-----
  MIIEvQIBADAaBgkqhkiG9wBAQEFAASCBCkwggSjAgEAAoIBAAcX4nyc2xwaVOzf
  ...
  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.

You can create wildcard certificates by using letsencrypt <https://letsencrypt.org/>. After you have the certificate, you can define the `certificate` and `privateKey` properties under the `ingressSecret` property to specify the certificate on the configuration YAML.

- 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
```

- Install Learning Center operator by running:

**Remember to change the 0.x.x version**

```
tanzu package install learning-center --package-name learningcenter.tanzu.vmware.com --version 0.x.x -f learning-center-config.yaml
```

The preceding command creates a default namespace in your Kubernetes cluster called `learningcenter`, and the operator, and any required namespaced resources, are created in it. A set of custom resource definitions and a global cluster role binding are also created.

You can confirm that the operator deployed successfully by running:

```
kubectl get all -n learningcenter
```

The pod for the operator should be marked as running.

## Install the Self-Guided Tour Training Portal and Workshop

To install the Self-Guided Tour Training Portal and Workshop:

- Confirm you have the workshop package installed by running:

```
tanzu package available list workshops.learningcenter.tanzu.vmware.com --namespace tap-install
```

- 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
```

- Check for the Training Portals available in your environment by running:

```
kubectl get trainingportals
```

Example output:

NAME	URL	ADMINURL
learningcenter-tutorials	http://learningcenter-tutorials.example.com	learningcenter
	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

```

```

ed.
network:
  blockCIDRs:
  - 169.254.169.254/32
  - fd00:ec2::254/128

#! clusterSecurity and policyEngine settings are used to change the security policy engine used by Learning Center. Possible values are:
#! "pod-security-policies" - kubernetes <= 1.24
#! "pod-security-standards" - kubernetes >= 1.22
#! "security-context-constraints" - Openshift <= 4.10
#! "security-context-constraints-v2" - Openshift >= 4.11
#! "none" - Disable security - Not recommended

clusterSecurity:
  policyEngine: "pod-security-policies"  #! default value

```

See [Restricting Network Access](#) for more information on blocking CIDRs.

## About Learning Center workshops

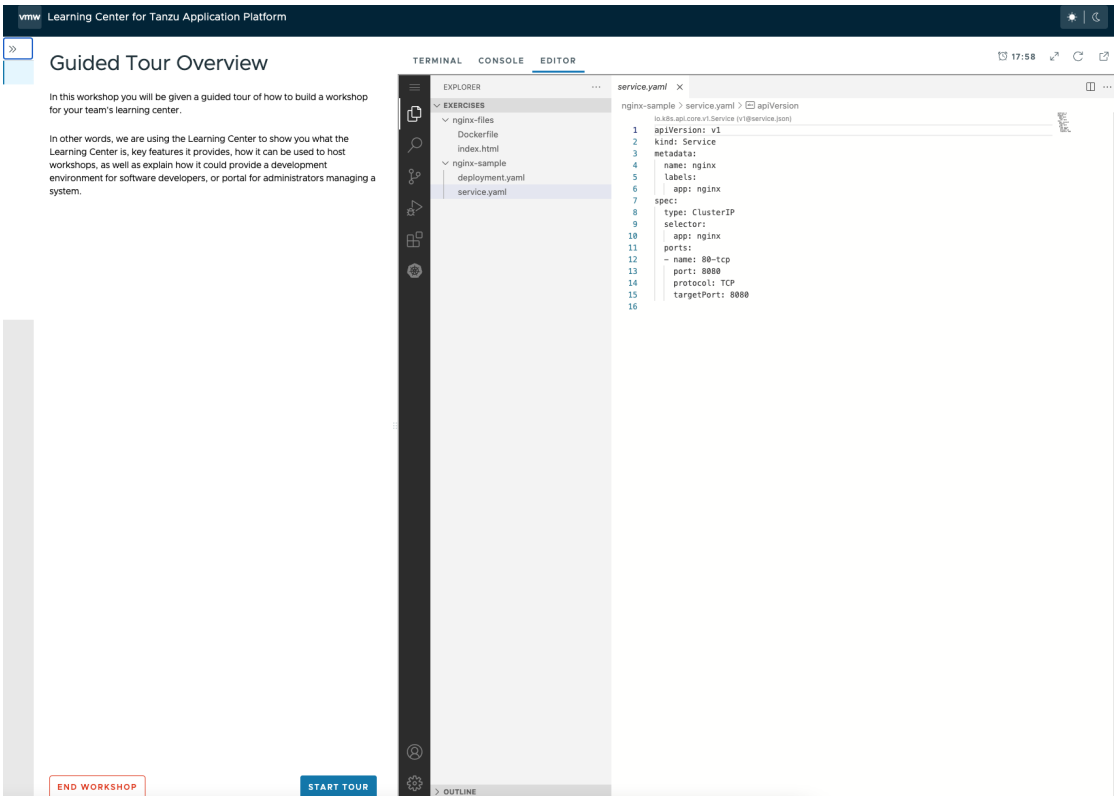
This topic gives you an overview of Learning Center workshops.

The Learning Center workshop dashboard comprises a set of workshop instructions on the left-hand side and a series of tabbed views on the right-hand side. For workshops that require users to run commands, one or more terminal shells are provided. For more information about workshops including creating your own, see [Create workshops](#).

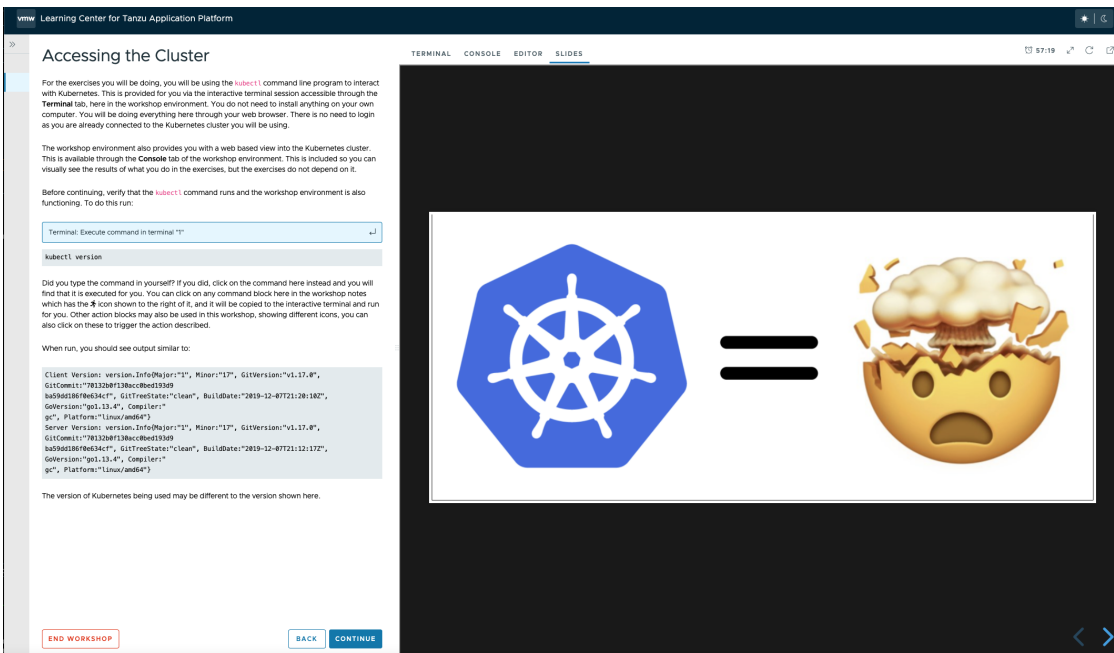
The screenshot displays the Learning Center interface. On the left, the 'Guided Tour Overview' section provides instructions for building a workshop. The main area on the right is a terminal window showing the output of the 'kubectl version' command. The output includes client and server version information, build dates, and a warning about version skew. The terminal window is numbered '1' and '2' in the bottom right corner. At the bottom of the terminal area, there is a 'START TOUR' button.

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.

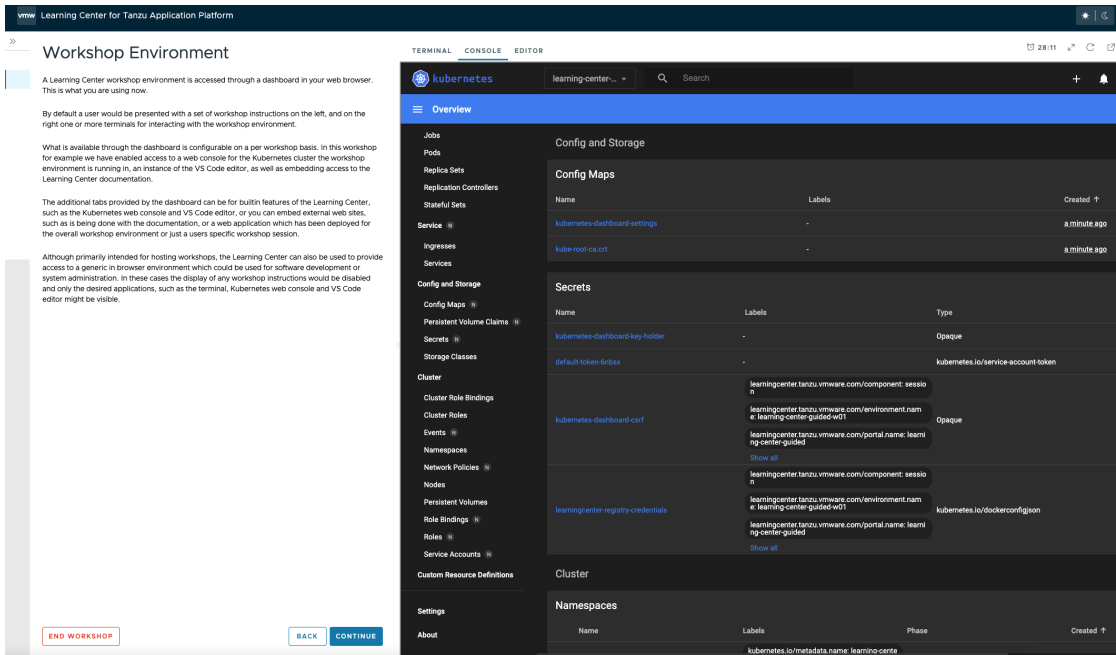




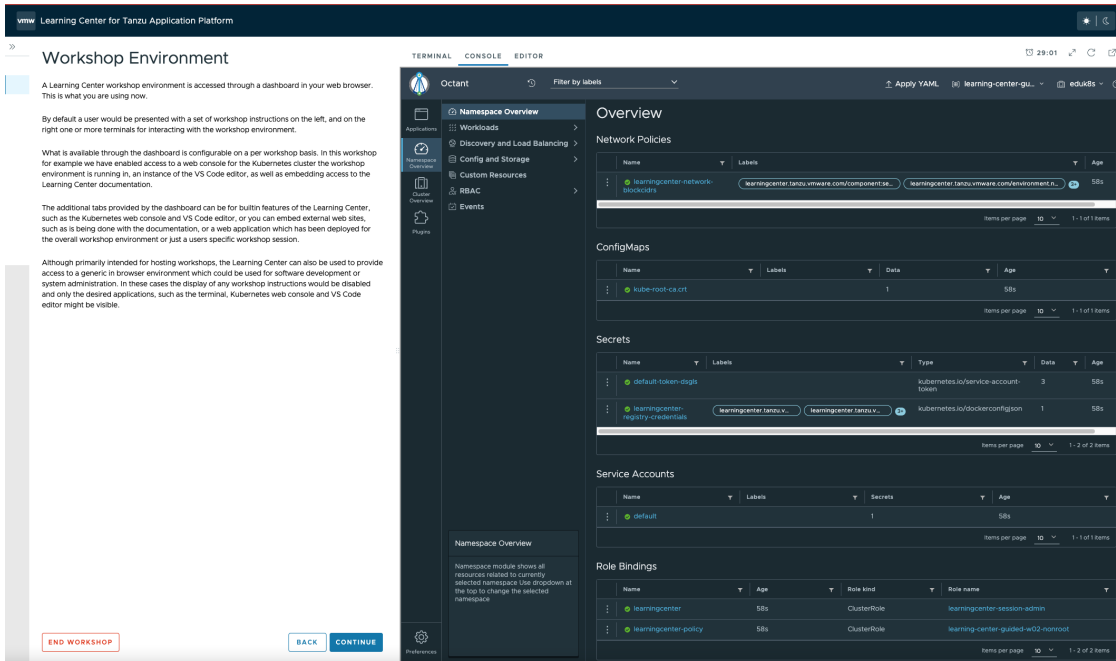
To complement the workshop instructions, or to be available for use by the instructor, you can include slides with a workshop. For slides you can use HTML based slide presentation tools such as [reveal.js](#), or you can embed a PDF file.



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.



Alternatively, you can enable Octant as the web console.



## Get started with Learning Center

This topic describes how you can get started with Learning Center for Tanzu Application Platform. For information about Learning Center and its use cases, see [Learning Center for Tanzu Application Platform](#).

## Installing Learning Center

Before deploying workshops, you must install a Kubernetes operator for Learning Center. The operator manages the setup of the environment for each workshop and deploys instances of a workshop for each person.

For basic information about installing the Learning Center operator, see [Install Learning Center](#).

## Get started

See the following useful information about getting started with Learning Center:

- [Install and configure the Learning Center operator](#)
- [Get started with workshops](#)
- [Get started with training portals](#)
- [Delete an operator](#)

## Get started with Learning Center

This topic describes how you can get started with Learning Center for Tanzu Application Platform. For information about Learning Center and its use cases, see [Learning Center for Tanzu Application Platform](#).

## Installing Learning Center

Before deploying workshops, you must install a Kubernetes operator for Learning Center. The operator manages the setup of the environment for each workshop and deploys instances of a workshop for each person.

For basic information about installing the Learning Center operator, see [Install Learning Center](#).

## Get started

See the following useful information about getting started with Learning Center:

- [Install and configure the Learning Center operator](#)
- [Get started with workshops](#)
- [Get started with training portals](#)
- [Delete an operator](#)

## Install and configure the Learning Center operator

This topic gives you information about installing and configuring the Learning Center operator.

Before deploying workshops, you must install a Kubernetes operator for Learning Center. The operator manages the setup of the environment for each workshop and deploys instances of a workshop for each person.

For basic information about installing the operator, see [Install Learning Center](#).

## Installing and setting up Learning Center operator

You can deploy the Learning Center operator to any Kubernetes cluster supporting custom resource definitions and the concept of operators. The cluster must have an ingress router configured, though only a basic deployment of the ingress controller is usually required. You do not need to configure the ingress controller to handle cluster wide edge termination of secure HTTP connections. Learning Center creates Kubernetes Ingress resources and supplies any secret for use with secure HTTP connections for each ingress.

For the ingress controller, VMware recommends the use of Contour over alternatives such as nginx. An nginx-based ingress controller has a less than optimal design. Every time a new ingress is created or deleted, the nginx config is reloaded. This causes websocket connections to terminate

after a period of time. Learning Center terminals reconnect automatically in the case of the websocket connection being lost. However, not all applications you might use with specific workshops can handle loss of websocket connections so gracefully, and they might be impacted due to the use of an nginx ingress controller. This problem is not specific to Learning Center. It can impact any application when an nginx ingress controller is used frequently and ingresses are created or deleted frequently.

You can use a hosted Kubernetes solution from an IaaS provider such as Google, AWS, or Azure. If you do, as needed increase any HTTP request timeout specified on the inbound load balancer for the ingress controller so that you can use long-lived websocket connections. In some cases, load balancers of hosted Kubernetes solutions only have a 30-second timeout. If possible, configure the timeout applying to websockets to be 1 hour.

If you deploy the web-based training portal, the cluster must have available persistent volumes of type `ReadWriteOnce (RWO)`. A default storage class must be defined so that persistent volume claims do not need to specify a storage class. For some Kubernetes distributions, including from IBM, you must configure Learning Center as to what user and group must be used for persistent volumes. If no default storage class is specified, or a specified storage class is required, you can configure Learning Center with the name of the storage class.

To install the Learning Center operator, you must have cluster admin access.

## Cluster pod security policies

The Learning Center operator defines pod security policies to limit what users can do from workshops when deploying workloads to the cluster. The default policy prohibits running of images as the `root` user or using a privileged pod. Specified workshops can relax these restrictions and apply a policy that enables additional privileges required by the workshop.

To enforce a security policy around what a user can do, different mechanisms have been provided with standard Kubernetes distributions and derivatives such as OpenShift. These are:

- Pod security policies (Kubernetes <= 1.25)
- Pod security standards (Kubernetes >= 1.22)
- Security context constraints (OpenShift)

For pod security policies and pod security standards, these both must be enabled in the Kubernetes cluster at the time the cluster is created. They cannot be enabled afterwards. For some Kubernetes distributions, such as Tanzu Community Edition, it is not possible to enable pod security policies. Because pod security standards are new, they might also not be supported.

VMware recommends that the pod security policy admission controller be enabled for the cluster to ensure that the pod security policies are applied. If the admission controller is not enabled, users can deploy workloads that run as the `root` user in a container, or run privileged pods.

If you are unable to enable the pod security policy admission controller, you should only provide access to workshops deployed using the Learning Center operator to users you trust.

Whether the absence of the pod security policy admission controller causes issues with access to persistent volumes depends on the cluster. Although minikube does not enable the pod security policy admission controller, it works as persistent volumes when mounted to give write permissions to all users.

No matter whether pod security policies are enabled, individual workshops must be reviewed as to what added privileges they grant before allowing their use in a cluster.

## Specifying the ingress domain

When deploying instances of workshop environments, the operator must expose the instances by using an external URL for access to define the domain name that is used as a suffix to host names for instances.



#### Note

For the custom domain you are using, configure your DNS with a wildcard domain to forward all requests for subdomains of the custom domain to the ingress router of the Kubernetes cluster.

VMware recommends that you avoid using a `.dev` or `.app` domain name, because such domain names require browsers to use HTTPS and not HTTP. Although you can provide a certificate for secure connections under the domain name for use by Learning Center, this doesn't extend to what a workshop may do. If workshop instructions require that you create ingresses in Kubernetes using HTTP only, a `.dev` or `.app` domain name cannot work in the browser.



#### Note

If you are running Kubernetes on your local machine using a system such as `minikube` and you don't have a custom domain name that maps to the IP address for the cluster, you can use a `nip.io` address. For example, if `minikube ip` returned `192.168.64.1`, you can use the `192.168.64.1.nip.io` domain. You cannot use an address of form `127.0.0.1.nip.io` or `subdomain.localhost`. This causes a failure as internal services needing to connect to each other end up connecting to themselves instead, because the address resolves to the host loopback address of `127.0.0.1`.

If needed, you can override the `shared.ingress_domain` in the values file of Tanzu Application Platform with the `ingressDomain` parameter of learning center:

```
ingressDomain: learningcenter.my-domain.com
```

## Set the environment variable manually

Set the `INGRESS_DOMAIN` environment variable on the operator deployment. To set the `INGRESS_DOMAIN` environment variable, run:

```
kubectl set env deployment/learningcenter-operator -n learningcenter INGRESS_DOMAIN=test
```

Where `test` is the domain name for your Kubernetes cluster.

Or if using a `nip.io` address:

```
kubectl set env deployment/learningcenter-operator -n learningcenter INGRESS_DOMAIN=192.168.64.1.nip.io
```

Use of environment variables to configure the operator is a shortcut for a simple use. VMware recommends using Tanzu CLI, or for more complicated scenarios, you can use the `SystemProfile` custom resource.

## Enforcing secure connections

By default, the workshop portal and workshop sessions are accessible over HTTP connections. To use secure HTTPS connections, you must have access to a wildcard SSL certificate for the domain under which you want to host the workshops. You cannot use a self-signed certificate.

You can create wildcard certificates by using `letsencrypt` <<https://letsencrypt.org/>>. After you have the certificate, you can define it as follows.

## Configuration YAML

The easiest way to define the certificate is with the configuration passed to Tanzu CLI. So define the `certificate` and `privateKey` properties under the `ingressSecret` property to specify the certificate on the configuration YAML passed to Tanzu CLI:

```
ingressSecret:
  certificate: |
    -----BEGIN CERTIFICATE-----
    MIIFLTCCBBWgAwIBAgSAys/V2NCTG9uXa9aAiYt7WJ3MA0GCSqGSIb3DQEBCwUA
    ...
    dHa6Ly9yMy5vamxlbmNyLm9yZzAiBggrBgEFBQawAoYWAHR0cDoaL3IzLmkubGVu
    -----END CERTIFICATE-----
  privateKey: |
    -----BEGIN PRIVATE KEY-----
    MIEvQIBADAaBgkqhkiG9wBAQEFAASCBCwggSjAgEAAoIBAAcX4nyc2xwaVOzf
    ...
    IY/9SatMcJZivH3F1a7SXL98PawPIOSR7986P7rLFHzNjaQQ0DWTaXBrt+oUDxpN
    -----END PRIVATE KEY-----
```

If you already have a TLS secret, follow these steps **before deploying any workshops**:

1. Create the `learningcenter` namespace manually or the one you defined.
2. Copy the TLS secret to the `learningcenter` namespace or to the one you defined, and use the `secretName` property as in this example:

```
ingressSecret:
  secretName: workshops.example.com-tls
```

## Create the TLS secret manually

To add the certificate as a secret in the `learningcenter` namespace or in the one you defined, the secret must be of type `tls`. You can create it using the `kubect1 create secret tls` command:

```
kubect1 create secret tls -n learningcenter workshops.example.com-tls --cert=workshop
s.example.com/fullchain.pem --key=workshops.example.com/privkey.pem
```

Having created the secret, if it is the secret corresponding to the default ingress domain you specified earlier, set the `INGRESS_SECRET` environment variable. This way you do not use the configuration passed to Tanzu CLI on the operator deployment. This ensures the secret is applied automatically to any ingress created:

```
kubect1 set env deployment/learningcenter-operator -n learningcenter INGRESS_SECRET=wo
rkshops.example.com-tls
```

If the certificate isn't that of the default ingress domain, you can supply the domain name and name of the secret when creating a workshop environment or training portal. In either case, you must create secrets for the wildcard certificates in the `learningcenter` namespace or the one that you defined.

## Specifying the ingress class

Any ingress routes created use the default ingress class. If you have multiple ingress class types available, and you must override which is used, you can define the `ingressClass` property on the configuration YAML as follows.

## Configuration YAML

Define the `ingressClass` property on the configuration YAML passed to Tanzu CLI:

```
ingressClass: contour
```

## Set the environment variable manually

Set the `INGRESS_CLASS` environment variable for the learningcenter operator:

```
kubectl set env deployment/learningcenter-operator -n learningcenter INGRESS_CLASS=contour
```

This applies only to the ingress created for the training portal and workshop sessions. It does not apply to any ingress created from a workshop as part of the workshop instructions.

This can be necessary when a specific ingress provider is not reliable in maintaining websocket connections. For example, in the case of the nginx ingress controller when there are frequent creation or deletions of ingresses occurring in the cluster. See the earlier section, [Installing and setting up Learning Center operator](#).

## Trusting unsecured registries

One of the options available for workshops is to automatically deploy a container image registry each workshop session. When the Learning Center operator is configured to use a secure ingress with a valid wildcard certificate, the image registry works out of the box.

If the Learning Center operator is not set up to use secure ingress, the image registry is accessed over HTTP and is regarded as not secure.

When using the optional support for building container images using `docker`, the docker daemon deployed for the workshop session is configured for the image registry being not secure yet pushing images to the image registry still works.

In this case of an image registry that is not secure, deploying images from the image registry to the Kubernetes cluster does not work unless the Kubernetes cluster is configured to trust the registry that is not secure.

How you configure a Kubernetes cluster to trust an unsecured registry varies based on how the Kubernetes cluster is deployed and what container runtime it uses.

If you are using `minikube` with `dockerd`, to ensure that the registry is trusted, you must set up the trust the first time you create the minikube instance.

To do this, first determine which IP subnet minikube uses for the inbound ingress router of the cluster. If you already have a minikube instance running, you can determine this by running `minikube ip`. If, for example, this reported `192.168.64.1`, the subnet used is `129.168.64.0/24`.

With this information, when you create a fresh `minikube` instance, you must supply the `--insecure-registry` option with the subnet:

```
minikube start --insecure-registry="129.168.64.0/24"
```

This option tells `dockerd` to regard as not secure any image registry deployed in the Kubernetes cluster and accessed through a URL exposed using an ingress route of the cluster itself.

Currently, there is no way to configure `containerd` to treat as not secure image registries that match a wildcard subdomain or reside in a subnet. It is therefore not possible to run workshops that must deploy images from the per session image registry when using `containerd` as the underlying Kubernetes cluster container runtime. This is a limitation of `containerd`, and there are no known plans for `containerd` to support this ability. This limits your ability to use Kubernetes clusters deployed with a tool such as `kind`, which relies on using `containerd`.

## Get started with Learning Center workshops

This topic helps you to get started working with Learning Center workshops. Workshops are where you create your content. You can create a workshop for individual use or group multiple workshops together with a [Training Portal](#).

For more detailed instructions, go to [Working with Learning Center Workshops](#)

## Creating the workshop environment

With the definition of a workshop already in existence, the first step to deploying a workshop is to create the workshop environment.

To create the workshop environment run:

```
kubectl apply -f {YOUR-GIT-REPO-URL}/lab-k8s-fundamentals/main/resources/workshop-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 URL	NAMESPACE	WORKSHOP	IMAGE
lab-k8s-fundamentals /lab-k8s-fundamentals:main	lab-k8s-fundamentals	lab-k8s-fundamentals	{YOUR-REGISTRY-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	http://lab-k8s-fundamentals-cvh51.test	learningcenter	buQ OgZvfHM7m

The additional fields provide the URL where the workshop instance can be accessed and the username and password for you to provide when prompted by your web browser.

The user name and password only come into play when you use the lower-level resources to set up workshops. If you use the [TrainingPortal](#) custom resource, you will see that these fields are empty. This is because, for that case, the workshop instances are deployed so that they rely on user registration and access mediated by the web-based training portal. Visiting the URL for a workshop instance directly when using [TrainingPortal](#), redirects you back to the web portal in order to log in if necessary.

You can monitor the progress of this workshop deployment by listing the deployments in the namespace created for the workshop environment:

```
kubectl get all -n lab-k8s-fundamentals
```

For each workshop instance a separate namespace is created for the session. This is linked to the workshop instance, and is where any applications are deployed as part of the workshop. If the definition of the workshop includes a set of resources that should be automatically created for each session namespace, they are created by the Learning Center Operator. It is therefore possible to pre-deploy applications for each session.

In this case, we used [WorkshopRequest](#); whereas when using [TrainingPortal](#), we created a [WorkshopSession](#). The workshop request does result in creating a [WorkshopSession](#), but [TrainingPortal](#) skips the workshop request and directly creates a [WorkshopSession](#).

The purpose of having [WorkshopRequest](#) as a separate custom resource is to allow RBAC and other controls to be used to allow non-cluster administrators to create workshop instances.

## Deleting the workshop instance

When you have finished with the workshop instance, you can delete it by deleting the custom resource for the workshop request:

```
kubectl delete workshoprequest/lab-k8s-fundamentals
```

## Deleting the workshop environment

If you want to delete the whole workshop environment, it is recommended to first delete all workshop instances. Once this has been done, you can then delete the custom resource for the workshop environment:

```
kubectl delete workshopenvironment/lab-k8s-fundamentals
```

If you don't delete the custom resources for the workshop requests, the workshop instances are still cleaned up and removed when the workshop environment is removed. The custom resources for the workshop requests still remain, however, and need to be deleted separately.

## Get started with Learning Center training portals

This topic describes how you configure and use a [TrainingPortal](#), which deploys a set of workshops for attendees.

## Working with multiple workshops

The quickest way to deploy a set of workshops to use in a training session is to deploy a [TrainingPortal](#). This deploys a set of workshops with one instance of each workshop for each attendee. A web-based portal is provided for registering attendees and allocating them to workshops.

The [TrainingPortal](#) custom resource provides a high-level mechanism for creating a set of workshop environments and populating it with workshop instances. When the Learning Center operator processes this custom resource, it creates other custom resources to trigger the creation of the workshop environment and the workshop instances. If you want more control, you can use these latter custom resources directly instead.

## Loading the workshop definition

A custom resource of type [Workshop](#) describes each workshop. Before you can create a workshop environment, you must load the definition of the workshop.

Here is an example [Workshop](#) custom resource:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-k8s-fundamentals
spec:
  title: Kubernetes Fundamentals
  description: Workshop on getting started with Kubernetes
  url: {YOUR-GIT-REPO-URL}/lab-k8s-fundamentals
  vendor: learningcenter.io
  authors:
  - Graham Dumbleton
  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
PASSWORD		
lab-k8s-fundamentals-w01-s001	http://lab-k8s-fundamentals-w01-s001.test	

Only one workshop instance is created. Though the maximum capacity is set to three, the reserved number of instances (hot spares) is defined as one. Additional workshops instances are only created as workshop sessions are allocated to users. One reserved instance is always maintained until capacity is reached.

If you need a different number of workshop instances, set the `portal.capacity` field of the [TrainingPortal](#) custom resource YAML input file before creating the resource. Changing the values after the resource is created has no effect.

In this case, only one workshop is listed to be hosted by the training portal. You can deploy more than one workshop at the same time by adding the names of other workshops to [workshops](#).

The first time you deploy the workshop, it can take a few moments to pull down the workshop image and start.

To access the workshops, attendees of a training session need to visit the web-based portal for the training session. Find the URL for the web portal by running:

```
kubectl get trainingportals
```

This should yield output similar to:

NAME	URL	ADMINUSERNAME	ADMINPASSWO
RD			
lab-k8s-fundamentals	https://lab-k8s-fundamentals-ui.test	learningcenter	mGI
2C1TkHEBoFgKiZetxMnwAldRU80aN			

Attendees should only be given the URL. The password listed is only for use by the instructor of the training session if required.

## Accessing workshops via the web portal

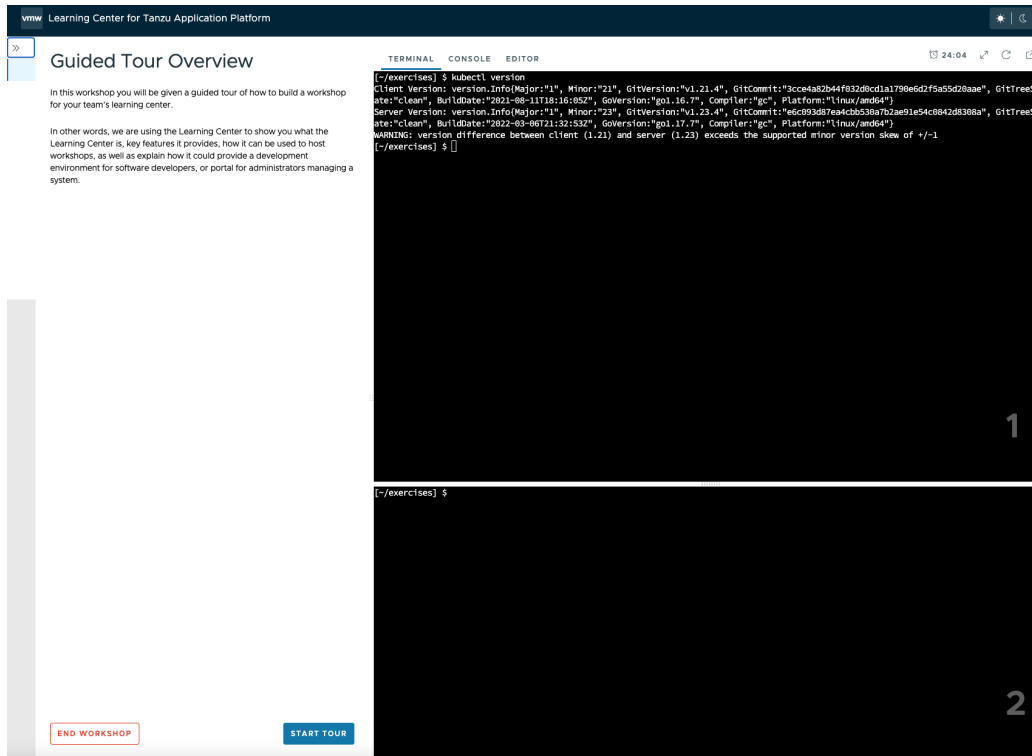
Attendees can access workshops through the web portal by following two steps:

1. The attendee visits the web-based portal for the training session and is presented with a login page. However, before logging in, the attendee must register for an account. The attendee clicks the link to the registration page and fills it in.

Registration is required so if the attendee’s web browser exits or the attendee needs to switch web browsers, the attendee can log in again and access the same workshop instance.

2. Upon registering, the attendee is presented with a list of workshops available for the training session.
  - o 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.
  - o A red dot indicates there are no more workshop instances available.
  - o 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
```

## Delete Learning Center

This topic describes how you can delete Learning Center.

1. Delete all current workshop environments by running:

```
kubectl delete workshops,trainingportals,workshoprequests,workshopsessions,workshopenvironments --all
```

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
```

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
```

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.

- 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
```

## Local install guides

The following topics describe how you install Learning Center on your local environment:

- [Install on Kind](#)
- [Install on Minikube](#)

## Local install guides

The following topics describe how you install Learning Center on your local environment:

- [Install on Kind](#)
- [Install on Minikube](#)

## Install Learning Center on Kind

This topic describes how you install Learning Center on your local machine with Kind.

Kind was developed as a means to support development and testing of Kubernetes. Though it exists primarily for that purpose, Kind clusters are often used for local development of user applications as well. For Learning Center, you can use a local Kind cluster to develop workshop content or self-learning when deploying other people's workshops.

Because you are deploying to a local machine, you are unlikely to have access to your own custom domain name and certificate you can use with the cluster. If you don't, you can be restricted as to the sorts of workshops you can develop or run using the Learning Center in Kind. Kind uses `containerd`, which lacks certain features that allow you to trust any image registries hosted within a subnet. This means you cannot readily run workshops that use a local container image registry for each workshop session. If you must run workshops on your local computer that uses an image registry for each session, VMware recommends you use minikube with `dockerd` instead. For more information, see [Install 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 VMware Tanzu Network account and have access to your Tanzu Network credentials.
- Install Kind on your local machine.
- Install Tanzu CLI on your local machine.
- Install Kubernetes command-line tool (kubectl) 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
```

For information about installing Contour, which comes with Tanzu Application Platform, see [Install cert-manager](#), [Contour](#).



## 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, `1.4.13`.



### Note

We are currently on build 7. If this changes, we need to update the command with the correct build version after the `-url` flag.

4. To check the package repository install status, run:

```
tanzu package repository get tanzu-tap-repository --namespace tap-install
```

Wait for a reconciled successful status before attempting to install any other packages.

## Create a configuration YAML file for Learning Center package

To create a configuration YAML file:

See [Supported yaml file configurations](#) to see a list of configurations you can provide to Learning Center.

1. Create a file called `learningcenter-value.yaml` in your current directory with the following data:

```
ingressDomain: workshops.example.com
```

Where:

- `ingressDomain` is `<your-local-ip>.nip.io` if you are using a `nip.io` DNS address. Details about this are provided in the following section.
- `workshops.example.com` with is `<your-local-ip>.nip.io`.

## Using a `nip.io` DNS address

Before you can start deploying workshops, you must configure the operator to tell it what domain name can be used to access anything deployed by the operator.

Being a local cluster that isn't exposed to the Internet with its own custom domain name, you can use a `nip.io` address.

To calculate the `nip.io` address to use, first work out the IP address for the ingress controller exposed by Kind. This is usually the IP address of the local machine itself, even when you use Docker for Mac.

How you get the IP address for your local machine depends on the operating system you are using.

For example on a Mac, you can find your IP address by searching for network using spotlight and selecting the network option under system preferences. Here you can see your IP address under status.

After you have the IP address, add this as a prefix to the domain name `nip.io`. For example, if the address was `192.168.1.1`, use the domain name of `192.168.1.1.nip.io`.

To configure the Learning Center operator with this cluster domain, run:

```
kubectl set env deployment/learningcenter-operator -n eduk8s INGRESS_DOMAIN=192.168.1.1.nip.io
```

This causes the Learning Center operator to redeploy with the new configuration. You can now deploy workshops.



### Note

Some home Internet gateways implement what is called rebind protection. These gateways do not allow DNS names from the public Internet bind to local IP address ranges inside the home network. If your home Internet gateway has such a feature and it is enabled, it blocks `nip.io` addresses from working. In this case, you must configure your home Internet gateway to allow `*.nip.io` names to be bound to local addresses. Also, you cannot use an address of form `127.0.0.1.nip.io` or `subdomain.localhost`. This causes a failure, because when internal services need to

connect to each other, they connect to themselves instead. This happens because the address resolves to the host loopback address of `127.0.0.1`.

## Install Learning Center package onto a Kubernetes cluster

To install Learning Center on a Kubernetes cluster:

```
tanzu package install learningcenter --package-name learningcenter.tanzu.vmware.com --version 0.1.0 -f ./learningcenter-value.yaml --namespace tap-install
```

This package installation uses the installed Package repository with a configuration `learningcenter-value.yaml` to install our Learning Center package.

## Install workshop tutorial package onto a Kubernetes cluster

To install a workshop tutorial on a Kubernetes cluster:

```
tanzu package install learningcenter-tutorials --package-name workshops.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-r
egistry.192.168.1.1.nip.io"]
      endpoint = ["http://lab-docker-testing-w01-s001-registry.192.168.1.1.nip.io"]
    [plugins."io.containerd.grpc.v1.cri".registry.mirrors."lab-docker-testing-w01-s002-r
egistry.192.168.1.1.nip.io"]
      endpoint = ["http://lab-docker-testing-w01-s002-registry.192.168.1.1.nip.io"]
    [plugins."io.containerd.grpc.v1.cri".registry.mirrors."lab-docker-testing-w01-s003-r
egistry.192.168.1.1.nip.io"]
      endpoint = ["http://lab-docker-testing-w01-s003-registry.192.168.1.1.nip.io"]
    [plugins."io.containerd.grpc.v1.cri".registry.mirrors."lab-docker-testing-w01-s004-r
egistry.192.168.1.1.nip.io"]
      endpoint = ["http://lab-docker-testing-w01-s004-registry.192.168.1.1.nip.io"]
    [plugins."io.containerd.grpc.v1.cri".registry.mirrors."lab-docker-testing-w01-s005-r
egistry.192.168.1.1.nip.io"]
      endpoint = ["http://lab-docker-testing-w01-s005-registry.192.168.1.1.nip.io"]
EOF
```

This allows you to run five workshop sessions before you have to delete the training portal and recreate it.

If you use this, you can use the feature of the training portal to automatically update when a workshop definition is changed. This is because the **wMM** value identifying the workshop environment changes any time you update the workshop definition.

There is no other known workaround for this limitation of **containerd**. As such, VMware recommends you use minikube with **dockerd** instead. For more information, see [Install on Minikube](#).

## Install Learning Center on Minikube

This topic describes how you install Learning Center on your local machine with Minikube.

Minikube enables local deployment of Kubernetes for developing workshop content or for self-learning when deploying other people's workshops.

Because you are deploying to a local machine, you are unlikely to have access to your own custom domain name and certificate you can use with the cluster. You must take extra steps over a standard install of Minikube to ensure you can run certain types of workshops.

Also, because Minikube generally has limited memory resources available and is only a single-node cluster, you might be restricted from running workshops that have large memory requirements or that demonstrate the use of third-party applications requiring a multinode cluster.

Requirements and setup instructions specific to Minikube are detailed in this document. Otherwise, you can follow normal installation instructions for the Learning Center operator.

## Trusting insecure registries

Workshops can optionally deploy a container image registry for a workshop session. This image registry is secured with a password specific to the workshop session and is exposed through a Kubernetes ingress so it can be accessed from the workshop session.

In a typical scenario, Minikube uses insecure ingress routes. Even were you to generate a self-signed certificate to use for ingress, it is not trusted by `dockerd` that runs within Minikube. You must tell Minikube to trust any insecure registry running inside of Minikube.

You must configure Minikube to trust insecure registries the first time you start a new cluster with it. That is, you must supply the details to `minikube start`, which means you must know the IP subnet Minikube uses.

If you already have a cluster running using Minikube, run `minikube ip` to discover the IP address it uses. From that you can discover the trusted subnet. For example, if `minikube ip` returned `192.168.64.1`, the trusted subnet is `192.168.64.0/24`.

With this information, when you start a new cluster with Minikube, run:

```
minikube start --insecure-registry=192.168.64.0/24
```

If you already have a cluster started with Minikube, you cannot stop it and then provide this option when it is restarted. You can only use this option for a completely new cluster.

You must also use `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
```

Type “y” and enter to continue when prompted during installation of both kapp and secret-gen controllers.

## Install Tanzu package repository

Follow these steps to install the Tanzu package repository:

1. To create a namespace, run:

```
kubectl create ns tap-install
```

2. Create a registry secret:

```
tanzu secret registry add tap-registry \
  --username "TANZU-NET-USER" --password "TANZU-NET-PASSWORD" \
  --server registry.tanzu.vmware.com \
  --export-to-all-namespaces --yes --namespace tap-install
```

Where:

- `TANZU-NET-USER` and `TANZU-NET-PASSWORD` are your credentials for Tanzu Network.

3. Add a package repository to your cluster:

```
tanzu package repository add tanzu-tap-repository \
  --url registry.tanzu.vmware.com/tanzu-application-platform/tap-packages:VERSION-NUMBER \
  --namespace tap-install
```

Where `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.4.13`.



#### 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.

- 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:

See [Supported yaml file configurations](#) to see a list of configurations you can provide to Learning Center.

```
ingressDomain: workshops.example.com
```

Where:

- `ingressDomain` is `<your-local-ip>.nip.io` if you are using a `nip.io` DNS address. Details about this are provided in the following section.
- `workshops.example.com` is `<your-local-ip>.nip.io`

## Using a `nip.io` DNS address

After the Learning Center operator is installed, before you can start deploying workshops, you must configure the operator to tell it what domain name can be used to access anything deployed by the operator.

Being a local cluster that isn't exposed to the Internet with its own custom domain name, you can use a `nip.io` address.

To calculate the `nip.io` address to use, first work out the IP address of the cluster created by Minikube by running `minikube ip`. Add this as a prefix to the domain name `nip.io`. For example, if `minikube ip` returns `192.168.64.1`, use the domain name of `192.168.64.1.nip.io`.

To configure the Learning Center operator with this cluster domain, run:

```
kubectl set env deployment/learningcenter-operator -n learningcenter INGRESS_DOMAIN=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.

## Create workshops for Learning Center

This section provides information about how you create Learning Center workshops.

- [Workshop configuration](#)
- [Workshop images](#)
- [Workshop content](#)
- [Build an image](#)
- [Workshop instructions](#)
- [Workshop runtime](#)
- [Workshop slides](#)
- [Air-gapped environment requirements](#)

## Create workshops for Learning Center

This section provides information about how you create Learning Center workshops.

- [Workshop configuration](#)
- [Workshop images](#)
- [Workshop content](#)
- [Build an image](#)
- [Workshop instructions](#)

- [Workshop runtime](#)
- [Workshop slides](#)
- [Air-gapped environment requirements](#)

## Configure your Learning Center workshop

This topic describes the two main steps required to configure your Learning Center workshop. The first specifies the structure of the workshop content and the second defines the runtime requirements for deploying the workshop.

### Specifying structure of the content

There are multiple ways you can configure a workshop to specify the structure of the content. The sample workshops use YAML files.

The `workshop/modules.yaml` file provides details about the list of available modules that make up your workshop and data variables for use in content.

The list of available modules represents all of the modules available to you. You might not use all of them. You might want to run variations of your workshop, such as for different programming languages. As such, which modules are active and are used for a specific workshop are listed in the separate `workshop/workshop.yaml` file. The active modules are listed with the name to be given to that workshop.

By default the `workshop.yaml` file specifies what modules are used. When you want to deliver different variations of the workshop content, you can provide multiple workshop files with different names. For example, you can name the workshop files `workshop-java.yaml` and `workshop-python.yaml`.

Where you have multiple workshop files and don't have the default `workshop.yaml` file, you can specify the default workshop file by setting the `WORKSHOP_FILE` environment variable in the runtime configuration.

The format for listing the available modules in the `workshop/modules.yaml` file is:

```
modules:
  workshop-overview:
    name: Workshop Overview
    exit_sign: Setup Environment
  setup-environment:
    name: Setup Environment
    exit_sign: Start Workshop
  exercises/01-sample-content:
    name: Sample Content
  workshop-summary:
    name: Workshop Summary
    exit_sign: Finish Workshop
```

Each available module is listed under `modules`, where the name used corresponds to the path to the file containing the content for that module. Any extension identifying the content type is left off.

For each module, set the `name` field to the page title to be displayed for that module. If no fields are provided and `name` is not set, the title for the module is derived from the name of the module file.

The corresponding `workshop/workshop.yaml` file, where all available modules are used, would have the format:

```

name: Markdown Sample
modules:
  activate:
    - workshop-overview
    - setup-environment
    - exercises/01-sample-content
    - workshop-summary

```

The top-level `name` field in this file is the name of this variation of the workshop content.

The `modules.activate` field is a list of modules to be used for the workshop. The names in this list must match the names as they appear in the modules file.

The order in which modules are listed under the `modules.activate` field in the workshop configuration file dictates the order pages are traversed. The order in which modules appear in the modules configuration file is not relevant.

At the bottom of each page, a **Continue** button is displayed to allow the user to go to the next page in sequence. You can customize the label on this button by setting the `exit_sign` field in the entry for the module in the modules configuration file.

In the last module in the workshop, a button is displayed, but where the user goes after clicking it varies. If you want the user to go to a different website upon completion, you can set the `exit_link` field of the final module to an external URL. Alternatively, you can set the `RESTART_URL` environment variable in a workshop environment to control where the user goes. If a destination for the final page is not provided, the user is redirected back to the starting page of the workshop.

When the user uses the training portal, the training portal overrides this environment variable so, at the completion of a workshop, the user returns to the training portal.

VMware recommends that for the last page, the `exit_sign` be set to “Finish Workshop” and `exit_link` not be specified. This enables the destination to be controlled from the workshop environment or training portal.

## Specifying the runtime configuration

You can deploy workshop images directly to a container runtime. The Learning Center Operator is provided to manage deployments into a Kubernetes cluster. You define the configuration for the Learning Center Operator with a `Workshop` CRD in the `resources/workshop.yaml` file:

```

apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  vendor: learningcenter.tanzu.vmware.com
  title: Markdown Sample
  description: A sample workshop using Markdown
  url: YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE
  content:
    image: {YOUR-REGISTRY-URL}/lab-markdown-sample:main
  duration: 15m
  session:
    namespaces:
      budget: small
    applications:
      console:
        enabled: true
      editor:
        enabled: true

```

Where:

- `YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE` is the Git repository URL for `lab-markdown-sample`. For example, `{YOUR-GIT-REPO-URL}/lab-markdown-sample`.

In this sample, a custom workshop image bundles the workshop content into its own container image. The `content.image` setting specifies this. To instead download workshop content from a GitHub repository at runtime, use:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  vendor: learningcenter.tanzu.vmware.com
  title: Markdown Sample
  description: A sample workshop using Markdown
  url: YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE
  content:
    files: YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE
  duration: 15m
  session:
    namespaces:
      budget: small
    applications:
      console:
        enabled: true
      editor:
        enabled: true
```

Where:

- `YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE` is the Git repository URL for `lab-markdown-sample`. For example, `{YOUR-GIT-REPO-URL}/lab-markdown-sample`.

The difference is the use of the `content.files` setting. Here, the workshop content is overlaid on top of the standard workshop base image. To use an alternate base image with additional applications or packages installed, specify the alternate image against the `content.image` setting at the same time you set `content.files`.

## Next steps

- Learn about configuration options for the `workshop.yaml` custom resource definitions (CRD) in [Workshop resource](#).

## Create the image for your Learning Center workshop

The workshop environment for the Learning Center is packaged as a container image. This topic describes how you create the Learning Center workshop image.

You can execute the image with remote content pulled down from GitHub or a web server. Alternatively, you can bundle your workshop content, including any extra tools required, in a new container image derived from the workshop environment base image.

## Templates for creating a workshop

To get you started with your own workshop content, VMware provides a number of sample workshops. Different templates in Markdown or AsciiDoc are available to use depending on the syntax you use to create the workshop. These templates are available in a zip file called `LEARNING-CENTER-WORKSHOP-SAMPLES.ZIP` on the [Tanzu Network TAP Product Page](#). The zip file contains the following projects that you can upload to your own Git repository:

- lab-markdown-sample
- lab-asciidoc-sample

When creating your own workshops, a suggested convention is to prefix the directory name with the Git repository name where it is hosted. For example, you can make the prefix `lab-`. This way it stands out as a workshop or lab when you have a number of Git repositories on the same Git hosting service account or organization.



#### Note

Do not make the name you use for a workshop too long. The DNS host name used for applications deployed from the workshop, when using certain methods of deployment, might exceed the 63 character limit. This is because the workshop deployment name is used as part of the namespace for each workshop session. This is in turn used in the DNS host names generated for the ingress host name. VMware suggests keeping the workshop name, and so your repository name, to 25 characters or less.

## Workshop content directory layout

After creating a copy of the sample workshop content, you can see a number of files located in the top-level directory and a number of subdirectories forming a hierarchy. The files in the top-level directory are:

- `README.md` - A file stating what the workshop in your Git repository is about and how to deploy it. Replace the current content provided in the sample workshop with your own.
- `LICENSE` - A license file so people are clear about how they can use your workshop content. Replace this with what license you want to apply to your workshop content.
- `Dockerfile` - Steps to build your workshop into an image ready for deployment. Leave this as is, unless you want to customize it to install additional system packages or tools.
- `kustomization.yaml` - A kustomize resource file for loading the workshop definition. The Learning Center operator must be deployed before using this file.
- `.dockerignore` - List of files to ignore when building the workshop content into an image.
- `.educ8signore` - List of files to ignore when downloading workshop content into the workshop environment at runtime.

Key subdirectories and the files contained within them are:

- `workshop` - Directory under which your workshop files reside.
- `workshop/modules.yaml` - Configuration file with details of available modules that make up your workshop and data variables for use in content.
- `workshop/workshop.yaml` - Configuration file that gives the name of the workshop, the list of active modules for the workshop, and any overrides for data variables.
- `workshop/content` - Directory under which your workshop content resides, including images to be displayed in the content.
- `resources` - Directory under which Kubernetes custom resources are stored for deploying the workshop using the Learning Center.
- `resources/workshop.yaml` - The custom resources for the Learning Center, which describe your workshop and requirements for deployment.

- `resources/training-portal.yaml` - A sample custom resource for the Learning Center for creating a training portal for the workshop, encompassing the workshop environment and a workshop instance.

A workshop can include other configuration files and directories with other types of content, but this is the minimal set of files to get you started.

## Directory for workshop exercises

The number of files and directories can quickly add up at the top level of your repository. The same is true of the home directory for the user when running the workshop environment. To help with this proliferation of files, you can push files required for exercises during the workshop into the `exercises` subdirectory under the root of the repository.

With an `exercises` subdirectory, the initial working directory for the embedded terminal when created is set to `$HOME/exercises` instead of `$HOME`. If the embedded editor is enabled, the subdirectory is opened as the workspace for the editor. Only directories and files in that subdirectory are visible through the default view of the editor.

However, the `exercises` directory isn't set as the home directory of the user. This means if a user inadvertently runs `cd` with no arguments from the terminal, they go back to the home directory.

To avoid confusion and help a user return to where they must be, VMware recommends that when you instruct users to change directories, provide a full path relative to the home directory. For example, use a path of the form `~/exercises/example-1` rather than `example-1` for the `cd` command when changing directories. By using a full path, users can execute the command and be assured of going to the required location.

## Working on your Learning Center workshop content

This topic tells you about the best practices for speeding up the iterative loop of editing and testing a Learning Center workshop when developing the content.

Workshop content is either embedded in a custom workshop image or downloaded from a Git repository or web server when the workshop session is created.

## Deactivating reserved sessions

Deactivate the reserved sessions by setting the `reserved` field to `0` in your training portal instance:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-sample-workshop
spec:
  portal:
    sessions:
      maximum: 1
  workshops:
  - name: lab-sample-workshop
    reserved: 0
    expires: 120m
    orphaned: 15m
```

If you do not deactivate reserved sessions, a new session is always created ready for the next workshop session when there is available capacity to do so. If you modify workshop content while testing the current workshop session, terminate the session and start a new one, the workshop picks up the reserved session. The reserved session has a copy of the old content.

By deactivating reserved sessions, a new workshop session is always created on demand. This ensures the latest workshop content is used.

Because you might have to wait to create a new workshop, shut down the existing workshop session first. The new workshop session might also take some time to start if an updated version of the workshop image also has to be pulled down.

## Live updates to the content

If you download workshop content from a Git repository or web server, and you are only doing simple updates to workshop instructions, scripts, or files bundled with the workshop, you can update the content in place without needing to restart the workshop session. To perform an update, download the workshop content after you have pushed back any changes to the hosted Git repository or updated the content available through the web server. From the workshop session terminal, run:

```
update-workshop
```

This command downloads any workshop content from the Git repository or web server, unpacks it into the live workshop session, and re-runs any script files found in the `workshop/setup.d` directory.

Find the location where the workshop content is downloading by viewing the file:

```
cat ~/.educ8s/workshop-files.txt
```

You can change the location saved in this file if, for example, it references a specific version of the workshop content and you want to test with a different version.

Once the workshop content has been updated, reload the current page of the workshop instructions by clicking the reload icon on the dashboard while holding down the shift key.

If additional pages are added to the workshop instructions or pages are renamed, you must restart the workshop renderer process by running:

```
restart-workshop
```

If you didn't rename the current pager or if the name changed, you can trigger a reload of the current page. Click the home icon or refresh the webpage if the name of the first page didn't change.

If action blocks within the workshop instructions are broken, to change and test the workshop instructions within the live workshop session, you can edit the appropriate page under `/opt/workshop/content`. Navigate to the modified page or reload it to verify the change.

To change set up scripts that create files specific to a workshop session, edit the script under `/opt/workshop/setup.d` directory.

To trigger running of any setup scripts, run:

```
rebuild-workshop
```

If local changes to the workshop session take effect, you can restore the file in the original Git repository.

Updating workshop content in a live session in this way does not undo any deployments or changes you make in the Kubernetes cluster for that session. To retest parts of the workshop instructions, you might have to manually undo the changes in the cluster to replay them. This depends on your specific workshop content.

## Custom workshop image changes

If your workshop uses a custom workshop image to provide additional tools and you have included the workshop instructions as part of the workshop image, you must use an image tag of `main`, `develop`, or `latest` during the development of workshop content. Do not use a version image reference.

For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-sample-workshop
spec:
  title: Sample Workshop
  description: A sample workshop
  content:
    image: {YOUR-GIT-REPO-URL}/lab-sample-workshop:main
```

When you use an image tag of `main`, `develop`, or `latest`, the image pull policy is set to `Always` to ensure that the custom workshop image is pulled down again for a new workshop session if the remote image changes. If the image tag is for a specific version, you must change the workshop definition every time when the workshop image changes.

## Custom workshop image overlay

For a custom workshop image, you can set up the workshop definition to pull down the workshop content from the hosted Git repository or web server as the follows:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-sample-workshop
spec:
  title: Sample Workshop
  description: A sample workshop
  content:
    image: {YOUR-REGISTRY-URL}/lab-sample-workshop:main
    files: {YOUR-GIT-REPO-URL}/lab-sample-workshop
```

By pulling down the workshop content as an overlay of the custom workshop image when the workshop session starts, you only need to rebuild the custom workshop image when you need to make changes such as to include additional tools or to ensure the latest workshop instructions are included in the final custom workshop image.

Because the location of the workshop files is known, you can live update the workshop content in the session by following [Live updates to the content](#).

If the additional set of tools required for a workshop is not specific to a workshop, VMware recommends that you create a standalone workshop base image where you can add the tools. You can always pull down content for a specific workshop from a Git repository or web server when the workshop session starts.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-sample-workshop
spec:
  title: Sample Workshop
  description: A sample workshop
```



```

content:
  image: {YOUR-REGISTRY-URL}/custom-environment:main
  files: {YOUR-GIT-REPO-URL}/lab-sample-workshop

```

This separates generic tooling from specific workshops and so you can use the custom workshop base image for multiple workshops on different, but related topics that require the same tooling.

## Changes to workshop definition

By default, to modify the definition for a workshop, you need to delete the training portal instance, update the workshop definition in the cluster, and recreate the training portal.

During the workshop content development, to change resource allocations, role access, or to specify what resource objects to be automatically created for the workshop environment or a specific workshop session, you can enable automatic updates in the training portal definition by setting `updates.workshop` field as `true`:

```

apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-sample-workshop
spec:
  portal:
    sessions:
      maximum: 1
    updates:
      workshop: true
  workshops:
  - name: lab-sample-workshop
    expires: 120m
    orphaned: 15m

```

With automatic updates enabled, if the workshop definition in the cluster is modified, the existing workshop environment managed by the training portal for that workshop is shut down and replaced with a new workshop environment by using the updated workshop definition.

When an active workshop session is running, the actual deletion of the old workshop environment is delayed until that workshop session is terminated.

## Local build of workshop image

If you do not package a workshop into a custom workshop image, VMware recommends to build a custom workshop image locally on your own machine by using `docker` to avoid keeping pushing changes to a hosted Git repository and using a Kubernetes cluster for local workshop content development.

Furthermore, to avoid pushing the image to a public image registry on the Internet, you must deploy an image registry to your local Kubernetes cluster where you run the Learning Center. In most cases, a basic deployment of an image registry in a local cluster access is not secure. As a result, you have to configure the Kubernetes cluster to trust the registry that is not secure. This can be difficult to do depending on the Kubernetes cluster you use, but it can enable quicker turnaround because you do not have to push or pull the custom workshop image across the public Internet.

After pushing the custom workshop image built locally to the local image registry, you can set the image reference in the workshop definition to pull the custom workshop from the local registry in the same cluster. To ensure that the custom workshop image is always pulled for a new workshop session after update, use the `latest` tag when tagging and pushing the image to the local registry.

## Build an image for your Learning Center workshop

This topic describes how you include an extra system, third-party tool, or configuration in your image by bundling workshop content from the Learning Center workshop base image.

The following sample workshop template provides a `Dockerfile`.

### Structure of the Dockerfile

The structure of the `Dockerfile` in the sample workshop template is:

```
FROM registry.tanzu.vmware.com/tanzu-application-platform/tap-packages@sha256:xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
XXXXXXXXXXXXXXXXXXXX

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 base environment image according to the version of Tanzu Application Platform. To get the image ID, run:

```
kubectl get ds -n learningcenter learningcenter-prepare -o=jsonpath="{.spec.template.spec.initContainers[0].image}"
```

Example image ID:

```
registry.tanzu.vmware.com/tanzu-application-platform/tap-packages@sha256:a8870aa60b45495d298df5b65c69b3d7972608da4367bd6e69d6e392ac969dd4
```

- You can build the workshop images directly on the base environment image, or you can 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.

## 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 ${IMAGE_REPOSITORY}/pkgs-java-tools as java-tools
FROM registry.tanzu.vmware.com/tanzu-application-platform/tap-packages@sha256:xxxxxxxx
xxxxxxxx
COPY --from=java-tools --chown=1001:0 /opt/jdk11 /opt/java
COPY --from=java-tools --chown=1001:0 /opt/gradle /opt/gradle
COPY --from=java-tools --chown=1001:0 /opt/maven /opt/maven
COPY --from=java-tools --chown=1001:0 /opt/code-server/extensions/. /opt/code-server/
extensions/
COPY --from=java-tools --chown=1001:0 /home/educ8s/. /home/educ8s/
COPY --from=java-tools --chown=1001:0 /opt/educ8s/. /opt/educ8s/
ENV PATH=/opt/java/bin:/opt/gradle/bin:/opt/maven/bin:$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.

## Writing instructions for your Learning Center workshop

This topic describes how you write and format the instructions for a Learning Center workshop. You can use either [Markdown](#) with file extension `.md` or [AsciiDoc](#) with file extension `.adoc` as the markup format for the individual module files that comprise the workshop instructions.

### Annotation of executable commands

In conjunction with the standard Markdown and AsciiDoc, you can apply additional annotations to code blocks. The annotations indicate that a user can click the code block and have it copied to the terminal and executed.

If using Markdown, to annotate a code block so it is copied to the terminal and executed, use:

```
```execute
echo "Execute command."
```
```

When the user clicks the code block, the command is executed in the first terminal of the workshop dashboard.

If using AsciiDoc, you can instead use the `role` annotation in an existing code block:

```
[source,bash,role=execute]
----
echo "Execute command."
----
```

When the workshop dashboard is configured to display multiple terminals, you can qualify which terminal the command must be executed in by adding a suffix to the `execute` annotation. For the first terminal, use `execute-1`, for the second terminal `execute-2`, and so on:

```
```execute-1
echo "Execute command."
```

```execute-2
echo "Execute command."
```
```

To execute a command in all terminal sessions on the terminals tab of the dashboard, you can use `execute-all`:

```
```execute-all
clear
```
```

In most cases, a command the user executes completes immediately. To run a command that never returns, with the user needing to interrupt it to stop it, you can use the special string `<ctrl+c>` in a subsequent code block.

```
```execute
<ctrl+c>
```
```

When the user clicks on this code block, the command running in the corresponding terminal is interrupted.



#### Note

Using the special string `<ctrl+c>` is deprecated, and you must use the `terminal:interrupt` clickable action instead.

## Annotation of text to be copied

To copy the content of the code block into the paste buffer instead of running the command, you can use:

```
```copy
echo "Text to copy."
```
```

After the user clicks this code block, they can then paste the content into another window.

If you have a situation where the text being copied must be modified before use, you can denote this special case by using `copy-and-edit` instead of `copy`. The text is still copied to the paste buffer, but is displayed in the browser in a way to highlight that it must be changed before use.

```
```copy-and-edit
echo "Text to copy and edit."
```
```

For AsciiDoc, similar to `execute`, you can add the `role` of `copy` or `copy-and-edit`:

```
[source,bash,role=copy]
----
echo "Text to copy."
----

[source,bash,role=copy-and-edit]
----
echo "Text to copy and edit."
----
```

For `copy` only, to mark an inline code section within a paragraph of text as copyable when clicked, you can append the special data variable reference `{{copy}}` immediately after the inline code block:

```
Text to `copy`{{copy}}.
```

## Extensible clickable actions

The preceding means to annotate code blocks were the original methods used to indicate code blocks to be executed or copied when clicked. To support a growing number of clickable actions with different customizable purposes, annotation names are now name-spaced. The preceding annotations are still supported, but the following are now recommended, with additional options available to customize the way the actions are presented.

For code execution, instead of:

```
```execute
echo "Execute command."
```
```

you can use:

```
```terminal:execute
command: echo "Execute command."
```
```

The contents of the code block is YAML. The executable command must be set as the `command` property. By default when the user clicks the command, it is executed in terminal session 1. To select a different terminal session, you can set the `session` property.

```
```terminal:execute
command: echo "Execute command."
session: 1
```
```

To define a command the user clicks that executes in all terminal sessions on the terminals tab of the dashboard, you can also use:

```
```terminal:execute-all
command: echo "Execute command."
```
```

For `terminal:execute` or `terminal:execute-all`, to clear the terminal before the command is executed, set the `clear` property to `true`:

```
```terminal:execute
command: echo "Execute command."
clear: true
```
```

This clears the full terminal buffer and not just the displayed portion of the buffer.

With the new clickable actions, to indicate that a running command in a terminal session must be interrupted, use:

```
```terminal:interrupt
session: 1
```
```

(Optional) Set the `session` property within the code block to indicate an alternate terminal session to session 1.

To allow the user to send an interrupt to all terminals sessions on the terminals tab of the dashboard, use:

```
``terminal:interrupt-all
```
```

Where you want the user to enter input into a terminal rather than a command, such as when a running command prompts for a password, use:

```
``terminal:input
text: password
```
```

To allow the user to run commands or interrupt a command, set the `session` property to indicate a specific terminal to send it to if you don't want to send it to terminal session 1:

```
``terminal:input
text: password
session: 1
```
```

When providing terminal input in this way, the text by default still has a newline appended to the end, making it behave the same as using `terminal:execute`. If you do not want a newline appended, set the `endl` property to `false`.

```
``terminal:input
text: input
endl: false
```
```

To allow the user to clear all terminal sessions on the terminals tab of the dashboard, use:

```
``terminal:clear-all
```
```

This clears the full terminal buffer and not just the displayed portion of the terminal buffer. It does not have any effect when an application is running in the terminal using visual mode. To clear only the displayed portion of the terminal buffer when a command dialog box is displayed, use `terminal:execute` and run the `clear` command.

To allow the user to copy content to the paste buffer, use:

```
``workshop:copy
text: echo "Text to copy."
```
```

or:

```
``workshop:copy-and-edit
text: echo "Text to copy and edit."
```
```

A benefit of using these over the original methods is that by using the appropriate YAML syntax, you can control whether:

- A multiline string value is concatenated into one line.
- Line breaks are preserved.
- Initial or terminating new lines are included.

In the original methods, the string was always trimmed before use. By using the different forms as appropriate, you can annotate the displayed code block with a different message letting the user know what will happen.

The method for using AsciiDoc is similar, using the `role` for the name of the annotation and YAML as the content:

```
[source,bash,role=terminal:execute]
----
command: echo "Execute command."
----
```

## Supported workshop editor

Learning Center currently **only** supports the code-server v4.4.0 of VS Code as an editor in workshops.

## Clickable actions for the dashboard

In addition to the clickable actions related to the terminal and copying of text to the paste buffer, other actions are available for controlling the dashboard and opening URL links.

To allow the user to click in the workshop content to open a URL in a new browser, use:

```
```dashboard:open-url
url: https://www.example.com/
```
```

To allow the user to click in the workshop content to display a specific dashboard tab if hidden, use:

```
```dashboard:open-dashboard
name: Terminal
```
```

To allow the user to click in the workshop content to display the console tab, use:

```
```dashboard:open-dashboard
name: Console
```
```

To allow the user to click in the workshop content to display a specific view within the Kubernetes web console by using a clickable action block, rather than requiring the user to find the correct view, use:

```
```dashboard:reload-dashboard
name: Console
prefix: Console
title: List pods in namespace {{session_namespace}}
url: {{ingress_protocol}}://{{session_namespace}}-console.{{ingress_domain}}/#/pod?namespace={{session_namespace}}
description: ""
```
```

To allow the user to create a new dashboard tab with a specific URL, use:

```
```dashboard:create-dashboard
name: Example
url: https://www.example.com/
```
```

To allow the user to create a new dashboard tab with a new terminal session, use:



```
```dashboard:create-dashboard
name: Example
url: terminal:example
```
```

The value must be of the form `terminal:<session>`, where `<session>` is replaced with the name you want to give the terminal session. The terminal session name must be restricted to lowercase letters, numbers, and `-`. You must avoid using numeric terminal session names such as `"1"`, `"2"`, and `"3"`, because these are used for the default terminal sessions.

To allow the user to reload an existing dashboard, using the URL it is currently targeting, use:

```
```dashboard:reload-dashboard
name: Example
```
```

If the dashboard is for a terminal session, there is no effect unless the terminal session was disconnected, in which case it is reconnected.

To allow the user to change the URL target of an existing dashboard by entering the new URL when reloading a dashboard, use:

```
```dashboard:reload-dashboard
name: Example
url: https://www.example.com/
```
```

The user cannot change the target of a dashboard that includes a terminal session.

To allow the user to delete a dashboard, use:

```
```dashboard:delete-dashboard
name: Example
```
```

The user cannot delete dashboards corresponding to builtin applications provided by the workshop environment, such as the default terminals, console, editor, or slides.

Deleting a custom dashboard including a terminal session does not destroy the underlying terminal session, and the user can reconnect it by creating a new custom dashboard for the same terminal session name.

## Clickable actions for the editor

If the embedded editor is enabled, special actions are available that control the editor.

To allow the user to open an existing file you can use:

```
```editor:open-file
file: ~/exercises/sample.txt
```
```

You can use `~/` prefix to indicate the path relative to the home directory of the session. When the user opens the file, if you want the insertion point left on a specific line, provide the `line` property. Lines numbers start at `1`.

```
```editor:open-file
file: ~/exercises/sample.txt
line: 1
```
```

To allow the user to highlight certain lines of a file based on an exact string match, use:

```
```editor:select-matching-text
file: ~/exercises/sample.txt
text: "int main()"
```
```

The region of the match is highlighted by default. To allow the user to highlight any number of lines before or after the line with the match, you can set the `before` and `after` properties:

```
```editor:select-matching-text
file: ~/exercises/sample.txt
text: "int main()"
before: 1
after: 1
```
```

Setting both `before` and `after` to 0 causes the complete line that matched to be highlighted instead of a region within the line.

To match based on a regular expression, rather than an exact match, set `isRegex` to `true`:

```
```editor:select-matching-text
file: ~/exercises/sample.txt
text: "image: (.*)"
isRegex: true
```
```

When a regular expression is used, and subgroups are specified within the pattern, you can indicate which subgroup is selected:

```
```editor:select-matching-text
file: ~/exercises/sample.txt
text: "image: (.*)"
isRegex: true
group: 1
```
```

Where there are multiple possible matches in a file, and the one you want to match is not the first, you can set a range of lines to search:

```
```editor:select-matching-text
file: ~/exercises/sample.txt
text: "image: (.*)"
isRegex: true
start: 8
stop: 12
```
```

Absence of `start` means start at the beginning of the file. Absence of `stop` means stop at the end of the file. The line number given by `stop` is not included in the search.

For both an exact match and regular expression, the text to be matched must all be on one line. It is not possible to match text that spans across lines.

To allow the user to replace text within the file, first match it exactly or use a regular expression so it is marked as selected, then use:

```
```editor:replace-text-selection
file: ~/exercises/sample.txt
```

```
text: nginx:latest
...

```

To allow the user to append lines to the end of a file, use:

```
``editor:append-lines-to-file
file: ~/exercises/sample.txt
text: |
  Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed
  do eiusmod tempor incididunt ut labore et dolore magna aliqua.
...

```

If the user runs the action `editor:append-lines-to-file` and the file doesn't exist, it is created. You can use this to create new files for the user.

To allow the user to insert lines before a specified line in the file, use:

```
``editor:insert-lines-before-line
file: ~/exercises/sample.txt
line: 8
text: |
  Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed
  do eiusmod tempor incididunt ut labore et dolore magna aliqua.
...

```

To allow the user to insert lines after matching a line containing a specified string, use:

```
``editor:append-lines-after-match
file: ~/exercises/sample.txt
match: Lorem ipsum
text: |
  Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed
  do eiusmod tempor incididunt ut labore et dolore magna aliqua.
...

```

Where the file contains YAML, to allow the user to insert a new YAML value into an existing structure, use:

```
``editor:insert-value-into-yaml
file: ~/exercises/deployment.yaml
path: spec.template.spec.containers
value:
- name: nginx
  image: nginx:latest
...

```

To allow the user to execute a registered VS code command, use:

```
``editor:execute-command
command: spring.initializr.maven-project
args:
- language: Java
  dependencies: [ "actuator", "webflux" ]
  artifactId: demo
  groupId: com.example
...

```

## Clickable actions for file download

If file downloads are enabled for the workshop, you can use the `files:download-file` clickable action:

```
```files:download-file
path: .kube/config
```
```

The action triggers saving the file to the user's local computer, and the file is not displayed in the user's web browser.

## Clickable actions for the examiner

If the test examiner is enabled, special actions are available to run verification checks to verify whether a workshop user has performed a required step. You can trigger these verification checks by clicking on the action, or you can configure them to start running when the page loads.

For a single verification check that the user must click to run, use:

```
```examiner:execute-test
name: test-that-pod-exists
title: Verify that pod named "one" exists.
args:
- one
```
```

The `title` field is displayed as the title of the clickable action and must describe the nature of the test. If required, you can provide a `description` field for a longer explanation of the test. This is displayed in the body of the clickable action but is shown as preformatted text.

There must be an executable program (script or compiled application) in the `workshop/examiner/tests` directory with name matching the value of the `name` field.

The list of program arguments against the `args` field is passed to the test program.

The executable program for the test must exit with a status of 0 if the test is successful and nonzero if the test is a failure. The test should aim to return as quickly as possible and should not be a persistent program.

```
#!/bin/bash

kubectl get pods --field-selector=status.phase=Running -o name | egrep -e "^pod/$1$"

if [ "$?" != "0" ]; then
    exit 1
fi

exit 0
```

By default, the program for a test is stopped after a timeout of 15 seconds, and the test is deemed to have failed. To adjust the timeout, you can set the `timeout` value, which is in seconds. A value of 0 causes the default 15 seconds timeout to be applied. It is not possible to deactivate stopping the test program after running for the default or a specified `timeout` value.

```
```examiner:execute-test
name: test-that-pod-exists
title: Verify that pod named "one" exists
args:
- one
timeout: 5
```
```

To apply the test multiple times, you can enable the retry when a failure occurs. For this you must set the number of times to retry and the delay between retries. The value for the delay is in

seconds.

```

```examiner:execute-test
name: test-that-pod-exists
title: Verify that pod named "one" exists
args:
- one
timeout: 5
retries: 10
delay: 1
```

```

When you use retries, the testing stops as soon as the test program returns that it was successful.

To have retries continue for as long as the page of the workshop instructions displays, set `retries` to the special YAML value of `.INF`:

```

```examiner:execute-test
name: test-that-pod-exists
title: Verify that pod named "one" exists
args:
- one
timeout: 5
retries: .INF
delay: 1
```

```

Rather than require a workshop user to click the action to run the test, you can have the test start as soon as the page is loaded, or when a section the page is contained in is expanded. Do this by setting `autostart` to `true`:

```

```examiner:execute-test
name: test-that-pod-exists
title: Verify that pod named "one" exists
args:
- one
timeout: 5
retries: .INF
delay: 1
autostart: true
```

```

When a test succeeds, to immediately start the next test in the same page, set `cascade` to `true`.

```

```examiner:execute-test
name: test-that-pod-exists
title: Verify that pod named "one" exists
args:
- one
timeout: 5
retries: .INF
delay: 1
autostart: true
cascade: true
```

```examiner:execute-test
name: test-that-pod-does-not-exist
title: Verify that pod named "one" does not exist
args:
- one
retries: .INF
```

```

```
delay: 1
...
```

## Clickable actions for sections

For optional instructions, or instructions you want to hide until the workshop user is ready for them, you can designate sections to be hidden. When the user clicks the appropriate action, the section expands to show its content. You can use this for examples that initially hide a set of questions or a test at the end of each workshop page.

In order to designate a section of content as hidden, you must use two separate action code blocks marking the beginning and end of the section:

```
```section:begin
title: Questions
...

To show you understand ...

```section:end
...
```

The `title` must be set to the text you want to include in the banner for the clickable action.

A clickable action is only shown for the beginning of the section, and the action for the end is always hidden. Clicking the action for the beginning expands the section. The user can collapse the section again by clicking the action.

To create nested sections, you must name the action blocks for the beginning and end so they are correctly matched:

```
```section:begin
name: questions
title: Questions
...

To show you understand ...

```section:begin
name: question-1
prefix: Question
title: 1
...

...

```section:end
name: question-1
...

```section:end
name: questions
...`
```

The `prefix` attribute allows you to override the default `Section` prefix used on the title for the action.

If a collapsible section includes an examiner action block set to automatically run, it only starts when the user expands the collapsible section.

In case you want a section header showing in the same style as other clickable actions, you can use:

```
```section:heading
title: Questions
```
```

When the user clicks on this, the action is still marked as completed, but it does not trigger any other action.

## Overriding title and description

Clickable action blocks default to use a title with the prefix dictated by what the action block does. The body of the action block also defaults to use a value commensurate with the action.

Especially for complicated scenarios involving editing of files, the defaults might not be the most appropriate and be confusing, so you can override them. To override these defaults, set the `prefix`, `title`, and `description` fields of a clickable action block:

```
```action:name
prefix: Prefix
title: Title
description: Description
```
```

The banner of the action block in this example displays “Prefix: Title”, with the body showing “Description”.



### Note

The description is always displayed as pre-formatted text within the rendered page.

## Escaping of code block content

Because the [Liquid](#) template engine is applied to workshop content, you must escape content in code blocks that conflict with the syntactic elements of the Liquid template engine. To escape such elements, you can suspend processing by the template engine for that section of workshop content to ensure it is rendered correctly. Do this by using a Liquid `{% raw %}...{% endraw %}` block.

```
{% raw %}
```execute
echo "Execute command."
```
{% endraw %}
```

This has the side effect of preventing interpolation of data variables, so restrict it to only the required scope.

## Interpolation of data variables

When creating page content, you can reference a number of predefined data variables. The values of the data variables are substituted into the page when rendered in the user’s browser.

The workshop environment provides the following built-in data variables:

- `workshop_name`: The name of the workshop.
- `workshop_namespace`: The name of the namespace used for the workshop environment.

- `session_namespace`: The name of the namespace the workshop instance is linked to and into which any deployed applications run.
- `training_portal`: The name of the training portal the workshop is hosted by.
- `ingress_domain`: The host domain must be used in the any generated host name of ingress routes for exposing applications.
- `ingress_protocol`: The protocol (http/https) used for ingress routes created for workshops.

To use a data variable within the page content, surround it by matching pairs of brackets:

```
{{ session_namespace }}
```

Do this inside of code blocks, including clickable actions, as well as in URLs:

```
http://myapp-{{ session_namespace }}.{{ ingress_domain }}
```

When the workshop environment is hosted in Kubernetes and provides access to the underlying cluster, the following data variables are also available.

- `kubernetes_token`: The Kubernetes access token of the service account the workshop session is running as.
- `kubernetes_ca_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%">
  <thead>
    <tr>
      <th>Firstname</th>
      <th>Lastname</th>
      <th>Age</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Jill</td>
      <td>Smith</td>
      <td>50</td>
    </tr>
    <tr>
      <td>Eve</td>
      <td>Jackson</td>
      <td>94</td>
    </tr>
  </tbody>
</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%">
  <thead>
    <tr>
      <th>Firstname</th>
      <th>Lastname</th>
      <th>Age</th>
```

```

</tr>
</thead>
<tbody>
<tr>
  <td>Jill</td>
  <td>Smith</td>
  <td>50</td>
</tr>
<tr>
  <td>Eve</td>
  <td>Jackson</td>
  <td>94</td>
</tr>
</tbody>
</table>
</div>
++++

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin justo.

```

In both cases, VMware recommends that the HTML consist of only a single HTML element. If you have more than one, include them all in a `div` element. The latter is necessary if any of the HTML elements are marked as hidden and the embedded HTML is a part of a collapsible section. If you don't ensure the hidden HTML element is placed under the single top-level `div` element, the hidden HTML element is visible when the collapsible section is expanded.

In addition to visual HTML elements, you can also include elements for embedded scripts or style sheets.

If you have HTML markup that must be added to multiple pages, extract it into a separate file and use the include file mechanism of the Liquid template engine. You can also use the partial render mechanism of Liquid as a macro mechanism for expanding HTML content with supplied values.

## Automate your Learning Center workshop runtime

Your workshop content can script the steps a user must run for a workshop. This topic tells you how to set this up.

In some cases, you must parameterize 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, you can also reference a number of predefined environment variables.

Key environment variables are:

- `WORKSHOP_NAMESPACE` - The name of the namespace used for the workshop environment.

- `SESSION_NAMESPACE` - The name of the namespace the workshop instance is linked to and into which any deployed applications run.
- `INGRESS_DOMAIN` - The host domain that must be used in any generated host name of ingress routes for exposing applications.
- `INGRESS_PROTOCOL` - The protocol (http/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/eduk8s` directory as the current working directory.

If you are creating or updating files in the file system and using a custom workshop image, ensure that the workshop image is created with correct file permissions to allow updates.

## Running background applications

The setup scripts run once on container startup. You can use the script to start a background application needed to run in the container for the life of the workshop, but if that application stops, it does not restart.

If you must run a background application, you can integrate the management of the background application with the supervisor daemon run within the container. To have the supervisor daemon manage the application for you, add a configuration file snippet for the supervisor daemon in the `workshop/supervisor` directory. This configuration file must have a `.conf` extension.

The form of the configuration file snippet must be:

```
[program:myapplication]
process_name=myapplication
command=/opt/myapplication/sbin/start-myapplication
stdout_logfile=/proc/1/fd/1
stdout_logfile_maxbytes=0
redirect_stderr=true
```

The application must send any logging output to `stdout` or `stderr`, and the configuration snippet must direct log output to `/proc/1/fd/1` so it is captured in the container log file. If you must restart or shut down the application within the workshop interactive terminal, you can use the `supervisorctl` control script.

## Terminal user shell environment

Neither the setup scripts that run when the container starts nor background applications affect the user environment of the terminal shell. The shell environment makes use of `bash` and the `$HOME/.bash_profile` script is read to perform added setup for the user environment. Because some default setup is included in `$HOME/.bash_profile`, you must not replace it, because you can 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
```

## Add presenter slides to your Learning Center workshop

If your workshop includes a presentation, include slides by placing them in the `workshop/slides` directory. Anything in this directory is served up as static files through a HTTP web server. The default webpage must be provided as `index.html`.

## Use reveal.js presentation tool

To support the use of `reveal.js`, static media assets for that package are already bundled and available at the standard URL paths that the package expects. You can drop your slide presentation using `reveal.js` into the `workshop/slides` directory and it will work with no additional setup.

If you are using `reveal.js` for the slides and you have history enabled or are using section IDs to support named links, you can use an anchor to a specific slide and that slide will be opened when clicked on:

```
%slides_url%#/questions
```

When using embedded links to the slides in workshop content, if the workshop content is displayed as part of the dashboard, the slides open in the tab to the right rather than as a separate browser window or tab.

## Use a PDF file for presenter slides

For slides bundled as a PDF file, add the PDF file to `workshop/slides` and then add an `index.html` which displays the PDF `embedded` in the page.

## Requirements for Learning Center in an air-gapped environment

This topic gives you the list of configurations required for Learning Center to properly function in an air-gapped environment.

Learning Center can run in an air-gapped environment but workshops do not have this capability by default. Users must therefore take the following steps to ensure Learning Center functions as expected.

## Workshop yaml changes

In an air-gapped environment a user has no Internet access, so workshop yamls should be `modified` to use:

1. Private container registries.
2. Private Maven, NPM, Python, Go, or any other language repository.

For example, in NPM you can modify the `npmrc` file to use:

```
// .npmrc
registry=https://myregistry-url
```

## Self-signed certificates

Air-gapped environments normally use private Certificate Authorities (CA) that may require the use of self-signed certificates. You can allow the injection of CAs by:

1. Setting the env variable `NODE_EXTRA_CA_CERTS` to the path of the file that contains one or more trusted certificates in PEM format.
2. Add the following to your workshop definition:

```
spec:
  session:
    env:
      - name: NODE_EXTRA_CA_CERTS
        value: "$my-cert-pathway"
```

## Internet dependencies

If the workshop requires the installation of any Internet dependency, such as a Linux Tool or any other tool, it must be done in the workshop image. See [Build an image](#)

## Define custom resources for Learning Center

This topic describes how you define custom resources for Learning Center workshops and training portals.

You can deploy workshop images directly to a container runtime. The Learning Center Operator enables managing the deployments into a Kubernetes cluster. A set of Kubernetes custom resource definitions (CRDs) controls the operation of the Learning Center Operator.



### Note

The examples do not show all the possible fields of each custom resource type. Later documentation may go in-depth on possible fields and their definitions.

## Workshop definition resource

The `Workshop` custom resource defines a workshop. It specifies the title and description of the workshop, the location of the workshop content or container image that you deploy, any resources that you pre-create in the workshop environment or for each instance of the workshop.

You can also define environment variables for the workshop image, the amount of CPU and memory resources for the workshop instance, any overall quota you will apply to the created namespaces and what the workshop uses.

A minimal example of the `Workshop` custom resource looks like this:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  title: Markdown Sample
```

```

description: A sample workshop using Markdown
content:
  files: {YOUR-GIT-REPO-URL}/lab-markdown-sample
session:
  namespaces:
    budget: small
  applications:
    console:
      enabled: true
    editor:
      enabled: true

```

When you create an instance of the `Workshop` custom resource, the Learning Center Operator does not take any immediate action. This custom resource exists only to define the workshop.

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.

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.

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.

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.

You create the `SystemProfile` custom resource at the cluster scope.

## Loading the workshop CRDs

The custom resource definitions for the custom resource described earlier are created in the Kubernetes cluster when you deploy the Learning Center operator by using the Tanzu CLI.

This is because `v1` versions of CRDs are only supported from Kubernetes v1.17. If you want to use the `v1` versions of the CRDs, you must create a copy of the Learning Center operator deployment resources and override the configuration.

## Define custom resources for Learning Center

This topic describes how you define custom resources for Learning Center workshops and training portals.

You can deploy workshop images directly to a container runtime. The Learning Center Operator enables managing the deployments into a Kubernetes cluster. A set of Kubernetes custom resource definitions (CRDs) controls the operation of the Learning Center Operator.



### Note

The examples do not show all the possible fields of each custom resource type. Later documentation may go in-depth on possible fields and their definitions.

## Workshop definition resource

The `Workshop` custom resource defines a workshop. It specifies the title and description of the workshop, the location of the workshop content or container image that you deploy, any resources that you pre-create in the workshop environment or for each instance of the workshop.

You can also define environment variables for the workshop image, the amount of CPU and memory resources for the workshop instance, any overall quota you will apply to the created namespaces and what the workshop uses.

A minimal example of the `Workshop` custom resource looks like this:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  title: Markdown Sample
  description: A sample workshop using Markdown
  content:
    files: {YOUR-GIT-REPO-URL}/lab-markdown-sample
  session:
    namespaces:
      budget: small
    applications:
      console:
        enabled: true
      editor:
        enabled: true
```

When you create an instance of the `Workshop` custom resource, the Learning Center Operator does not take any immediate action. This custom resource exists only to define the workshop.

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.

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.

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.

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.

You create the `SystemProfile` custom resource at the cluster scope.

## Loading the workshop CRDs

The custom resource definitions for the custom resource described earlier are created in the Kubernetes cluster when you deploy the Learning Center operator by using the Tanzu CLI.

This is because `v1` versions of CRDs are only supported from Kubernetes v1.17. If you want to use the `v1` versions of the CRDs, you must create a copy of the Learning Center operator deployment resources and override the configuration.

## Configure the Workshop resource

This topic describes how you configure the `Workshop` custom resource, which defines a Learning Center workshop.

## Workshop title and description

Each workshop must have the `title` and `description` fields. If you do not supply these fields, the `Workshop` resource is rejected when you attempt to load it into the Kubernetes cluster.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  title: Markdown Sample
  description: A sample workshop using Markdown
  content:
    files: {YOUR-GIT-REPO-URL}/lab-markdown-sample
```

Where:

- The `title` field has a single-line value specifying the subject of the workshop.
- The `description` field has a longer description of the workshop.

You can also supply the following optional information for the workshop:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  title: Markdown Sample
  description: A sample workshop using Markdown
  url: YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE
  difficulty: beginner
  duration: 15m
  vendor: learningcenter.tanzu.vmware.com
  authors:
    - John Smith
  tags:
    - template
  logo: data:image/png;base64,...
  content:
    files: YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE
```

Where:

- The `url` field is the Git repository URL for `lab-markdown-sample`. For example, `{YOUR-GIT-REPO-URL}/lab-markdown-sample`. It must be a URL you can use to get more information about the workshop.
- The `difficulty` field indicates the target audiences of the workshop. The value can be `beginner`, `intermediate`, `advanced`, or `extreme`.
- The `duration` field gives the maximum amount of time the workshop takes to complete. This field provides informational value and does not guarantee how long a workshop instance lasts. The field format is an integer number with `s`, `m`, or `h` suffix.
- The `vendor` field must be a value that identifies the company or organization with which the authors are affiliated. This is a company or organization name or a DNS host name under the control of whoever has created the workshop.
- The `authors` field must list the people who create the workshop.
- The `tags` field must list labels identifying what the workshop is about. This is used in a searchable catalog of workshops.

- The `logo` field must be an image provided in embedded data URI format that depicts the topic of the workshop. The image must be 400 by 400 pixels. You can use it in a searchable catalog of workshops.
- The `files` field is the Git repository URL for `lab-markdown-sample`. For example, `{YOUR-GIT-REPO-URL}/lab-markdown-sample`.

When referring to a workshop definition after you load it into a Kubernetes cluster, use the value of the `name` field given in the metadata. To experiment with different variations of a workshop, copy the original workshop definition YAML file and change the value of `name`. Make your changes and load it into the Kubernetes cluster.

## Downloading workshop content

You can download workshop content when you create the workshop instance. If the amount of content is moderate, the download doesn't increase startup time for the workshop instance. The alternative is to bundle the workshop content in a container image you build from the Learning Center workshop base image.

To download workshop content at the time the workshop instance starts, set the `content.files` field to the location of the workshop content:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  title: Markdown Sample
  description: A sample workshop using Markdown
  content:
    files: {YOUR-GIT-REPO-URL}/lab-markdown-sample
```

The location is a GitHub or GitLab repository, a URL to a tarball hosted on a HTTP server, or a reference to an OCI image artifact on a registry.

For a GitHub or GitLab repository, do not prefix the location with `https://` as it uses symbolic reference and is not a URL.

The format of the reference to a GitHub or GitLab repository is similar to what you use with Kustomize when referencing remote repositories. For example:

- `github.com/organisation/project?ref=develop` Or `github.com/organisation/project?ref=main`: Use the workshop content you host at the root of the GitHub repository. Use the `develop` or `main` branch. Be sure to specify the ref branch, because not specifying the branch may lead to content download errors.
- `github.com/organisation/project/subdir?ref=develop`: Use the workshop content you host at `subdir` of the GitHub repository. Use the `develop` branch.
- `gitlab.com/organisation/project`: Use the workshop content you host at the root of the GitLab repository. Use the `main` branch.
- `gitlab.com/organisation/project/subdir?ref=develop`: Use the workshop content you host at `subdir` of the GitLab repository. Use the `develop` branch.

For a URL to a tarball hosted on a HTTP server, the URL is in the following formats:

- `https://example.com/workshop.tar` - Use the workshop content from the top-level directory of the unpacked tarball.
- `https://example.com/workshop.tar.gz` - Use the workshop content from the top-level directory of the unpacked tarball.

- `https://example.com/workshop.tar?path=subdir` - Use the workshop content from the subdirectory path of the unpacked tarball.
- `https://example.com/workshop.tar.gz?path=subdir` - Use the workshop content from the subdirectory path of the unpacked tarball.

The tarball referenced by the URL is either uncompressed or compressed.

For GitHub, instead of referencing the Git repository containing the workshop content, use a URL to refer directly to the downloadable tarball for a specific version of the Git repository:

- `https://github.com/organization/project/archive/develop.tar.gz?path=project-develop`

You must reference the `.tar.gz` download and cannot use the `.zip` file. The base name of the tarball file is the branch or commit name. You must enter the `path` query string parameter where the argument is the name of the project and branch or project and commit. You must supply the path because the contents of the repository are not returned at the root of the archive.

GitLab also provides a means of downloading a package as a tarball:

- `https://gitlab.com/organization/project/-/archive/develop/project-develop.tar.gz?path=project-develop`

If the GitHub or GitLab repository is private, you can generate a personal access token providing read-only access to the repository and include the credentials in the URL:

- `https://username@token:github.com/organization/project/archive/develop.tar.gz?path=project-develop`

With this method, you supply a full URL to request a tarball of the repository and it does not refer to the repository itself. You can also reference private enterprise versions of GitHub or GitLab and the repository doesn't need to be on the public `github.com` or `gitlab.com` sites.

The last case is a reference to an OCI image artifact stored on a registry. This is not a full container image with the operating system, but an image containing only the files making up the workshop content. The URI formats for this are:

- `imgpkg+https://harbor.example.com/organisation/project:version` - Use the workshop content from the top-level directory of the unpacked OCI artifact. The registry in this case must support `https`.
- `imgpkg+https://harbor.example.com/organisation/project:version?path=subdir` - Use the workshop content from the subdirectory path of the unpacked OCI artifact you specify. The registry in this case must support `https`.
- `imgpkg+http://harbor.example.com/organisation/project:version` - Use the workshop content from the top-level directory of the unpacked OCI artifact. The registry in this case can only support `http`.
- `imgpkg+http://harbor.example.com/organisation/project:version?path=subdir` - Use the workshop content from the subdirectory path of the unpacked OCI artifact you specify. The registry in this case can only support `http`.

You can use `imgpkg://` instead of the prefix `imgpkg+https://`. The registry in this case must still support `https`.

For any of the formats, you can supply credentials as part of the URI:

- `imgpkg+https://username:password@harbor.example.com/organisation/project:version`

Access to the registry using a secure connection of `https` must have a valid certificate.

You can create the OCI image artifact by using `imgpkg` from the Carvel tool set. For example, from the top-level directory of the Git repository containing the workshop content, run:



```
imgpkg push -i harbor.example.com/organisation/project:version -f .
```

In all cases for downloading workshop content, the `workshop` subdirectory holding the actual workshop content is relocated to `/opt/workshop` so that it is not visible to a user. If you want to ignore other files so the user can not see them, you can supply a `.eduk8signore` file in your repository or tarball and list patterns for the files in it.

The contents of the `.eduk8signore` file are processed as a list of patterns and each is applied recursively to subdirectories. To ensure that a file is only ignored if it resides in the root directory, prefix it with `./`:

```
./.dockerignore
./.gitignore
./Dockerfile
./LICENSE
./README.md
./kustomization.yaml
./resources
```

## Container image for the workshop

When you bundle the workshop content into a container image, the `content.image` field must specify the image reference identifying the location of the container image that you will deploy for the workshop instance:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  title: Markdown Sample
  description: A sample workshop using Markdown
  content:
    image: {YOUR-REGISTRY-URL}/lab-markdown-sample:main
```

Even though you can download workshop content when the workshop environment starts, you might still want to override the workshop image that is used as a base. You can do this when you have a custom workshop base image that includes added language runtimes or tools that the specialized workshops require.

For example, if running a Java workshop, you can enter the `jdk11-environment` for the workshop image. The workshop content is still downloaded from GitHub:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-spring-testing
spec:
  title: Spring Testing
  description: Playground for testing Spring development
  content:
    image: registry.tanzu.vmware.com/learning-center/jdk11-environment:latest
    files: {YOUR-GIT-REPO-URL}/lab-spring-testing
```

If you want to use the latest version of an image, always include the `:latest` tag. This is important because the Learning Center Operator looks for version tags `:main`, `:main`, `:develop` and `:latest`. When using these tags, the Operator sets the image pull policy to `Always` to ensure that a newer version is always pulled if available. Otherwise, the image is cached on the Kubernetes nodes and only pulled when it is initially absent. Any other version tags are always assumed to be unique and

are never updated. Be aware of image registries that use a content delivery network (CDN) as front end. When using these image tags, the CDN can still regard them as unique and not do pull through requests to update an image even if it uses a tag of `:latest`.

When special custom workshop base images are available as part of the Learning Center project, instead of specifying the full location for the image, including the image registry, you can specify a short name. The Learning Center Operator then fills in the rest of the details:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-spring-testing
spec:
  title: Spring Testing
  description: Playground for testing Spring development
  content:
    image: jdk11-environment:latest
    files: github.com/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: {YOUR-GIT-REPO-URL}/lab-markdown-sample
  session:
    env:
      - name: REPOSITORY-URL
        value: YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE
```

Where:

- The `session.env` field is a list of dictionaries with the `name` and `value` fields.
- The `value` field is the Git repository for `lab-markdown-sample`. For example, `{YOUR-GIT-REPO-URL}/lab-markdown-sample`.

Values of fields in the list of resource objects can reference a number of predefined parameters. The available parameters are:

- `session_id`: A unique ID for the workshop instance within the workshop environment.

- `session_namespace`: The namespace you create for and bind to the workshop instance. This is the namespace unique to the session. A workshop can create its own resources.
- `environment_name`: The name of the workshop environment. Its current value is the name of the namespace for the workshop environment and subject to change.
- `workshop_namespace`: The namespace for the workshop environment. This is the namespace where you create all deployments of the workshop instances. It is also the namespace where the service account that the workshop instance runs.
- `service_account`: The name of the service account that the workshop instance runs as. It has access to the namespace you create for that workshop instance.
- `ingress_domain`: The host domain under which you can create host names when creating ingress routes.
- `ingress_protocol`: The protocol (http/https) you use for ingress routes and create for workshops.

The syntax for referencing the parameters is `$(parameter_name)`.

Use the `session.env` field to override environment variables only when they are required for the workshop. To set or override an environment for a specific workshop environment, set environment variables in the `WorkshopEnvironment` custom resource for the workshop environment instead.

## Overriding the memory available

By default the container the workshop environment runs in is allocated 512Mi. If the editor is enabled, a total of 1Gi is allocated.

The memory allocation is sufficient for the workshop that is mainly aimed at deploying workloads into the Kubernetes cluster. If you run workloads in the workshop environment container and need more memory, you can override the default by setting `memory` under `session.resources`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  title: Markdown Sample
  description: A sample workshop using Markdown
  content:
    image: {YOUR-REGISTRY-URL}/lab-markdown-sample:main
  session:
    resources:
      memory: 2Gi
```

## Mounting a persistent volume

In circumstances where a workshop needs persistent storage to ensure no loss of work, you can request a persistent volume be mounted into the workshop container after the workshop environment container is stopped and restarted:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  title: Markdown Sample
  description: A sample workshop using Markdown
```

```

content:
  image: {YOUR-REGISTRY-URL}/lab-markdown-sample:main
session:
  resources:
    storage: 5Gi

```

The persistent volume is mounted on top of the `/home/eduk8s` directory. Because this hides any workshop content bundled with the image, an init container is automatically configured and run, which copies the contents of the home directory to the persistent volume before the persistent volume is mounted on top of the home directory.

## Resource budget for namespaces

In conjunction with each workshop instance, a namespace is created during the workshop. From the terminal of the workshop, you can deploy dashboard applications into the namespace through the Kubernetes REST API by using tools such as `kubectl`.

By default, this namespace has all the limit ranges and resource quotas the Kubernetes cluster can enforce. In most cases, this means there are no limits or quotas.

To control how much resources you can use when you set no limit ranges and resource quotas, or override any default limit ranges and resource quotas, you can set a resource budget for any namespace of the workshop instance in the `session.namespaces.budget` field:

```

apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  title: Markdown Sample
  description: A sample workshop using Markdown
  content:
    image: {YOUR-REGISTRY-URL}/lab-markdown-sample:main
  session:
    namespaces:
      budget: small

```

The resource budget sizings and quotas for CPU and memory are:

| Budget    | CPU   | Memory |
|-----------|-------|--------|
| small     | 1000m | 1Gi    |
| medium    | 2000m | 2Gi    |
| large     | 4000m | 4Gi    |
| x-large   | 8000m | 8Gi    |
| xx-large  | 8000m | 12Gi   |
| xxx-large | 8000m | 16Gi   |

A value of 1000m is equivalent to 1 CPU.

Separate resource quotas for CPU and memory are applied for terminating and non-terminating workloads.

Only the CPU and memory quotas are listed in the preceding table, but limits also apply to the number of resource objects of certain types you can create, such as:

- persistent volume claims
- replication controllers

- services
- secrets

For each budget type, a limit range is created with fixed defaults. The limit ranges for CPU usage on a container are as follows:

| Budget    | Minimum | Maximum | Request | Limit |
|-----------|---------|---------|---------|-------|
| small     | 50m     | 1000m   | 50m     | 250m  |
| medium    | 50m     | 2000m   | 50m     | 500m  |
| large     | 50m     | 4000m   | 50m     | 500m  |
| x-large   | 50m     | 8000m   | 50m     | 500m  |
| xx-large  | 50m     | 8000m   | 50m     | 500m  |
| xxx-large | 50m     | 8000m   | 50m     | 500m  |

The limit ranges for memory are as follows:

| Budget    | Minimum | Maximum | Request | Limit |
|-----------|---------|---------|---------|-------|
| small     | 32Mi    | 1Gi     | 128Mi   | 256Mi |
| medium    | 32Mi    | 2Gi     | 128Mi   | 512Mi |
| large     | 32Mi    | 4Gi     | 128Mi   | 1Gi   |
| x-large   | 32Mi    | 8Gi     | 128Mi   | 2Gi   |
| xx-large  | 32Mi    | 12Gi    | 128Mi   | 2Gi   |
| xxx-large | 32Mi    | 16Gi    | 128Mi   | 2Gi   |

The request and limit values are the defaults of a container when there is no resources specification in a pod specification.

You can supply overrides in `session.namespaces.limits` to override the limit ranges and defaults for request and limit values when a budget sizing for CPU and memory is sufficient and there is no resources specification in a pod specification:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  title: Markdown Sample
  description: A sample workshop using Markdown
  content:
    image: {YOUR-REGISTRY-URL}/lab-markdown-sample:main
  session:
    namespaces:
      budget: medium
      limits:
        min:
          cpu: 50m
          memory: 32Mi
        max:
          cpu: 1
          memory: 1Gi
      defaultRequest:
        cpu: 50m
        memory: 128Mi
      default:
```

```
cpu: 500m
memory: 1Gi
```

Although all the configurable properties are listed in this example, you only need to supply the property for the value that you want to override.

If you need more control over the limit ranges and resource quotas, you can set the resource budget to `custom`. This removes any default limit ranges and resource quota that might be applied to the namespace. You can enter your own `LimitRange` and `ResourceQuota` resources as part of the list of resources created for each session.

Before disabling the quota and limit ranges or contemplating any switch to using a custom set of `LimitRange` and `ResourceQuota` resources, consider if that is what is really required.

The default requests defined by these for memory and CPU are fallbacks only. In most cases, instead of changing the defaults, you can enter the memory and CPU resources in the pod template specification of your deployment resources used in the workshop to indicate what the application requires. This allows you to control exactly what the application can use and so fit into the minimum quota required for the task.

This budget setting and the memory values are distinct from the amount of memory the container the workshop environment runs in. To change how much memory is available to the workshop container, set the `memory` setting under `session.resources`.

## Patching workshop deployment

In order to set or override environment variables, you can provide `session.env`. To make other changes to the Pod template for the deployment used to create the workshop instance, provide an overlay patch. You can use this patch to override the default CPU and memory limit applied to the workshop instance or to mount a volume.

The patches are provided by setting `session.patches`. The patch is applied to the `spec` field of the pod template:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-resource-testing
spec:
  title: Resource testing
  description: Play area for testing memory resources
  content:
    files: github.com/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:
              - kpack.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`, `pks-restricted` or `vmware-system-tmc-restricted`, no matter what the name of the global policy default.

## Custom security policies for user containers

You can also set the value of the `session.namespaces.security.policy` setting as `custom`. This gives you more fine-grained control of the security policy applied to the pods and containers that a user deploys during a session. In this case you must provide your own resources that define and map the pod security policy.

For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-policy-testing
spec:
  title: Policy Testing
  description: Play area for testing policy override
  content:
    files: github.com/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](https://learningcenter.tanzu.vmware.com/session.security.policy: custom) annotation.

## Defining additional ingress points

If running additional background applications, by default they are only accessible to other processes within the same container. For an application to be accessible to a user through their web browser, an ingress must be created mapping to the port for the application.

You can do this by supplying a list of the ingress points and the internal container port they map to by setting the `session.ingresses` field in the workshop definition:

```

apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    ingresses:
    - name: application
      port: 8080

```

The form of the host name used in the URL to access the service is:

```
$(session_namespace)-application.$(ingress_domain)
```

This name cannot be `terminal`, `console`, `slides`, `editor`, or the name of any built-in dashboard. These values are reserved for the corresponding built-in capabilities providing those features.

In addition to specifying ingresses for proxying to internal ports within the same Pod, you can enter a `host`, `protocol` and `port` corresponding to a separate service running in the Kubernetes cluster:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    ingresses:
      - name: application
        protocol: http
        host: service.namespace.svc.cluster.local
        port: 8080
```

You can use variables providing information about the current session within the `host` property if required:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    ingresses:
      - name: application
        protocol: http
        host: service.$(session_namespace).svc.cluster.local
        port: 8080
```

Available variables are:

- `session_namespace`: The namespace you create for and bind to the workshop instance. This is the namespace unique to the session and where a workshop can create its own resources.
- `environment_name`: The name of the workshop environment. Its current value is the name of the namespace for the workshop environment and subject to change.
- `workshop_namespace`: The namespace for the workshop environment. This is the namespace where you create all deployments of the workshop instances and where the service account that the workshop instance runs.
- `ingress_domain`: The host domain under which you can create host names when creating ingress routes.

If the service uses standard `http` or `https` ports, you can leave out the `port` property, and the port is set based on the value of `protocol`.

When a request is proxied, you can specify additional request headers that must be passed to the service:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    ingresses:
      - name: application
        protocol: http
        host: service.${session_namespace}.svc.cluster.local
        port: 8080
        headers:
          - name: Authorization
            value: "Bearer ${kubernetes_token}"
```

The value of a header can reference the following variable:

- `kubernetes_token`: The access token of the service account for the current workshop session, used for accessing the Kubernetes REST API.

Access controls enforced by the workshop environment or training portal protect accessing any service through the ingress. If you use the training portal, this must be transparent. Otherwise, supply any login credentials for the workshop again when prompted by your web browser.

## External workshop instructions

In place of using workshop instructions provided with the workshop content, you can use externally hosted instructions instead. To do this set `sessions.applications.workshop.url` to the URL of an external web site:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      workshop:
        url: https://www.example.com/instructions
```

The external web site must be displayed in an HTML iframe, as shown, and must provide its own page navigation and table of contents if required.

The URL value can reference a number of predefined parameters. The available parameters are:

- `session_namespace`: The namespace you create for and bind to the workshop instance. This is the namespace unique to the session and where a workshop can create its own resources.
- `environment_name`: The name of the workshop environment. Its current value is the name of the namespace for the workshop environment and subject to change.

- `workshop_namespace`: The namespace for the workshop environment. This is the namespace where you create all deployments of the workshop instances and where the service account that the workshop instance runs.
- `ingress_domain`: The host domain under which you can create host names when creating ingress routes.
- `ingress_protocol`: The protocol (http/https) used for ingress routes that you create for workshops.

These could be used, for example, to reference workshops instructions hosted as part of the workshop environment:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      workshop:
        url: $(ingress_protocol)://$(workshop_namespace)-instructions.$(ingress_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.

## Deactivating workshop instructions

The aim of the workshop environment is to provide instructions for a workshop that users can follow. If you want instead to use the workshop environment as a development environment or as an administration console that provides access to a Kubernetes cluster, you can deactivate the display of workshop instructions provided with the workshop content. In this case, only the work area with the terminals, console, and so on, is displayed. To deactivate display of workshop instructions, add a `session.applications.workshop` section and set the `enabled` property to `false`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      workshop:
        enabled: false
```

## Enabling the Kubernetes console

By default the Kubernetes console is not enabled. To enable it and make it available through the web browser when accessing a workshop, add a `session.applications.console` section to the workshop definition, and set the `enabled` property to `true`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      console:
        enabled: true
```

The Kubernetes dashboard provided by the Kubernetes project is used. To use Octant as the console, you can set the `vendor` property to `octant`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      console:
        enabled: true
        vendor: octant
```

When `vendor` is not set, `kubernetes` is assumed.

## Enabling the integrated editor

By default the integrated web based editor is not enabled. To enable it and make it available through the web browser when accessing a workshop, add a `session.applications.editor` section to the workshop definition, and set the `enabled` property to `true`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      editor:
        enabled: true
```

The integrated editor used is based on Visual Studio Code. For more information about the editor, see <https://github.com/cdr/code-server> in GitHub.

To install additional VS Code extensions, do this from the editor. Alternatively, if building a custom workshop, you can install them from your `Dockerfile` into your workshop image by running:

```
code-server --install-extension vendor.extension
```

Replace `vendor.extension` with the name of the extension, where the name identifies the extension on the VS Code extensions marketplace used by the editor or provide a path name to a local `.vsix` file.

This installs the extensions into `$HOME/.config/code-server/extensions`.

If downloading extensions yourself and unpacking them or extensions are part of your Git repository, you can instead locate them in the `workshop/code-server/extensions` directory.

## Enabling workshop downloads

You can provide a way for a workshop user to download files as part of the workshop content. Enable this by adding the `session.applications.files` section to the workshop definition and setting the `enabled` property to `true`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      files:
        enabled: true
```

The recommended way of providing access to files from workshop instructions is using the `files:download-file` clickable action block. This action ensures any file is downloaded to the local machine and is not displayed in the browser in place of the workshop instructions.

By default the user can access any files located under the home directory of the workshop user account. To restrict where the user can download files from, set the `directory` setting:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      files:
        enabled: true
        directory: exercises
```

When the specified directory is a relative path, it is evaluated relative to the home directory of the workshop user.

## Enabling the test examiner

The test examiner is a feature that allows a workshop to have verification checks that the workshop instructions can trigger. The test examiner is deactivated by default. To enable it, add a `session.applications.examiner` section to the workshop definition and set the `enabled` property to `true`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      examiner:
        enabled: true
```

You must provide any executable test programs for verification checks in the `workshop/examiner/tests` directory.

The test programs must return an exit status of 0 if the test is successful and nonzero if it fails. Test programs must not be persistent programs that can run forever.

Clickable actions for the test examiner are used within the workshop instructions to trigger the verification checks. You can configure them to start when the page of the workshop instructions is loaded.

## Enabling session image registry

Workshops using tools such as `kpack` or `tekton` and which need a place to push container images when built can enable a container image registry. A separate registry is deployed for each workshop session.

The container image registry is currently fully usable only if workshops are deployed under a Learning Center Operator configuration that uses secure ingress. This is because a registry that is not secure is not trusted by the Kubernetes cluster as the source of container images when doing deployments.

To enable the deployment of a registry per workshop session, add a `session.applications.registry` section to the workshop definition and set the `enabled` property to `true`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      registry:
        enabled: true
```

The registry mounts a persistent volume for storing of images. By default the size of that persistent volume is 5Gi. To override the size of the persistent volume, add the `storage` property under the `registry` section:

```

apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      registry:
        enabled: true
        storage: 20Gi

```

The amount of memory provided to the registry defaults to 768Mi. To increase this, add the `memory` property under the `registry` section.

```

apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      registry:
        enabled: true
        memory: 1Gi

```

The registry is secured with a user name and password unique to the workshop session, and must be accessed over a secure connection.

To allow access from the workshop session, the file `$(HOME)/.docker/config.json` containing the registry credentials are injected into the workshop session. This is used by tools such as `docker`.

For deployments in Kubernetes, a secret of type `kubernetes.io/dockerconfigjson` is created in the namespace and applied to the `default` service account in the namespace. This means deployments made using the default service account can pull images from the registry without additional configuration. If creating deployments using other service accounts, add configuration to the service account or deployment to add the registry secret for pulling images.

If you need access to the raw registry host details and credentials, they are provided as environment variables in the workshop session. The environment variables are:

- `REGISTRY_HOST`: Contains the host name for the registry for the workshop session.
- `REGISTRY_AUTH_FILE`: Contains the location of the `docker` configuration file. Must be the equivalent of `$(HOME)/.docker/config.json`.
- `REGISTRY_USERNAME`: Contains the user name for accessing the registry.
- `REGISTRY_PASSWORD`: Contains the password for accessing the registry. This is different for each workshop session.
- `REGISTRY_SECRET`: Contains the name of a Kubernetes secret of type `kubernetes.io/dockerconfigjson` added to the session namespace, which contains the registry credentials.

The URL for accessing the registry adopts the HTTP protocol scheme inherited from the environment variable `INGRESS_PROTOCOL`. This is the same HTTP protocol scheme the workshop



sessions use.

To use any of the variables as data variables in workshop content, use the same variable name but in lowercase: `registry_host`, `registry_auth_file`, `registry_username`, `registry_password` and `registry_secret`.

## Enabling ability to use Docker

To build container images in a workshop using `docker`, first enable it. Each workshop session is provided with its own separate Docker daemon instance running in a container.

Enabling support for running `docker` requires the use of a privileged container for running the Docker daemon. Because of the security implications of providing access to Docker with this configuration, VMware recommends that if you don't trust the people taking the workshop, any workshops that require Docker only be hosted in a disposable Kubernetes cluster that is destroyed at the completion of the workshop. You must not enable Docker for workshops hosted on a public service that is always kept running and where arbitrary users can access the workshops.

To enable support for using `docker` add a `session.applications.docker` section to the workshop definition and set the `enabled` property to `true`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      docker:
        enabled: true
```

The container that runs the Docker daemon mounts a persistent volume for storing of images which are pulled down or built locally. By default the size of that persistent volume is 5Gi. To override the size of the persistent volume, add the `storage` property under the `docker` section:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      docker:
        enabled: true
        storage: 20Gi
```

The amount of memory provided to the container running the Docker daemon defaults to 768Mi. To increase this, add the `memory` property under the `registry` section:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
```

```
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      docker:
        enabled: true
        memory: 1Gi
```

Access to the Docker daemon from the workshop session uses a local UNIX socket shared with the container running the Docker daemon. If it uses a local tool to access the socket connection for the Docker daemon directly rather than by running `docker`, it must use the `DOCKER_HOST` environment variable to set the location of the socket.

The Docker daemon is only available from within the workshop session and cannot be accessed outside of the pod by any tools deployed separately to Kubernetes.

## Enabling WebDAV access to files

You can access or update local files within the workshop session from the terminal command line or editor of the workshop dashboard. The local files reside in the file system of the container the workshop session is running in.

To access the files remotely, you can enable WebDAV support for the workshop session.

To enable support for accessing files over WebDAV, add a `session.applications.webdav` section to the workshop definition, and set the `enabled` property to `true`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      webdav:
        enabled: true
```

This causes a WebDAV server running within the workshop session environment. A set of credentials is also generated and are available as environment variables. The environment variables are:

- `WEBDAV_USERNAME`: Contains the user name that must be used when authenticating over WebDAV.
- `WEBDAV_PASSWORD`: Contains the password that must be used when authenticating over WebDAV.

To use any of the environment variables related to the container image registry as data variables in workshop content, declare this in the `workshop/modules.yaml` file in the `config.vars` section:

```
config:
  vars:
    - name: WEBDAV_USERNAME
    - name: WEBDAV_PASSWORD
```

The URL endpoint for accessing the WebDAV server is the same as the workshop session, with `/webdav/` path added. This can be constructed from the terminal using:

```
$INGRESS_PROTOCOL://$SESSION_NAMESPACE.$INGRESS_DOMAIN/webdav/
```

In workshop content it can be constructed using:

```
{{ingress_protocol}}://{{session_namespace}}.{{ingress_domain}}/webdav/
```

You can use WebDAV client support provided by your operating system or by using a standalone WebDAV client, such as [CyberDuck](#).

Using WebDAV can make it easier to transfer files to or from the workshop session.

## Customizing the terminal layout

By default a single terminal is provided in the web browser when accessing the workshop. If required, you can enable alternate layouts which provide additional terminals. To set the layout, add the `session.applications.terminal` section and include the `layout` property with the desired layout:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      terminal:
        enabled: true
        layout: split
```

The options for the `layout` property are:

- `default`: Single terminal.
- `split`: Two terminals stacked above each other in ratio 60/40.
- `split/2`: Three terminals stacked above each other in ratio 50/25/25.
- `lower`: A single terminal is placed below any dashboard tabs, rather than being a tab of its own. The ratio of dashboard tab to terminal is 70/30.
- `none`: No terminal is displayed but can still be created from the drop down menu.

When adding the `terminal` section, you must include the `enabled` property and set it to `true` as it is a required field when including the section.

If you don't want a terminal displayed and also want to deactivate the ability to create terminals from the drop-down menu, set `enabled` to `false`.

## Adding custom dashboard tabs

Exposed applications, external sites and additional terminals, can be given their own custom dashboard tab. This is done by specifying the list of dashboard panels and the target URL:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
```

```

metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    ingresses:
      - name: application
        port: 8080
    dashboards:
      - name: Internal
        url: "${(ingress_protocol)}://${(session_namespace)}-application.${(ingress_domain)}/"
      - name: External
        url: http://www.example.com

```

The URL values can reference a number of predefined parameters. The available parameters are:

- `session_namespace`: The namespace you create for and bind to the workshop instance. This is the namespace unique to the session and where a workshop can create its own resources.
- `environment_name`: The name of the workshop environment. Its current value is the name of the namespace for the workshop environment and subject to change.
- `workshop_namespace`: The namespace for the workshop environment. This is the namespace where all deployments of the workshop instances you create and where the service account that the workshop instance runs.
- `ingress_domain`: The host domain under which you can create host names when creating ingress routes.
- `ingress_protocol`: The protocol (http/https) used for ingress routes that you create for workshops.

The URL can reference an external web site, however, that web site must not prohibit being embedded in an HTML iframe.

In the case of wanting to have a custom dashboard tab provide an additional terminal, the `url` property must use the form `terminal:<session>`, where `<session>` is replaced with the name of the terminal session. The name of the terminal session can be any name you choose, but must be restricted to lowercase letters, numbers, and dashes. You should avoid using numeric terminal session names such as “1”, “2”, and “3” as these are used for the default terminal sessions.

```

apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    dashboards:
      - name: Example
        url: terminal:example

```

## Configure the WorkshopEnvironment resource

This topic describes how you configure the `WorkshopEnvironment` custom resource, which defines a Learning Center workshop environment.

## Specifying the workshop definition

Creating a workshop environment is performed as a separate step to loading the workshop definition. This allows multiple distinct workshop environments using the same workshop definition to be created if necessary.

To specify which workshop definition is to be used for a workshop environment, set the `workshop.name` field of the specification for the workshop environment.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
```

The workshop environment name specified in the workshop environment metadata does not need to be the same. It has to be different if you create multiple workshop environments from the same workshop definition.

When the workshop environment is created, the namespace created for the workshop environment uses the `name` specified in the `metadata`. This name is also used in the unique names of each workshop instance created under the workshop environment.

## Overriding environment variables

A workshop definition can set a list of environment variables that must be set for all workshop instances. To override an environment variable specified in the workshop definition, or one defined in the container image, you can supply a list of environment variables as `session.env`.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  session:
    env:
      - name: REPOSITORY-URL
        value: YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE
```

Where `YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE` is the Git repository URL for `lab-markdown-sample`. For example, `{YOUR-GIT-REPO-URL}/lab-markdown-sample`.

You can use this to set the location of a back-end service, such as an image registry, used by the workshop.

Values of fields in the list of resource objects can reference several predefined parameters. The available parameters are:

- `session_` - A unique ID for the workshop instance within the workshop environment.
- `session_` - The namespace created for and bound to the workshop instance. This is the namespace unique to the session and where a workshop can create its own resources.
- `environment_` - The name of the workshop environment. Currently, this is the same as the name of the namespace for the workshop environment. It is suggested that you do not rely on workshop environment name and namespace being the same, and use the most appropriate to cope with any future change.

- `workshop_` - The namespace for the workshop environment. This is the namespace where all deployments of the workshop instances are created and where the workshop instance runs the service account exists.
- `service_` - The workshop instance service account's name and access to the namespace created for that workshop instance.
- `ingress_` - The host domain under which host names are created when creating ingress routes.
- `ingress_` - The protocol (http/https) used for ingress routes created for workshops.

The syntax for referencing one of the parameters is `$(parameter_name)`.

## Overriding the ingress domain

To access a workshop instance using a public URL, you must specify an ingress domain. If an ingress domain is not specified, the default ingress domain that the Learning Center operator configured with is used.

When setting a custom domain, DNS must be configured with a wildcard domain to forward all requests for subdomains of the custom domain to the ingress router of the Kubernetes cluster.

To provide the ingress domain, you can set the `session.ingress.domain` field.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  session:
    ingress:
      domain: training.learningcenter.tanzu.vmware.com
```

By default, the workshop session is exposed using an HTTP connection if overriding the domain. If you require a secure HTTPS connection, you must have access to a wildcard SSL certificate for the domain. A secret of type `tls` must be created for the certificate in the `learningcenter` namespace or the namespace where the Learning Center Operator is deployed. The name of that secret must then be set in the `session.ingress.secret` field.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  session:
    ingress:
      domain: training.learningcenter.tanzu.vmware.com
      secret: training.learningcenter.tanzu.vmware.com-tls
```

If HTTPS connections are terminated using an external load balancer and not by specifying a secret for ingresses managed by the Kubernetes ingress controller, then routing traffic into the Kubernetes cluster as HTTP connections, you can override the ingress protocol without specifying an ingress secret by setting the `session.ingress.protocol` field.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
```

```

name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  session:
    ingress:
      domain: training.learningcenter.tanzu.vmware.com
      protocol: https

```

To override or set the ingress class, which dictates which ingress router is used when more than one option is available, you can add `session.ingress.class`.

```

apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  session:
    ingress:
      domain: training.learningcenter.tanzu.vmware.com
      secret: training.learningcenter.tanzu.vmware.com-tls
      class: nginx

```

## Controlling access to the workshop

By default, requesting a workshop using the `WorkshopRequest` custom resource is deactivated and must be enabled for a workshop environment by setting `request.enabled` to `true`.

```

apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  request:
    enabled: true

```

With this enabled, anyone who can create a `WorkshopRequest` custom resource can request the creation of a workshop instance for the workshop environment.

To further control who can request a workshop instance in the workshop environment, you can first set an access token, which a user must know and supply with the workshop request. This is done by setting the `request.token` field.

```

apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  request:
    enabled: true
    token: lab-markdown-sample

```

The same name as the workshop environment is used in this example, which is probably not a good practice. Use a random value instead. The token value may be multiline.

As a second control measure, you can specify what namespaces the `WorkshopRequest` must be created. This means a user must have the specific ability to create `WorkshopRequest` resources in one of those namespaces.

You can specify the list of namespaces from which workshop requests for the workshop environment by setting `request.namespaces`.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  request:
    enabled: true
    token: lab-markdown-sample
    namespaces:
      - default
```

To add the workshop namespace in the list, rather than list the literal name, you can reference a predefined parameter specifying the workshop namespace by including `$(workshop_namespace)`.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  request:
    enabled: true
    token: lab-markdown-sample
    namespaces:
      - $(workshop_namespace)
```

## Overriding the login credentials

When requesting a workshop using `WorkshopRequest`, a login dialog box is presented to the user when accessing the workshop instance URL. By default, the user name is `learningcenter`. The password is a random value the user must query from the `WorkshopRequest` status after creating the custom resource.

To override the user name, you can set the `session.username` field. To set the same fixed password for all workshop instances, you can set the `session.password` field.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  session:
    username: workshop
    password: lab-markdown-sample
```

## Additional workshop resources



The workshop definition defined by the `Workshop` custom resource already declares a set of resources to be created with the workshop environment. You can use this when you have shared service applications the workshop needs, such as a 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.

## Configure the WorkshopRequest resource

This topic describes how you configure the `WorkshopRequest` custom resource, which defines a Learning Center workshop request.

### Specifying workshop environment

The `WorkshopRequest` custom resource is used to request a workshop instance. It does not provide details needed to perform the deployment of the workshop instance. That information is sourced by the Learning Center Operator from the `WorkshopEnvironment` and `Workshop` custom resources.

The minimum required information in the workshop request is the name of the workshop environment. You supply this by setting the `environment.name` field.

For example:

```

apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopRequest
metadata:
  name: lab-markdown-sample

```

```
spec:
  environment:
    name: lab-markdown-sample
```

A request is successful only if requesting a workshop instance for a workshop environment is enabled for that workshop. You can enable requests in the `WorkshopEnvironment` custom resource for the workshop environment.

If multiple workshop requests, for the same workshop environment or different ones, are created in the same namespace, the `name` defined in the `metadata` for the workshop request must be different for each. The value of this name is not used to name workshop instances. You need the `name` value to delete the workshop instance, which is done by deleting the workshop request.

## Specifying required access token

If a workshop environment is configured to require an access token when making a workshop request against that environment, you can specify the token by setting the `environment.token` field.

For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopRequest
metadata:
  name: lab-markdown-sample
spec:
  environment:
    name: lab-markdown-sample
    token: lab-markdown-sample
```

Even with the token, the request fails if the following is true:

- The workshop environment has restricted the namespaces from which a workshop request was made
- The workshop request was not created in one of the permitted namespaces

## Configure the TrainingPortal resource

This topic describes how you configure the `TrainingPortal` custom resource, which triggers the deployment of a set of Learning Center workshop environments and a set number of workshop instances.

### Specifying the workshop definitions

You run multiple workshop instances to perform training to a group of people by creating the workshop environment and then creating each workshop instance. The `TrainingPortal` workshop resource bundles that up as one step.

Before creating the training environment, you must load the workshop definitions as a separate step.

To specify the names of the workshops to be used for the training, list them under the `workshops` field of the training portal specification. Each entry needs to define a `name` property, matching the name of the `Workshop` resource you created.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: sample-workshops
```

```
spec:
  portal:
    sessions:
      maximum: 8
  workshops:
  - name: lab-asciidoc-sample
  - name: lab-markdown-sample
```

When the training portal is created, it:

- Sets up the underlying workshop environments.
- Creates any workshop instances required to be created initially for each workshop.
- Deploys a web portal for attendees of the training to access their workshop instances.

## Limit the number of sessions

When defining the training portal, you can set a limit on the workshop sessions that can be run concurrently. Set this limit by using the `portal.sessions.maximum` property:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: sample-workshops
spec:
  portal:
    sessions:
      maximum: 8
  workshops:
  - name: lab-asciidoc-sample
  - name: lab-markdown-sample
```

When you specify this, the maximum capacity of each workshop is set to the maximum value for the training portal as a whole. This means that any one workshop can have as many sessions running as specified by the maximum for the portal. However, to achieve this maximum for a given workshop, only instances of that workshop can be created. In other words, the maximum capacity can be spread across a number of workshops or it can be used in its entirety by a single workshop.

If you do not set `portal.sessions.maximum`, you must set the capacity for each individual workshop as detailed in the following section. In only setting the capacities of each workshop and not an overall maximum for sessions, you cannot share the overall capacity of the training portal across multiple workshops.

## Capacity of individual workshops

When you have more than one workshop, you can want to limit how many instances of each workshop you can have so that they cannot grow to the maximum number of sessions for the whole training portal. This means you can stop a specific workshop from using all of the capacity of the training portal. To do this, set the `capacity` field under the entry for the workshop:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: sample-workshops
spec:
  portal:
    sessions:
      maximum: 8
  workshops:
  - name: lab-asciidoc-sample
```

```

capacity: 4
- name: lab-markdown-sample
  capacity: 6

```

The value of `capacity` limits the number of workshop sessions for a specific workshop to that value. It must be less than or equal to the maximum number of workshops sessions for the portal, because the latter always sets the absolute limit.

## Set reserved workshop instances

By default one instance of each of the listed workshops is created so when the initial user requests that workshop, it's available for use immediately.

When such a reserved instance is allocated to a user, provided that the workshop capacity hasn't been reached, a new instance of the workshop is created as a reserve ready for the next user. When a user ends a workshop and the workshop is at capacity, when the instance is deleted, a new reserve is created. The total of allocated and reserved sessions for a workshop cannot exceed the capacity for that workshop.

To override for a specific workshop how many reserved instances are kept on standby ready for users, you can set the `reserved` setting against the workshop:

```

apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: sample-workshops
spec:
  portal:
    sessions:
      maximum: 8
  workshops:
  - name: lab-asciidoc-sample
    capacity: 4
    reserved: 2
  - name: lab-markdown-sample
    capacity: 6
    reserved: 4

```

You can set the value of `reserved` to 0 if you never want any reserved instances for a workshop and only want instances of that workshop created on demand when required for a user. Creating instances of a workshop on demand can result in a user waiting longer to access a workshop session.

When workshop instances are always created on demand, the oldest reserved instance is terminated to allow a new session of a desired workshop to be created. This also happens when reserved instances tie up capacity that could be used for a new session of another workshop. This occurs if any caps for specific workshops are met.

## Override initial number of sessions

The initial number of workshop instances created for each workshop is specified by `reserved` or 1 if the setting hasn't been provided.

In the case where `reserved` is set in order to keep workshop instances on standby, you can indicate that initially you want more than the reserved number of instances created. This is useful when running a workshop for a set period of time. You might create up-front instances of the workshop corresponding to 75% of the expected number of attendees but with a smaller reserve number. With this configuration, new reserve instances only start to be created when the total number approaches 75% and all extra instances created up front have been allocated to users. This ensures

you have enough instances ready for when most people come, but you can also create other instances later if necessary:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: kubernetes-fundamentals
spec:
  portal:
    sessions:
      maximum: 100
  workshops:
  - name: lab-kubernetes-fundamentals
    initial: 75
    reserved: 5
```

## Setting defaults for all workshops

If you have a list of workshops, and they all must be set with the same values for `capacity`, `reserved`, and `initial`, rather than add settings to each, you can set defaults to apply to all workshops under the `portal` section:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: sample-workshops
spec:
  portal:
    sessions:
      maximum: 10
    capacity: 6
    reserved: 2
    initial: 4
  workshops:
  - name: lab-asciidoc-sample
  - name: lab-markdown-sample
```

## Set caps on individual users

By default a single user can run more than one workshop at a time. You can cap this to ensure that workshops run only one at a time. This prevents a user from wasting resources by starting more than one workshop and only working on one without shutting the other down.

To apply a limit on how many concurrent workshop sessions a user can start, use the `portal.sessions.registered` setting:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: sample-workshops
spec:
  portal:
    sessions:
      maximum: 8
      registered: 1
  workshops:
  - name: lab-asciidoc-sample
    capacity: 4
    reserved: 2
  - name: lab-markdown-sample
```

```
capacity: 6
reserved: 4
```

This limit also applies to anonymous users when anonymous access is enabled through the training portal web interface or if sessions are being created through the REST API. To set a limit on anonymous users, you can set `portal.sessions.anonymous` instead:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: sample-workshops
spec:
  portal:
    sessions:
      maximum: 8
      anonymous: 1
  workshops:
  - name: lab-asciidoc-sample
    capacity: 4
    reserved: 2
  - name: lab-markdown-sample
    capacity: 6
    reserved: 4
```

## Expiration of workshop sessions

After you reach the maximum capacity, no more workshops sessions can be created. After a workshop session is allocated to a user, it cannot be reassigned to another user.

If you are running a supervised workshop, set the capacity higher than the anticipated number of users in case you have more users than you expect. Use the setting for the reserved number of instances. This way, even if you set a higher capacity than needed, workshop sessions are only created as required and not all up front.

In supervised workshops, when the training is over, delete the whole training environment. All workshop sessions are then deleted.

To host a training portal over an extended period but don't know when users want to do a workshop, you can set up workshop sessions to expire after a set time. When expired, the workshop session is deleted and a new workshop session can be created in its place.

The maximum capacity is therefore the maximum at any one point in time, while the number can grow and shrink over time. So over an extended time, you can handle many more sessions than the set maximum capacity. The maximum capacity ensures you don't try to allocate more workshop sessions than you have resources for at a given time.

To set a maximum time allowed for a workshop session, use the `expires` setting:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  workshops:
  - name: lab-markdown-sample
    capacity: 8
    reserved: 1
    expires: 60m
```

The value needs to be an integer, followed by a suffix of 's', 'm' or 'h', corresponding to seconds, minutes, or hours.

The time period is calculated from when the workshop session is allocated to a user. When the time period is up, the workshop session is automatically deleted.

When an expiration period is specified, or when a user finishes a workshop or restarts the workshop, the workshop is also deleted.

To cope with users who claim a workshop session, but leave and don't use it, you can set a time period for when a workshop session with no activity is deemed orphaned and so is deleted. Do this using the `orphaned` setting:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  workshops:
  - name: lab-markdown-sample
    capacity: 8
    reserved: 1
    expires: 60m
    orphaned: 5m
```

Avoid this setting for supervised workshops where the whole event only lasts a certain length of time. This prevents a user's session from being deleted when the user takes breaks and the computer goes to sleep.

The `expires` and `orphaned` settings can also be set against `portal` to apply them to all workshops.

## Updates to workshop environments

The list of workshops for an existing training portal can be changed by modifying the training portal definition applied to the Kubernetes cluster.

If you remove a workshop from the list of workshops, the workshop environment is marked as stopping and is deleted when all active workshop sessions have completed.

If you add a workshop to the list of workshops, a new workshop environment for it is created.

Changes to settings, such as the maximum number of sessions for the training portal or capacity settings for individual workshops, are applied to existing workshop environments.

By default a workshop environment is left unchanged if the corresponding workshop definition is changed. So in the default configuration, you must explicitly delete the workshop from the list of workshops managed by the training portal and then add it back again if the workshop definition changed.

If you prefer that workshop environments be replaced when the workshop definition changes, enable this by using the `portal.updates.workshop` setting:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    sessions:
      maximum: 8
    updates:
      workshop: true
  workshops:
  - name: lab-markdown-sample
    reserved: 1
```



```
expires: 60m
orphaned: 5m
```

When using this option, use the `portal.sessions.maximum` setting to limit the number of workshop sessions that can be run for the training portal as a whole. When replacing the workshop environment, the old workshop environment is retained if there is still an active workshop session being used. If the limit isn't set, the new workshop environment is still able to grow to its specific capacity and is not limited by how many workshop sessions are running against old instances of the workshop environment.

Overall, VMware recommends updating workshop environments when workshop definitions change only in development environments when working on workshop content. This is an especially good practice until you are familiar with how the training portal replaces existing workshop environments, and the resource implications of having old and new instances of a workshop environment running at the same time.

## Override the ingress domain

To access a workshop instance using a public URL, specify an ingress domain. If an ingress domain isn't specified, the default ingress domain that the Learning Center Operator is configured with is used.

When setting a custom domain, DNS must have been configured with a wildcard domain to forward all requests for sub-domains of the custom domain to the ingress router of the Kubernetes cluster.

To provide the ingress domain, set the `portal.ingress.domain` field:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    ingress:
      domain: learningcenter.tanzu.vmware.com
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

If overriding the domain, by default the workshop session is exposed using a HTTP connection. For a secure HTTPS connection, you must have access to a wildcard SSL certificate for the domain. A secret of type `tls` should be created for the certificate in the `learningcenter` namespace or the namespace where the Learning Center Operator is deployed. The name of that secret must be set in the `portal.ingress.secret` field:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    ingress:
      domain: learningcenter.tanzu.vmware.com
      secret: learningcenter.tanzu.vmware.com-tls
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

You can terminate HTTPS connections by using an external load balancer instead of specifying a secret for ingresses managed by the Kubernetes ingress controller. In that case, when routing traffic into the Kubernetes cluster as HTTP connections, you can override the ingress protocol without specifying an ingress secret. Instead, set the `portal.ingress.protocol` field:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    ingress:
      domain: learningcenter.tanzu.vmware.com
      protocol: https
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

To override or set the ingress class, which dictates which ingress router is used when more than one option is available, you can add `portal.ingress.class`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    ingress:
      domain: learningcenter.tanzu.vmware.com
      secret: learningcenter.tanzu.vmware.com-tls
      class: nginx
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

## Override the portal host name

The default host name given to the training portal is the name of the resource with `-ui` suffix, followed by the domain specified by the resource or the default inherited from the configuration of the Learning Center Operator.

To override the generated host name, you can set `portal.ingress.hostname`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    ingress:
      hostname: labs
      domain: learningcenter.tanzu.vmware.com
      secret: learningcenter.tanzu.vmware.com-tls
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

This causes the host name to be `labs.learningcenter.tanzu.vmware.com` rather than the default generated name for this example of `lab-markdown-sample-ui.learningcenter.tanzu.vmware.com`.

## Set extra environment variables

To override any environment variables for workshop instances created for a specific work, provide the environment variables in the `env` field of that workshop:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
    env:
    - name: REPOSITORY-URL
      value: YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE
```

Where `YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE` is the Git repository URL for `lab-markdown-sample`. For example, `{YOUR-GIT-REPO-URL}/lab-markdown-sample`.

Values of fields in the list of resource objects can reference a number of predefined parameters. The available parameters are:

- `session_id` - A unique ID for the workshop instance within the workshop environment.
- `session_namespace` - The namespace created for and bound to the workshop instance. This is the namespace unique to the session and where a workshop can create its own resources.
- `environment_name` - The name of the workshop environment. For now this is the same as the name of the namespace for the workshop environment. Don't rely on them being the same, and use the most appropriate to cope with any future change.
- `workshop_namespace` - The namespace for the workshop environment. This is the namespace where all deployments of the workshop instances are created and where the service account that the workshop instance runs as exists.
- `service_account` - The name of the service account the workshop instance runs as and which has access to the namespace created for that workshop instance.
- `ingress_domain` - The host domain under which host names can be created when creating ingress routes.
- `ingress_protocol` - The protocol (http/https) used for ingress routes created for workshops.

The syntax for referencing one of the parameters is `$(parameter_name)`.

## Override portal credentials

When a training portal is deployed, the user name for the admin and robot accounts uses the defaults of `learningcenter` and `robot@learningcenter`. The passwords for each account are randomly set.

For the robot account, the OAuth application client details used with the REST API are also randomly generated.

You can see what the credentials and client details are by running `kubectl describe` against the training portal resource. This will yield output that includes:

```
Status:
  learningcenter:
    Clients:
```

```

Robot:
  Id:      ACZpcaLIT3qr725YWmXu8et9REl4HBg1
  Secret:  t5IfXbGZQThAKR43apoc9usOFVDv2BLE
Credentials:
  Admin:
    Password: 0kGmM1Yw46BZT2vCntyrRuFf1gQq5ohi
    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 want to allow only the administrator to log in, you can deactivate the registration page. Do this if you are using the REST API to create and allocate workshop sessions from a separate application:

```

apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:

```

```
portal:
  registration:
    type: one-step
    enabled: false
workshops:
- name: lab-markdown-sample
  capacity: 3
  reserved: 1
```

If rather than requiring users to register, you want to allow anonymous access, you can switch the registration type to anonymous:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    registration:
      type: anonymous
  workshops:
- name: lab-markdown-sample
  capacity: 3
  reserved: 1
```

When a user visits the training portal home page in anonymous mode, an account for that user is automatically created and the user is logged in.

## Specify an event access code

When deploying the training portal with anonymous access or open registration, anyone who knows the URL can access workshops. To at least restrict access to those who know a common event access code or password, you can set `portal.password`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    password: workshops-2020-07-01
  workshops:
- name: lab-markdown-sample
  capacity: 3
  reserved: 1
```

When anonymous access is enabled and the training portal URL is accessed, users are asked to enter an event access code before they are redirected to the list of workshops or to the login page.

## Make a list of workshops public

By default the index page providing the catalog of available workshop images is only available after a user has logged in, either through a registered account or as an anonymous user.

To make the catalog of available workshops public so they can be viewed before logging in, set the `portal.catalog.visibility` property:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
```

```
spec:
  portal:
    catalog:
      visibility: public
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

By default the catalog has visibility set to `private`. Use `public` to expose it.

This also makes it possible to access the list of available workshops from the catalog through the REST API, without authenticating against the REST API.

## Use an external list of workshops

If you are using the training portal with registration deactivated, and you are using the REST API from a separate website to control creation of sessions, you can specify an alternate URL for providing the list of workshops.

This helps when the REST API creates a session and cookies are deleted or a session URL is shared with a different user. This means the value for the `index_url` supplied with the REST API request is lost.

To set the URL for the external site, use the `portal.index` property:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    index: https://www.example.com/
    registration:
      type: one-step
      enabled: false
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

If you supply this property, passing the `index_url` when creating a workshop session using the REST API is optional, and the value of this property is used. You can still supply `index_url` when using the REST API for a user to be redirected back to a sub-category of workshops on the site. The URL provided in the training portal definition then acts only as a fallback. That is, when the redirect URL becomes unavailable, it directs the user back to the top-level page for the external list of workshops.

If a user has logged into the training portal as the admin user, the user is not redirected to the external site and still sees the training portal's list of workshops.

## Override portal title and logo

By default the web interface for the training portal displays a generic Learning Center logo and a page title of "Workshops." To override these, you can set `portal.title` and `portal.logo`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
```

```
portal:
  title: Workshops
  logo: data:image/png;base64,...
workshops:
- name: lab-markdown-sample
  capacity: 3
  reserved: 1
```

The `logo` field should be a graphical image provided in embedded data URI format. The image is displayed with a fixed height of “40px”. The field can also be a URL for an image stored on a remote web server.

## Allow the portal in an iframe

By default it is prohibited to display the web interface for the training portal in an iframe of another web site, because of content security policies applying to the training portal website.

To enable the ability to iframe the full training portal web interface or even a specific workshop session created using the REST API, provide the host name of the site that embeds it. Do this by using the `portal.theme.frame.ancestors` property:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    theme:
      frame:
        ancestors:
          - https://www.example.com
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

The property is a list of hosts, not a single value. To use a URL for the training portal in an iframe of a page, which, in turn, is embedded in another iframe of a page on a different site, list the host names of all sites.

Because the sites that embed iframes must be secure and use HTTPS, they cannot use plain HTTP. Browser policies prohibit promoting cookies to an insecure site when embedding using an iframe. If cookies cannot be stored, a user cannot authenticate against the workshop session.

## Collect analytics on workshops

To collect analytics data on usage of workshops, supply a webhook URL. When you supply a webhook URL, events are posted to the webhook URL, including:

- Workshops started
- Pages of a workshop viewed
- Expiration of a workshop
- Completion of a workshop
- Termination of a workshop

For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
```

```

metadata:
  name: lab-markdown-sample
spec:
  analytics:
    webhook:
      url: https://metrics.learningcenter.tanzu.vmware.com/?client=name&token=password
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1

```

At present there is no metrics collection service compatible with the portal webhook reporting mechanism, so create a custom service or integrate it with any existing web front end for the portal REST API service.

If the collection service needs to be provided with a client ID or access token, it must accept using query string parameters set in the webhook URL.

Include the details of the event as HTTP POST data by using the `application/json` content type:

```

{
  "portal": {
    "name": "lab-markdown-sample",
    "uid": "91dfa283-fb60-403b-8e50-fb30943ae87d",
    "generation": 3,
    "url": "https://lab-markdown-sample-ui.learningcenter.tanzu.vmware.com"
  },
  "event": {
    "name": "Session/Started",
    "timestamp": "2021-03-18T02:50:40.861392+00:00",
    "user": "c66db34e-3158-442b-91b7-25391042f037",
    "session": "lab-markdown-sample-w01-s001",
    "environment": "lab-markdown-sample-w01",
    "workshop": "lab-markdown-sample",
    "data": {}
  }
}

```

When an event has associated data, it is included in the `data` dictionary:

```

{
  "portal": {
    "name": "lab-markdown-sample",
    "uid": "91dfa283-fb60-403b-8e50-fb30943ae87d",
    "generation": 3,
    "url": "https://lab-markdown-sample-ui.learningcenter.tanzu.vmware.com"
  },
  "event": {
    "name": "Workshop/View",
    "timestamp": "2021-03-18T02:50:44.590918+00:00",
    "user": "c66db34e-3158-442b-91b7-25391042f037",
    "session": "lab-markdown-sample-w01-s001",
    "environment": "lab-markdown-sample-w01",
    "workshop": "lab-markdown-sample",
    "data": {
      "current": "workshop-overview",
      "next": "setup-environment",
      "step": 1,
      "total": 4
    }
  }
}

```



The `user` field is the same portal user identity returned by the REST API when creating workshop sessions.

The event stream only produces events for things as they happen. For a snapshot of all current workshop sessions, use the REST API to request the catalog of available workshop environments, enabling the inclusion of current workshop sessions.

## Track using Google Analytics

To record analytics data on usage of workshops by using Google Analytics, enable tracking by supplying a tracking ID for Google Analytics:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  analytics:
    google:
      trackingId: UA-XXXXXXX-1
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

Custom dimensions are used in Google Analytics to record details about the workshop a user is taking, including through which training portal and cluster it was accessed. So you can use the same Google Analytics tracking ID for multiple training portal instances running on different Kubernetes clusters.

To support use of custom dimensions in Google Analytics, configure the Google Analytics property with the following custom dimensions. They must be added in the order shown, because Google Analytics doesn't allow you to specify the index position for a custom dimension. It allocates them for you. You can't already have custom dimensions defined for the property, as the new custom dimensions must start at index of 1.

| Custom Dimension Name | Index |
|-----------------------|-------|
| workshop_name         | 1     |
| session_namespace     | 2     |
| workshop_namespace    | 3     |
| training_portal       | 4     |
| ingress_domain        | 5     |
| ingress_protocol      | 6     |

In addition to custom dimensions against page accesses, events are also generated. These include:

- Workshop/Start
- Workshop/Finish
- Workshop/Expired

If you provide a Google Analytics tracking ID with the `TrainingPortal` resource definition, it takes precedence over the `SystemProfile` resource definition.



### Note

Google Analytics is not a reliable way to collect data. Individuals or corporate firewalls can block the reporting of Google Analytics data. For more precise statistics, use the webhook URL for collecting analytics with a custom data collection platform.

## Configure the SystemProfile resource

This topic describes how you use the `SystemProfile` custom resource to configure the Learning Center operator.

You can use the default system profile to set defaults for ingress and image pull secrets. You can also select an alternate profile for specific deployments if required.



### Important

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.blockCIDRs`. This must be a CIDR block range corresponding to the subnet or a portion of a subnet you want to block. A Kubernetes `NetworkPolicy` is used to enforce the restriction. So the Kubernetes cluster must use a network layer supporting network policies, and the necessary Kubernetes controllers supporting network policies must be enabled when the cluster is installed.

If deploying to AWS, it is important to block access to the AWS endpoint for querying EC2 metadata, because it can expose sensitive information that workshop users should not 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:
    blockCIDRs:
      - 169.254.169.254/32
      - fd00:ec2::254/128
```

## Running Docker daemon rootless

If `docker` is enabled for workshops, Docker-in-Docker is run using a sidecar container. Because of the current state of running Docker-in-Docker and portability across Kubernetes environments, the `docker` daemon by default runs as `root`. Because a privileged container is also being used, this represents a security risk. Only run workshops requiring `docker` in disposable Kubernetes clusters or for users whom you trust.

You can partly mediate the risks of running `docker` in the Kubernetes cluster by running the `docker` daemon in rootless mode. However, not all Kubernetes clusters can support this due to the Linux kernel configuration or other incompatibilities.

To enable rootless mode, you can set the `dockerd.rootless` property to `true`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
```

```

metadata:
  name: default-system-profile
spec:
  dockerd:
    rootless: true

```

Use of `docker` can be made even more secure by avoiding the use of a privileged container for the `docker` daemon. This requires that you set up a specific configuration for nodes in the Kubernetes cluster. With this configuration, you can disallow the use of a privileged container by setting `dockerd.privileged` to `false`:

```

apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  dockerd:
    rootless: true
    privileged: false

```

For further details about the requirements for running rootless Docker-in-Docker and using a non-privileged container, see the [Docker documentation](#).

## Overriding network packet size

When you enable support for building container images using `docker` for workshops, because of network layering that occurs when doing `docker build` or `docker run`, you must adjust the network packet size (MTU) used for containers run from `dockerd` hosted inside the workshop container.

The default MTU size for networks is 1500, but, when containers are run in Kubernetes, the size available to containers is often reduced. To deal with this possibility, the MTU size used when `dockerd` is run for a workshop is set as 1400 instead of 1500.

You might need to override this value to an even lower value if you experience problems building or running images with `docker` support. These problems could include errors or timeouts in pulling images or when pulling software packages such as PyPi, npm, and so on.

To lower the value, set the `dockerd.mtu` property:

```

apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  dockerd:
    mtu: 1400

```

To discover the maximum viable size, access the `docker` container run with a workshop and run `ifconfig eth0`. This yields something similar to:

```

eth0      Link encap:Ethernet  HWaddr 02:42:AC:11:00:07
          inet addr:172.17.0.7  Bcast:172.17.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1350  Metric:1
          RX packets:270018  errors:0  dropped:0  overruns:0  frame:0
          TX packets:283882  errors:0  dropped:0  overruns:0  carrier:0
          collisions:0 txqueuelen:0
          RX bytes:86363656 (82.3 MiB)  TX bytes:65183730 (62.1 MiB)

```

If the `MTU` size is less than 1400, use the value given, or a smaller value, for the `dockerd.mtu` setting.

## Image registry pull through cache

When running or building container images with `docker`, if the container image is hosted on Docker Hub, it is pulled down directly from the Docker Hub for each separate workshop session of that workshop.

Because the image is pulled from Docker Hub, this can be slow for all users, especially for large images. With Docker Hub introducing limits on how many images can be pulled anonymously from an IP address within a set period, this also can result in the cap on image pulls being reached. This prevents the workshop from being used until the period expires.

Docker Hub has a higher limit when pulling images as an authenticated user, but with the limit applied to the user rather than by IP address. For authenticated users with a paid plan on Docker Hub, there is no limit.

To attempt to avoid the impact of the limit, the first thing you can do is enable an image registry mirror with image pull-through. This is enabled globally and results in an instance of an image registry mirror being created in the workshop environment of workshops that enable `docker` support. This mirror is used for all workshops sessions created against that workshop environment. When the first user attempts to pull an image, it is pulled down from Docker Hub and cached in the mirror. Subsequent users are served up from the image registry mirror, avoiding the need to pull the image from Docker Hub again. Subsequent users also see a speed up in pulling the image, because the mirror is deployed to the same cluster.

To enable the use of an image registry mirror against Docker Hub, use:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  dockerd:
    mirror:
      remote: https://registry-1.docker.io
```

For authenticated access to Docker Hub, create an access token under your Docker Hub account. Then set the `username` and `password` using the access token as the `password`. Do not use the password for the account itself. Using an access token makes it easier to revoke the token if necessary.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  dockerd:
    mirror:
      remote: https://registry-1.docker.io
      username: username
      password: access-token
```

An access token provides write access to Docker Hub. It is therefore also recommended you use a separate robot account in Docker Hub that is not used to host images and doesn't have write access to any other organizations. In other words, use it purely for reading images from Docker Hub.

If this is a free account, the higher limit on image pulls then applies. If the account is paid, there might be higher limits or no limit 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:
      "{YOUR-REGISTRY-URL}/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
```

## Configure the WorkshopSession resource

This topic describes how you configure the `WorkshopSession` custom resource, which defines a Learning Center workshop session.

### Specifying the session identity

When running training for multiple people, typically you'll use the `TrainingPortal` custom resource to set up a training environment. Alternatively, you can set up a workshop environment by using the `WorkshopEnvironment` custom resource, and then create requests for workshop instances by using the `WorkshopRequest` custom resource. If you're creating requests for workshop instances, and you need more control over how the workshop instances are set up, you can use `WorkshopSession` custom resource instead of `WorkshopRequest`.

To specify the workshop environment the workshop instance is created against, set the `environment.name` field of the specification for the workshop session. You must also specify the session ID for the workshop instance. For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopSession
metadata:
  name: lab-markdown-sample-user1
spec:
  environment:
    name: lab-markdown-sample
  session:
    id: user1
```

The `name` of the workshop specified in the `metadata` of the training environment must be globally unique for the workshop instance you're creating. You must create a separate `WorkshopSession` custom resource for each workshop instance.

The session ID must be unique within the workshop environment that you're creating the workshop instance against.

## Specifying the login credentials

You can control access to each workshop instance using login credentials. This ensures one workshop attendee cannot interfere with another.

To set login credentials for a workshop instance, set the `session.username` and `session.password` fields. For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopSession
metadata:
  name: lab-markdown-sample
spec:
  environment:
    name: lab-markdown-sample-user1
  session:
    username: learningcenter
    password: lab-markdown-sample
```

If you do not specify login credentials, the workshop instance has no access controls and anyone can access it.

## Specifying the ingress domain

To access the workshop instance by using a public URL, you must specify an ingress domain. If an ingress domain isn't specified, use the default ingress domain that the Learning Center operator was configured with.

When setting a custom domain, configure DNS with a wildcard domain to forward all requests for sub-domains of the custom domain to the ingress router of the Kubernetes cluster.

To provide the ingress domain, you can set the `session.ingress.domain` field. For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopSession
metadata:
  name: lab-markdown-sample
spec:
  environment:
    name: lab-markdown-sample-user1
  session:
    ingress:
      domain: training.learningcenter.tanzu.vmware.com
```

You can create a full host name for the session by prefixing the ingress domain with a host name constructed from the name of the workshop environment and the session ID.

If overriding the domain, by default, the workshop session is exposed by using a HTTP connection. If you require a secure HTTPS connection, you must have access to a wildcard SSL certificate for the domain.

You must create a secret of type `tls` for the certificate in the `learningcenter` namespace or in the namespace where the Learning Center operator is deployed. You must then set the name of that secret in the `session.ingress.secret` field. For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopSession
metadata:
  name: lab-markdown-sample
spec:
  environment:
    name: lab-markdown-sample-user1
```

```

session:
  ingress:
    domain: training.learningcenter.tanzu.vmware.com
    secret: training.learningcenter.tanzu.vmware.com-tls

```

You can terminate HTTPS connections by using an external load balancer rather than by specifying a secret for ingresses managed by the Kubernetes ingress controller. When routing traffic into the Kubernetes cluster as HTTP connections, you can override the ingress protocol without specifying an ingress secret by setting the `session.ingress.protocol` field.

For example:

```

apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopSession
metadata:
  name: lab-markdown-sample
spec:
  environment:
    name: lab-markdown-sample-user1
  session:
    ingress:
      domain: training.learningcenter.tanzu.vmware.com
      protocol: https

```

To override or set the ingress class, add `session.ingress.class`. This dictates which ingress router is used when more than one option is available.

For example:

```

apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopSession
metadata:
  name: lab-markdown-sample
spec:
  environment:
    name: lab-markdown-sample-user1
  session:
    ingress:
      domain: training.learningcenter.tanzu.vmware.com
      secret: training.learningcenter.tanzu.vmware.com-tls
      class: nginx

```

## Setting the environment variables

To set the environment variables for the workshop instance, provide the environment variables in the `session.env` field.

```

apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopSession
metadata:
  name: lab-markdown-sample
spec:
  environment:
    name: lab-markdown-sample
  session:
    id: user1
    env:
      - name: REPOSITORY-URL
        value: YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE

```

Where `YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE` is the Git repository URL for `lab-markdown-sample`. For example, `{YOUR-GIT-REPO-URL}/lab-markdown-sample`.

Values of fields in the list of resource objects can reference a number of predefined parameters. Available parameters are:

- `session_id` is a unique ID for the workshop instance within the workshop environment.
- `session_namespace` is the namespace created for and bound to the workshop instance. This is the namespace unique to the session and where a workshop can create their own resources.
- `environment_name` is the name of the workshop environment. For now this is the same as the name of the namespace for the workshop environment. Don't rely on them being the same, and use the most appropriate to cope with any future change.
- `workshop_namespace` is the namespace for the workshop environment. This is the namespace where all deployments of the workshop instances are created, and where the service account that the workshop instance runs as exists.
- `service_account` is the name of the service account the workshop instance runs as, and which has access to the namespace created for that workshop instance.
- `ingress_domain` is the host domain under which host names can be created when creating ingress routes.
- `ingress_protocol` is the protocol (http/https) used for ingress routes created for workshops.

The syntax for referencing one of the parameters is `$(parameter_name)`.

If the workshop environment had specified a set of extra environment variables to be set for workshop instances, it is up to you to incorporate those in the set of environment variables you list under `session.env`. That is, anything listed in `session.env` of the `WorkshopEnvironment` custom resource of the workshop environment is ignored.

## Enable anonymous access to a Learning Center training portal

This topic describes how you enable anonymous access to a Learning Center training portal. The REST API with client authentication provides a means to have the portal create and manage workshop sessions on your behalf but allow a separate system handle user authentication.

If you do not need to authenticate users but still want to provide your own front end from which users select a workshop, such as when integrating workshops into an existing web property, you can enable anonymous mode and redirect users to a URL for workshop session creation.



### Note

Anonymous mode is only recommended for temporary deployments and not for a permanent web site providing access to workshops.

## Enabling anonymous access

Set the registration type to `anonymous` to enable full anonymous access to the training portal:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    registration:
```

```

type: anonymous
workshops:
...

```



### Note

Users can still visit the training portal directly and view the catalog of available workshops, so instead of linking to the main page of the training portal, link from your custom index page to the individual links for creating each workshop.

## Triggering workshop creation

Direct users' browsers to a URL that is specific to a workshop to trigger creation and allocation of the workshop.

The URL format looks like this:

```

TRAINING-PORTAL-URL/workshops/environment/NAME/create/?index_url=INDEX

```

Where:

- **NAME** is the name of the workshop environment corresponding to the workshop that you creates.
- **INDEX** is the URL of your custom index page that contains the workshops.

The user is redirected back to this index page when:

- a user completes the workshop
- an error occurs

When a user is redirected back to the index page, a query string parameter is supplied to display a banner or other indication about why the user was returned.

The name of the query string parameter is `notification` and the possible values are:

- `session-deleted` - Used when the workshop session is completed or restarted.
- `workshop-invalid` - Used when the name of the workshop environment created is invalid.
- `session-unavailable` - Used when capacity is reached and a workshop session cannot be created.
- `session-invalid` - Used when an attempt is made to access a session that doesn't exist. This can occur when the workshop dashboard is refreshed after the workshop session is expired and deleted.

## Enable anonymous access to a Learning Center training portal

This topic describes how you enable anonymous access to a Learning Center training portal. The REST API with client authentication provides a means to have the portal create and manage workshop sessions on your behalf but allow a separate system handle user authentication.

If you do not need to authenticate users but still want to provide your own front end from which users select a workshop, such as when integrating workshops into an existing web property, you can enable anonymous mode and redirect users to a URL for workshop session creation.



### Note

Anonymous mode is only recommended for temporary deployments and not for a permanent web site providing access to workshops.

## Enabling anonymous access

Set the registration type to `anonymous` to enable full anonymous access to the training portal:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    registration:
      type: anonymous
  workshops:
    ...
```



### Note

Users can still visit the training portal directly and view the catalog of available workshops, so instead of linking to the main page of the training portal, link from your custom index page to the individual links for creating each workshop.

## Triggering workshop creation

Direct users' browsers to a URL that is specific to a workshop to trigger creation and allocation of the workshop.

The URL format looks like this:

```
TRAINING-PORTAL-URL/workshops/environment/NAME/create/?index_url=INDEX
```

Where:

- `NAME` is the name of the workshop environment corresponding to the workshop that you creates.
- `INDEX` is the URL of your custom index page that contains the workshops.

The user is redirected back to this index page when:

- a user completes the workshop
- an error occurs

When a user is redirected back to the index page, a query string parameter is supplied to display a banner or other indication about why the user was returned.

The name of the query string parameter is `notification` and the possible values are:

- `session-deleted` - Used when the workshop session is completed or restarted.
- `workshop-invalid` - Used when the name of the workshop environment created is invalid.
- `session-unavailable` - Used when capacity is reached and a workshop session cannot be created.
- `session-invalid` - Used when an attempt is made to access a session that doesn't exist. This can occur when the workshop dashboard is refreshed after the workshop session is



expired and deleted.

## Use the Learning Center workshop catalog

A single training portal can host one or more workshops. This topic describes how you use the workshop catalog to list the available workshops and get information about them using the REST API.

### Listing available workshops

The URL sub path for accessing the list of available workshop environments is `/workshops/catalog/environments/`. When making the request, you must supply the access token in the HTTP `Authorization` header with type set as `Bearer`:

```
curl -v -H "Authorization: Bearer <access-token>" \
<training-portal-url>/workshops/catalog/environments/
```

The JSON response looks like this:

```
{
  "portal": {
    "name": "learningcenter-tutorials",
    "uid": "9b82a7b1-97db-4333-962c-97be6b5d7ee0",
    "generation": 451,
    "url": "<training_portal_url>",
    "sessions": {
      "maximum": 10,
      "registered": 0,
      "anonymous": 0,
      "allocated": 0
    }
  },
  "environments": [
    {
      "name": "learningcenter-tutorials-w01",
      "state": "RUNNING",
      "workshop": {
        "name": "lab-et-self-guided-tour",
        "id": "15e5f1a569496f335049bb00c370ee20",
        "title": "Workshop Building Tutorial",
        "description": "A guided tour of how to build a workshop for your team's learning 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.

## Use session management for your Learning Center workshops

This topic describes how you use the REST API endpoints for session management, which allows you to request a workshop session to be allocated.

## Deactivating portal user registration

When you use the REST API to trigger creation of workshop sessions, VMware recommends that you deactivate user registration through the training portal web interface. This means that only the admin user is able to directly access the web interface for the training portal.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: learningcenter-tutorials
spec:
  portal:
    registration:
      type: one-step
      enabled: false
  workshops:
  ...
```

## Requesting a workshop session

The form of the URL sub path for requesting the allocation of a workshop environment by using the REST API is `/workshops/environment/<name>/request/`. The name segment must be replaced with the name of the workshop environment. When making the request, the access token must be supplied in the HTTP `Authorization` header with type set as `Bearer`:

```
curl -v -H "Authorization: Bearer <access-token>" \
<training-portal-url>/workshops/environment/<name>/request/?index_url=https://hub.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=6UIW4D8Bhf0egVmsEKYlaOcTywrpQJGi&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 learning 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.

## Use client authentication for Learning Center

This topic describes how you can use the portal REST API to integrate access to workshops into an existing website or to create a custom web interface for accessing workshops hosted across one or more training portals.

The training portal web interface is a quick way of providing access to a set of workshops when running a supervised training workshop. The REST API gives you access to the list of workshops hosted by a training portal instance and allow you to request and access workshop sessions. This bypasses the training portal's own user registration and log in so you can implement whatever access controls you need. This can include anonymous access or access integrated into an organization's single sign-on system.

## Querying the credentials

To provide access to the REST API, a robot account is automatically provisioned. Obtain the login credentials and details of the OAuth client endpoint used for authentication by querying the resource definition for the training portal after it is created and the deployment completed. If using `kubectl describe`, use:

```

kubectl describe trainingportal.learningcenter.tanzu.vmware.com/<training-portal-name>

```

The status section of the output reads:

```
Status:
  learningcenter:
    Clients:
      Robot:
        Id:      ACZpcaLIT3qr725YWmXu8et9REl4HBg1
        Secret:  t5IfXbGZQThAKR43apoc9usOFVDv2BLE
    Credentials:
      Admin:
        Password: 0kGmM1Yw46BZT2vCntyrRuFflgQq5ohi
        Username:  learningcenter
      Robot:
        Password: QrnY67ME9yGasNhq20TbgWA4RzipUvo5
        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 topic gives you troubleshooting and recovery steps for Learning Center known issues.

### Training portal stays in pending state

The training portal stays in a “pending” state.

The Training Portal custom resource (CR) has a status property. To see the status, run:

```
kubectl get trainingportals.learningcenter.tanzu.vmware.com
```

#### Explanation

If the status stays in a pending state, the TLS secret `tls` might not be available. Other errors can also cause the status to stay in a pending state, so it is important to check the operator and portal logs to execute the right steps.

#### Solution

1. Access the operator logs by running:

```
kubectl logs deployment/learningcenter-operator -n learningcenter
```

Access the portal logs by running:

```
kubectl logs deployment/learningcenter-portal -n {PORTAL_NAMESPACE}
```

2. Check whether the TLS secret `tls` is available. The TLS secret must be on the Learning Center operator namespace (by default `learningcenter`). If the TLS secret is not on the Learning Center operator namespace, the operator logs contain the following error:

```
ERROR:kopf.objects:Handler 'learningcenter' failed temporarily: TLS secret tls is not available
```

3. Follow the steps in [Enforcing Secure Connections in Learning Center Operator](#) to create the TLS secret.
4. Redeploy the `trainingPortal` resource.

### image-policy-webhook-service not found

You are installing a Tanzu Application Platform profile and you get this error:

```
Internal error occurred: failed calling webhook "image-policy-webhook.signing.run.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.

### Updates to Tanzu Application Platform values file not reflected in Learning Center Training Portal



If you installed Learning Center through Tanzu profiles, then your installation made use of a `tap-values.yml` file where configurations were specified for Learning Center. If you make updates to these configurations using this command:

```
tanzu package installed update tap --package-name tap.tanzu.vmware.com --version {VERSION} -f tap-values.yml -n tap-install
```

then the changes are not reflected in the deployed Learning Center Training Portal resource. Tap package updates currently **DO NOT** update running Learning Center Training Portal resources.

Run one of these commands to validate changes made to parameters provided to the Learning Center Operator. These parameters include `ingressDomain`, `TLS secret`, `ingressClass`, and others.

Command:

```
kubectl describe systemprofile
```

Command:

```
kubectl describe pod -n learningcenter
```

### Explanation

By design, the training portal resources do not react to any changes on the parameters provided when the training portals were created. This prevents any change on the `trainingportal` resource from affecting any online user running a workshop.

### Solution

You must restart the operator resource by first deleting the operator pod:

```
kubectl delete pod -n learningcenter learningcenter-operator-$OPERATOR_POD_NAME
```

Then delete the training portal resource. Redeploy `trainingportal` in a maintenance window where Learning Center is unavailable while the `systemprofile` is updated.

## Increase your cluster's resources

If you don't have enough nodes or enough resources on nodes for deploying the workloads, node pressure might occur. In this case, follow your cloud provider's instructions on how to scale out or scale up your cluster.

## Kubernetes Api Timeout error

The following operator error log means there is a connection error with the Kubernetes API server:

```
operator-log: unexpected error occurred. Read timed out.
```

This error has been found when running Learning Center with the Azure AkS cloud provider.

### Solution

To fix this error:

1. Delete the operator pod on the learningcenter namespace.
2. Delete the training portal once the operator is running again by using:

```
kubectl delete trainingportals $PORTAL_NAME
```

1. Redeploy the `trainingPortal` resource.

## No URL returned to your trainingportal

After deploying the Learning Center Operator and Trainingportal resources, the following command can yield the resource with no URL, even though your resources deployed correctly and are running:

```
kubectl get trainingportals
```

You also already specified `learningcenter.mydomain.com` in your tap values YAML file if installed through Tanzu Application Platform. See [specifying ingress domain](#)

### Solution

Learning center requires that you use a wildcard domain (Wildcard DNS entry) to access your training portal in the browser. This configuration must be done in your DNS provider with a rule that points your wildcard domain to your IP/Load balancer.

For example, if using the default workshop on an Elastic Kubernetes Service (EKS) cluster, your URL could look something like:

```
learning-center-guided.learningcenter.yourdomain.com
```

Where `learningcenter.yourdomain.com` needs a DNS configuration made to point to your default ingress controller.

In this case, the wildcard domain configuration needed is `*.learningcenter.yourdomain.com`.

After this configuration is made, you might need to restart your operator resource by deleting and redeploying to see the URL update.

## Overview of Supply Chain Choreographer for Tanzu

This topic introduces you to Supply Chain Choreographer.

### Overview

Supply Chain Choreographer is based on open source [Cartographer](#). It allows App Operators to create pre-approved paths to production by integrating Kubernetes resources with the elements of their existing toolchains, for example, Jenkins.

Each pre-approved supply chain creates a path to production. Orchestrating supply chain resources including, test, build, scan, and deploy allows developers to focus on delivering value to their users and provides App Operators the assurance that all code in production has passed through all the steps of an approved workflow.

### Out of the Box Supply Chains

Out of the box supply chains are provided with Tanzu Application Platform.

The following three supply chains are included:

- [Out of the Box Supply Chain Basic](#)
- [Out of the Box Supply Chain with Testing](#)
- [Out of the Box Supply Chain with Testing and Scanning](#)

As auxiliary components, Tanzu Application Platform also includes:

- [Out of the Box Templates](#), for providing templates used by the supply chains to perform common tasks such as fetching source code, running tests, and building container images.

- [Out of the Box Delivery Basic](#), for delivering to a Kubernetes cluster the configuration built throughout a supply chain

Both Templates and Delivery Basic are requirements for the Supply Chains.

Supply Chain Choreographer supports the following pipeline types:

- [Tekton pipelines](#)
- [Jenkins pipelines](#)

## Overview of Supply Chain Choreographer for Tanzu

This topic introduces you to Supply Chain Choreographer.

### Overview

Supply Chain Choreographer is based on open source [Cartographer](#). It allows App Operators to create pre-approved paths to production by integrating Kubernetes resources with the elements of their existing toolchains, for example, Jenkins.

Each pre-approved supply chain creates a path to production. Orchestrating supply chain resources including, test, build, scan, and deploy allows developers to focus on delivering value to their users and provides App Operators the assurance that all code in production has passed through all the steps of an approved workflow.

### Out of the Box Supply Chains

Out of the box supply chains are provided with Tanzu Application Platform.

The following three supply chains are included:

- [Out of the Box Supply Chain Basic](#)
- [Out of the Box Supply Chain with Testing](#)
- [Out of the Box Supply Chain with Testing and Scanning](#)

As auxiliary components, Tanzu Application Platform also includes:

- [Out of the Box Templates](#), for providing templates used by the supply chains to perform common tasks such as fetching source code, running tests, and building container images.
- [Out of the Box Delivery Basic](#), for delivering to a Kubernetes cluster the configuration built throughout a supply chain

Both Templates and Delivery Basic are requirements for the Supply Chains.

Supply Chain Choreographer supports the following pipeline types:

- [Tekton pipelines](#)
- [Jenkins pipelines](#)

## Install Supply Chain Choreographer

This document describes how to install Supply Chain Choreographer from the Tanzu Application Platform package repository.



#### Note

Follow the steps in this topic if you do not want to use a profile to install Supply Chain Choreographer. For more information about profiles, see [Components and](#)

### installation profiles..

The Supply Chain Choreographer is now bundled with the Cartographer Conventions. For information on configuring and using Cartographer Conventions, see [Creating conventions](#).

Supply Chain Choreographer provides the custom resource definitions the supply chain uses. Each pre-approved supply chain creates a clear road to production and orchestrates supply chain resources. You can test, build, scan, and deploy. Developers can focus on delivering value to users. Application operators can rest assured that all code in production has passed through an approved workflow.

For example, Supply Chain Choreographer passes the results of fetching source code to the component that builds a container image of it, and then passes the container image to a component that deploys the image.

## Prerequisites

Before installing Supply Chain Choreographer:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see [Prerequisites](#).

## Install

To install Supply Chain Choreographer:

1. Get the values schema to see what properties can be configured during installation. Run:

```
tanzu package available get cartographer.tanzu.vmware.com/0.4.0 --values-schema
--namespace tap-install
```

| KEY                 | DEFAULT | TYPE   | DESCRIPTION                                                                    |
|---------------------|---------|--------|--------------------------------------------------------------------------------|
| aws_iam_role_arn    |         | string | Optional: Arn role that has access to pull images from ECR container registry  |
| ca_cert_data        |         | string | Optional: PEM Encoded certificate data for image registries with private CA.   |
| excluded_components | []      | array  | Optional: List of components to exclude from installation (e.g. [conventions]) |

2. Install v0.4.0 of the `cartographer.tanzu.vmware.com` package, naming the installation `cartographer`. Run:

```
tanzu package install cartographer \
  --namespace tap-install \
  --package-name cartographer.tanzu.vmware.com \
  --version 0.4.0
```

Example output:

```
| Installing package 'cartographer.tanzu.vmware.com'
| Getting namespace 'tap-install'
| Getting package metadata for 'cartographer.tanzu.vmware.com'
| Creating service account 'cartographer-tap-install-sa'
| Creating cluster admin role 'cartographer-tap-install-cluster-role'
| Creating cluster role binding 'cartographer-tap-install-cluster-rolebinding'
- Creating package resource
\ Package install status: Reconciling
```

```
Added installed package 'cartographer' in namespace 'tap-install'
```

## Out of the Box 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, Maven repository, or local directory for changes
  2. Building a container image out of the source code with Buildpacks
  3. Applying operator-defined conventions to the container definition
  4. Creating a deliverable object for deploying the application to a cluster
- Using a prebuilt application image:
  1. Applying operator-defined conventions to the container definition
  2. Creating a deliverable object for deploying the application to a cluster

## Prerequisites

To use this package, you must:

- Install [Out of the Box Templates](#).
- Configure the Developer namespace with auxiliary objects that are used by the supply chain as described in the following section.
- (Optionally) install [Out of the Box Delivery Basic](#), if you are willing to deploy the application to the same cluster as the workload and supply chains.

## Developer Namespace

The supply chains provide definitions of many of the objects that they create to transform the source code to a container image and make it available as an application in a cluster.

The developer must provide or configure particular objects in the developer namespace so that the supply chain can provide credentials and use permissions granted to a specific development team.

The objects that the developer must provide or configure include:

- **registries secrets**: Kubernetes secrets of type `kubernetes.io/dockerconfigjson` that contain credentials for pushing and pulling the container images built by the supply chain and the installation of Tanzu Application Platform.
- **service account**: The identity to be used for any interaction with the Kubernetes API made by the supply chain.
- **rolebinding**: Grant to the identity the necessary roles for creating the resources prescribed by the supply chain.

### Registries Secrets

Regardless of the supply chain that a workload goes through, there must be Kubernetes secrets in the developer namespace containing credentials for both pushing and pulling the container image

that gets built by the supply chains when source code is provided. The developer namespace must also contain registry credentials for Kubernetes to run pods using images from the installation of Tanzu Application Platform.

1. Add read/write registry credentials for pushing and pulling application images:

```
tanzu secret registry add registry-credentials \
  --server REGISTRY-SERVER \
  --username REGISTRY-USERNAME \
  --password REGISTRY-PASSWORD \
  --namespace YOUR-NAMESPACE
```

Where:

- o `YOUR-NAMESPACE` is the name you want to use for the developer namespace. For example, use `default` for the default namespace.
  - o `REGISTRY-SERVER` is the URL of the registry. For Docker Hub, this must be `https://index.docker.io/v1/`. Specifically, it must have the leading `https://`, the `v1` path, and the trailing `/`. For Google Container Registry (GCR), this is `gcr.io`. Based on the information used in [Installing the Tanzu Application Platform package and profiles](#), you can use the same registry server as in `ootb_supply_chain_basic - registry - server`.
2. Add a placeholder secret for gathering the credentials used for pulling container images from the installation of Tanzu Application Platform:

```
cat <<EOF | kubectl -n YOUR-NAMESPACE apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: tap-registry
  annotations:
    secretgen.carvel.dev/image-pull-secret: ""
type: kubernetes.io/dockerconfigjson
data:
  .dockerconfigjson: e30K
EOF
```

With the two secrets created:

- `tap-registry` is a placeholder secret filled indirectly by `secretgen-controller` Tanzu Application Platform credentials set up during the installation of Tanzu Application Platform.
- `registry-credentials` is a secret providing credentials for the registry where application container images are pushed to.

The following section discusses setting up the identity required for the workload.

## ServiceAccount

In a Kubernetes cluster, a ServiceAccount provides a way of representing an actor within the Kubernetes role-based access control (RBAC) system. In the case of a developer namespace, this represents a developer or development team.

You can directly attach secrets to the ServiceAccount through both the `secrets` and `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 v1.2 ships with two ClusterRoles that describe all of the necessary permissions to grant to the service account:

- `workload` clusterrole, providing the necessary roles for the supply chains to be able to manage the resources prescribed by them.
- `deliverable` clusterrole, providing the roles for deliveries to deploy to the cluster the application Kubernetes objects produced by the supply chain.

To provide those permissions to the identity we created for this workload, bind the `workload` ClusterRole to the ServiceAccount we created above:

```

apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: default-permit-workload
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: workload
subjects:
  - kind: ServiceAccount
    name: default

```

If this is just a Build cluster, and you do not intend to run the application in it, this single RoleBinding is all that's necessary.

If you intend to also deploy the application that's been built, bind to the same ServiceAccount the `deliverable` ClusterRole too:

```

apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: default-permit-deliverable
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: deliverable
subjects:
  - kind: ServiceAccount
    name: default

```

For more information about authentication and authorization in Tanzu Application Platform, see [Overview of Default roles for Tanzu Application Platform](#).

## 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
```

Where:

- `FLAGS` are the one or more flags you want to include.
- `WORKLOAD-NAME` is the name of the workload you want to target.

Depending on what you are aiming to achieve, you can set different flags. To know more about those (including details about different features of the supply chains), see the following sections:

- [Building from source](#), for more information about different ways of creating a workload where the application is built from source code.
- [Using an existing image](#), for more information about how to leverage prebuilt images in the supply chains.
- [GitOps vs RegistryOps](#), for a description of the different ways of propagating the deployment configuration through external systems (Git repositories and image registries).

## Out of the Box Supply Chain Basic

This package contains Cartographer supply chains that tie together a series of Kubernetes resources that drive a developer-provided workload from source code to a Kubernetes configuration ready to be deployed to a cluster. It contains the most basic supply chains that focus on providing a quick path to deployment making no use of testing or scanning resources.

The supply chains included in this package perform the following:

- Building from source code:
  1. Watching a Git repository, Maven repository, or local directory for changes
  2. Building a container image out of the source code with Buildpacks
  3. Applying operator-defined conventions to the container definition
  4. Creating a deliverable object for deploying the application to a cluster
- Using a prebuilt application image:
  1. Applying operator-defined conventions to the container definition
  2. Creating a deliverable object for deploying the application to a cluster

## Prerequisites

To use this package, you must:

- Install [Out of the Box Templates](#).
- Configure the Developer namespace with auxiliary objects that are used by the supply chain as described in the following section.



- (Optionally) install [Out of the Box Delivery Basic](#), if you are willing to deploy the application to the same cluster as the workload and supply chains.

## Developer Namespace

The supply chains provide definitions of many of the objects that they create to transform the source code to a container image and make it available as an application in a cluster.

The developer must provide or configure particular objects in the developer namespace so that the supply chain can provide credentials and use permissions granted to a specific development team.

The objects that the developer must provide or configure include:

- **registries secrets:** Kubernetes secrets of type `kubernetes.io/dockerconfigjson` that contain credentials for pushing and pulling the container images built by the supply chain and the installation of Tanzu Application Platform.
- **service account:** The identity to be used for any interaction with the Kubernetes API made by the supply chain.
- **rolebinding:** Grant to the identity the necessary roles for creating the resources prescribed by the supply chain.

### Registries Secrets

Regardless of the supply chain that a workload goes through, there must be Kubernetes secrets in the developer namespace containing credentials for both pushing and pulling the container image that gets built by the supply chains when source code is provided. The developer namespace must also contain registry credentials for Kubernetes to run pods using images from the installation of Tanzu Application Platform.

1. Add read/write registry credentials for pushing and pulling application images:

```
tanzu secret registry add registry-credentials \
  --server REGISTRY-SERVER \
  --username REGISTRY-USERNAME \
  --password REGISTRY-PASSWORD \
  --namespace YOUR-NAMESPACE
```

Where:

- `YOUR-NAMESPACE` is the name you want to use for the developer namespace. For example, use `default` for the default namespace.
  - `REGISTRY-SERVER` is the URL of the registry. For Docker Hub, this must be `https://index.docker.io/v1/`. Specifically, it must have the leading `https://`, the `v1` path, and the trailing `/`. For Google Container Registry (GCR), this is `gcr.io`. Based on the information used in [Installing the Tanzu Application Platform package and profiles](#), you can use the same registry server as in `ootb_supply_chain_basic - registry - server`.
2. Add a placeholder secret for gathering the credentials used for pulling container images from the installation of Tanzu Application Platform:

```
cat <<EOF | kubectl -n YOUR-NAMESPACE apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: tap-registry
  annotations:
    secretgen.carvel.dev/image-pull-secret: ""
type: kubernetes.io/dockerconfigjson
```

```
data:
  .dockerconfigjson: e30K
EOF
```

With the two secrets created:

- `tap-registry` is a placeholder secret filled indirectly by `secretgen-controller` Tanzu Application Platform credentials set up during the installation of Tanzu Application Platform.
- `registry-credentials` is a secret providing credentials for the registry where application container images are pushed to.

The following section discusses setting up the identity required for the workload.

## ServiceAccount

In a Kubernetes cluster, a ServiceAccount provides a way of representing an actor within the Kubernetes role-based access control (RBAC) system. In the case of a developer namespace, this represents a developer or development team.

You can directly attach secrets to the ServiceAccount through both the `secrets` and `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 v1.2 ships with two ClusterRoles that describe all of the necessary permissions to grant to the service account:

- `workload` clusterrole, providing the necessary roles for the supply chains to be able to manage the resources prescribed by them.
- `deliverable` clusterrole, providing the roles for deliveries to deploy to the cluster the application Kubernetes objects produced by the supply chain.

To provide those permissions to the identity we created for this workload, bind the `workload` ClusterRole to the ServiceAccount we created above:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
```

```

metadata:
  name: default-permit-workload
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: workload
subjects:
- kind: ServiceAccount
  name: default

```

If this is just a Build cluster, and you do not intend to run the application in it, this single RoleBinding is all that's necessary.

If you intend to also deploy the application that's been built, bind to the same ServiceAccount the `deliverable` ClusterRole too:

```

apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: default-permit-deliverable
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: deliverable
subjects:
- kind: ServiceAccount
  name: default

```

For more information about authentication and authorization in Tanzu Application Platform, see [Overview of Default roles for Tanzu Application Platform](#).

## 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
```

Where:

- `FLAGS` are the one or more flags you want to include.
- `WORKLOAD-NAME` is the name of the workload you want to target.

Depending on what you are aiming to achieve, you can set different flags. To know more about those (including details about different features of the supply chains), see the following sections:

- [Building from source](#), for more information about different ways of creating a workload where the application is built from source code.
- [Using an existing image](#), for more information about how to 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**

Follow the steps in this topic if you do not want to use a profile to install Out of the Box Supply Chain Basic. For more information about profiles, see [Components and installation profiles](#).

The Out of the Box Supply Chain Basic package provides the most basic ClusterSupplyChain that brings an application from source code to a deployed instance of it running in a Kubernetes environment.

## Prerequisites

Fulfill the following prerequisites:

- Fulfill the [prerequisites](#) for installing Tanzu Application Platform.
- [Install Supply Chain Choreographer](#).

## Install

To install Out of the Box Supply Chain Basic:

1. Familiarize yourself with the set of values of the package that can be configured by running:

```
tanzu package available get ootb-supply-chain-basic.tanzu.vmware.com/0.7.0 \
  --values-schema \
  -n tap-install
```

For example:

| KEY                     | DESCRIPTION                                                                                                                                                                               |
|-------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| registry.repository     | Name of the repository in the image 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.server_address   | Default server address to be used for forming Git URLs for pushing Kubernetes configuration produced by the supply chain. This must include the scheme/protocol (e.g. https:// or ssh://) |
| gitops.repository_owner | Default project or user of the repository. Used to create URLs for pushing Kubernetes configuration produced by the supply chain.                                                         |
| gitops.repository_name  | Default repository name used for forming Git URLs for pushing Kubernetes configuration produced by the supply chain.                                                                      |

|                                                     |                                                                                                                                                                                                       |
|-----------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>gitops.username</code>                        | 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 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.                                                                               |
| <code>gitops.email</code>                           | Default user email to be used for the commits produced by the supply chain.                                                                                                                           |
| <code>gitops.ssh_secret</code>                      | 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.                                  |
| <code>gitops.commit_strategy</code>                 | Specification of how commits are made to the branch; directly or through a pull request.                                                                                                              |
| <code>gitops.repository_prefix</code>               | DEPRECATED: Use <code>server_address</code> and <code>repository_owner</code> instead.                                                                                                                |
| <code>gitops.repository_owner</code>                | Default prefix to be used for forming Git SSH URLs for pushing Kubernetes configuration produced by the supply chain.                                                                                 |
| <code>gitops.pull_request.server_kind</code>        | The git source control platform used                                                                                                                                                                  |
| <code>gitops.pull_request.commit_branch</code>      | The branch to which commits will be made, before opening a pull request                                                                                                                               |
| <code>gitops.pull_request.commit_branch</code>      | If the string "" is specified, an essentially random string will be used for the branch name, in order to prevent collisions.                                                                         |
| <code>gitops.pull_request.pull_request_title</code> | The title for the pull request                                                                                                                                                                        |
| <code>gitops.pull_request.pull_request_body</code>  | Any further information to add to the pull request                                                                                                                                                    |
| <code>cluster_builder</code>                        | Name of the Tanzu Build Service 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 Image objects as well as deploying the application. |
| <code>maven.repository.url</code>                   | The URL of the Maven repository to be used when pulling Maven artifacts. HTTP is not supported. e.g.: "https://repo.maven.apache.org/maven"                                                           |

|                              |                                                                    |
|------------------------------|--------------------------------------------------------------------|
| maven.repository.secret_name | The name of the Secret resource that contains the credentials used |
|                              | to access the Maven repository.                                    |

2. Create a file named `ootb-supply-chain-basic-values.yaml` that specifies the corresponding values to the properties you want to change. For example:

```
registry:
  server: REGISTRY-SERVER
  repository: REGISTRY-REPOSITORY

gitops:
  server_address: https://github.com/
  repository_owner: vmware-tanzu
  branch: main
  username: supplychain
  email: supplychain
  commit_message: supplychain@cluster.local
  ssh_secret: git-ssh
  commit_strategy: direct

maven:
  repository:
    url: https://my-maven-repository/releases
    secret_name: my-maven-repository-credentials

cluster_builder: default
service_account: default
```

3. With the configuration ready, install the package by running:

```
tanzu package install ootb-supply-chain-basic \
  --package-name ootb-supply-chain-basic.tanzu.vmware.com \
  --version 0.7.0 \
  --namespace tap-install \
  --values-file ootb-supply-chain-basic-values.yaml
```

Example output:

```
\ Installing package 'ootb-supply-chain-basic.tanzu.vmware.com'
| Getting package metadata for 'ootb-supply-chain-basic.tanzu.vmware.com'
| Creating service account 'ootb-supply-chain-basic-tap-install-sa'
| Creating cluster admin role 'ootb-supply-chain-basic-tap-install-cluster-role'
| Creating cluster role binding 'ootb-supply-chain-basic-tap-install-cluster-rolebinding'
| 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

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

By default, the workload is matched to the corresponding pipeline to run using labels. Pipelines must have the label `apps.tanzu.vmware.com/pipeline: test` at a minimum, but you can add additional labels for granularity. This provides a default match in the event that no other labels are provided. The pipeline expects two parameters:

- `source-url`, an HTTP address where a `.tar.gz` file containing all the source code to be tested 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 pipeline resources to include labels that differentiate between the pipelines. For example, differentiate between Java and Go pipelines by adding labels for Java and Go:

```

apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: java-tests
  labels:
    apps.tanzu.vmware.com/pipeline: test
    apps.tanzu.vmware.com/language: java
spec:
  #...
  steps:
    - name: test
      image: gradle
      script: |-
        # ...
        ./mvnw test
---
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: go-tests

```

```

labels:
  apps.tanzu.vmware.com/pipeline: test
  apps.tanzu.vmware.com/language: go
spec:
  #...
  steps:
    - name: test
      image: golang
      script: |-
        # ...
        go test -v ./...

```

To match the correct pipeline, you add a `testing_pipeline_matching_labels` parameter to the workload. For example, if you want to match to the Java pipeline, you have the following `workload.yaml`:

```

apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: sample-java-app
  labels:
    apps.tanzu.vmware.com/has-tests: true
    apps.tanzu.vmware.com/workload-type: web
    app.kubernetes.io/part-of: sample-java-app
spec:
  params:
    - name: testing_pipeline_matching_labels
      value:
        apps.tanzu.vmware.com/pipeline: test
        apps.tanzu.vmware.com/language: java
    ...

```

This matches the workload to the pipeline with the `apps.tanzu.vmware.com/language: java` label.

## Developer Workload

With the Tekton Pipeline object submitted to the same namespace as the one where the Workload will be submitted to, you can submit your Workload.

Regardless of the workflow being targeted (local development or gitops), the Workload configuration details are the same as in Out of the Box Supply Chain Basic, except that you mark the workload with tests enabled by means of the `has-tests` label.

For example:

```

tanzu apps workload create tanzu-java-web-app \
  --git-repo ${GIT_PROJECT_URL} \
  --git-branch ${GIT_BRANCH} \
  --label apps.tanzu.vmware.com/has-tests=true \
  --label app.kubernetes.io/part-of=tanzu-java-web-app \
  --type web

```

```

Create workload:
1 + |---
2 + |apiVersion: carto.run/v1alpha1
3 + |kind: Workload
4 + |metadata:
5 + |  labels:
6 + |    apps.tanzu.vmware.com/workload-type: web
7 + |    apps.tanzu.vmware.com/has-tests: "true"
8 + |    app.kubernetes.io/part-of: tanzu-java-web-app
9 + |  name: tanzu-java-web-app
10 + |  namespace: default

```

```

11 + |spec:
12 + |  source:
13 + |    git:
14 + |      ref:
15 + |        branch: main
16 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
17 + |      subPath: tanzu-java-web-app

```

## Out of the Box Supply Chain with Testing

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

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

By default, the workload is matched to the corresponding pipeline to run using labels. Pipelines must have the label `apps.tanzu.vmware.com/pipeline: test` at a minimum, but you can add additional labels for granularity. This provides a default match in the event that no other labels are provided. The pipeline expects two parameters:

- `source-url`, an HTTP address where a `.tar.gz` file containing all the source code to be tested 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 pipeline resources to include labels that differentiate between the pipelines. For example, differentiate between Java and Go pipelines by adding labels for Java and Go:

```

apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:

```

```

name: java-tests
labels:
  apps.tanzu.vmware.com/pipeline: test
  apps.tanzu.vmware.com/language: java
spec:
  #...
  steps:
    - name: test
      image: gradle
      script: |-
        # ...
        ./mvnw test
---
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: go-tests
  labels:
    apps.tanzu.vmware.com/pipeline: test
    apps.tanzu.vmware.com/language: go
spec:
  #...
  steps:
    - name: test
      image: golang
      script: |-
        # ...
        go test -v ./...

```

To match the correct pipeline, you add a `testing_pipeline_matching_labels` parameter to the workload. For example, if you want to match to the Java pipeline, you have the following `workload.yaml`:

```

apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: sample-java-app
  labels:
    apps.tanzu.vmware.com/has-tests: true
    apps.tanzu.vmware.com/workload-type: web
    app.kubernetes.io/part-of: sample-java-app
spec:
  params:
    - name: testing_pipeline_matching_labels
      value:
        apps.tanzu.vmware.com/pipeline: test
        apps.tanzu.vmware.com/language: java
  ...

```

This matches the workload to the pipeline with the `apps.tanzu.vmware.com/language: java` label.

## Developer Workload

With the Tekton Pipeline object submitted to the same namespace as the one where the Workload will be submitted to, you can submit your Workload.

Regardless of the workflow being targeted (local development or gitops), the Workload configuration details are the same as in Out of the Box Supply Chain Basic, except that you mark the workload with tests enabled by means of the `has-tests` label.

For example:

```
tanzu apps workload create tanzu-java-web-app \
  --git-repo ${GIT_PROJECT_URL} \
  --git-branch ${GIT_BRANCH} \
  --label apps.tanzu.vmware.com/has-tests=true \
  --label app.kubernetes.io/part-of=tanzu-java-web-app \
  --type web
```

```
Create workload:
 1 + |---
 2 + |apiVersion: carto.run/v1alpha1
 3 + |kind: Workload
 4 + |metadata:
 5 + |  labels:
 6 + |    apps.tanzu.vmware.com/workload-type: web
 7 + |    apps.tanzu.vmware.com/has-tests: "true"
 8 + |    app.kubernetes.io/part-of: tanzu-java-web-app
 9 + |  name: tanzu-java-web-app
10 + |  namespace: default
11 + |spec:
12 + |  source:
13 + |    git:
14 + |      ref:
15 + |        branch: main
16 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
17 + |      subPath: tanzu-java-web-app
```

## Install Out of the Box Supply Chain with Testing

This document describes how to install Out of the Box Supply Chain with Testing from the Tanzu Application Platform package repository.



### Note

Follow the steps in this topic if you do not want to use a profile to install Out of the Box Supply Chain with Testing. For more information about profiles, see [Components and installation profiles](#).

The Out of the Box Supply Chain with Testing package provides a ClusterSupplyChain that brings an application from source code to a deployed instance that:

- Runs in a Kubernetes environment.
- Runs developer-provided tests in the form of Tekton/Pipeline objects to validate the source code before building container images.

## Prerequisites

Before installing Out of the Box Supply Chain with Testing:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see [Prerequisites](#).
- Install cartographer. For more information, see [Install Supply Chain Choreographer](#).
- Install [Out of the Box Delivery Basic](#)
- Install [Out of the Box Templates](#)

## Install

Install by following these steps:

1. Ensure you do not have Out of the Box Supply Chain With Testing and Scanning (`ootb-supply-chain-testing-scanning.tanzu.vmware.com`) installed:

1. Run the following command:

```
tanzu package installed list --namespace tap-install
```

2. Verify `ootb-supply-chain-testing-scanning` is in the output:

| NAME                    | PACKAGE-NAME                             |
|-------------------------|------------------------------------------|
| ootb-delivery-basic     | ootb-delivery-basic.tanzu.vmware.com     |
| ootb-supply-chain-basic | ootb-supply-chain-basic.tanzu.vmware.com |
| ootb-templates          | ootb-templates.tanzu.vmware.com          |

3. If you see `ootb-supply-chain-testing-scanning` in the list, uninstall it by running:

```
tanzu package installed delete ootb-supply-chain-testing-scanning --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   | 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.server_address | Default server address to be used for forming Git URLs for pushing                                                              |
|                       | Kubernetes configuration produced by the                                                                                        |



|                                                                                                                                     |                                                                                                                                                                                                                   |
|-------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| supply chain. This must include the scheme/protocol (e.g. http s:// or ssh://)                                                      |                                                                                                                                                                                                                   |
| gitops.repository_owner y. Used to create URLs for pushing supply chain.                                                            | Default project or user of the repository. Kubernetes configuration produced by the supply chain.                                                                                                                 |
| gitops.repository_name Git URLs for pushing Kubernetes in.                                                                          | Default repository name used for forming configuration produced by the supply chain.                                                                                                                              |
| gitops.username commits produced by the supply chain.                                                                               | Default user name to be used for the commits produced by the supply chain.                                                                                                                                        |
| gitops.branch etes configuration files produced by the supply chain.                                                                | Default branch to use for pushing Kubernetes configuration files produced by the supply chain.                                                                                                                    |
| gitops.commit_message publishing Kubernetes configuration files produces by the supply chain to git.                                | Default git commit message to write when publishing Kubernetes configuration files produced by the supply chain to git.                                                                                           |
| gitops.email mmits produced by the supply chain.                                                                                    | Default user email to be used for the commits produced by the supply chain.                                                                                                                                       |
| gitops.ssh_secret H credentials to lookup in the to fetch source code from and push configuration to.                               | 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.commit_strategy the branch; directly or through a pull request.                                                              | Specification of how commits are made to the branch; directly or through a pull request.                                                                                                                          |
| gitops.repository_prefix itory_owner instead. Default prefix to be used for forming Git SSH URLs for pushing Kubernetes in.         | DEPRECATED: Use server_address and repository_owner instead. Default prefix to be used for forming Git SSH URLs for pushing Kubernetes configuration produced by the supply chain.                                |
| gitops.pull_request.server_kind                                                                                                     | The git source control platform used                                                                                                                                                                              |
| gitops.pull_request.commit_branch e, before opening a pull request h If the string "" is specified, d for the branch name, in order | The branch to which commits will be made to the branch specified in .gitops.branch. If the string "" is specified, an essentially random string will be used for the branch name, in order to prevent collisions. |
| gitops.pull_request.pull_request_title                                                                                              | The title for the pull request                                                                                                                                                                                    |
| gitops.pull_request.pull_request_body ull request                                                                                   | Any further information to add to the pull request                                                                                                                                                                |
| cluster_builder chain.                                                                                                              | Name of the Tanzu Build Service ClusterBuilder to use by default on image objects managed by the supply chain.                                                                                                    |
| service_account                                                                                                                     | Name of the service account in the namespace where the                                                                                                                                                            |

```
e Workload
                                is submitted to utilize for providing registry creden
tials to
                                Tanzu Build Service Image objects as well as deployin
g the
                                application.
```

3. Create a file named `ootb-supply-chain-testing-values.yaml` that specifies the corresponding values to the properties you want to change. For example:

```
registry:
  server: REGISTRY-SERVER
  repository: REGISTRY-REPOSITORY

gitops:
  server_address: https://github.com/
  repository_owner: vmware-tanzu
  branch: main
  username: supplychain
  email: supplychain
  commit_message: supplychain@cluster.local
  ssh_secret: git-ssh
  commit_strategy: direct

cluster_builder: default
service_account: default
```



### Important

it's required that the `gitops.repository_prefix` field ends with a `/`.

4. With the configuration ready, install the package by running:

```
tanzu package install ootb-supply-chain-testing \
  --package-name ootb-supply-chain-testing.tanzu.vmware.com \
  --version 0.7.0 \
  --namespace tap-install \
  --values-file ootb-supply-chain-testing-values.yaml
```

Example output:

```
\ Installing package 'ootb-supply-chain-testing.tanzu.vmware.com'
| Getting package metadata for 'ootb-supply-chain-testing.tanzu.vmware.com'
| Creating service account 'ootb-supply-chain-testing-tap-install-sa'
| Creating cluster admin role 'ootb-supply-chain-testing-tap-install-cluster-ro
le'
| Creating cluster role binding 'ootb-supply-chain-testing-tap-install-cluster-
rolebinding'
| Creating secret 'ootb-supply-chain-testing-tap-install-values'
| Creating package resource
- Waiting for 'PackageInstall' reconciliation for 'ootb-supply-chain-testing'
\ 'PackageInstall' resource install status: Reconciling

Added installed package 'ootb-supply-chain-testing' in namespace 'tap-install'
```

## Out of the Box Supply Chain with Testing and Scanning

This package contains Cartographer Supply Chains that tie together a series of Kubernetes resources that drive a developer-provided workload from source code to a Kubernetes

configuration ready to be deployed to a cluster. It contains supply chains that pass the source code through testing and vulnerability scanning, and also the container image.

This package includes all the capabilities of the Out of the Box Supply Chain With Testing, but adds source and image scanning using Grype.

Workloads that use source code or prebuilt images perform the following:

- Building from source code:
  1. Watching a Git Repository or local directory for changes
  2. Running tests from a developer-provided Tekton pipeline
  3. Scanning the source code for known vulnerabilities using Grype
  4. Building a container image out of the source code with Buildpacks
  5. Scanning the image for known vulnerabilities
  6. Applying operator-defined conventions to the container definition
  7. Deploying the application to the same cluster
- Using a prebuilt application image:
  1. Scanning the image for known vulnerabilities
  2. Applying operator-defined conventions to the container definition
  3. Creating a deliverable object for deploying the application to a cluster

## Prerequisites

To use this supply chain, verify that:

- Tanzu Application Platform GUI is configured to [enable CVE scan results](#). This configuration enables the Supply Chain Choreographer Tanzu Application Platform GUI plug-in to retrieve metadata about project packages and their vulnerabilities.
- Out of the Box Templates is installed.
- Out of the Box Supply Chain With Testing is **NOT installed**.
- Out of the Box Supply Chain With Testing and Scanning is **installed**.
- Developer namespace is configured with the objects according to Out of the Box Supply Chain With Testing guidance. This supply chain is in addition to the Supply Chain with testing.
- (Optionally) install [Out of the Box Delivery Basic](#), if you are willing to deploy the application to the same cluster as the workload and supply chains.

Verify that you have the supply chains with scanning, not with testing, installed. Run:

```
tanzu apps cluster-supply-chain list
```

| NAME                    | LABEL_SELECTOR                                                               |
|-------------------------|------------------------------------------------------------------------------|
| source-test-scan-to-url | apps.tanzu.vmware.com/has-tests=true,apps.tanzu.vmware.com/workload-type=web |
| source-to-url           | apps.tanzu.vmware.com/workload-type=web                                      |

If you see `source-test-to-url` in the list, the setup is wrong. You **must not have the `source-test-to-url` installed** at the same time as `source-test-scan-to-url`.

## Developer namespace

This example builds on the previous Out of the Box Supply Chain examples, so only additions are included here.

To ensure that you configured the namespace correctly, it is important that the namespace has the objects that you configured in the other supply chain setups:

- **registries secrets:** Kubernetes secrets of type `kubernetes.io/dockerconfigjson` that contain credentials for pushing and pulling the container images built by the supply chain and the installation of Tanzu Application Platform.
- **service account:** The identity to be used for any interaction with the Kubernetes API made by the supply chain.
- **rolebinding:** Grant to the identity the necessary roles for creating the resources prescribed by the supply chain.

For more information about the preceding objects, see [Out of the Box Supply Chain Basic](#).

- **Tekton pipeline:** A pipeline runs whenever the supply chain hits the stage of testing the source code.

For more information, see [Out of the Box Supply Chain Testing](#).

And the new objects, that you create here:

- **scan policy:** Defines what to do with the results taken from scanning the source code and image produced. For more information, see [ScanPolicy section](#).
- **source scan template:** A template of how 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](#).

The following section includes details about the new objects, compared to Out of the Box Supply Chain With Testing.

## Updates to the developer namespace

For source and image scans, scan templates and scan policies must exist in the same namespace as the workload. These define:

- `ScanTemplate`: how to run a scan, allowing one to change details about the execution of the scan (either for images or source code)
- `ScanPolicy`: how to evaluate whether the artifacts scanned are compliant. For example, allowing one to be either very strict, or restrictive about particular vulnerabilities found.

The names of the objects **must** match the names in the example with default installation configurations. This is overridden either by using the `ootb_supply_chain_testing_scanning` package configuration in the `tap-values.yaml` file or by using workload parameters:

- To override by using the `ootb_supply_chain_testing_scanning` package configuration, make the following modification to your `tap-values.yaml` file and perform a [Tanzu Application Platform update](#).

```
ootb_supply_chain_testing_scanning:
  scanning:
    source:
      policy: SCAN-POLICY
      template: SCAN-TEMPLATE
    image:
      policy: SCAN-POLICY
      template: SCAN-TEMPLATE
```

Where `SCAN-POLICY` and `SCAN-TEMPLATE` are the names of the `ScanPolicy` and `ScanTemplate`.

- To override through workload parameters, use the following commands. For more information, see [Tanzu apps workload apply](#).

```
tanzu apps workload apply WORKLOAD --param "scanning_source_policy=SCAN-POLICY"
-n DEV-NAMESPACE
tanzu apps workload apply WORKLOAD --param "scanning_source_template=SCAN-TEMPL
ATE" -n DEV-NAMESPACE
```

Where:

- `WORKLOAD` is the name of the workload.
- `SCAN-POLICY` and `SCAN-TEMPLATE` are the names of the `ScanPolicy` and `ScanTemplate`.
- `DEV-NAMESPACE` is the developer namespace.

## ScanPolicy

The `ScanPolicy` defines a set of rules to evaluate for a particular scan to consider the artifacts (image or source code) either compliant or not.

When a `ImageScan` or `SourceScan` is created to run a scan, those reference a policy whose name **must** match the following sample `scan-policy`:

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: scan-policy
  labels:
    'app.kubernetes.io/part-of': 'enable-in-gui'
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "UnknownSeve
    rity"
    notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]
    ignoreCves := []

    contains(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
      severities := { e | e := match.ratings.rating.severity } | { e | e := match.rati
      ngs.rating[_].severity }
      some i
      fails := contains(notAllowedSeverities, severities[i])
      not fails
    }

    isSafe(match) {
      ignore := contains(ignoreCves, match.id)
      ignore
    }

    deny[msg] {
      comps := { e | e := input.bom.components.component } | { e | e := input.bom.comp
      onents.component[_] }
      some i
      comp := comps[i]
      vulns := { e | e := comp.vulnerabilities.vulnerability } | { e | e := comp.vulne
```

```

rabilities.vulnerability[_] }
  some j
  vuln := vulns[j]
  ratings := { e | e := vuln.ratings.rating.severity } | { e | e := vuln.ratings.r
ating[_].severity }
  not isSafe(vuln)
  msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
}

```

See [Writing Policy Templates](#).

## ScanTemplate

A ScanTemplate defines the PodTemplateSpec used by a Job to run a particular scan (image or source). When the supply chain initiates an ImageScan or SourceScan, they reference these templates which must live in the same namespace as the workload with the names matching the following:

- source scanning ([blob-source-scan-template](#))
- image scanning ([private-image-scan-template](#))

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 following the Getting Started guide, VMware recommends that you follow what is provided in the installation of `grype.scanning.apps.tanzu.vmware.com`. This is created in the same namespace as configured by using `grype.namespace` in either Tanzu Application Platform profiles or individual component installation as in the earlier example. For more information, see [About Source and Image Scans](#).

## Enable storing scan results

To enable SCST - Scan to store scan results by using SCST - Store, see [Developer namespace setup](#) for exporting the SCST - Store CA certificate and authentication token to the developer namespace.

## Allow multiple Tekton pipelines in a namespace

You can configure your developer namespace to include more than one pipeline using either of the following methods:

- Use a single pipeline running on a container image that includes testing tools and runs a common script to execute tests. This allows you to accommodate multiple workloads based in different languages in the same namespace that use a common make test script. For example:

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: developer-defined-tekton-pipeline
  labels:
    apps.tanzu.vmware.com/pipeline: test
spec:
  #...
  steps:
    - name: test
      image: <image_that_has_JDK_and_Go>
      script: |-
        cd `mktemp -d`
        wget -qO- $(params.source-url) | tar xvz -m
        make test
```

- Update the template to include labels that differentiate the pipelines. Then configure the labels to differentiate between pipelines. For example:

```
selector:
  resource:
    apiVersion: tekton.dev/v1beta1
    kind: Pipeline
    matchingLabels:
      apps.tanzu.vmware.com/pipeline: test
+   apps.tanzu.vmware.com/language: #@ data.values.workload.metadata.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, such as local development or gitops, the workload configuration details are the same as in Out of the Box Supply Chain Basic, except that you mark the workload as having tests enabled.

For example:

```

tanzu apps workload create tanzu-java-web-app \
  --git-repo ${GIT_PROJECT_URL} \
  --git-branch ${GIT_BRANCH} \
  --label apps.tanzu.vmware.com/has-tests=true \
  --label app.kubernetes.io/part-of=tanzu-java-web-app \
  --type web

```

Expect to see output similar to the following:

```

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/my/tanzu-project.git

```

## CVE triage workflow

The Supply Chain halts progression if either a SourceScan

([sourcescans.scanning.apps.tanzu.vmware.com](#)) or an ImageScan

([imagescans.scanning.apps.tanzu.vmware.com](#)) fails policy enforcement through the ScanPolicy

([scanpolicies.scanning.apps.tanzu.vmware.com](#)). This can prevent source code from building or images deploying that contain vulnerabilities that are in violation of the user-defined scan policy.

For information about learning how to handle these vulnerabilities and unblock your Supply Chain, see [Triaging and Remediating CVEs](#).

## Scan Images using a different scanner

[Supply Chain Security Tools - Scan](#) includes additional integrations for running an image scan using Snyk and VMware Carbon Black.



## Out of the Box Supply Chain with Testing and Scanning

This package contains Cartographer Supply Chains that tie together a series of Kubernetes resources that drive a developer-provided workload from source code to a Kubernetes configuration ready to be deployed to a cluster. It contains supply chains that pass the source code through testing and vulnerability scanning, and also the container image.

This package includes all the capabilities of the Out of the Box Supply Chain With Testing, but adds source and image scanning using Gype.

Workloads that use source code or prebuilt images perform the following:

- Building from source code:
  1. Watching a Git Repository or local directory for changes
  2. Running tests from a developer-provided Tekton pipeline
  3. Scanning the source code for known vulnerabilities using 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 use this supply chain, verify that:

- Tanzu Application Platform GUI is configured to [enable CVE scan results](#). This configuration enables the Supply Chain Choreographer Tanzu Application Platform GUI plug-in to retrieve metadata about project packages and their vulnerabilities.
- Out of the Box Templates is installed.
- Out of the Box Supply Chain With Testing is **NOT installed**.
- Out of the Box Supply Chain With Testing and Scanning is **installed**.
- Developer namespace is configured with the objects according to Out of the Box Supply Chain With Testing guidance. This supply chain is in addition to the Supply Chain with testing.
- (Optionally) install [Out of the Box Delivery Basic](#), if you are willing to deploy the application to the same cluster as the workload and supply chains.

Verify that you have the supply chains with scanning, not with testing, installed. Run:

```
tanzu apps cluster-supply-chain list
```

| NAME                    | LABEL SELECTOR                                                               |
|-------------------------|------------------------------------------------------------------------------|
| source-test-scan-to-url | apps.tanzu.vmware.com/has-tests=true,apps.tanzu.vmware.com/workload-type=web |
| source-to-url           | apps.tanzu.vmware.com/workload-type=web                                      |

If you see `source-test-to-url` in the list, the setup is wrong. You **must not have the `source-test-to-url` installed** at the same time as `source-test-scan-to-url`.

## Developer namespace

This example builds on the previous Out of the Box Supply Chain examples, so only additions are included here.

To ensure that you configured the namespace correctly, it is important that the namespace has the objects that you configured in the other supply chain setups:

- **registries secrets:** Kubernetes secrets of type `kubernetes.io/dockerconfigjson` that contain credentials for pushing and pulling the container images built by the supply chain and the installation of Tanzu Application Platform.
- **service account:** The identity to be used for any interaction with the Kubernetes API made by the supply chain.
- **rolebinding:** Grant to the identity the necessary roles for creating the resources prescribed by the supply chain.

For more information about the preceding objects, see [Out of the Box Supply Chain Basic](#).

- **Tekton pipeline:** A pipeline runs whenever the supply chain hits the stage of testing the source code.

For more information, see [Out of the Box Supply Chain Testing](#).

And the new objects, that you create here:

- **scan policy:** Defines what to do with the results taken from scanning the source code and image produced. For more information, see [ScanPolicy section](#).
- **source scan template:** A template of how 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](#).

The following section includes details about the new objects, compared to Out of the Box Supply Chain With Testing.

## Updates to the developer namespace

For source and image scans, scan templates and scan policies must exist in the same namespace as the workload. These define:

- `ScanTemplate`: how to run a scan, allowing one to change details about the execution of the scan (either for images or source code)
- `ScanPolicy`: how to evaluate whether the artifacts scanned are compliant. For example, allowing one to be either very strict, or restrictive about particular vulnerabilities found.

The names of the objects **must** match the names in the example with default installation configurations. This is overridden either by using the `ootb_supply_chain_testing_scanning` package configuration in the `tap-values.yaml` file or by using workload parameters:

- To override by using the `ootb_supply_chain_testing_scanning` package configuration, make the following modification to your `tap-values.yaml` file and perform a [Tanzu Application Platform update](#).

```
ootb_supply_chain_testing_scanning:
  scanning:
```

```

source:
  policy: SCAN-POLICY
  template: SCAN-TEMPLATE
image:
  policy: SCAN-POLICY
  template: SCAN-TEMPLATE

```

Where `SCAN-POLICY` and `SCAN-TEMPLATE` are the names of the `ScanPolicy` and `ScanTemplate`.

- To override through workload parameters, use the following commands. For more information, see [Tanzu apps workload apply](#).

```

tanzu apps workload apply WORKLOAD --param "scanning_source_policy=SCAN-POLICY"
-n DEV-NAMESPACE
tanzu apps workload apply WORKLOAD --param "scanning_source_template=SCAN-TEMPL
ATE" -n DEV-NAMESPACE

```

Where:

- `WORKLOAD` is the name of the workload.
- `SCAN-POLICY` and `SCAN-TEMPLATE` are the names of the `ScanPolicy` and `ScanTemplate`.
- `DEV-NAMESPACE` is the developer namespace.

## ScanPolicy

The `ScanPolicy` defines a set of rules to evaluate for a particular scan to consider the artifacts (image or source code) either compliant or not.

When a `ImageScan` or `SourceScan` is created to run a scan, those reference a policy whose name **must** match the following sample `scan-policy`:

```

---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: scan-policy
  labels:
    'app.kubernetes.io/part-of': 'enable-in-gui'
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "UnknownSeverity"
    notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]
    ignoreCves := []

    contains(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
      severities := { e | e := match.ratings.rating.severity } | { e | e := match.rati
ngs.rating[_].severity }
      some i
      fails := contains(notAllowedSeverities, severities[i])
      not fails
    }

    isSafe(match) {
      ignore := contains(ignoreCves, match.id)
      ignore
    }

```

```

deny[msg] {
  comps := { e | e := input.bom.components.component } | { e | e := input.bom.components.component[_] }
  some i
  comp := comps[i]
  vulns := { e | e := comp.vulnerabilities.vulnerability } | { e | e := comp.vulnerabilities.vulnerability[_] }
  some j
  vuln := vulns[j]
  ratings := { e | e := vuln.ratings.rating.severity } | { e | e := vuln.ratings.rating[_].severity }
  not isSafe(vuln)
  msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
}

```

See [Writing Policy Templates](#).

## ScanTemplate

A ScanTemplate defines the PodTemplateSpec used by a Job to run a particular scan (image or source). When the supply chain initiates an ImageScan or SourceScan, they reference these templates which must live in the same namespace as the workload with the names matching the following:

- source scanning ([blob-source-scan-template](#))
- image scanning ([private-image-scan-template](#))

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 following the Getting Started guide, VMware recommends that you follow what is provided in the installation of `grype.scanning.apps.tanzu.vmware.com`. This is created in the same namespace as configured by using `grype.namespace` in either Tanzu Application Platform profiles or individual component installation as in the earlier example. For more information, see [About Source and Image Scans](#).

## Enable storing scan results

To enable SCST - Scan to store scan results by using SCST - Store, see [Developer namespace setup](#) for exporting the SCST - Store CA certificate and authentication token to the developer namespace.

### Allow multiple Tekton pipelines in a namespace

You can configure your developer namespace to include more than one pipeline using either of the following methods:

- Use a single pipeline running on a container image that includes testing tools and runs a common script to execute tests. This allows you to accommodate multiple workloads based in different languages in the same namespace that use a common make test script. For example:

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: developer-defined-tekton-pipeline
  labels:
    apps.tanzu.vmware.com/pipeline: test
spec:
  #...
  steps:
    - name: test
      image: <image_that_has_JDK_and_Go>
      script: |-
        cd `mktemp -d`
        wget -qO- $(params.source-url) | tar xvz -m
        make test
```

- Update the template to include labels that differentiate the pipelines. Then configure the labels to differentiate between pipelines. For example:

```
selector:
  resource:
    apiVersion: tekton.dev/v1beta1
    kind: Pipeline
    matchingLabels:
      apps.tanzu.vmware.com/pipeline: test
+     apps.tanzu.vmware.com/language: #@ data.values.workload.metadata.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, such as local development or gitops, the workload configuration details are the same as in Out of the Box Supply Chain Basic, except that you mark the workload as having tests enabled.

For example:

```

tanzu apps workload create tanzu-java-web-app \
  --git-repo ${GIT_PROJECT_URL} \
  --git-branch ${GIT_BRANCH} \
  --label apps.tanzu.vmware.com/has-tests=true \
  --label app.kubernetes.io/part-of=tanzu-java-web-app \
  --type web

```

Expect to see output similar to the following:

```

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/my/tanzu-project.git

```

## CVE triage workflow

The Supply Chain halts progression if either a SourceScan ([sourcescans.scanning.apps.tanzu.vmware.com](https://docs.tanzu.vmware.com/docs/1.4/scanning/sourcescans)) or an ImageScan ([imagescans.scanning.apps.tanzu.vmware.com](https://docs.tanzu.vmware.com/docs/1.4/scanning/imagescans)) fails policy enforcement through the ScanPolicy ([scanpolicies.scanning.apps.tanzu.vmware.com](https://docs.tanzu.vmware.com/docs/1.4/scanning/scanpolicies)). This can prevent source code from building or images deploying that contain vulnerabilities that are in violation of the user-defined scan policy.

For information about learning how to handle these vulnerabilities and unblock your Supply Chain, see [Triaging and Remediating CVEs](#).

## Scan Images using a different scanner

[Supply Chain Security Tools - Scan](#) includes additional integrations for running an image scan using Snyk and VMware Carbon Black.

## Install Out of the Box Supply Chain with Testing and Scanning

This document describes how to install Out of the Box Supply Chain with Testing and Scanning from the Tanzu Application Platform package repository.



### Note

Follow the steps in this topic if you do not want to use a profile to install Out of the Box Supply Chain with Testing and Scanning. For more information about profiles, see [Components and installation profiles](#).

The Out of the Box Supply Chain with Testing and Scanning package provides a ClusterSupplyChain that brings an application from source code to a deployed instance that:

- Runs in a Kubernetes environment.
- Performs validations in terms of running application tests.
- Scans the source code and image for vulnerabilities.

## Prerequisites

- Complete all prerequisites to install Tanzu Application Platform. For more information, see [Prerequisites](#).
- Install cartographer. For more information, see [Install Supply Chain Choreographer](#).
- Install [Out of the Box Delivery Basic](#)
- Install [Out of the Box Templates](#)

## Install

To install Out of the Box Supply Chain with Testing and Scanning:

1. Ensure you do not have Out of The Box Supply Chain With Testing (`ootb-supply-chain-testing.tanzu.vmware.com`) installed:

1. Run the following command:

```
tanzu package installed list --namespace tap-install
```

2. Verify `ootb-supply-chain-testing` is in the output:

| NAME                    | PACKAGE-NAME                             |
|-------------------------|------------------------------------------|
| ootb-delivery-basic     | ootb-delivery-basic.tanzu.vmware.com     |
| ootb-supply-chain-basic | ootb-supply-chain-basic.tanzu.vmware.com |
| ootb-templates          | ootb-templates.tanzu.vmware.com          |

3. If you see `ootb-supply-chain-testing` in the list, uninstall it by running:

```
tanzu package installed delete ootb-supply-chain-testing --namespace tap-
install
```

Example output:

```
Deleting installed package 'ootb-supply-chain-testing' in namespace 'tap-
install'.
Are you sure? [y/N]: y

| Uninstalling package 'ootb-supply-chain-testing' from namespace 'tap-in
stall'
\ Getting package install for 'ootb-supply-chain-testing'
- Deleting package install 'ootb-supply-chain-testing' from namespace 'ta
p-install'
| Deleting admin role 'ootb-supply-chain-testing-tap-install-cluster-rol
e'
| Deleting role binding 'ootb-supply-chain-testing-tap-install-cluster-ro
lebinding'
| Deleting secret 'ootb-supply-chain-testing-tap-install-values'
| Deleting service account 'ootb-supply-chain-testing-tap-install-sa'

Uninstalled package 'ootb-supply-chain-testing' from namespace 'tap-inst
all'
```

2. Check the values of the package that can be configured by running:

```
tanzu package available get ootb-supply-chain-testing-scanning.tanzu.vmware.co
m/0.7.0 \
--values-schema \
-n tap-install
```

For example:

| KEY                     | DESCRIPTION                                                                                                                                                                               |
|-------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| registry.repository     | Name of the repository in the image 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.server_address   | Default server address to be used for forming Git URLs for pushing Kubernetes configuration produced by the supply chain. This must include the scheme/protocol (e.g. https:// or ssh://) |
| gitops.repository_owner | Default project or user of the repository. Used to create URLs for pushing Kubernetes configuration produced by the supply chain.                                                         |
| gitops.repository_name  | Default repository name used for forming Git URLs for pushing Kubernetes                                                                                                                  |



|                                                     |                                                                                                                                                                                                       |
|-----------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>in.</code>                                    | configuration produced by the supply chain.                                                                                                                                                           |
| <code>gitops.username</code>                        | 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 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.                                                                               |
| <code>gitops.email</code>                           | Default user email to be used for the commits produced by the supply chain.                                                                                                                           |
| <code>gitops.ssh_secret</code>                      | 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.                                  |
| <code>gitops.commit_strategy</code>                 | Specification of how commits are made to the branch; directly or through a pull request.                                                                                                              |
| <code>gitops.repository_prefix</code>               | DEPRECATED: Use <code>server_address</code> and <code>repository_owner</code> instead.                                                                                                                |
| <code>gitops.repository_owner</code>                | Default prefix to be used for forming Git SSH URLs for pushing Kubernetes configuration produced by the supply chain.                                                                                 |
| <code>gitops.pull_request.server_kind</code>        | The git source control platform used                                                                                                                                                                  |
| <code>gitops.pull_request.commit_branch</code>      | The branch to which commits will be made, before opening a pull request                                                                                                                               |
| <code>gitops.pull_request.commit_branch</code>      | If the string "" is specified, an essentially random string will be used for the branch name, in order to prevent collisions.                                                                         |
| <code>gitops.pull_request.pull_request_title</code> | The title for the pull request                                                                                                                                                                        |
| <code>gitops.pull_request.pull_request_body</code>  | Any further information to add to the pull request                                                                                                                                                    |
| <code>cluster_builder</code>                        | Name of the Tanzu Build Service 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 Image objects as well as deploying the application. |

3. Create a file named `ootb-supply-chain-testing-scanning-values.yaml` that specifies the corresponding values to the properties you want to change. For example:

```

registry:
  server: REGISTRY-SERVER
  repository: REGISTRY-REPOSITORY

gitops:
  server_address: https://github.com/
  repository_owner: vmware-tanzu
  branch: main
  username: supplychain
  email: supplychain
  commit_message: supplychain@cluster.local
  ssh_secret: git-ssh
  commit_strategy: direct

cluster_builder: default
service_account: default

```



### Important

The `gitops.repository_prefix` field must end with `/`.

4. With the configuration ready, install the package by running:

```

tanzu package install ootb-supply-chain-testing-scanning \
  --package-name ootb-supply-chain-testing-scanning.tanzu.vmware.com \
  --version 0.7.0 \
  --namespace tap-install \
  --values-file ootb-supply-chain-testing-scanning-values.yaml

```

Example output:

```

\ Installing package 'ootb-supply-chain-testing-scanning.tanzu.vmware.com'
| Getting package metadata for 'ootb-supply-chain-testing-scanning.tanzu.vmw
e.com'
| Creating service account 'ootb-supply-chain-testing-scanning-tap-install-sa'
| Creating cluster admin role 'ootb-supply-chain-testing-scanning-tap-install-c
luster-role'
| Creating cluster role binding 'ootb-supply-chain-testing-scanning-tap-install
-cluster-rolebinding'
| Creating secret 'ootb-supply-chain-testing-scanning-tap-install-values'
| Creating package resource
- Waiting for 'PackageInstall' reconciliation for 'ootb-supply-chain-testing-sc
anning'
\ 'PackageInstall' resource install status: Reconciling

Added installed package 'ootb-supply-chain-testing-scanning' in namespace 'tap-
install'

```

## Out of the Box Templates for Supply Chain Choreographer

This topic describes the templates you can use with Supply Chain Choreographer.

Templates define Kubernetes objects based on configuration in the workload, supply chain Tanzu Application Platform values, and results output from other templated objects. A supply chain organizes a set of templates into a directed acyclic graph. This package contains templates that are used by the Out of the Box Supply Chains and the Out of the Box Delivery. You must install this package to have Workloads delivered properly.

The OOTB Template package includes:

- [Cartographer Templates](#): See [reference](#)

- [Cartographer ClusterRunTemplates](#): See [reference](#)
- [Tekton ClusterTasks](#)
- [ClusterRoles](#)
- [openshift SecurityContextConstraints](#)

For information about OOTB Supply Chains and Delivery, see:

- [Out of the Box Supply Chain Basic](#)
- [Out of the Box Supply Chain with Testing](#)
- [Out of the Box Supply Chain with Testing and Scanning](#)
- [Out of the Box Delivery Basic](#)

## Out of the Box Templates for Supply Chain Choreographer

This topic describes the templates you can use with Supply Chain Choreographer.

Templates define Kubernetes objects based on configuration in the workload, supply chain Tanzu Application Platform values, and results output from other templated objects. A supply chain organizes a set of templates into a directed acyclic graph. This package contains templates that are used by the Out of the Box Supply Chains and the Out of the Box Delivery. You must install this package to have Workloads delivered properly.

The OOTB Template package includes:

- [Cartographer Templates](#): See [reference](#)
- [Cartographer ClusterRunTemplates](#): See [reference](#)
- [Tekton ClusterTasks](#)
- [ClusterRoles](#)
- [openshift SecurityContextConstraints](#)

For information about OOTB Supply Chains and Delivery, see:

- [Out of the Box Supply Chain Basic](#)
- [Out of the Box Supply Chain with Testing](#)
- [Out of the Box Supply Chain with Testing and Scanning](#)
- [Out of the Box Delivery Basic](#)

## Install Out of the Box Templates

This document describes how to install Out of the Box Templates from the Tanzu Application Platform package repository.



### Note

Follow the steps in this topic if you do not want to use a profile to install Out of the Box Templates. For more information about profiles, see [Components and installation profiles](#).

The Out of the Box Templates package is used by all the Out of the Box Supply Chains to provide the templates that are used by the Supply Chains to create the objects that drive source code all the way to a deployed application in a cluster.

## Prerequisites

Before installing Out of the Box Templates:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see [Prerequisites](#).
- Install cartographer. For more information, see [Install Supply Chain Choreographer](#).
- Install [Tekton Pipelines](#).

## Install

To install Out of the Box Templates:

1. View the configurable values of the package by running:

```
tanzu package available get ootb-templates.tanzu.vmware.com/0.7.0 \
  --values-schema \
  -n tap-install
```

For example:

| KEY                | DEFAULT | TYPE  | DESCRIPTION                                                              |
|--------------------|---------|-------|--------------------------------------------------------------------------|
| excluded_templates | []      | array | List of templates to exclude from the installation (e.g. ['git-writer']) |

2. Create a file named `ootb-templates.yaml` that specifies the corresponding values to the properties you want to change.

For example, the contents of the file might look like this:

```
excluded_templates: []
```

3. After the configuration is ready, install the package by running:

```
tanzu package install ootb-templates \
  --package-name ootb-templates.tanzu.vmware.com \
  --version 0.7.0 \
  --namespace tap-install \
  --values-file ootb-templates-values.yaml
```

Example output:

```
\ Installing package 'ootb-templates.tanzu.vmware.com'
| Getting package metadata for 'ootb-templates.tanzu.vmware.com'
| Creating service account 'ootb-templates-tap-install-sa'
| Creating cluster admin role 'ootb-templates-tap-install-cluster-role'
| Creating cluster role binding 'ootb-templates-tap-install-cluster-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'
```

## Out of the Box Delivery Basic

This package provides a reusable ClusterDelivery object that delivers the Kubernetes configuration that the Out of the Box Supply Chain produces to an environment, including [Basic](#), [Testing](#), and

Testing With Scanning supply chains.

## Prerequisites

To make use of this package you must have installed:

- [Supply Chain Cartographer](#)
- [Out of the Box Templates](#)

## Using Out of the Box Delivery Basic

Out of the Box Delivery Basic support both GitOps and local development workflows:

```

GITOPS

  Deliverable:
    points at a git repository where source code is found and
    kubernetes configuration is pushed to

LOCAL DEVELOPMENT

  Deliverable:

    points at a container image registry where the supplychain
    pushes source code and configuration to

---

DELIVERY

  takes a Deliverable (local or gitops) and passes is through
  a series of resources:

      config-provider <---[config]--- deployer
          .
          .
GitRepository/ImageRepository      kapp-ctrl/App
                                     - knative/Service
                                     - ResourceClaim
                                     - ServiceBinding
                                     ...

```

You must install this package to have Workloads delivered properly with the [Basic](#), [Testing](#), and [Testing With Scanning](#) Out of the Box Supply Chains.

Consumers do not interact directly with this package. Instead, this package is used after the supply chains create a [carto.run/Deliverable](#) object to express the intention of having the Workloads that go through them delivered to an environment. The environment is the same Kubernetes cluster as the Supply Chains.

## More information

- [Reference](#)
- [Installation](#)

## Out of the Box Delivery Basic

This package provides a reusable ClusterDelivery object that delivers the Kubernetes configuration that the Out of the Box Supply Chain produces to an environment, including [Basic](#), [Testing](#), and [Testing With Scanning](#) supply chains.

## Prerequisites

To make use of this package you must have installed:

- [Supply Chain Cartographer](#)
- [Out of the Box Templates](#)

## Using Out of the Box Delivery Basic

Out of the Box Delivery Basic support both GitOps and local development workflows:

```

GITOPS

  Deliverable:
    points at a git repository where source code is found and
    kubernetes configuration is pushed to

LOCAL DEVELOPMENT

  Deliverable:

    points at a container image registry where the supplychain
    pushes source code and configuration to

---

DELIVERY

  takes a Deliverable (local or gitops) and passes is through
  a series of resources:

      config-provider <---[config]--- deployer
      .
      .
  GitRepository/ImageRepository      kapp-ctrl/App
                                      - knative/Service
                                      - ResourceClaim
                                      - ServiceBinding
                                      ...

```

You must install this package to have Workloads delivered properly with the [Basic](#), [Testing](#), and [Testing With Scanning](#) Out of the Box Supply Chains.

Consumers do not interact directly with this package. Instead, this package is used after the supply chains create a [carto.run/Deliverable](#) object to express the intention of having the Workloads that go through them delivered to an environment. The environment is the same Kubernetes cluster as the Supply Chains.

## More information

- [Reference](#)
- [Installation](#)

## Install Out of the Box Delivery Basic

This document describes how to install Out of the Box Delivery Basic from the Tanzu Application Platform package repository.



### Note

Follow the steps in this topic if you do not want to use a profile to install Out of the Box Delivery Basic. For more information about profiles, see [Components and installation profiles](#).

The Out of the Box Delivery Basic package is used by all the Out of the Box Supply Chains to deliver the objects that have been produced by them to a Kubernetes environment.

## Prerequisites

Before installing Out of the Box Delivery Basic:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see [Prerequisites](#).
- Install cartographer. For more information, see [Install Supply Chain Choreographer](#).

## Install

To install Out of the Box Delivery Basic:

1. Familiarize yourself with the set of values of the package that can be configured by running:

```
tanzu package available get ootb-delivery-basic.tanzu.vmware.com/0.7.0 \
  --values-schema \
  -n tap-install
```

For example:

| KEY                | DEFAULT | TYPE   | DESCRIPTION                                                                         |
|--------------------|---------|--------|-------------------------------------------------------------------------------------|
| service_account    | default | string | Name of the service account in the namespace where the Deliverable is submitted to. |
| git_implementation | go-git  | string | Which git client library to use. Valid options are go-git or libgit2.               |

2. Create a file named `ootb-delivery-basic-values.yaml` that specifies the corresponding values to the properties you want to change.

For example, the contents of the file might look like this:

```
service_account: default
```

3. With the configuration ready, install the package by running:

```
tanzu package install ootb-delivery-basic \
  --package-name ootb-delivery-basic.tanzu.vmware.com \
  --version 0.7.0 \
  --namespace tap-install \
  --values-file ootb-delivery-basic-values.yaml
```

Example output:

```
\ Installing package 'ootb-delivery-basic.tanzu.vmware.com'
| Getting package metadata for 'ootb-delivery-basic.tanzu.vmware.com'
| Creating service account 'ootb-delivery-basic-tap-install-sa'
| Creating cluster admin role 'ootb-delivery-basic-tap-install-cluster-role'
| Creating cluster role binding 'ootb-delivery-basic-tap-install-cluster-rolebi
nding'
| Creating secret 'ootb-delivery-basic-tap-install-values'
| Creating package resource
- Waiting for 'PackageInstall' reconciliation for 'ootb-delivery-basic'
/ 'PackageInstall' resource install status: Reconciling

Added installed package 'ootb-delivery-basic' in namespace 'tap-install'
```

## How-to guides for Supply Chain Choreographer for Tanzu

This topic describes the how-to guides you can use for Supply Chain Choreographer for Tanzu.

### How-to guides

The following how-to guides apply to Supply Chain Choreographer for Tanzu:

- [Install Supply Chain Choreographer](#)
- [Install Out of the Box Delivery Basic](#)
- [Install Out of the Box Supply Chain Basic](#)
- [Install Out of the Box Supply Chain with Testing](#)
- [Install Out of the Box Supply Chain with Testing and Scanning](#)
- [Install Out of the Box Templates](#)
- [Tanzu Build Service Integration](#)
- [Building from source](#)
- [Git authentication](#)

### Out of the Box Supply Chain with testing on Jenkins

The Out of the Box templates package now includes a Tekton `ClusterTask` resource, which triggers a build for a specified Jenkins job.

You can configure the Jenkins task in both the [Out of the Box Supply Chain with Testing](#) and [Out of the Box Supply Chain With Testing and Scanning](#) to trigger a Jenkins job. The task is implemented as a Tekton `ClusterTask` and can now run from a Tekton `Pipeline`.

### Prerequisites

Follow the instructions from [Out of the Box Supply Chain With Testing](#) or [Out of the Box Supply Chain With Testing and Scanning](#) to install the required packages. You must set up only one of these packages.

These supply chains can use the Jenkins service during the `source-tester` phase of the pipeline.

### Making a Jenkins test job

The intent of the Jenkins task for the Out of the Box templates is to help Tanzu Application Platform users keep their existing test suites on their Jenkins services and still integrate with the



modern application deployment pipeline that Tanzu Application Platform provides.

This section of the guide instructs you on how to configure a Jenkins job triggered by the Tanzu Application Platform Jenkins task.

It is assumed that you are using the Jenkins job to run test suites on code. For the Jenkins job to know which source code to test, the Jenkins task calls the Jenkins job with the `Workload` and `job-params` parameters, even if they are not declared in `Workload` or `job-params`. The Jenkins tasks only pass these parameters if they are defined in the Jenkins job itself.

- **SOURCE-URL string** The URL of the source code being tested. The `source-provider` resource in the supply chain provides this code and is only resolvable inside the Kubernetes cluster. This URL is only useful if your Jenkins service is running inside the cluster or if there is ingress set up and the Jenkins service can make requests to services inside the cluster.
- **SOURCE-REVISION string** The revision of the source code being tested. The format of this value can vary depending on the implementation of the `source_provider` resource. If the `source-provider` is the Flux CD `GitRepository` resource, then the value of the `SOURCE-REVISION` is the Git branch name followed by the commit SHA, both separated by a (/) slash character. For example: `main/2b1ed6c3c4f74f15b0e4de2732234eafd050eb1ca`. Your Jenkins pipeline script must extract the commit SHA from the `SOURCE-REVISION` to be useful. See the example in the [Example Jenkins Job](#) for guidance.

If you can't use the `SOURCE-URL` because your Jenkins service cannot make requests into the Kubernetes cluster, then you can supply the source code URL to the Jenkins job with other parameters instead. See the following example.

### Example Jenkins Job

Add the following parameters to your Jenkins job:

- **SOURCE-REVISION string**
- **GIT-URL string**

Use the following script in your pipeline:

```
#!/bin/env groovy

pipeline {
  agent {
    kubernetes {
      label 'maven'
    }
  }

  stages {
    stage('Checkout code') {
      steps {
        script {
          sourceUrl = params.SOURCE-REVISION
          indexSlash = sourceUrl.indexOf("/")
          revision = sourceUrl.substring(indexSlash + 1)
        }
        sh "git clone ${params.GIT-URL} target"
        dir("target") {
          sh "git checkout ${revision}"
        }
      }
    }

    stage('Maven test') {
      steps {
```

```

    container('maven') {
      dir("target") {
        // Example tests with maven
        sh "mvn clean test --no-transfer-progress"
      }
    }
  }
}
}
}
}
}

```

To configure your `Workload` to pass the `GIT-URL` parameter into the Jenkins task:

```

tanzu apps workload create workload \
  --namespace your-test-namespace \
  --git-branch main \
  --git-repo https://your.git/repository.git \
  --label apps.tanzu.vmware.com/has-tests=true \
  --label app.kubernetes.io/part-of=test-workload \
  --param-yaml testing_pipeline_matching_labels='{"apps.tanzu.vmware.com/pipeline":"jenkins-pipeline"}' \
  --param-yaml testing_pipeline_params='{"secret-name":"my-secret","job-name":"jenkins-job-name","job-params":[{"name":"GIT_URL","value":"https://your.git/repository.git"}]}' \
  --type web \
  --yes

```

The `Workload` is described in the later [Developer Workload](#) section.

## Updates to the developer namespace

### Create a secret

A secret must be created in the developer namespace with the following properties:

- **JENKINS-URL required:** URL of the Jenkins instance that hosts the job, including the scheme. For example: `https://my-jenkins.com`.
- **USERNAME required:** User name of the user that has access to trigger a build on Jenkins.
- **PASSWORD required:** Password of the user that has access to trigger a build on Jenkins.
- **PEM-CA-CERT optional:** The PEM-encoded CA certificate to verify the Jenkins instance identity.

For example:

```

apiVersion: v1
kind: Secret
metadata:
  name: MY-SECRET
type: Opaque
stringData:
  url: JENKINS-URL
  username: USERNAME
  password: PASSWORD
  ca-cert: PEM-CA-CERT

```

You cannot use the Tanzu CLI to create secrets such as this, but you can use the Kubernetes CLI tool (`kubectl`) instead.

If you saved the password to a file, and you saved the optional PEM-encoded CA certificate in a file, here is an example command to create this kind of secret:

```
kubectl create secret generic my-secret \
  --from-literal=url=https://jenkins.instance \
  --from-literal=username=literal-username \
  --from-file=password=/path/to/file/with/password.txt \
  --from-file=ca-cert=/path/to/ca-certificate.pem \
```

## Create a Tekton pipeline

The developer must create a Tekton `Pipeline` object with the following parameters:

- `source-url`, **required**: An HTTP address where a `.tar.gz` file containing all the source code being tested is supplied.
- `source-revision`, **required**: The revision of the commit or image reference found by the `source-provider`.
- `secret-name`, **required**: The secret that contains the URL, user name, password, and certificate (optional) to the Jenkins instance that houses the job that is required to run.
- `job-name`, **required**: The name of the Jenkins job that is required to run.
- `job-params`, **required**: A list of key-value pairs, encoded as a JSON string, that passes in parameters needed for the Jenkins job.

Tasks:

- `jenkins-task`, **required**: This `ClusterTask` is one of the tasks that the pipeline runs to trigger the Jenkins job. It is installed in the cluster by the “Out of the Box Templates” package.

Results:

- `jenkins-job-url`: A string result that outputs the URL of the Jenkins build that the Tekton task triggered. The `jenkins-task ClusterTask` populates the output.

For example:

```
---
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: developer-defined-jenkins-tekton-pipeline
  namespace: developer-namespace
  labels:
    #! This label should be provided to the Workload so that
    #! the supply chain can find this pipeline
    apps.tanzu.vmware.com/pipeline: jenkins-pipeline
spec:
  results:
  - name: jenkins-job-url    #! To show the job URL on the
    #! Tanzu Application Platform GUI
    value: ${tasks.jenkins-task.results.jenkins-job-url}
  params:
  - name: source-url        #! Required
  - name: source-revision   #! Required
  - name: secret-name       #! Required
  - name: job-name          #! Required
  - name: job-params        #! Required
  tasks:
    #! Required: Include the built-in task that triggers the
    #! given job in Jenkins
  - name: jenkins-task
    taskRef:
      name: jenkins-task
      kind: ClusterTask
```

```

params:
  - name: source-url
    value: $(params.source-url)
  - name: source-revision
    value: $(params.source-revision)
  - name: secret-name
    value: $(params.secret-name)
  - name: job-name
    value: $(params.job-name)
  - name: job-params
    value: $(params.job-params)

```

Save the earlier YAML definition to a file, for example, `pipeline.yaml`. Run:

```
kubectl apply -f pipeline.yaml
```

## Patch the Service Account

The `jenkins-task ClusterTask` resource uses a container image with the Jenkins Adapter application to trigger the Jenkins job and wait for it to complete. This container image is distributed with Tanzu Application Platform on VMware Tanzu Network, but it is not installed at the same time as the other packages. It is pulled at the time that the supply chain executes the job. As a result, it does not implicitly have access to the `imagePullSecrets` with the required credentials.



### Important

The `ServiceAccount` that a developer can configure with their `Workload` is *not* passed to the task and is not used to pull the Jenkins Adapter container image. If you followed the Tanzu Application Platform Install Guide, then you have a `Secret` named `tap-registry` in each of your cluster's namespaces. You can patch the default Service Account in your workload's namespace so that your supply chain can pull the Jenkins Adapter image. For example:

```

kubectl patch serviceaccount default \
  --patch '{"imagePullSecrets": [{"name": "tap-registry"}]}' \
  --namespace developer-namespace

```

These tasks are run by Tekton. Tekton has other methods for configuring the [custom service account credentials](#) used by running tasks, if you prefer.

## Developer Workload

Submit your `Workload` to the same namespace as the Tekton `Pipeline` defined earlier.

To enable the supply chain to run Jenkins tasks, the `Workload` must include the following parameters:

```

parameters:

  #! Required: selects the pipeline
  - name: testing_pipeline_matching_labels
    value:
      #! This label must match the label on the pipeline created earlier
      apps.tanzu.vmware.com/pipeline: jenkins-pipeline

  #! Required: Passes parameters to pipeline
  - name: testing_pipeline_params
    value:

```

```

#! Required: Name of the Jenkins job
job-name: my-jenkins-job

#! Required: The secret created earlier to access Jenkins
secret-name: my-secret

#! Required: The `job-params` element is required, but the parameter string
#! might be empty. If empty, then set this value to `[]`. If non-empty then the
#! value contains a JSON-encoded list of parameters to pass to the Jenkins job.
#! Ensure that the quotation marks inside the JSON-encoded string are escaped.
job-params: "[{\"name\":\"A\",\"value\":\"x\"},{\"name\":\"B\",\"value\":\"y
\"},...]"

```

You can create the workload by using the `apps` CLI plug-in:

```

tanzu apps workload create my-workload-name \
  --namespace developer-namespace \
  --git-branch my-branch \
  --git-repo https://my-source-code-repository \
  --label apps.tanzu.vmware.com/has-tests=true \
  --label app.kubernetes.io/part-of=my-workload-name \
  --param-yaml testing_pipeline_matching_labels="{\"apps.tanzu.vmware.com/pipeline\":\"jenkins-pipeline\"}" \
  --param-yaml testing_pipeline_params="{\"secret-name\":\"my-secret\", \"job-name\": \"jenkins-job\", \"job-params\": \"SEE BELOW\"}"
--type web

```

The value of the `job-params` parameter is a list of zero-or-more parameters that are sent to the Jenkins job. The parameter is entered into the `Workload` as a list of name-value pairs. For example:

```

[{"name":"GIT-URL", "value":"https://github.com/spring-projects/spring-petclinic"},
{"name":"GIT-BRANCH", "value":"main"}]

```

Where:

- `GIT-URL` is the URL of your GitHub repository.
- `GIT-BRANCH` is the branch you want to target.



### Important

None of the fields in the `Workload` resource are implicitly passed to the Jenkins job. You have to set them in the `job-params` explicitly. An exception to this is the `SOURCE_URL` and `SOURCE_REVISION` parameters are sent to the Jenkins job implicitly by the Jenkins Adapter trigger application. For example, you can use the `SOURCE_REVISION` to verify which commit SHA to test. See [Making a Jenkins Test Job](#) earlier for details about how to use the Git URL and source revision in a Jenkins test job.

Watch the quoting of the `job-params` value closely. In the earlier `tanzu apps workload create` example, the `job-params` value is a string with a JSON structure in it. The value of the `--param-yaml testing_pipeline_params` parameter is a JSON string. Add backslash (`\`) escape characters before the double quote characters (`"`) in the `job-params` value.

Example output from the `tanzu apps workload create` command:

```

Create workload:
 1 + |---
 2 + |apiVersion: carto.run/v1alpha1
 3 + |kind: Workload

```

```

4 + |metadata:
5 + |  labels:
6 + |    app.kubernetes.io/part-of: my-workload-name
7 + |    apps.tanzu.vmware.com/has-tests: "true"
8 + |  name: my-workload-name
9 + |  namespace: developer-namespace
10 + |spec:
11 + |  params:
12 + |    - name: testing_pipeline_matching_labels
13 + |      value:
14 + |        apps.tanzu.vmware.com/pipeline: jenkins-pipeline
15 + |    - name: testing_pipeline_params
16 + |      value:
17 + |        job-name: jenkins-job
18 + |        job-params:
19 + |          - name: param1
20 + |            value: value1
21 + |          secret-name: my-secret
22 + |      source:
23 + |        git:
24 + |          ref:
25 + |            branch: my-branch
26 + |          url: https://my-source-code-repository

```

## Building container images with Supply Chain Choreographer

This topic describes the methods you can use to build container images for Supply Chain Choreographer for Tanzu.

### Methods for building container images

You can build a container image by using:

- A Maven artifact. See [Building from source](#)
- A Dockerfile based build. See [Dockerfile-based builds](#)
- Tanzu Build Service with buildpacks. See [Tanzu Build Service Integration](#)

### Building from source with Supply Chain Choreographer

You can build from source by providing source code for the workload with any Supply Chain package.

You can provide source code for the workload from one of three places:

1. A Git repository.
2. A directory in your local computer's file system.
3. A Maven repository.

```

Supply Chain
-- fetch source          * either from Git or local directory
-- test
-- build
-- scan
-- apply-conventions
-- push config

```

This document provides details about each approach.

**Note**

To provide a prebuilt container image instead of building the application from the beginning by using the supply chain, see [Using an existing image](#).

## Git source

To provide source code from a Git repository to the supply chains, you must fill `workload.spec.source.git`. With the Tanzu CLI, you can do so by using the following flags:

- `--git-branch`: branch within the Git repository to checkout
- `--git-commit`: commit SHA within the Git repository to checkout
- `--git-repo`: Git URL to remote source code
- `--git-tag`: tag within the Git repository to checkout

For example, after installing `ootb-supply-chain-basic`, to create a `Workload` the source code for which comes from the `main` branch of the `github.com/vmware-tanzu/application-accelerator-samples` Git repository, and the subdirectory `tanzu-java-web-app` run:

```
tanzu apps workload create tanzu-java-web-app \
  --app tanzu-java-web-app \
  --type web \
  --git-repo ${GIT_PROJECT_URL} \
  --sub-path tanzu-java-web-app \
  --git-branch ${GIT_BRANCH}
```

Expect to see similar 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/my/tanzu-project.git
```

**Important**

The Git repository URL must include the scheme: `http://`, `https://`, or `ssh://`.

## Private `GitRepository`

To fetch source code from a repository that requires credentials, you must provide those by using a Kubernetes secret object that the `GitRepository` object created for that workload references. See [How It Works](#) to learn more about detecting changes to the repository.

```

Workload/tanzu-java-web-app
└─GitRepository/tanzu-java-web-app
    └───> secretRef: {name: GIT-SECRET-NAME}
        |
        | either a default from TAP installation or
        | gitops_ssh_secret Workload parameter

```

Platform operators who install the Out of the Box Supply Chain packages by using Tanzu Application Platform profiles can customize the default name of the secret (`git-ssh`) by editing the corresponding `ootb_supply_chain*` property in the `tap-values.yaml` file:

```

ootb_supply_chain_basic:
  gitops:
    ssh_secret: GIT-SECRET-NAME

```

For platform operators who install the `ootb-supply-chain-*` package individually by using `tanzu package install`, they can edit the `ootb-supply-chain-*-values.yml` as follows:

```

gitops:
  ssh_secret: GIT-SECRET-NAME

```

You can also override the default secret name directly in the workload by using the `gitops_ssh_secret` parameter, regardless of how Tanzu Application Platform is installed. You can use the `--param` flag in Tanzu CLI. For example:

```

tanzu apps workload create tanzu-java-web-app \
  --app tanzu-java-web-app \
  --type web \
  --git-repo ${GIT_PROJECT_URL} \
  --git-branch ${GIT_BRANCH} \
  --param gitops_ssh_secret=SECRET-NAME

```

Expect to see similar 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/my/tanzu-project.git

```



#### Note

A secret reference is only provided to `GitRepository` if `gitops_ssh_secret` is set to a non-empty string in some fashion, either by a package property or a workload



parameter. To force a `GitRepository` to not reference a secret, set the value to an empty string (`""`).

After defining the name of the Kubernetes secret, you can define the secret.

### HTTP(S) Basic-authentication and Token-based authentication

Despite both the package value and workload parameter being called `gitops.ssh_secret`, you can use HTTP(S) transports as well:

1. Ensure that the repository in the `Workload` specification uses `http://` or `https://` schemes in any URLs that relate to the repositories. For example, `https://github.com/my-org/my-repo` instead of `github.com/my-org/my-repo` or `ssh://github.com:my-org/my-repo`.
2. In the same namespace as the workload, create a Kubernetes secret object of type `kubernetes.io/basic-auth` with the name matching the one expected by the supply chain. For example:

```
apiVersion: v1
kind: Secret
metadata:
  name: GIT-SECRET-NAME
  annotations:
    tekton.dev/git-0: GIT-SERVER      # ! required
type: kubernetes.io/basic-auth
stringData:
  username: GIT-USERNAME
  password: GIT-PASSWORD
```

3. With the secret created with the name matching the one configured for `gitops.ssh_secret`, attach it to the `ServiceAccount` used by the workload. For example:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: default
secrets:
  - name: registry-credentials
  - name: tap-registry
  - name: GIT-SECRET-NAME
imagePullSecrets:
  - name: registry-credentials
  - name: tap-registry
```

For more information about the credentials and setting up the Kubernetes secret, see [Git Authentication's HTTP section](#).

### SSH authentication

Aside from using HTTP(S) as a transport, you can also use SSH:

1. Ensure that the repository URL in the workload specification uses `ssh://` as the scheme in the URL, for example, `ssh://git@github.com:my-org/my-repo.git`
2. Create a Kubernetes secret object of type `kubernetes.io/ssh-auth`:

```
apiVersion: v1
kind: Secret
metadata:
  name: GIT-SECRET-NAME
  annotations:
```

```

tekton.dev/git-0: GIT-SERVER
type: kubernetes.io/ssh-auth
stringData:
  ssh-privatekey: SSH-PRIVATE-KEY      # private key with push-permissions
  identity: SSH-PRIVATE-KEY            # private key with pull permissions
  identity.pub: SSH-PUBLIC-KEY          # public of the `identity` private key
  known_hosts: GIT-SERVER-PUBLIC-KEYS  # git server public keys

```

3. With the secret created with the name matching the one configured for `gitops.ssh_secret`, attach it to the ServiceAccount used by the workload. For example:

```

apiVersion: v1
kind: ServiceAccount
metadata:
  name: default
secrets:
  - name: registry-credentials
  - name: tap-registry
  - name: GIT-SECRET-NAME
imagePullSecrets:
  - name: registry-credentials
  - name: tap-registry

```

For information about how to generate the keys and set up SSH with the Git server, see [Git Authentication's SSH section](#).

## How it works

With the `workload.spec.source.git` filled, the supply chain takes care of managing a child `GitRepository` object that keeps track of commits made to the Git repository stated in `workload.spec.source.git`.

For each revision found, `gitrepository.status.artifact` gets updated providing information about an HTTP endpoint that the controller makes available for other components to fetch the source code from within the cluster.

The digest of the latest commit:

```

apiVersion: source.toolkit.fluxcd.io/v1beta1
kind: GitRepository
metadata:
  name: tanzu-java-web-app
spec:
  gitImplementation: go-git
  ignore: '!.git'
  interval: 1m0s
  ref: {branch: main}
  timeout: 20s
  url: https://github.com/vmware-tanzu/application-accelerator-samples
status:
  artifact:
    checksum: 375c2daee5fc8657c5c5b49711a8e94d400994d7
    lastUpdateTime: "2022-04-07T15:02:30Z"
    path: gitrepository/default/tanzu-java-web-app/d85df1fc.tar.gz
    revision: main/d85df1fc28c6b86ca54bd613f55991645d3b257c
    url: http://source-controller.flux-system.svc.cluster.local./gitrepository/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 is fetched, without having to process any Git-specific details in multiple places.

## Workload parameters

You can pass the following parameters by using the workload object's `workload.spec.params` field to override the default behavior of the `GitRepository` object created for keeping track of the changes to a repository:

- `gitImplementation`: name of the Git implementation (either `libgit2` or `go-git`) to fetch the source code.
- `gitops_ssh_secret`: name of the secret in the same namespace as the workload where credentials to fetch the repository are found.

You can also customize the following parameters with defaults for the whole cluster. Do this by using properties for either `tap-values.yaml` when installing supply chains by using Tanzu Application Platform profiles, or `ootb-supply-chain-*-values.yml` when installing the OOTB packages individually):

- `git_implementation`: the same as `gitImplementation` workload parameter
- `gitops.ssh_secret`: the same as `gitops_ssh_secret` workload parameter

## Local source

You can provide source code from a local directory such as, from a directory in the developer's file system. The Tanzu CLI provides two flags to specify the source code location in the file system and where the source code is pushed to as a container image:

- `--local-path`: path on the local file system to a directory of source code to build for the workload
- `--source-image`: destination image repository where source code is staged before being built

This way, whether the cluster the developer targets is local (a cluster in the developer's machine) or not, the source code is made available by using a container image registry.

For example, if a developer has source code under the current directory (`.`) and access to a repository in a container image registry, you can create a workload as follows:

```
tanzu apps workload create tanzu-java-web-app \
--app tanzu-java-web-app \
--type web \
--local-path . \
--source-image $REGISTRY/test
```

```
Publish source in "." to "REGISTRY-SERVER/REGISTRY-REPOSITORY"?
It may be visible to others who can pull images from that repository

Yes

Publishing source in "." to "REGISTRY-SERVER/REGISTRY-REPOSITORY"...
Published source

Create workload:
 1 + |---
```

```

2 + |apiVersion: carto.run/v1alpha1
3 + |kind: Workload
4 + |metadata:
5 + |  labels:
6 + |    app.kubernetes.io/part-of: tanzu-java-web-app
7 + |    apps.tanzu.vmware.com/workload-type: web
8 + |  name: tanzu-java-web-app
9 + |  namespace: default
10 + |spec:
11 + |  source:
12 + |    image: REGISTRY-SERVER/REGISTRY-REPOSITORY:latest@<digest>

```

Where:

- `REGISTRY-SERVER` is the container image registry.
- `REGISTRY-REPOSITORY` is the repository in the container image registry.

## Authentication

Both the cluster and the developer's machine must be configured to properly provide credentials for accessing the container image registry where the local source code is published to.

### Developer

The Tanzu CLI must push the source code to the container image registry indicated by `--source-image`. To do so, the CLI must find the credentials, so the developer must configure their machine accordingly.

To ensure credentials are available, use `docker` to make the necessary credentials available for the Tanzu CLI to perform the image push. Run:

```
docker login REGISTRY-SERVER -u REGISTRY-USERNAME -p REGISTRY-PASSWORD
```

### Supply chain components

Aside from the developer's ability to push source code to the container image registry, the cluster must also have the proper credentials, so it can pull that container image, unpack it, run tests, and build the application.

To provide the cluster with the credentials, point the ServiceAccount used by the workload at the Kubernetes secret that contains the credentials.

If the registry that the developer targets is the same one for which credentials were provided while setting up the workload namespace, no further action is required. Otherwise, follow the same steps as recommended for the application image.

## How it works

A workload specifies that source code must come from an image by setting `workload.spec.source.image` to point at the registry provided by using `--source-image`. Instead of having a `GitRepository` object created, an `ImageRepository` object is instantiated, with its specification filled in such a way to keep track of images pushed to the registry provided by the user.

Take the following workload as an example:

```

apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: app

```

```

labels:
  app.kubernetes.io/part-of: app
  apps.tanzu.vmware.com/workload-type: web
spec:
  source:
    image: 10.188.0.3:5000/test:latest

```

Instead of a `GitRepository` object, an `ImageRepository` is created:

```

Workload/app
|
- |--GitRepository/app
+ |--ImageRepository/app
  |
  |--Image/app
  | |--Build/app-build-1
  | |   |--Pod/app-build-1-build-pod
  | |   |--PersistentVolumeClaim/app-cache
  | |   |--SourceResolver/app-source
  |
  |--PodIntent/app
  |--ConfigMap/app
  |--Runnable/app-config-writer
  |   |--TaskRun/app-config-writer-2zj7w
  |   |--Pod/app-config-writer-2zj7w-pod

```

`ImageRepository` provides the same semantics as `GitRepository`, except that it looks for source code in container image registries rather than Git repositories.

## Maven Artifact

This approach aids integration with existing CI systems, such as Jenkins, and can pull artifacts from existing Maven repositories, including Jfrog Artifactory.

There are no dedicated fields in the `Workload` resource for specifying the Maven artifact configuration. You must fill in the `name/value` pairs in the `params` structure.

For example:

```

apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: my-workload
  labels:
    apps.tanzu.vmware.com/workload-type: web
spec:
  params:
    - name: maven
      value:
        groupId: com.example
        artifactId: springboot-initial
        version: RELEASE # latest 'RELEASE' or a specific version (e.g.: '1.2.2')
        type: jar # optional (defaults to 'jar')
        classifier: sources # optional

```

There are two ways to create a workload that defines a specific version of a Maven artifact as source in the Tanzu CLI.

The first way is to define the source through CLI flags. For example:

```
tanzu apps workload apply my-workload \
  --maven-artifact springboot-initial \
  --maven-version 2.6.0 \
  --maven-group com.example \
  --type web --app spring-boot-initial -y
```

Another flag that can be used alongside the others in this type of command is `--maven-type`, which refers to the Maven packaging type and defaults to `jar` if not specified.

The second one is through complex params (in JSON or YAML format). To specify the Maven info with this method, run:

```
tanzu apps workload apply my-workload \
  --param-yaml maven='{ "artifactId": "springboot-initial", "version": "2.6.0", "gr
oupId": "com.example"}'\
  --type web --app spring-boot-initial -y
```

To create a workload that defines the `RELEASE` version of a maven artifact as source, run:

```
tanzu apps workload apply my-workload \
  --param-yaml maven='{ "artifactId": "springboot-initial", "version": "RELEASE",
"groupId": "com.example"}'\
  --type web --app spring-boot-initial -y
```

The Maven repository URL and required credentials are defined in the supply chain, not the workload. For more information, see [Installing OOTB Basic](#).

## Maven Repository Secret

The MavenArtifact only supports authentication using basic authentication.

Additionally, MavenArtifact supports security using the TLS protocol. The Application Operator can configure the MavenArtifact to use a custom, or self-signed certificate authority (CA).

The MavenArtifact expects that all of the earlier credentials are provided in one secret, formatted as shown later:

```
---
apiVersion: v1
kind: Secret
metadata:
  name: maven-credentials
type: Opaque
data:
  username: <BASE64> # basic auth user name
  password: <BASE64> # basic auth password
  caFile: <BASE64> # PEM Encoded certificate data for custom CA
```

You cannot use the Tanzu CLI to create secrets such as this, but you can use the kubectl CLI instead.

For example:

```
kubectl create secret generic maven-credentials \
  --from-literal=username=literal-username \
  --from-file=password=/path/to/file/with/password.txt \
  --from-file=caFile=/path/to/ca-certificate.pem
```

## Use Dockerfile-based builds with Supply Chain Choreographer

This topic explains how you can use Dockerfile-based builds with Supply Chain Choreographer.

For any source-based supply chains, when you specify the new `dockerfile` parameter in a workload, the builds switch from using Kpack to using Kaniko. Source-based supply chains are supply chains that don't take a pre-built image. Kaniko is an open-source tool for building container images from a Dockerfile without running Docker inside a container.

## Use Dockerfile-based builds with Supply Chain Choreographer

| Parameter name                       | Description                                                                                  | Example                                         |
|--------------------------------------|----------------------------------------------------------------------------------------------|-------------------------------------------------|
| <code>dockerfile</code>              | relative path to the Dockerfile file in the build context                                    | <code>./Dockerfile</code>                       |
| <code>docker_build_context</code>    | relative path to the directory where the build context is                                    | <code>.</code>                                  |
| <code>docker_build_extra_args</code> | list of flags to pass directly to Kaniko (such as providing arguments, and so on to a build) | <code>- --build-arg=MY_KEY<br/>=MY_VALUE</code> |

To build a container image from the `github.com/my-foo/bar` repository where the Dockerfile resides in the root of that repository, you can switch from using Kpack to building from that Dockerfile by passing the `dockerfile` parameter:

```
$ tanzu apps workload create my-foo \
  --git-repo https://github.com/my-foo/bar \
  --git-branch dev \
  --param dockerfile=./Dockerfile \
  --type web

Create workload:
 1 + |---
 2 + |apiVersion: carto.run/v1alpha1
 3 + |kind: Workload
 4 + |metadata:
 5 + |  labels:
 6 + |    apps.tanzu.vmware.com/workload-type: web
 7 + |  name: my-foo
 8 + |  namespace: dev
 9 + |spec:
10 + |  params:
11 + |    - name: dockerfile
12 + |      value: ./Dockerfile
13 + |  source:
14 + |    git:
15 + |      ref:
16 + |        branch: dev
17 + |        url: https://github.com/my-foo/bar
```

Similarly, if the context to be used for the build must be set to a different directory within the repository, you can make use of the `docker_build_context` to change that:

```
$ tanzu apps workload create my-foo \
  --git-repo https://github.com/my-foo/bar \
  --git-branch dev \
  --param dockerfile=MyDockerfile \
  --param docker_build_context=./src
```

**Important**

This feature has no platform operator configurations to be passed through `tap-values.yaml`, but if `ootb-supply-chain-*.registry.ca_cert_data` or `shared.ca_cert_data` is configured in `tap-values`, the certificates are considered when pushing the container image.

## OpenShift

Despite that Kaniko can perform container image builds without needing either a Docker daemon or privileged containers, it does require the use of:

- Capabilities usually dropped from the more restrictive SecurityContextConstraints (SCC) enabled by default in OpenShift.
- The root user.

To overcome such limitations imposed by the default unprivileged SecurityContextConstraints (SCC), Tanzu Application Platform installs:

- `SecurityContextConstraints/ootb-templates-kaniko-restricted-v2-with-anyuid` with enough extra privileges for Kaniko to operate.
- `ClusterRole/ootb-templates-kaniko-restricted-v2-with-anyuid` to permit the use of such SCC to any actor binding to that cluster role.

Each developer namespace needs a role binding that binds the role to an actor: `ServiceAccount`. For more information, see [Set up developer namespaces to use your installed packages](#).

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: workload-kaniko-scc
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: ootb-templates-kaniko-restricted-v2-with-anyuid
subjects:
- kind: ServiceAccount
  name: default
```

With the SCC created and the ServiceAccount bound to the role that permits the use of the SCC, OpenShift accepts the pods created to run Kaniko to build the container images.

**Note**

Such restrictions are due to well-known limitations in how Kaniko performs the image builds, and there is currently no solution. For more information, see [kaniko#105](#).

## Tanzu Build Service integration for Supply Chain Choreographer

This topic describes how you can configure and use the Tanzu Build Service integration for Supply Chain Choreographer.



By default, the Out of the Box supply chains (`ootb-supply-chain-*`) in Tanzu Application Platform make use of Tanzu Build Service for building container images out of source code.

You can configure a **platform operator** by using `tap-values.yaml`:

1. The default container image registry where application images must be pushed:

```
ootb_supply_chain_basic:
  registry:
    server: <>
    repository: <>
```

2. The name of the Kpack `ClusterBuilder` used by default:

```
ootb_supply_chain_basic:
  cluster_builder: my-custom-cluster-builder
```

You can configure an **application operator** by using `Workload`:

- `spec.build.env` are the environment variables used during the build:

```
kind: Workload
apiVersion: carto.run/v1alpha1
metadata:
  name: tanzu-java-web-app
spec:
  # ...
  build:
    env:
      - name: PORT
        value: "8080"
      - name: CA_CERTIFICATE
        valueFrom:
          secretKeyRef:
            name: secret-in-the-same-namespace-as-workload
            key: crt.pem
```

- `spec.params.clusterBuilder` is the name of the ClusterBuilder to use for builds of that Workload:

```
kind: Workload
apiVersion: carto.run/v1alpha1
metadata:
  name: tanzu-java-web-app
spec:
  # ...
  params:
    - name: clusterBuilder
      value: nodejs-cluster-builder
```

- `spec.params.buildServiceBindings` is the object carrying the definition of a list of service bindings to use at build time:

```
---
kind: Workload
apiVersion: carto.run/v1alpha1
metadata:
  name: tanzu-java-web-app
spec:
  # ...
  params:
    - name: buildServiceBindings
      value:
```

```

- name: settings-xml
  kind: Secret
  apiVersion: v1
---
apiVersion: v1
kind: Secret
metadata:
  name: settings-xml
  type: service.binding/maven
stringData:
  type: maven
  provider: sample
  settings.xml: <settings>...</settings>

```



### Note

See the Kpack [ServiceBinding](#) documentation in GitHub for more details about build-time service bindings.

these configuration only take effect when Kpack is used for building a container image. If you use Dockerfile-based builds by leveraging the `dockerfile` parameter, see [dockerfile-based builds](#) for more information.

## Use an existing image with Supply Chain Choreographer

This topic describes how you can use an existing image with Supply Chain Choreographer.

For apps that build container images in a predefined way, the supply chains in the Out of the Box packages enable you to specify a prebuilt image. This uses the same stages as any other workload.

## Requirements for prebuilt images

Supply chains aim at Knative as the runtime for the container image you provide. Your app must adhere to the following Knative standards:

- **Container port listens on port 8080**

The Knative service is created with the container port set to `8080` in the pod template spec. Therefore, your container image must have a socket listening on `8080`.

```

ports:
- containerPort: 8080
  name: user-port
  protocol: TCP

```

- **Non-privileged user ID**

By default, the container initiated as part of the pod is run as user 1000.

```

securityContext:
  runAsUser: 1000

```

- **Arguments other than the image's default ENTRYPOINT**

In most cases the container image runs using the `ENTRYPOINT` it was configured with. In the case of Dockerfiles, the combination of `ENTRYPOINT` and `CMD`.

If you need extra configuration for your image, use `--env` flags with the `tanzu apps workload create` command or modify `spec.env` in your `workload.yaml` file.

- **Credentials for pulling the container image at runtime**

The image you provide is not relocated to an internal container image registry. Any components associated with the image must have the necessary credentials to pull it. For the service accounts used for the workload and deliverable, you must attach a secret that contains the credentials to pull the container image.

If the image is hosted in a registry that has certificates signed by a private certificate authority, the components of the supply chains, delivery, and the Kubernetes nodes in the run cluster must trust the certificate.

## Configure your workload to use a prebuilt image

To select a prebuilt image, set the `spec.image` field in your `workload.yaml` file with the name of the container image that contains the app to deploy by running:

```
tanzu apps workload create WORKLOAD-NAME \
  --app APP-NAME \
  --type TYPE \
  --image IMAGE
```

Where:

- `WORKLOAD-NAME` is the name you choose for your workload.
- `APP-NAME` is the name of your app.
- `TYPE` is the type of your app.
- `IMAGE` is the container image that contains the app you want to deploy.

For example, if you have an image named `IMAGE`, you can create a workload with the flag mentioned earlier:

```
tanzu apps workload create tanzu-java-web-app \
  --app tanzu-java-web-app \
  --type web \
  --image IMAGE
```

Expected output:

```
Create workload:
 1 + |---
 2 + |apiVersion: carto.run/v1alpha1
 3 + |kind: Workload
 4 + |metadata:
 5 + |  labels:
 6 + |    app.kubernetes.io/part-of: hello-world
 7 + |    apps.tanzu.vmware.com/workload-type: web
 8 + |  name: tanzu-java-web-app
 9 + |  namespace: default
10 + |spec:
11 + |  image: IMAGE
```

When you run `tanzu apps workload create` command with the `--image` field, the source resolution and build phases of the supply chain are skipped.

## Examples

The following examples show ways that you can build container images for a Java-based app and complete the supply chains to a running service.

## Using a Dockerfile

Using a Dockerfile is the most common way of building container images. You can select a base image, on top of which certain operations must occur, such as compiling code, and mutate the contents of the file system to a final container image that has a build of your app and any required runtime dependencies.

Here you use the `maven` base image for compiling your app code, and then the minimal distroless `java17-debian11` image for providing a JRE that can run your app when it is built.

After building the image, you push it to a container image registry, and then reference it in the workload.

1. Create a Dockerfile that describes how to build your app and make it available as a container image:

```
ARG BUILDER_IMAGE=maven
ARG RUNTIME_IMAGE=gcr.io/distroless/java17-debian11

FROM $BUILDER_IMAGE AS build

    ADD . .
    RUN unset MAVEN_CONFIG && ./mvnw clean package -B -DskipTests

FROM $RUNTIME_IMAGE AS runtime

    COPY --from=build /target/demo-0.0.1-SNAPSHOT.jar /demo.jar
    CMD [ "/demo.jar" ]
```

2. Push the container image to a container image registry by running:

```
docker build -t IMAGE .
docker push IMAGE
```

3. Create a workload by running:

```
tanzu apps workload create tanzu-java-web-app \
  --type web \
  --app tanzu-java-web-app \
  --image IMAGE
```

Expected output:

```
Create workload:
 1 + |---
 2 + |apiVersion: carto.run/v1alpha1
 3 + |kind: Workload
 4 + |metadata:
 5 + |  labels:
 6 + |    app.kubernetes.io/part-of: hello-world
 7 + |    apps.tanzu.vmware.com/workload-type: web
 8 + |  name: tanzu-java-web-app
 9 + |  namespace: default
10 + |spec:
11 + |  image: IMAGE
```

4. Run the following workload:

```
tanzu apps workload get tanzu-java-web-app
```

Expected output:

```
# tanzu-java-web-app: Ready
---
lastTransitionTime: "2022-04-06T19:32:46Z"
message: ""
reason: Ready
status: "True"
type: Ready

Workload pods
NAME                                STATUS    RESTARTS   A
GE
tanzu-java-web-app-00001-deployment-7d7df5ccf5-k58rt  Running    0           3
2s
tanzu-java-web-app-config-writer-xjmvw-pod            Succeeded  0           8
9s

Workload Knative Services
NAME          READY    URL
tanzu-java-web-app  Ready    http://tanzu-java-web-app.default.example.com
```

## Using Spring Boot's `build-image` Maven target

You can use Spring Boot's `build-image` target to build a container image that runs your app. The `build-image` target must use a Dockerfile.

For example, using the same sample repository as mentioned before (<https://github.com/vmware-tanzu/application-accelerator-samples/tree/main/tanzu-java-web-app>):

1. Build the image by running the following command from the root of the repository:

```
IMAGE=ghcr.io/kontinue/hello-world:tanzu-java-web-app
./mvnw spring-boot:build-image -Dspring-boot.build-image.imageName=$IMAGE
```

Expected output:

```
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.example:demo >-----
[INFO] Building demo 0.0.1-SNAPSHOT
[INFO] -----[ jar ]-----
[INFO]
...
[INFO] 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: sha256:7140722ea396af69fb3d0ad12e9b4419bc3e67d9c5d8
a2f6a1421decc4828ace size: 4497
```

After you push the container image, you see the same results as building the image [using a Dockerfile](#).

For more information about building container images for a Spring Boot app, see [Spring Boot with Docker](#)

## About Out of the Box Supply Chains

In Tanzu Application Platform, the `ootb-supply-chain-basic`, `ootb-supply-chain-testing`, and `ootb-supply-chain-testing-scanning` packages each receive a new supply chain that provides a prebuilt container image for your app.

```
ootb-supply-chain-basic

(cluster)  basic-image-to-url  ClusterSupplyChain  (!) new
^          source-to-url     ClusterSupplyChain

ootb-supply-chain-testing

(cluster)  testing-image-to-url  ClusterSupplyChain  (!) new
^          source-test-to-url  ClusterSupplyChain

ootb-supply-chain-testing-scanning

(cluster)  scanning-image-scan-to-url  ClusterSupplyChain  (!) new
^          source-test-scan-to-url  ClusterSupplyChain
```

To leverage the supply chains that expect a prebuilt image, you must set the `spec.image` field in the workload to the name of the container image that contains the app to deploy.

The new supply chains use a Cartographer feature that lets VMware increase the specificity of supply chain selection by using the `matchFields` selector rule.

The selection takes place as follows:

- `ootb-supply-chain-basic`
  - From source: label `apps.tanzu.vmware.com/workload-type: web`
  - Prebuilt image: label `apps.tanzu.vmware.com/workload-type: web` and set `spec.image` in the `workload.yaml`
- `ootb-supply-chain-testing`
  - From source: labels `apps.tanzu.vmware.com/workload-type: web` and `apps.tanzu.vmware.com/has-tests: true`
  - Prebuilt image: label `apps.tanzu.vmware.com/workload-type: web` and set `spec.image` in the `workload.yaml`
- `ootb-supply-chain-testing-scanning`
  - From source: labels `apps.tanzu.vmware.com/workload-type: web` and `apps.tanzu.vmware.com/has-tests: true`

- o 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.



### Important

To use the pull request feature, you must use HTTP(S) authentication with an access token.

1. To provide the credentials for any Git operations with SSH, create the Kubernetes secret as follows:

```

apiVersion: v1
kind: Secret
metadata:
  name: GIT-SECRET-NAME
  annotations:
    tekton.dev/git-0: GIT-SERVER
type: kubernetes.io/ssh-auth
stringData:
  ssh-privatekey: SSH-PRIVATE-KEY # private key with push-permissions
  identity: SSH-PRIVATE-KEY # private key with pull permissions
  identity.pub: SSH-PUBLIC-KEY # public of the `identity` private key
  known_hosts: GIT-SERVER-PUBLIC-KEYS # Git server public keys

```

2. Generate a new SSH keypair: `identity` and `identity.pub`.

```
ssh-keygen -t ecdsa -b 521 -C "" -f "identity" -N ""
```

3. Go to your Git provider and add the `identity.pub` as a deployment key for the repository of interest or add to an account that has access to it. For example, for GitHub, visit <https://github.com/<repository>/settings/keys/new>.



#### Note

Keys of type SHA-1/RSA are recently deprecated by GitHub.

4. Gather public keys from the provider, for example, GitHub:

```
ssh-keyscan github.com > ./known_hosts
```

5. Create the Kubernetes secret by using the contents of the files in the first step:

```
apiVersion: v1
kind: Secret
metadata:
  name: GIT-SECRET-NAME
  annotations: {tekton.dev/git-0: GIT-SERVER}
type: kubernetes.io/ssh-auth
stringData:
  ssh-privatekey: SSH-PRIVATE-KEY
  identity: SSH-PRIVATE-KEY
  identity.pub: SSH-PUBLIC-KEY
  known_hosts: GIT-SERVER-PUBLIC-KEYS
```

For example, edit the credentials:

```
apiVersion: v1
kind: Secret
metadata:
  name: git-ssh
  annotations: {tekton.dev/git-0: github.com}
type: kubernetes.io/ssh-auth
stringData:
  ssh-privatekey: |
    -----BEGIN OPENSSH PRIVATE KEY-----
    AAAA
    ....
    ....
    -----END OPENSSH PRIVATE KEY-----
  known_hosts: |
    <known hosts entrys for git provider>
  identity: |
    -----BEGIN OPENSSH PRIVATE KEY-----
    AAAA
    ....
    ....
    -----END OPENSSH PRIVATE KEY-----
  identity.pub: ssh-ed25519 AAAABBBCCCCDDDeeeeFFFFF user@example.com
```

6. After you create the secret, attach it to the ServiceAccount configured for the workload by including it in its set of secrets. For example:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: default
secrets:
```

```

- name: registry-credentials
- name: tap-registry
- name: GIT-SECRET-NAME
imagePullSecrets:
- name: registry-credentials
- name: tap-registry

```

## Read more on Git

For information about Git, see [Git Reference](#).

## Author your supply chains

The Out of the Box Supply Chain, Delivery Basic, and Templates Supply Chain Choreographer packages give you Kubernetes objects that cover a reference path to production. Because VMware recognizes that you have your own needs, these objects are customizable, including individual templates for each resource, whole supply chains, or delivery objects.

Depending on how you installed Tanzu Application Platform, there are different ways to customize the Out of the Box Supply Chains. The following sections describe the ways supply chains and templates are authored within the context of profile-based Tanzu Application Platform installations.

## Providing your own supply chain

To create a new supply chain and make it available for workloads, ensure that the supply chain does not conflict with those installed on the cluster, as those objects are cluster-scoped.

If this is your first time creating a supply chain, follow the tutorials from the [Cartographer documentation](#).

Any supply chain installed in a Tanzu Application Platform cluster might encounter two possible cases of collisions:

- **object name:** Supply chains are cluster scoped, such as any Cartographer resource prefixed with `cluster`. So the name of the custom supply chain must be different from the ones provided by the Out of the Box packages.

Either create a supply chain whose name is different, or remove the installation of the corresponding `ootb-supply-chain-*` from the Tanzu Application Platform.

- **workload selection:** A workload is reconciled against a particular supply chain based on a set of selection rules as defined by the supply chains. If the rules for the supply chain to match a workload are ambiguous, the workload does not make any progress.

Either create a supply chain whose selection rules are different from the ones the Out of the Box Supply Chain packages use, or remove the installation of the corresponding `ootb-supply-chain-*` from Tanzu Application Platform.

See [Selectors](#).

For Tanzu Application Platform 1.2, the following selection rules are in place for the supply chains of the corresponding packages:

- `ootb-supply-chain-basic`
  - ClusterSupplyChain/**basic-image-to-url**
    - label `apps.tanzu.vmware.com/workload-type: web`
    - `workload.spec.image` text box set
  - ClusterSupplyChain/**source-to-url**

- label `apps.tanzu.vmware.com/workload-type: web`
- *ootb-supply-chain-testing*
  - ClusterSupplyChain/**testing-image-to-url**
    - label `apps.tanzu.vmware.com/workload-type: web`
    - `workload.spec.image` text box set
  - ClusterSupplyChain/**source-test-to-url**
    - label `apps.tanzu.vmware.com/workload-type: web`
    - label `apps.tanzu.vmware.com/has-test: true`
- *ootb-supply-chain-testing-scanning*
  - ClusterSupplyChain/**scanning-image-scan-to-url**
    - label `apps.tanzu.vmware.com/workload-type: web`
    - `workload.spec.image` text box set
  - ClusterSupplyChain/**source-test-scan-to-url**
    - label `apps.tanzu.vmware.com/workload-type: web`
    - label `apps.tanzu.vmware.com/has-test: true`

For details about how to edit an existing supply chain, see [Modifying an Out of the Box Supply Chain](#) section.

You can exclude a supply chain package from the installation to prevent the conflicts mentioned earlier, by using the `excluded_packages` property in `tap-values.yaml`. For example:

```
# add to 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.

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**
- ClusterTask/**kaniko-build**
- ClusterImageTemplate/**kaniko-template**
- ClusterRole/**ootb-templates-app-viewer**

- ClusterRole/**ootb-templates-deliverable**
- ClusterRole/**ootb-templates-workload**
- ClusterRunTemplate/**tekton-source-pipelinerun**
- ClusterRunTemplate/**tekton-taskrun**
- ClusterSourceTemplate/**delivery-source-template**
- ClusterSourceTemplate/**source-scanner-template**
- ClusterSourceTemplate/**source-template**
- ClusterSourceTemplate/**testing-pipeline**
- ClusterTask/**git-writer**
- ClusterTask/**image-writer**
- ClusterTemplate/**config-writer-template**
- ClusterTemplate/**deliverable-template**

Before submitting your own, either ensure that the name and resource has no conflicts with those installed by `ootb-templates`, or exclude from the installation the template you want to override by using the `excluded_templates` property of `ootb-templates`.

For example, perhaps you want to override the `ClusterConfigTemplate` named `config-template` to provide your own with the same name, so that you don't need to edit the supply chain. In `tap-values.yaml`, you can exclude template provided by Tanzu Application Platform:

```
ootb_templates:
  excluded_templates:
    - 'config-writer'
```

For details about how to edit an existing template, see [Modifying an Out of the Box Supply template](#) section.

## Modifying an Out of the Box Supply Chain

To change the shape of a supply chain or the template that it points to, do the following:

1. Copy one of the reference supply chains.
2. Remove the old supply chain. See [preventing Tanzu Application Platform supply chains from being installed](#).
3. Edit the supply chain object.
4. Submit the modified supply chain to the cluster.

### Example

In this example, you have a new `ClusterImageTemplate` object named `foo` that you want use for building container images instead of the out of the box object that makes use of Kpack. The supply chain that you want to apply the modification to is `source-to-url` provided by the `ootb-supply-chain-basic` package.

1. Find the image that contains the supply chain definition:

```
kubectl get app ootb-supply-chain-basic \
  -n tap-install \
  -o jsonpath={.spec.fetch[0].imgpkgBundle.image}
```

```
registry.tanzu.vmware.com/tanzu-application-platform/tap-packages@sha256: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 supply chain that you want to exchange the template with another:

```
--- a/supply-chain.yaml
+++ b/supply-chain.yaml
@@ -52,7 +52,7 @@ spec:
  - name: image-builder
    templateRef:
      kind: ClusterImageTemplate
  - name: kpack-template
+ name: foo
  params:
    - name: serviceAccount
      value: #@ data.values.service_account
```

5. Submit the supply chain to Kubernetes:

The supply chain definition found in the bundle expects the values you provided by using `tap-values.yaml` to be interpolated by using YTT before they are submitted to Kubernetes. So before applying the modified supply chain to the cluster, use YTT to interpolate those values. After that, run:

```
ytt \
  --ignore-unknown-comments \
  --file ./ootb-supply-chain-basic/config \
  --data-value registry.server=REGISTRY-SERVER \
  --data-value registry.repository=REGISTRY-REPOSITORY |
  kubectl apply -f-
```



### Important

The modified supply chain does not outlive the destruction of the cluster. VMware recommends that you save it, for example, in a Git repository to

install on every cluster where you expect the supply chain to exist.

## Modifying an Out of the Box Supply template

The Out of the Box Templates package (`ootb-templates`) includes all of the templates and shared Tekton tasks used by the supply chains shipped by using `ootb-supply-chain-*` packages. Any template that you want to edit, for example, to change details about the resources that are created based on them, is part of this package.

The workflow for updating a template is as follows:

1. Copy one of the reference templates from `ootb-templates`.
2. Exclude that template from the set of objects provided by `ootb-templates`. For more information, see `excluded_templates` in [Providing your Own Templates](#).
3. Edit the template.
4. Submit the modified template to the cluster.



### Note

You don't need to change anything related to supply chains, because you're preserving the name of the object referenced by the supply chain.

## Example

In this example, you want to update the `ClusterImageTemplate` object called `kpack-template`, which provides a template for creating `kpack/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
```

- Confirm that you downloaded all the templates:

```
tree ./ootb-templates
```

```
./ootb-templates
├── config
│   ├── cluster-roles.yaml
│   ├── config-template.yaml
│   ├── kpack-template.yaml          # ! the one we want to modify
│   └── ...
├── testing-pipeline.yaml
└── values.yaml
```

- 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"):
```

- Submit the template.

The name of the template is preserved but the contents are changed. So after the template is submitted, the supply chains are all embedded to the build of the application container images that have `FOO` environment variable.

## Live modification of supply chains and templates

Preceding sections covered how to update supply chains or templates installed in a cluster. This section shows how you can experiment by making small changes in a live setup with `kubectl edit`.

When you install Tanzu Application Platform by using profiles, a `PackageInstall` object is created. This in turn creates a set of children `PackageInstall` objects for installing the individual components that make up the platform.

```
PackageInstall/tap
├── App/tap
│   ├── PackageInstall/cert-manager
│   ├── PackageInstall/cartographer
│   ├── ...
└── PackageInstall/tekton-pipelines
```

Because the installation is based on Kubernetes primitives, `PackageInstall` tries to achieve the state where all packages are installed.

This is great but presents challenges for modifying the contents of some of the objects that the installation submits to the cluster. Namely, such modifications cause the original definition persisting instead of the changes.

For this reason, before you perform any customization to the Out of the Box packages, you must pause the top-level `PackageInstall/tap` object. Run:



```
kubectl edit -n tap-install packageinstall tap
```

```
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: tap
  namespace: tap-install
spec:
  paused: true # ! set this field to `paused: true`.
  packageRef:
    refName: tap.tanzu.vmware.com
    versionSelection:
# ...
```

With the installation of Tanzu Application Platform paused, all of the installed components are still there, but changes to those children `PackageInstall` objects are not overwritten.

Now you can pause the `PackageInstall` objects that relate to the templates or supply chains you want to edit.

For example:

- To edit templates: `kubectl edit -n tap-install packageinstall ootb-templates`
- To edit the basic supply chains: `kubectl edit -n tap-install packageinstall ootb-supply-chain-basic`

setting `packageinstall.spec.paused: true`.

With the installations paused, further live changes to templates or supply chains are persisted until you revert the `PackageInstalls` to not being paused. To persist the changes, follow the steps outlined in the earlier sections.

## Adding custom behavior to Supply Chains

Most behaviors in supply chains are supplied by Kubernetes controllers. For example, Cloud Native Buildpacks are created by the `kpack` controller when a `kpack Image` object is created. Sometimes there is need for behavior and no controller for it exists. In these instances, you might want to write a script that uses a CLI tool, or interact with an external API. To do this, you can bring the behavior to the supply chain by using Tekton.

You can look at the `kaniko image-building` as an example. You create a Tekton ClusterTask `kaniko-build` with instructions for how to build a Docker image using `kaniko` given a set of parameters. The ClusterTask has a set of steps. Each step refers to a container image and a set of instructions to run on the image. For example, it can be a Linux image against which a set of `bash` instructions are run. The ClusterTask is installed on the cluster.

You create the ClusterImageTemplate `kaniko-template` to create Tekton TaskRuns. TaskRuns are immutable, so you add the `lifecycle: tekton` field to the template's specifications. This ensures two things:

- When inputs to the template change, rather than updating the TaskRun, a new TaskRun is created.
- Only the values from the most recently created TaskRun that is successful are propagated forward in the supply chain.

To learn more about the `lifecycle: tekton` field, see the Cartographer tutorial [Lifecycle: Templating Objects That Cannot Update](#). To learn more about Tekton, see the [Tekton documentation](#).

## Reference guides for Supply Chain Choreographer for Tanzu

This topic describes the reference guides you can use for Supply Chain Choreographer for Tanzu.

### Reference guides

The following reference guides apply to Supply Chain Choreographer for Tanzu:

- [Tanzu Build Service Integration](#)

### Events reference for Supply Chain Choreographer

This topic describes each event you can view with Supply Chain Choreographer.

Events are emitted when Choreographer edits resources or notices a change in their output or healthy state. Don't treat events like logs, however they can offer valuable insight into what's happening in a supply chain over time. For example, very high occurrences of events in a short period of time might be a sign of resources thrashing.

Events are published on Workload, Deliverable, and Runnable resources. You can view them manually using:

```
kubectl describe workload.carto.run <workload-name> -n <workload-ns>
kubectl describe runnable.carto.run <runnable-name> -n <runnable-ns>
kubectl describe deliverable.carto.run <deliverable-name> -n <deliverable-ns>
```

### Events

The following sections define the different events.

#### StampedObjectApplied

This event is emitted every time Choreographer creates or updates a resource. The created or updated resource is referenced in the event message.

Example messages:

```
Created object [gitrepositories.source.toolkit.fluxcd.io/my-project]
Updated object [apps.kappctrl.k14s.io/my-project-app]
```

#### StampedObjectRemoved

This event is emitted every time Choreographer deletes a resource. This currently only occurs when Runnable resources expire. The deleted object is referenced in the event message.

Example message:

```
Deleted object [task.tekton.dev/my-project-a737bdf]
```

#### ResourceOutputChanged

This event is emitted every time Choreographer recognizes a new output from a resource.

Example message:

```
[source-provider] found a new output in [imagerepositories.source.apps.tanzu.vmware.co
```

```
m/app]
```

## ResourceHealthyStatusChanged

This event is emitted every time Choreographer recognizes that the healthy status of a resource has changed.

Example message:

```
[image-provider] found healthy status in [images.kpack.io/app] changed to [True]
[source-provider] found healthy status in [[gitrepositories.source.toolkit.fluxcd.io/my-project]] changed to [False]
```

## Workload Reference for Supply Chain Choreographer

This topic describes the fields you can use for Supply Chain Choreographer workloads.

This topic describes available fields for Supply Chain Choreographer workloads.

## Standard Fields

Cartographer workloads have standard fields leveraged by supply chains. See [Cartographer's Reference Documentation](#) in the Cartographer documentation.

## Labels

Workload labels affect which supply chain is selected. For information about which template is defined for a particular reference, see [Selectors](#) in the Cartographer documentation. Individual templates can also use workload labels.

The following are workload label keys whose values change the behavior of OOTB Supply Chains:

- [apps.tanzu.vmware.com/has-tests](#) by [Source-Test-to-URL](#) and [Source-Test-Scan-to-URL](#).
- [apps.tanzu.vmware.com/workload-type](#) by [all supply chains](#).
- [apis.apps.tanzu.vmware.com/register-api](#) by the [Api-Descriptors Template](#).

## Parameters

The OOTB templates are configured with parameters from the supply chain or workload. For information about Cartographer parameters, including precedence rules, see [Parameters](#) in the Cartographer documentation.

What parameters are relevant depends on the supply chain that selects the workload, for two reasons:

1. The OOTB supply chains refer to overlapping sets of templates. A workload selected by the Source-to-URL supply chain can provide a `scanning_image_template` parameter, but the supply chain does not refer to a template that leverages that parameter.
2. You can write Supply Chains to provide a parameter value to a template and prevent the workload from overriding the value. See [Further Information](#) in the Cartographer documentation.

The following list of parameters are respected by some OOTB supply chains. Each provides the templates that respect the parameter. The reference for the template details which supply chains include the template.

- gitImplementation: [source-template](#)

- gitops\_ssh\_secret: source-template, deliverable-template, external-deliverable-template
- serviceAccount: source-template, image-provider-template, kpack-template, kaniko-template, convention-template, config-writer-template, config-writer-and-pull-requester-template, deliverable-template, external-deliverable-template
- maven: source-template
- testing\_pipeline\_matching\_labels: testing-pipeline
- testing\_pipeline\_params: testing-pipeline
- scanning\_source\_template: source-scanner-template
- scanning\_source\_policy: source-scanner-template
- clusterBuilder: kpack-template
- buildServiceBindings: kpack-template
- live-update: kpack-template, convention-template
- dockerfile: kaniko-template
- docker\_build\_context: kaniko-template
- docker\_build\_extra\_args: kaniko-template
- scanning\_image\_template: image-scanner-template
- scanning\_image\_policy: image-scanner-template
- annotations: convention-template, service-bindings, api-descriptors
- debug: convention-template
- ports: server-template
- api-descriptors: api-descriptors
- gitops\_branch: config-writer-template, config-writer-and-pull-requester-template, deliverable-template, external-deliverable-template
- gitops\_user\_name: config-writer-template, config-writer-and-pull-requester-template
- gitops\_user\_email: config-writer-template, config-writer-and-pull-requester-template
- gitops\_commit\_message: config-writer-template, config-writer-and-pull-requester-template
- gitops\_repository: config-writer-template, deliverable-template, external-deliverable-template
- gitops\_repository\_prefix: config-writer-template, deliverable-template, external-deliverable-template
- gitops\_server\_address: config-writer-template, config-writer-and-pull-requester-template, deliverable-template, external-deliverable-template
- gitops\_repository\_owner: config-writer-template, config-writer-and-pull-requester-template, deliverable-template, external-deliverable-template
- gitops\_repository\_name: config-writer-template, config-writer-and-pull-requester-template, deliverable-template, external-deliverable-template
- gitops\_commit\_branch: config-writer-and-pull-requester-template
- gitops\_pull\_request\_title: config-writer-and-pull-requester-template
- gitops\_pull\_request\_body: config-writer-and-pull-requester-template

- `gitops_server_kind`: [config-writer-and-pull-requester-template](#)

## Service Account

To create the templated objects, Cartographer needs a reference to a service account with permissions to manage resources. This service account might be provided in the workload's `.spec.serviceAccountName` field or in the supply chain's `spec.serviceAccountRef` field. See [Service Account](#) and [Workload and Supply Chain Custom Resources](#) in the Cartographer documentation. When using the Tanzu CLI to create a workload, specify this service account's name with the `--service-account` flag.

After the templated objects are created, they often need a service account with permissions to do work. In the OOTB Templates and Supply Chains, the parameter `serviceAccount` must reference the service account for these objects. When using the Tanzu CLI to create a workload, specify this service account's name with `--param serviceAccount=...`

## Supply chains

Tanzu Application Platform includes a number of supply chains packages, each of which installs two [ClusterSupplyChains](#). You can only install one supply chain package at a time.

The supply chains provide some [parameters](#) to the referenced templates. The parameters provided by the workload might override the parameters in this topic.

## Source-to-URL

### Purpose

- Fetches application source code
- Builds it into an image
- Writes the Kubernetes configuration necessary to deploy the application
- Commits that configuration to either a Git repository or a container image registry

### Resources

This section describes the templates and their parameters.

#### source-provider

Refers to [source-template](#).

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `gitImplementation` from tap-value `git_implementation`. NOT overridable by workload.

#### image-provider

Refers to [kaniko-template](#) when the workload provides a parameter `dockerfile`. Refers to [kpack-template](#) otherwise.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `registry` from tap-value `registry`. NOT overridable by workload.

- `clusterBuilder` from tap-value `cluster_builder`. Overridable by workload.
- `dockerfile` value `./Dockerfile`. Overridable by workload.
- `docker_build_context` value `./`. Overridable by workload.
- `docker_build_extra_args` value `[]`. Overridable by workload.

### Common resources

- [Config-Provider](#)
- [App-Config](#)
- [Service-Bindings](#)
- [Api-Descriptors](#)
- [Config-Writer](#)
- [Deliverable](#)

### Parameters provided to all resources

- `maven_repository_url` from tap-value `maven.repository.url`. NOT overridable by workload.
- `maven_repository_secret_name` from tap-value `maven.repository.secret_name`. NOT overridable by workload.
- See [Params provided by all Supply Chains to all Resources](#)

### Package

[Out of the Box Supply Chain Basic](#)

### More information

See [Install Out of the Box Supply Chain Basic](#) for information about setting tap-values at installation time.

### Source-Test-to-URL

- Fetches application source code
- Runs user defined tests against the code
- Builds the code into an image
- Writes the Kubernetes configuration necessary to deploy the application
- Commits that configuration to either a Git repository or a container image registry

### Resources

#### source-provider

Refers to [source-template](#).

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `gitImplementation` from tap-value `git_implementation`. NOT overridable by workload.

#### source-tester

Refers to [testing-pipeline](#).

No parameters are provided by the supply-chain.

### image-provider

Refers to [kaniko-template](#) when the workload provides a parameter `dockerfile`. Refers to [kpack-template](#) otherwise.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `registry` from tap-value `registry`. NOT overridable by workload.
- `clusterBuilder` from tap-value `cluster_builder`. Overridable by workload.
- `dockerfile` value `./Dockerfile`. Overridable by workload.
- `docker_build_context` value `./`. Overridable by workload.
- `docker_build_extra_args` value `[]`. Overridable by workload.

### Common resources

- [Config-Provider](#)
- [App-Config](#)
- [Service-Bindings](#)
- [Api-Descriptors](#)
- [Config-Writer](#)
- [Deliverable](#)

### Parameters provided to all resources

- `maven_repository_url` from tap-value `maven.repository.url`. NOT overridable by workload.
- `maven_repository_secret_name` from tap-value `maven.repository.secret_name`. NOT overridable by workload.
- See [Params provided by all Supply Chains to all Resources](#).

### Package

[Out of the Box Supply Chain Testing](#)

### More information

See [Install Out of the Box Supply Chain with Testing](#) for information about setting tap-values at installation time.

### Source-Test-Scan-to-URL

- Fetches application source code
- Runs user defined tests against the code
- Scans the code for vulnerabilities
- Builds the code into an image
- Scans the image for vulnerabilities

- Writes the Kubernetes configuration necessary to deploy the application
- Commits that configuration to either a Git repository or an image registry

## Resources

### source-provider

Refers to [source-template](#).

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `gitImplementation` from tap-value `git_implementation`. NOT overridable by workload.

### source-tester

Refers to [testing-pipeline](#).

No parameters are provided by the supply-chain.

### source-scanner

Refers to [source-scanner-template](#).

Parameters provided:

- `scanning_source_policy` from tap-value `scanning.source.policy`. Overridable by workload.
- `scanning_source_template` from tap-value `scanning.source.template`. Overridable by workload.

### image-provider

Refers to [kaniko-template](#) when the workload provides a parameter `dockerfile`. Refers to [kpack-template](#) otherwise.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `registry` from tap-value `registry`. NOT overridable by workload.
- `clusterBuilder` from tap-value `cluster_builder`. Overridable by workload.
- `dockerfile` value `./Dockerfile`. Overridable by workload.
- `docker_build_context` value `./`. Overridable by workload.
- `docker_build_extra_args` value `[]`. Overridable by workload.

### image-scanner

Refers to [image-scanner-template](#).

Parameters provided:

- `scanning_image_policy` from tap-value `scanning.image.policy`. Overridable by workload.
- `scanning_image_template` from tap-value `scanning.image.template`. Overridable by workload.

### Common resources



- [Config-Provider](#)
- [App-Config](#)
- [Service-Bindings](#)
- [Api-Descriptors](#)
- [Config-Writer](#)
- [Deliverable](#)

## Parameters provided to all resources

- `maven_repository_url` from tap-value `maven.repository.url`. NOT overridable by workload.
- `maven_repository_secret_name` from tap-value `maven.repository.secret_name`. NOT overridable by workload.
- See [Params provided by all Supply Chains to all Resources](#)

## Package

[Out of the Box Supply Chain Testing Scanning](#)

## More information

See [Install Out of the Box Supply Chain with Testing and Scanning](#) for information about setting tap-values at installation time.

## Basic-Image-to-URL

- Fetches a prebuilt image.
- Writes the Kubernetes configuration necessary to deploy the application.
- Commits that configuration to either a Git repository or an image registry.

## Resources

### image-provider

Refers to [image-provider-template](#).

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.

### Common resources

- [Config-Provider](#)
- [App-Config](#)
- [Service-Bindings](#)
- [Api-Descriptors](#)
- [Config-Writer](#)
- [Deliverable](#)

## Parameters provided to all resources

- See [Params provided by all Supply Chains to all Resources](#)

## Package

[Out of the Box Supply Chain Basic](#)

## More information

See [Install Out of the Box Supply Chain Basic](#) for information about setting tap-values at installation time.

## Testing-Image-to-URL

- Fetches a prebuilt image.
- Writes the Kubernetes configuration necessary to deploy the application.
- Commits that configuration to either a Git repository or an image registry.

## Resources

### image-provider

Refers to [image-provider-template](#).

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.

### Common resources

- [Config-Provider](#)
- [App-Config](#)
- [Service-Bindings](#)
- [Api-Descriptors](#)
- [Config-Writer](#)
- [Deliverable](#)

## Parameters provided to all resources

- See [Params provided by all Supply Chains to all Resources](#)

## Package

[Out of the Box Supply Chain Testing](#)

## More information

See [Install Out of the Box Supply Chain with Testing](#) for information about setting tap-values at installation time.

## Scanning-image-scan-to-URL

- Fetches a prebuilt image.
- Scans the image for vulnerabilities.
- Writes the Kubernetes configuration necessary to deploy the application.
- Commits the configuration to either a Git repository or an image registry.

## Resources

### image-provider

Refers to [image-provider-template](#).

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.

### image-scanner

Refers to [image-scanner-template](#).

Parameters provided:

- `scanning_image_policy` from tap-value `scanning.image.policy`. Overridable by workload.
- `scanning_image_template` from tap-value `scanning.image.template`. Overridable by workload.

### Common resources

- [Config-Provider](#)
- [App-Config](#)
- [Service-Bindings](#)
- [Api-Descriptors](#)
- [Config-Writer](#)
- [Deliverable](#)

## Parameters provided to all resources

- See [Params provided by all Supply Chains to all Resources](#)

## Package

[Out of the Box Supply Chain Testing Scanning](#)

## More information

See [Install Out of the Box Supply Chain with Testing and Scanning](#) for information about setting tap-values at installation time.

## Source-to-URL-Package (experimental)

### Purpose

- Fetches the application source code.
- Builds the source code into an image.
- Bundles the Kubernetes configuration necessary to deploy the application into a Carvel Package.
- Commits the Package to a Git Repository.

## Resources

This section describes the templates and their parameters.

### source-provider

Refers to [source-template](#).

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `gitImplementation` from tap-value `git_implementation`. NOT overridable by workload.

### image-provider

Refers to [kaniko-template](#) when the workload provides a parameter `dockerfile`. Refers to [kpack-template](#) otherwise.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `registry` from tap-value `registry`. NOT overridable by workload.
- `clusterBuilder` from tap-value `cluster_builder`. Overridable by workload.
- `dockerfile` value `./Dockerfile`. Overridable by workload.
- `docker_build_context` value `./`. Overridable by workload.
- `docker_build_extra_args` value `[]`. Overridable by workload.

### carvel-package

Refers to [carvel-package](#).

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `registry` from tap-value `registry`. NOT overridable by workload.

### package-config-writer

Refers to the [package-config-writer-and-pull-requester-template](#) when the tap-value `gitops.commit_strategy` is `pull_request`. Otherwise, this resource refers to the [package-config-writer-template](#).

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `registry` from tap-value `registry`. NOT overridable by workload.

### Common resources

- [Config-Provider](#)
- [App-Config](#)
- [Service-Bindings](#)
- [Api-Descriptors](#)

### Parameters provided to all resources

- `maven_repository_url` from tap-value `maven.repository.url`. NOT overridable by workload.

- `maven_repository_secret_name` from tap-value `maven.repository.secret_name`. NOT overridable by workload.
- `carvel_package_gitops_subpath` from tap-value `carvel_package.gitops_subpath`. Overridable by workload.
- `carvel_package_name_suffix` from tap-value `carvel_package.name_suffix`. Overridable by workload.
- See [Params provided by all Supply Chains to all Resources](#)

## Package

[Out of the Box Supply Chain Basic](#)

## More information

See [Install Out of the Box Supply Chain Basic](#) for information about setting tap-values at installation time.

## Basic-Image-to-URL-Package (experimental)

- Fetches a prebuilt image.
- Bundles the Kubernetes configuration necessary to deploy the application into a Carvel Package.
- Commits the Package to a Git Repository.

## Resources

### image-provider

Refers to [image-provider-template](#).

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.

### carvel-package

Refers to [carvel-package](#).

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `registry` from tap-value `registry`. NOT overridable by workload.

### package-config-writer

Refers to the [package-config-writer-and-pull-requester-template](#) when the tap-value `gitops.commit_strategy` is `pull_request`. Otherwise, this resource refers to the [package-config-writer-template](#)

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `registry` from tap-value `registry`. NOT overridable by workload.

## Common resources

- [Config-Provider](#)
- [App-Config](#)
- [Service-Bindings](#)
- [Api-Descriptors](#)
- [Config-Writer](#)
- [Deliverable](#)

## Parameters provided to all resources

- `carvel_package_gitops_subpath` from tap-value `carvel_package.gitops_subpath`. Overridable by workload.
- `carvel_package_name_suffix` from tap-value `carvel_package.name_suffix`. Overridable by workload.
- See [Params provided by all Supply Chains to all Resources](#)

## Package

Out of the Box Supply Chain Basic

## More information

See [Install Out of the Box Supply Chain Basic](#) for information about setting tap-values at installation time.

## Resources common to all OOTB supply chains

### config-provider

Refers to [convention-template](#).

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.

### app-config

The tap-values field `supported_workloads` defines which templates are referred to by this resource. Default configuration is:

```
supported_workloads:
- type: web
  cluster_config_template_name: config-template
- type: server
  cluster_config_template_name: server-template
- type: worker
  cluster_config_template_name: worker-template
```

The workload's `apps.tanzu.vmware.com/workload-type` label determines which template is used at this step. For example, when the workload has a label `apps.tanzu.vmware.com/workload-type:web`, the supply chain references `config-template`.

No parameters are provided by the supply-chain.

### service-bindings

Refers to the [service-binding template](#).

No parameters are provided by the supply-chain.

## api-descriptors

Refers to the [api-descriptors template](#).

No parameters are provided by the supply-chain.

## config-writer

Refers to the [config-writer-and-pull-requester-template](#) when the tap-value `gitops.commit_strategy` is `pull_request`. Otherwise, this resource refers to the [config-writer-template](#)

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `registry` from tap-value `registry`. NOT overridable by workload.

## deliverable

Refers to the [external-deliverable-template](#) when the tap-value `external_delivery` evaluates to `true`. Otherwise the resource refers to the [deliverable-template](#).

Parameters provided:

- `registry` from tap-value `registry`. NOT overridable by workload.

## Parameters provided by all supply chains to all resources

All of the following parameters are overridable by the workload.

- `gitops_branch` from tap-value `gitops.branch`
- `gitops_user_name` from tap-value `gitops.username`
- `gitops_user_email` from tap-value `gitops.email`
- `gitops_commit_message` from tap-value `gitops.commit_message`
- `gitops_ssh_secret` from tap-value `gitops.ssh_secret`
- `gitops_repository_prefix` from tap-value `gitops.repository_prefix` when present.
- `gitops_server_address` from tap-value `gitops.server_address` when present.
- `gitops_repository_owner` from tap-value `gitops.repository_owner` when present.
- `gitops_repository_name` from tap-value `gitops.repository_name` when present.
- `gitops_server_kind` from tap-value `gitops.pull_request.server_kind` when present.
- `gitops_commit_branch` from tap-value `gitops.pull_request.commit_branch` when present.
- `gitops_pull_request_title` from tap-value `gitops.pull_request.pull_request_title` when present.
- `gitops_pull_request_body` from tap-value `gitops.pull_request.pull_request_body` when present.

## Template reference for Supply Chain Choreographer

This topic describes the objects from templates that you can use with Supply Chain Choreographer.

All the objects referenced in this topic are [Cartographer Templates](#) packaged in [Out of the Box Templates](#).

This topic describes:

- The purpose of the templates
- The one or more objects that the templates create
- The supply chains that include the templates
- The parameters that the templates use

## source-template

### Purpose

Creates an object to fetch source code and make that code available to other objects in the supply chain. See [Building from Source](#).

### Used by

- [Source-to-URL](#) in the `source-provider` step.
- [Source-Test-to-URL](#) in the `source-provider` step.
- [Source-Test-Scan-to-URL](#) in the `source-provider` step.
- [Source-to-URL-Package \(experimental\)](#) in the `source-provider` step.

### Creates

The source-template creates one of three objects, either:

- `GitRepository`. Created if the workload has `.spec.source.git` defined.
- `MavenArtifact`. Created if the template is provided a value for the parameter `maven`.
- `ImageRepository`. Created if the workload has `.spec.source.image` defined.

### GitRepository

`GitRepository` makes source code from a particular commit available as a tarball in the cluster. Other resources in the supply chain can then access that code.


### Parameters

## Template reference for Supply Chain Choreographer

| Parameter name                 | Meaning                                                                                                                                                                                                                                                                                                                  | Example                                                 |
|--------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------|
| <code>gitImplementation</code> | The library used to fetch source code. If not provided, Tanzu Application Platform's default implementation uses <code>go-git</code> , which works with the providers supported by Tanzu Application Platform: GitHub and GitLab. An alternate value that can be used with other Git providers is <code>libgit2</code> . | <pre> - name: gitImplementation   value: libgit2 </pre> |



| Parameter name                 | Meaning                                                                                                                                                                                                                                                                                                                                                                     | Example                                                       |
|--------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------|
| <code>gitops_ssh_secret</code> | Name of the secret used to provide credentials for the Git repository. The secret with this name must exist in the same namespace as the <code>Workload</code> . The credentials must be sufficient to read the repository. If not provided, Tanzu Application Platform defaults to look for a secret named <code>git-ssh</code> . See <a href="#">Git authentication</a> . | <pre>- name: gitops_ssh_secret   value: git-credentials</pre> |

 **Note**

Some Git providers, notably Azure DevOps, require you to use `libgit2` due to the server-side implementation providing support only for [git's v2 protocol](#). For information about the features supported by each implementation, see [git implementation](#) in the flux documentation.

**More information**

For an example using the Tanzu CLI to create a Workload using GitHub as the provider of source code, see [Create a workload from GitHub repository](#).


For information about GitRepository objects, see [GitRepository](#).

**ImageRepository**

`ImageRepository` makes the contents of a container image available as a tarball on the cluster.

**Parameters**

| Parameter name              | Meaning                                                                                                                                                                                               | Example                                            |
|-----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------|
| <code>serviceAccount</code> | Name of the service account, providing credentials to <code>ImageRepository</code> for fetching container images. The service account must exist in the same namespace as the <code>Workload</code> . | <pre>- name: serviceAccount   value: default</pre> |

 **Note**

When using the Tanzu CLI to configure this `serviceAccount` parameter, use `--param serviceAccount=...`. The similarly named `--service-account` flag sets a different value: the `spec.serviceAccountName` key in the `Workload` object.

**More information**

For information about the ImageRepository resource, see the [ImageRepository reference documentation](#).

For information about how to use the Tanzu CLI to create a workload leveraging ImageRepository, see [Create a workload from local source code](#).

## MavenArtifact

`MavenArtifact` makes a pre-built Java artifact available to as a tarball on the cluster.

While the `source-template` leverages the workload's `.spec.source` field when creating a `GitRepository` or `ImageRepository` object, the creation of the `MavenArtifact` relies only on parameters in the Workload.

### Parameters

| Parameter name                            | Meaning                                                                                                                                                      | Example                                                                                                                                                                                                                    |
|-------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>maven</code>                        | Points to the Maven artifact to fetch and the polling interval.                                                                                              | <pre> - name: maven   value:     artifactId: springboot-initial     groupId: com.example     version: RELEASE     classifier: sources # optional     type: jar # optional     artifactRetryTimeout: 1m0s # optional </pre> |
| <code>maven_repository_url</code>         | Specifies the Maven repository from which to fetch                                                                                                           | <pre> - name: maven_repository_url   value: https://repol.maven.org/maven2/ </pre>                                                                                                                                         |
| <code>maven_repository_secret_name</code> | Specifies the secret containing credentials necessary to fetch from the Maven repository. The secret named must exist in the same workspace as the workload. | <pre> - name: maven_repository_secret_name   value: auth-secret </pre>                                                                                                                                                     |

### More information

For information about the custom resource, see [MavenArtifact reference docs](#).

For information about how to use the custom resource with the `tanzu apps workload` CLI plug-in [Create a Workload from Maven repository artifact](#).

## testing-pipeline

## Purpose

Tests the source code provided in the supply chain. Testing depends on a user provided [Tekton Pipeline](#). Parameters for this template allow for selection of the proper Pipeline and for specification of additional values to pass to the Pipeline.

## Used by

- [Source-Test-to-URL](#) in the source-tester step.
- [Source-Test-Scan-to-URL](#) in the source-tester step.

These are used as the `source-tester` resource.

## Creates

`testing-pipeline` creates a [Runnable](#) object. This Runnable provides inputs to the [ClusterRunTemplate](#) named `tekton-source-pipelinerun`.

## Parameters

| Parameter name                                | Meaning                                                                                                                                                                                                 | Example                                                                                                                                  |
|-----------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| <code>testing_pipeline_matching_labels</code> | Set of labels to use when searching for Tekton Pipeline objects in the same namespace as the Workload. By default, a Pipeline labeled as <code>apps.tanzu.vmware.com/pipeline: test</code> is selected. | <pre> - name: testing_pipeline_matching_labels   value:     apps.tanzu.vmware.com/pipeline: test     my.comp.any/language: golang </pre> |
| <code>testing_pipeline_params</code>          | Set of parameters to pass to the Tekton Pipeline. To this set of parameters, the template always adds the source URL and revision as <code>source-url</code> and <code>source-revision</code> .         | <pre> - name: testing_pipeline_params   value:     - name: verbose       value: true     - name: foo       value: bar </pre>             |

## More information

For information about the ClusterRunTemplate that pairs with the Runnable, read [tekton-source-pipelinerun](#)

For information about the Tekton Pipeline that the user must create, read the [OOTB Supply Chain Testing documentation of the Pipeline](#)

## source-scanner-template

## Purpose

Scans the source code for vulnerabilities.

## Used by

- [Source-Test-Scan-to-URL](#) in the source-scanner step.

This is used as the `source-scanner` resource.

## Creates

[SourceScan](#)

## Parameters

| Parameter name                        | Meaning                                                                                                                                               | Example                                                                           |
|---------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|
| <code>scanning_source_template</code> | Name of the ScanTemplate object to use for running the scans. The ScanTemplate must be in the same namespace as the Workload.                         | <pre>- name: scanning_source_template   value: private-source-scan-template</pre> |
| <code>scanning_source_policy</code>   | Name of the ScanPolicy object to use when evaluating the scan results of a source scan. The ScanPolicy must be in the same namespace as the Workload. | <pre>- name: scanning_source_policy   value: allowlist-policy</pre>               |

## More information

For information about how to set up the Workload namespace with the ScanPolicy and ScanTemplate required for this resource, see [Out of the Box Supply Chain with Testing and Scanning](#).

For information about the SourceScan custom resource, see [SourceScan reference](#).

For information about how the artifacts found during scanning are catalogued, see [Supply Chain Security Tools for Tanzu – Store](#).

## image-provider-template

### Purpose

Fetches a container image of a prebuilt application, specified in the workload's `.spec.image` field. This makes the content-addressable name, (e.g. the image name containing the digest) available to other resources in the supply chain.

### Used by

- [Basic-Image-to-URL](#) in the image-provider step.
- [Testing-Image-to-URL](#) in the image-provider step.
- [Scanning-Image-Scan-to-URL](#) in the image-provider step.
- [Basic-Image-to-URL-Package \(experimental\)](#) in the image-provider step.

These are used as the `image-provider` resource.

## Creates

ImageRepository.source.apps.tanzu.vmware.com

## Parameters

| Parameter name              | Meaning                                                                                                                                                | Example                                                    |
|-----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------|
| <code>serviceAccount</code> | Name of the service account providing credentials for the target image registry. The service account must exist in the same namespace as the Workload. | <pre>- name: s   serviceAccount     value:   default</pre> |



### Note

When using the Tanzu CLI to configure this `serviceAccount` parameter, use `--param serviceAccount=...`. The similarly named `--service-account` flag sets a different value: the `spec.serviceAccountName` key in the Workload object.

## More information

For information about the ImageRepository resource, see [ImageRepository reference docs](#).

For information about prebuilt images, see [Using a prebuilt image](#).

## kpack-template

### Purpose

Builds a container image from source code using [cloud native buildpacks](#).

### Used by

- [Source-to-URL](#) in the image-provider step.
- [Source-Test-to-URL](#) in the image-provider step.
- [Source-Test-Scan-to-URL](#) in the image-provider step.
- [Source-to-URL-Package \(experimental\)](#) in the image-provider step.

These are used as the `image-provider` resource when the workload parameter `dockerfile` is not defined.

## Creates

Image.kpack.io

## Parameters

| Parameter name                    | Meaning                                                                                                                                                                                                                                              | Example                                                                                                                   |
|-----------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------|
| <code>serviceAccount</code>       | Name of the service account providing credentials for the configured image registry. <code>Image</code> uses these credentials to push built container images to the registry. The service account must exist in the same namespace as the Workload. | <pre>- name:   serviceAccount   value:   default</pre>                                                                    |
| <code>clusterBuilder</code>       | Name of the Kpack Cluster Builder to use.                                                                                                                                                                                                            | <pre>- name:   clusterBuilder   value:   nodejs-cluster-builder</pre>                                                     |
| <code>buildServiceBindings</code> | Definition of a list of service bindings to make use at build time. For example, providing credentials for fetching dependencies from repositories that require credentials.                                                                         | <pre>- name:   buildServiceBindings   value:     - name:       settings-xml       kind: Secret       apiVersion: v1</pre> |
| <code>live-update</code>          | Enable the use of Tilt's live-update function.                                                                                                                                                                                                       | <pre>- name:   live-update   value:   "true"</pre>                                                                        |



#### Note

When using the Tanzu CLI to configure this `serviceAccount` parameter, use `--param serviceAccount=...`. The similarly named `--service-account` flag sets a different value: the `spec.serviceAccountName` key in the Workload object.

## More information

For information about the integration with Tanzu Build Service, see [Tanzu Build Service Integration](#).

For information about `live-update`, see [Developer Conventions](#) and [Overview of Tanzu Developer Tools for IntelliJ](#).

For information about using Kpack builders with `clusterBuilder`, see [Builders](#).

For information about `buildServiceBindings`, see [Service Bindings](#).

## kaniko-template

### Purpose

Build an image for source code that includes a Dockerfile.

## Used by

- [Source-to-URL](#) in the image-provider step.
- [Source-Test-to-URL](#) in the image-provider step.
- [Source-Test-Scan-to-URL](#) in the image-provider step.
- [Source-to-URL-Package \(experimental\)](#) in the image-provider step.

These are used as the `image-provider` resource when the workload parameter `dockerfile` is defined.

## Creates

A `taskrun.tekton.dev` which provides configuration to a Tekton ClusterTask to build an image with kaniko.

This template uses the `lifecycle: tekton` flag to create new immutable objects rather than updating the previous object.

## Parameters

| Parameter name                       | Meaning                                                                                                                                                                                                                                                                                                   | Example                                                                                   |
|--------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|
| <code>dockerfile</code>              | relative path to the Dockerfile file in the build context                                                                                                                                                                                                                                                 | <pre>./Dockerfile</pre>                                                                   |
| <code>docker_build_context</code>    | relative path to the directory where the build context is                                                                                                                                                                                                                                                 | <pre>.</pre>                                                                              |
| <code>docker_build_extra_args</code> | List of flags to pass directly to kaniko, such as providing arguments to a build.                                                                                                                                                                                                                         | <pre>- --build-arg=FOO=BAR</pre>                                                          |
| <code>serviceAccount</code>          | Name of the service account to use for providing Docker credentials. The service account must exist in the same namespace as the Workload. The service account must have a secret associated with the credentials. See <a href="#">Configuring authentication for Docker</a> in the Tekton documentation. | <pre>- name: serviceAccount   value: default</pre>                                        |
| <code>registry</code>                | Specification of the registry server and repository in which the built image is placed.                                                                                                                                                                                                                   | <pre>- name: registry   value:     server: index.docker.io     repository: web-team</pre> |

## More information

For information about how to use Dockerfile-based builds and limits associated with the function, see [Dockerfile-based builds](#).

For information about `lifecycle:tekton`, read [Cartographer Lifecycle](#).

## image-scanner-template

### Purpose

Scans the container image for vulnerabilities, persists the results in a store, and prevents the image from moving forward if CVEs are found which are not compliant with its referenced ScanPolicy.

### Used by

- [Source-Test-Scan-to-URL](#) in the image-scanner step.
- [Scanning-Image-Scan-to-URL](#) in the image-scanner step.

### Creates

ImageScan.scanning.apps.tanzu.vmware.com

### Parameters

| Parameter name                       | Meaning                                                                                                                                          | Example                                                                         |
|--------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------|
| <code>scanning_image_template</code> | Name of the ScanTemplate object for running the scans against a container image. The ScanTemplate must be in the same namespace as the Workload. | <pre>- name: scanning_image_template   value: private-image-scan-template</pre> |
| <code>scanning_image_policy</code>   | Name of the ScanPolicy object for evaluating the scan results of an image scan. The ScanPolicy must be in the same namespace as the Workload.    | <pre>- name: scanning_image_policy   value: allowlist-policy</pre>              |

### More information

For information about the ImageScan custom resource, see [ImageScan reference](#).

For information about how the artifacts found during scanning are catalogued, see [Supply Chain Security Tools for Tanzu – Store](#).

## convention-template

### Purpose

Create the PodTemplateSpec for the Kubernetes configuration (e.g. the knative service or kubernetes deployment) which are applied to the cluster.

### Used by

- [Source-to-URL](#) in the config-provider step.
- [Basic-Image-to-URL](#) in the config-provider step.



- [Source-Test-to-URL](#) in the config-provider step.
- [Testing-Image-to-URL](#) in the config-provider step.
- [Source-Test-Scan-to-URL](#) in the config-provider step.
- [Scanning-Image-Scan-to-URL](#) in the config-provider step.
- [Source-to-URL-Package \(experimental\)](#) in the config-provider step.
- [Basic-Image-to-URL-Package \(experimental\)](#) in the config-provider step.

## Creates

Creates a [PodIntent](#) object. The PodIntent leverages conventions installed on the cluster. The PodIntent object is responsible for generating a PodTemplateSpec. The PodTemplateSpec is used in app configs, such as knative services and deployments, to represent the shape of the pods to run the application in containers.

## Parameters

| Parameter name              | Meaning                                                                                                                                                                                                                                                                                                                                                                                                    | Example                                                                                              |
|-----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|
| <code>serviceAccount</code> | Name of the serviceAccount providing necessary credentials to <a href="#">PodIntent</a> . The serviceAccount must be in the same namespace as the Workload. The serviceAccount is set as the <code>serviceAccountName</code> in the podtemplatespec. The credentials associated with the serviceAccount must allow fetching the container image used to inspect the metadata passed to convention servers. | <pre>- name: serviceAccount   value: default</pre>                                                   |
| <code>annotations</code>    | Extra set of annotations to pass down to the PodTemplateSpec.                                                                                                                                                                                                                                                                                                                                              | <pre>- name: annotations   value:     name: my-application     version: v1.2.3     team: store</pre> |
| <code>debug</code>          | Put the workload in debug mode.                                                                                                                                                                                                                                                                                                                                                                            | <pre>- name: debug   value: "true"</pre>                                                             |

| Parameter name           | Meaning                                                       | Example                                            |
|--------------------------|---------------------------------------------------------------|----------------------------------------------------|
| <code>live-update</code> | Enable live-updating of the code (for innerloop development). | <pre> - n ame: live -update v alue: "tr ue" </pre> |



#### Note

When using the Tanzu CLI to configure this `serviceAccount` parameter, use `--param serviceAccount=...`. The similarly named `--service-account` flag sets a different value: the `spec.serviceAccountName` key in the Workload object.

## More information

For information about `PodTemplateSpec`, see [PodTemplateSpec](#) in the Kubernetes documentation.

For information about conventions, see [Cartographer Conventions](#).

For information about the two convention servers enabled by default in Tanzu Application Platform installations, see [Developer Conventions](#) and [Spring Boot conventions](#).

## config-template

### Purpose

For workloads with the label `apps.tanzu.vmware.com/workload-type: web`, define a knative service.

### Used by

- [Source-to-URL](#) in the app-config step.
- [Basic-Image-to-URL](#) in the app-config step.
- [Source-Test-to-URL](#) in the app-config step.
- [Testing-Image-to-URL](#) in the app-config step.
- [Source-Test-Scan-to-URL](#) in the app-config step.
- [Scanning-Image-Scan-to-URL](#) in the app-config step.

### Creates

A ConfigMap, in which the data field has a key `delivery.yaml` whose value is the definition of a knative service.

### Parameters

None

### More information

See [workload types](#) for more details about the three different types of workloads.

## worker-template

### Purpose

For workloads with the label `apps.tanzu.vmware.com/workload-type: worker`, define a Kubernetes Deployment.

### Used by

- [Source-to-URL](#) in the app-config step.
- [Basic-Image-to-URL](#) in the app-config step.
- [Source-Test-to-URL](#) in the app-config step.
- [Testing-Image-to-URL](#) in the app-config step.
- [Source-Test-Scan-to-URL](#) in the app-config step.
- [Scanning-Image-Scan-to-URL](#) in the app-config step.

### Creates

A ConfigMap, in which the data field has a key `delivery.yaml` whose value is the definition of a Kubernetes Deployment.

### Parameters

None

### More information

For information about the three different types of workloads, see [workload types](#).

## server-template

### Purpose

For workloads with the label `apps.tanzu.vmware.com/workload-type: server`, define a Kubernetes Deployment and a Kubernetes Service.

### Used by

- [Source-to-URL](#) in the app-config step.
- [Basic-Image-to-URL](#) in the app-config step.
- [Source-Test-to-URL](#) in the app-config step.
- [Testing-Image-to-URL](#) in the app-config step.
- [Source-Test-Scan-to-URL](#) in the app-config step.
- [Scanning-Image-Scan-to-URL](#) in the app-config step.
- [Source-to-URL-Package \(experimental\)](#) in the app-config step.
- [Basic-Image-to-URL-Package \(experimental\)](#) in the app-config step.

### Creates

A ConfigMap, in which the data field has a key `delivery.yaml` whose value is the definitions of a Kubernetes Deployment and a Kubernetes Service to expose the pods.

## Parameters

| Parameter name     | Meaning                                                                        | Example                                                                                       |
|--------------------|--------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|
| <code>ports</code> | Set of network ports to expose from the application to the Kubernetes cluster. | <pre> - name: ports   value:     - containerPort: 2025       name: smtp       port: 25 </pre> |

## More information

For information about the three different types of workloads, see [workload types](#).

For information about the ports parameter, see [server-specific Workload parameters](#).

## service-bindings

### Purpose

Adds [ServiceBindings](#) to the set of Kubernetes configuration files.

### Used by

- [Source-to-URL](#) in the service-bindings step.
- [Basic-Image-to-URL](#) in the service-bindings step.
- [Source-Test-to-URL](#) in the service-bindings step.
- [Testing-Image-to-URL](#) in the service-bindings step.
- [Source-Test-Scan-to-URL](#) in the service-bindings step.
- [Scanning-Image-Scan-to-URL](#) in the service-bindings step.
- [Source-to-URL-Package \(experimental\)](#) in the service-bindings step.
- [Basic-Image-to-URL-Package \(experimental\)](#) in the service-bindings step.

### Creates

A ConfigMap. This template consumes input of multiple deployment YAML files and enriches the input with ResourceClaims and ServiceBindings if the workload contains serviceClaims.

## Parameters

| Parameter name           | Meaning                                                                                | Example                                                                                                |
|--------------------------|----------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|
| <code>annotations</code> | Extra set of annotations to pass down to the ServiceBinding and ResourceClaim objects. | <pre> - name: annotations   value:     name: my-application     version: v1.2.3     team: store </pre> |

## More information

For an example, see [-service-ref](#) in the Tanzu CLI documentation.

For an overview of the function, see [Consume services on Tanzu Application Platform](#).

## api-descriptors

### Purpose

The `api-descriptor` resource takes care of adding an `APIDescriptor` to the set of Kubernetes objects to deploy such that API auto registration takes place.

### Used by

- [Source-to-URL](#) in the api-descriptors step.
- [Basic-Image-to-URL](#) in the api-descriptors step.
- [Source-Test-to-URL](#) in the api-descriptors step.
- [Testing-Image-to-URL](#) in the api-descriptors step.
- [Source-Test-Scan-to-URL](#) in the api-descriptors step.
- [Scanning-Image-Scan-to-URL](#) in the api-descriptors step.
- [Source-to-URL-Package \(experimental\)](#) in the api-descriptors step.
- [Basic-Image-to-URL-Package \(experimental\)](#) in the api-descriptors step.

### Creates

A ConfigMap. This template consumes input of multiple YAML files and enriches the input with an `APIDescriptor` if the workload has a label `apis.apps.tanzu.vmware.com/register-api` set to `true`.

### Parameters

| Parameter name           | Meaning                                                                         | Example                                                                                                |
|--------------------------|---------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|
| <code>annotations</code> | Extra set of annotations to pass down to the <code>APIDescriptor</code> object. | <pre> - name: annotations   value:     name: my-application     version: v1.2.3     team: store </pre> |

| Parameter name              | Meaning                                                                                  | Example                                                                                                                                                                                                                                                |
|-----------------------------|------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>api_descriptor</code> | Information used to fill the state that you want of the APIDescriptor object (its spec). | <pre> - name: api_descriptor   value:     type: openapi     location:       baseURL: http://petclinic- hard-coded.my-apps.tapdemo.vmware.com/       path: "/v3/api     owner: team-petclinic     system: pet-clinics     description: "example" </pre> |

## More information

For information about API auto registration, see [Use API Auto Registration](#).

## config-writer-template

### Purpose

Persist in an external system, such as a registry or git repository, the Kubernetes configuration passed to the template.

### Used by

- [Source-to-URL](#) in the config-writer step.
- [Basic-Image-to-URL](#) in the config-writer step.
- [Source-Test-to-URL](#) in the config-writer step.
- [Testing-Image-to-URL](#) in the config-writer step.
- [Source-Test-Scan-to-URL](#) in the config-writer step.
- [Scanning-Image-Scan-to-URL](#) in the config-writer step.

### Creates

A runnable which creates a Tekton TaskRun that refers either to the Tekton Task `git-writer` or the Tekton Task `image-writer`.

### Parameters

| Parameter name              | Meaning                                                                                                                                                         | Example                                              |
|-----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------|
| <code>serviceAccount</code> | Name of the service account which provides the credentials to the registry or repository. The service account must exist in the same namespace as the Workload. | <pre> - name: serviceAccount   value: default </pre> |

| Parameter name                        | Meaning                                                                                               | Example                                                                                      |
|---------------------------------------|-------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------|
| <code>gitops_branch</code>            | Name of the branch to push the configuration to.                                                      | <pre>- name: gitops_branch   value: main</pre>                                               |
| <code>gitops_user_name</code>         | User name to use in the commits.                                                                      | <pre>- name: gitops_user_name   value: "Alice Lee"</pre>                                     |
| <code>gitops_user_email</code>        | User email address to use in the commits.                                                             | <pre>- name: gitops_user_email   value: alice@example.com</pre>                              |
| <code>gitops_commit_message</code>    | Message to write as the body of the commits produced for pushing configuration to the Git repository. | <pre>- name: gitops_commit_message   value: "ci bump"</pre>                                  |
| <code>gitops_repository</code>        | The full repository URL to which the configuration is committed. <b>DEPRECATED</b>                    | <pre>- name: gitops_repository   value: "https://github.com/vmware-tanzu/cartographer"</pre> |
| <code>gitops_repository_prefix</code> | The prefix of the repository URL. <b>DEPRECATED</b>                                                   | <pre>- name: gitops_repository   value: "https://github.com/vmware-tanzu/"</pre>             |
| <code>gitops_server_address</code>    | The server URL of the Git repository to which configuration is applied.                               | <pre>- name: gitops_server_address   value: "https://github.com/"</pre>                      |
| <code>gitops_repository_owner</code>  | The owner/organization to which the repository belongs.                                               | <pre>- name: gitops_repository_owner   value: vmware-tanzu</pre>                             |

| Parameter name                      | Meaning                                                                                   | Example                                                                                                                                                                                                                                      |
|-------------------------------------|-------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>gitops_repository_name</code> | The name of the repository.                                                               | <pre> - name: gitops_repository_name   value: cartographer </pre>                                                                                                                                                                            |
| <code>registry</code>               | Specification of the registry server and repository in which the configuration is placed. | <pre> - name: registry   value:     server: index.docker.io     repository: web - team   ca_cert_data:     -----BEGIN CERTIFICATE-----       MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEEDQUAMEY...     -----END CERTIFICATE----- </pre> |

## More information

For information about operating this template, see [Gitops vs RegistryOps](#) and the [config-writer-and-pull-requester-template](#).

## config-writer-and-pull-requester-template

### Purpose

Persist the passed in Kubernetes configuration to a branch in a repository and open a pull request to another branch. This process allows for manual review of configuration before deployment to a cluster.

### Used by

- [Source-to-URL](#) in the config-writer step.
- [Basic-Image-to-URL](#) in the config-writer step.
- [Source-Test-to-URL](#) in the config-writer step.
- [Testing-Image-to-URL](#) in the config-writer step.
- [Source-Test-Scan-to-URL](#) in the config-writer step.
- [Scanning-Image-Scan-to-URL](#) in the config-writer step.

### Creates

A runnable which provides configuration to the ClusterRunTemplate `commit-and-pr-pipelinerun` to create a Tekton TaskRun. The Tekton TaskRun refers to the Tekton Task `commit-and-pr`.

### Parameters



| Parameter name                         | Meaning                                                                                                                                                         | Example                                                                          |
|----------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------|
| <code>serviceAccount</code>            | Name of the service account which provides the credentials to the registry or repository. The service account must exist in the same namespace as the Workload. | <pre>- name: serviceAccount   value: default</pre>                               |
| <code>gitops_commit_branch</code>      | Name of the branch to which configuration is pushed.                                                                                                            | <pre>- name: gitops_commit_branch   value: feature</pre>                         |
| <code>gitops_branch</code>             | Name of the branch to which a pull request is opened.                                                                                                           | <pre>- name: gitops_branch   value: main</pre>                                   |
| <code>gitops_user_name</code>          | User name to use in the commits.                                                                                                                                | <pre>- name: gitops_user_name   value: "Alice Lee"</pre>                         |
| <code>gitops_user_email</code>         | User email address to use in the commits.                                                                                                                       | <pre>- name: gitops_user_email   value: alice@example.com</pre>                  |
| <code>gitops_commit_message</code>     | Message to write as the body of the commits produced for pushing configuration to the Git repository.                                                           | <pre>- name: gitops_commit_message   value: "ci bump"</pre>                      |
| <code>gitops_pull_request_title</code> | Title of the pull request to be opened.                                                                                                                         | <pre>- name: gitops_pull_request_title   value: "ready for review"</pre>         |
| <code>gitops_pull_request_body</code>  | Body of the pull request to be opened.                                                                                                                          | <pre>- name: gitops_pull_request_body   value: "generated by supply chain"</pre> |

| Parameter name                       | Meaning                                                                 | Example                                                                  |
|--------------------------------------|-------------------------------------------------------------------------|--------------------------------------------------------------------------|
| <code>gitops_server_address</code>   | The server URL of the Git repository to which configuration is applied. | <pre>- name: gito ps_server_address   value: "https://github.com/"</pre> |
| <code>gitops_repository_owner</code> | The owner/organization to which the repository belongs.                 | <pre>- name: gito ps_repository_owner   value: vmware-tanzu</pre>        |
| <code>gitops_repository_name</code>  | The name of the repository.                                             | <pre>- name: gito ps_repository_name   value: cartographer</pre>         |
| <code>gitops_server_kind</code>      | The kind of Git provider                                                | <pre>- name: gito ps_server_kind   value: gitlab</pre>                   |

## More information

For information about the operation of this template, see [Gitops vs RegistryOps](#) and the [config-writer-template](#).

## deliverable-template

### Purpose

Create a deliverable which [pairs with a Delivery](#) to deploy Kubernetes configuration on the cluster.

### Used by

- [Source-to-URL](#) in the deliverable step.
- [Basic-Image-to-URL](#) in the deliverable step.
- [Source-Test-to-URL](#) in the deliverable step.
- [Testing-Image-to-URL](#) in the deliverable step.
- [Source-Test-Scan-to-URL](#) in the deliverable step.
- [Scanning-Image-Scan-to-URL](#) in the deliverable step.

### Creates

A [Deliverable](#) preconfigured with reference to a repository or registry from which to fetch Kubernetes configuration.

## Parameters

| Parameter name                        | Meaning                                                                                                                                                                                                                                               | Example                                                                                      |
|---------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------|
| <code>serviceAccount</code>           | Name of the service account providing the necessary permissions for the Delivery to create children objects. Populates the Deliverable's <code>serviceAccount</code> parameter. The service account must be in the same namespace as the Deliverable. | <pre>- name: serviceAccount   value: default</pre>                                           |
| <code>gitops_ssh_secret</code>        | Name of the secret where credentials exist for fetching the configuration from a Git repository. Populates the Deliverable's <code>gitops_ssh_secret</code> parameter. The service account must be in the same namespace as the Deliverable.          | <pre>- name: gitops_ssh_secret   value: ssh-secret</pre>                                     |
| <code>gitops_branch</code>            | Name of the branch from which to fetch the configuration.                                                                                                                                                                                             | <pre>- name: gitops_branch   value: main</pre>                                               |
| <code>gitops_repository</code>        | The full repository URL to which the configuration is fetched. <b>DEPRECATED</b>                                                                                                                                                                      | <pre>- name: gitops_repository   value: "https://github.com/vmware-tanzu/cartographer"</pre> |
| <code>gitops_repository_prefix</code> | The prefix of the repository URL. <b>DEPRECATED</b>                                                                                                                                                                                                   | <pre>- name: gitops_repository   value: "https://github.com/vmware-tanzu/"</pre>             |
| <code>gitops_server_address</code>    | The server URL of the Git repository from which configuration is fetched.                                                                                                                                                                             | <pre>- name: gitops_server_address   value: "https://github.com/"</pre>                      |
| <code>gitops_repository_owner</code>  | The owner/organization to which the repository belongs.                                                                                                                                                                                               | <pre>- name: gitops_repository_owner   value: vmware-tanzu</pre>                             |
| <code>gitops_repository_name</code>   | The name of the repository.                                                                                                                                                                                                                           | <pre>- name: gitops_repository_name   value: cartographer</pre>                              |

| Parameter name        | Meaning                                                                                      | Example                                                                                                                                                                                                                                                               |
|-----------------------|----------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>registry</code> | Specification of the registry server and repository from which the configuration is fetched. | <pre> - name: registry   value:     server: index.docker.io     repository: web-team     ca_certificate:       -----BEGIN CERTIFICATE-----               MIIFXzCCA 0egAwIBAgIJAJYm37SFoc jlMA0GCSqGSIb3DQEBAQU AMEY...               -----END CERTIFICATE----- </pre> |



#### Note

When using the Tanzu CLI to configure this `serviceAccount` parameter, use `--param serviceAccount=...`. The similarly named `--service-account` flag sets a different value: the `spec.serviceAccountName` key in the Workload object.

## More information

For information about the ClusterDelivery shipped with `ootb-delivery-basic`, see [Out of the Box Delivery Basic](#).

## external-deliverable-template

### Purpose

Create a definition of a deliverable which a user can manually applied to an external kubernetes cluster. When a properly configured Delivery is installed on that external cluster, the Deliverable will [pair with the Delivery](#) to deploy Kubernetes configuration on the cluster. For example, the [OOTB Delivery](#).

### Used by

- [Source-to-URL](#) in the deliverable step.
- [Basic-Image-to-URL](#) in the deliverable step.
- [Source-Test-to-URL](#) in the deliverable step.
- [Testing-Image-to-URL](#) in the deliverable step.
- [Source-Test-Scan-to-URL](#) in the deliverable step.
- [Scanning-Image-Scan-to-URL](#) in the deliverable step.

### Creates

A configmap in which the `.data` field has a key `deliverable` for which the value is the YAML definition of a [Deliverable](#).

## Parameters

| Parameter name                        | Meaning                                                                                                                                                                                                                                               | Example                                                                                        |
|---------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------|
| <code>serviceAccount</code>           | Name of the service account providing the necessary permissions for the Delivery to create children objects. Populates the Deliverable's <code>serviceAccount</code> parameter. The service account must be in the same namespace as the Deliverable. | <pre> - name: serviceAccount   value: default </pre>                                           |
| <code>gitops_ssh_secret</code>        | Name of the secret where credentials exist for fetching the configuration from a Git repository. Populates the Deliverable's <code>gitops_ssh_secret</code> parameter. The service account must be in the same namespace as the Deliverable.          | <pre> - name: gitops_ssh_secret   value: ssh-secret </pre>                                     |
| <code>gitops_branch</code>            | Name of the branch from which to fetch the configuration.                                                                                                                                                                                             | <pre> - name: gitops_branch   value: main </pre>                                               |
| <code>gitops_repository</code>        | The full repository URL to which the configuration is fetched. <b>DEPRECATED</b>                                                                                                                                                                      | <pre> - name: gitops_repository   value: "https://github.com/vmware-tanzu/cartographer" </pre> |
| <code>gitops_repository_prefix</code> | The prefix of the repository URL. <b>DEPRECATED</b>                                                                                                                                                                                                   | <pre> - name: gitops_repository   value: "https://github.com/vmware-tanzu/" </pre>             |
| <code>gitops_server_address</code>    | The server URL of the Git repository from which configuration is fetched.                                                                                                                                                                             | <pre> - name: gitops_server_address   value: "https://github.com/" </pre>                      |
| <code>gitops_repository_owner</code>  | The owner/organization to which the repository belongs.                                                                                                                                                                                               | <pre> - name: gitops_repository_owner   value: vmware-tanzu </pre>                             |

| Parameter name                      | Meaning                                                                                      | Example                                                                                                                                                                                                                                                      |
|-------------------------------------|----------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>gitops_repository_name</code> | The name of the repository.                                                                  | <pre> - name: gitops_ repository_name   value: cartog rapher </pre>                                                                                                                                                                                          |
| <code>registry</code>               | Specification of the registry server and repository from which the configuration is fetched. | <pre> - name: registr y   value:     server: ind ex.docker.io     repository: web-team     ca_cert_dat a:     -----BEGIN CERTIFICATE-----                 MIIFXzCCA 0egAwIBAgIJAJYm37SFoc jlMA0GCSqGSib3DQEBDQU AMEY...     -----END CERTIFICATE----- </pre> |

## More information

For information about the ClusterDelivery shipped with `ootb-delivery-basic`, see [Out of the Box Delivery Basic](#).

For information about using the Deliverable object in a multicluster environment, see [Getting started with multicluster Tanzu Application Platform](#).

## delivery-source-template

### Purpose

Continuously fetches Kubernetes configuration files from a Git repository or container image registry and makes them available on the cluster.

### Used by

- [Delivery-Basic](#)

### Creates


The source-template creates one of three objects, either: - `GitRepository`. Created if the deliverable has `.spec.source.git` defined. - `ImageRepository`. Created if the deliverable has `.spec.source.image` defined.

### GitRepository

`GitRepository` makes source code from a particular commit available as a tarball in the cluster. Other resources in the supply chain can then access that code.

Parameters

| Parameter name                 | Meaning                                                                                                                                                                                                                                                                                                                                                                        | Example                                                       |
|--------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------|
| <code>gitImplementation</code> | The library used to fetch source code. If not provided, Tanzu Application Platform's default implementation uses <code>go-git</code> , which works with the providers supported by Tanzu Application Platform: GitHub and GitLab. An alternate value that you can use with other Git providers is <code>libgit2</code> .                                                       | <pre>- name: gitImplementation   value: libgit2</pre>         |
| <code>gitops_ssh_secret</code> | Name of the secret used to provide credentials for the Git repository. The secret with this name must exist in the same namespace as the <code>Deliverable</code> . The credentials must be sufficient to read the repository. If not provided, Tanzu Application Platform defaults to look for a secret named <code>git-ssh</code> . See <a href="#">Git authentication</a> . | <pre>- name: gitops_ssh_secret   value: git-credentials</pre> |



**Note**

Some Git providers, notably Azure DevOps, require you to use `libgit2` due to the server-side implementation providing support only for [git's v2 protocol](#). For information about the features supported by each implementation, see [git implementation](#) in the flux documentation.

More information

For an example using the Tanzu CLI to create a Workload using GitHub as the provider of source code, see [Create a workload from GitHub repository](#).

For information about GitRepository objects, see [GitRepository](#).

ImageRepository

`ImageRepository` makes the contents of a container image available as a tarball on the cluster.

Parameters

| Parameter name              | Meaning                                                                                                                                                                                                  | Example                                            |
|-----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------|
| <code>serviceAccount</code> | Name of the service account, providing credentials to <code>ImageRepository</code> for fetching container images. The service account must exist in the same namespace as the <code>Deliverable</code> . | <pre>- name: serviceAccount   value: default</pre> |

More information

For information about the ImageRepository resource, see [ImageRepository reference docs](#).

## app-deploy

### Purpose

Applies Kubernetes configuration to the cluster.

### Used by

- [Delivery-Basic](#)

### Creates

A `kapp` App.

### Parameters

| Parameter name                               | Meaning                                                                                                                                                                                                   | Example                                                      |
|----------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------|
| <code>serviceAccount</code>                  | Name of the service account providing the necessary privileges for <code>App</code> to apply the Kubernetes objects to the cluster. The service account must be in the same namespace as the Deliverable. | <pre>- name: serviceAccount t       value: e: default</pre>  |
| <code>gitops_sub_path</code><br>(deprecated) | Sub directory within the configuration bundle that is used for looking up the files to apply to the Kubernetes cluster. <b>DEPRECATED</b>                                                                 | <pre>- name: gitops_sub_pa th       value: e: ./config</pre> |



#### Note

The `gitops_sub_path` parameter is deprecated. Use `deliverable.spec.source.subPath` instead.

### More information

For details about RBAC and how `kapp-controller` makes use of the ServiceAccount provided through the Deliverable's `serviceAccount` parameter, see [kapp-controller's Security Model](#).

## carvel-package (experimental)

### Purpose

Bundles Kubernetes configuration into a Carvel Package.

### Used by

- [Source-to-URL-Package \(experimental\)](#) in the carvel-package step.
- [Basic-Image-to-URL-Package \(experimental\)](#) in the carvel-package step.



## Creates

A taskrun.tekton.dev which provides configuration to a Tekton ClusterTask to bundle Kubernetes configuration into a Carvel Package.

This template uses the `lifecycle: tekton` flag to create new immutable objects rather than updating the previous object.

## Parameters

| Parameter name                             | Meaning                                                                                                                                                                                                                                                                                                   | Example                                                                                   |
|--------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|
| <code>serviceAccount</code>                | Name of the service account to use for providing Docker credentials. The service account must exist in the same namespace as the Workload. The service account must have a secret associated with the credentials. See <a href="#">Configuring authentication for Docker</a> in the Tekton documentation. | <pre>- name: serviceAccount   value: default</pre>                                        |
| <code>registry</code>                      | Specification of the registry server and repository in which the built image is placed.                                                                                                                                                                                                                   | <pre>- name: registry   value:     server: index.docker.io     repository: web-team</pre> |
| <code>carvel_package_gitops_subpath</code> | Specifies the subpath to which Carvel Packages should be written.                                                                                                                                                                                                                                         | <pre>- name: carvel_package_gitops_subpath   value: path/to/my/dir</pre>                  |
| <code>carvel_package_name_suffix</code>    | Specifies the suffix to append to the Carvel Package name. The format is <code>WORKLOAD_NAME.WORKLOAD_NAMESPACE.carvel_package_name_suffix</code> . The full Carvel Package name must be a valid DNS subdomain name as defined in RFC 1123.                                                               | <pre>- name: carvel_package_name_suffix   value: vmware.com</pre>                         |

| Parameter name                         | Meaning                                        | Example                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|----------------------------------------|------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>carvel_package_parameters</code> | Specifies the custom Carvel Package parameters | <pre> - name: carvel_package_parameters   value:       - selector:       matchLabels:         apps.tanzu.vmware.com/workload-type: server       schema:           #@data/values-schema         ---         #@schema/title "Workload name"         #@schema/desc "Required. Name of the workload, used by K8s Ingress HTTP rules."         #@schema/example "tanzu-java-web-app"         #@schema/validation min_len=1         workload_name: ""          #@schema/title "Replicas"         #@schema/desc "Number of replicas."         replicas:           1          #@schema/title "Port"         #@schema/desc "Port number for the backend associated with K8s Ingress."         port: 8080          #@schema/title "Hostname"         #@schema/desc "If set, K8s Ingress will be created with HTTP rules for hostname."         #@schema/example "app.tanzu.vmware.com"         hostname:           ""          #@schema/title "Cluster Issuer"         #@schema/desc "CertManager Issuer to use to generate c </pre> |

| Parameter name | Meaning | Example                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|----------------|---------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                |         | <pre> ertificate for K8s Ingress."       cluster_issuer: "tap-ingress-selfsigned"       overlays:           #@ load("@ytt:overlay", "overlay")         #@ load("@ytt:data", "data")         #@overlay/match by=overlay.subset({"apiVersion":"apps/v1", "kind": "Deployment"})         ---         spec:           #@overlay/match missing_ok=True           replicas: #@ data.values.replicas            #@ if data.values.hostname != ""           ---           apiVersion: networking.k8s.io/v1           kind: Ingress           metadata:             name: #@ data.values.workload_name             annotations:               cert-manager.io/cluster-issuer: #@ data.values.cluster_issuer               ingress.kubernetes.io/force-ssl-redirect: "true"               kubernetes.io/ingress.class: contour               kapp.k14s.io/change-rule: "upsert after upserting Services"             labels:               app.kubernetes.io/component: "run"               carto.run/workload-name: #@ data.values.workload_name           spec: </pre> |

| Parameter name | Meaning | Example                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|----------------|---------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                |         | <pre> tls:   - secr etName: #@ data.value s.workload_name       host s:   - #@ data.values.hostname   rules:   - host: #@ data.values.hostnam e       http:       path s:   - pa thType: Prefix       pa th: /       ba ckend:  service:  name: #@ data.values.w orkload_name  port:  number: #@ data.value s.port       #@ end   - selector:       matchLabel s:       apps.tan zu.vmware.com/workload -type: web       schema:   #@data/val ues-schema       --- #@schema/v alidation min_len=1 workload_n ame: ""       overlays: ""   - selector:       matchLabel s:       apps.tan zu.vmware.com/workload -type: worker       schema:   #@data/val ues-schema       --- #@schema/v alidation min_len=1 workload_n ame: ""       replicas: 1 </pre> |

| Parameter name                               | Meaning                                                                                 | Example                                                                                                                                                                                                                                                                   |
|----------------------------------------------|-----------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                              |                                                                                         | <pre> overlays:     #@ load("@ytt:overlay", "overlay")   #@ load("@ytt:data", "data")   #@overlay/match by=overlay.subset({"apiVersion":"apps/v1", "kind": "Deployment"})   ---   spec:     #@overlay/match missing_ok=True     replica s: #@ data.values.replicas </pre> |
| <code>carvel_package_openapi3_enabled</code> | Specifies whether the Carvel Package should include a generated OpenAPIv3 specification | <pre> - name: carvel_package_openapi3_enabled   value: true </pre>                                                                                                                                                                                                        |

### More information

To read more about `lifecycle:tekton`, read [Cartographer Lifecycle](#).

## package-config-writer-template (experimental)

### Purpose

Persist in an external git repository the Carvel Package Kubernetes configuration passed to the template.

### Used by

- [Source-to-URL-Package \(experimental\)](#) in the config-writer step.
- [Basic-Image-to-URL-Package \(experimental\)](#) in the config-writer step.

### Creates

A runnable which creates a Tekton TaskRun that refers either to the Tekton Task `git-writer`.

### Parameters

| Parameter name                        | Meaning                                                                                                                                                         | Example                                                                                      |
|---------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------|
| <code>serviceAccount</code>           | Name of the service account which provides the credentials to the registry or repository. The service account must exist in the same namespace as the Workload. | <pre>- name: serviceAccount   value: default</pre>                                           |
| <code>gitops_branch</code>            | Name of the branch to push the configuration to.                                                                                                                | <pre>- name: gitops_branch   value: main</pre>                                               |
| <code>gitops_user_name</code>         | User name to use in the commits.                                                                                                                                | <pre>- name: gitops_user_name   value: "Alice Lee"</pre>                                     |
| <code>gitops_user_email</code>        | User email address to use in the commits.                                                                                                                       | <pre>- name: gitops_user_email   value: alice@example.com</pre>                              |
| <code>gitops_commit_message</code>    | Message to write as the body of the commits produced for pushing configuration to the Git repository.                                                           | <pre>- name: gitops_commit_message   value: "ci bump"</pre>                                  |
| <code>gitops_repository</code>        | The full repository URL to which the configuration is committed. <b>DEPRECATED</b>                                                                              | <pre>- name: gitops_repository   value: "https://github.com/vmware-tanzu/cartographer"</pre> |
| <code>gitops_repository_prefix</code> | The prefix of the repository URL. <b>DEPRECATED</b>                                                                                                             | <pre>- name: gitops_repository   value: "https://github.com/vmware-tanzu/"</pre>             |
| <code>gitops_server_address</code>    | The server URL of the Git repository to which configuration is applied.                                                                                         | <pre>- name: gitops_server_address   value: "https://github.com/"</pre>                      |

| Parameter name                             | Meaning                                                                                                                                                                                                                      | Example                                                                                                                                                                                                                                                                              |
|--------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>gitops_repository_owner</code>       | The owner/organization to which the repository belongs.                                                                                                                                                                      | <pre>- name: gitops   _repository_owner     value: vmware-tanzu</pre>                                                                                                                                                                                                                |
| <code>gitops_repository_name</code>        | The name of the repository.                                                                                                                                                                                                  | <pre>- name: gitops   _repository_name     value: carto-grapher</pre>                                                                                                                                                                                                                |
| <code>registry</code>                      | Specification of the registry server and repository in which the configuration is placed.                                                                                                                                    | <pre>- name: registry   value:     server: index.docker.io     repository: web-team     ca_certificate:       -----BEGIN       IN CERTIFICATE-----       MIIFXzCC       A0egAwIBAgIJAJYm37SF       ocjlMA0GCSqGSIb3DQEB       DQUAMEY...       -----END       CERTIFICATE-----</pre> |
| <code>carvel_package_gitops_subpath</code> | Specifies the subpath to which Carvel Packages should be written.                                                                                                                                                            | <pre>- name: carvel   _package_gitops_subpath     value: path/to/my/dir</pre>                                                                                                                                                                                                        |
| <code>carvel_package_name_suffix</code>    | Specifies the suffix to append to the Carvel Package name. The format is WORKLOAD_NAME.WORKLOAD_NAMESPACE.carvel_package_name_suffix The full Carvel Package name must be a valid DNS subdomain name as defined in RFC 1123. | <pre>- name: carvel   _package_name_suffix     value: vmware.com</pre>                                                                                                                                                                                                               |

## More information

See [Gitops vs RegistryOps](#) for more information about the operation of this template and of the [package-config-writer-and-pull-requester-template \(experimental\)](#).

## package-config-writer-and-pull-requester-template (experimental)

## Purpose

Persist the passed in Carvel Package Kubernetes configuration to a branch in a repository and open a pull request to another branch. (This process allows for manual review of configuration before deployment to a cluster)

## Used by

- [Source-to-URL-Package \(experimental\)](#) in the config-writer step.
- [Basic-Image-to-URL-Package \(experimental\)](#) in the config-writer step.

## Creates

A runnable which provides configuration to the ClusterRunTemplate `commit-and-pr-pipelinerun` to create a Tekton TaskRun. The Tekton TaskRun refers to the Tekton Task `commit-and-pr`.

## Parameters

| Parameter name                    | Meaning                                                                                                                                                         | Example                                                             |
|-----------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------|
| <code>serviceAccount</code>       | Name of the service account which provides the credentials to the registry or repository. The service account must exist in the same namespace as the Workload. | <pre>- name:   serviceAccount   value:   default</pre>              |
| <code>gitops_commit_branch</code> | Name of the branch to which configuration is pushed.                                                                                                            | <pre>- name:   gitops_commit_branch   value:   feature</pre>        |
| <code>gitops_branch</code>        | Name of the branch to which a pull request is opened.                                                                                                           | <pre>- name:   gitops_branch   value:   main</pre>                  |
| <code>gitops_user_name</code>     | User name to use in the commits.                                                                                                                                | <pre>- name:   gitops_user_name   value:   "Alice Lee"</pre>        |
| <code>gitops_user_email</code>    | User email address to use in the commits.                                                                                                                       | <pre>- name:   gitops_user_email   value:   alice@example.com</pre> |



| Parameter name                         | Meaning                                                                                               | Example                                                                                |
|----------------------------------------|-------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------|
| <code>gitops_commit_message</code>     | Message to write as the body of the commits produced for pushing configuration to the Git repository. | <pre>- name:   gitops_commit_message     value:   "ci bump"</pre>                      |
| <code>gitops_pull_request_title</code> | Title of the pull request to be opened.                                                               | <pre>- name:   gitops_pull_request_title     value:   "ready for review"</pre>         |
| <code>gitops_pull_request_body</code>  | Body of the pull request to be opened.                                                                | <pre>- name:   gitops_pull_request_body     value:   "generated by supply chain"</pre> |
| <code>gitops_server_address</code>     | The server URL of the Git repository to which configuration is applied.                               | <pre>- name:   gitops_server_address     value:   "https://github.com/"</pre>          |
| <code>gitops_repository_owner</code>   | The owner/organization to which the repository belongs.                                               | <pre>- name:   gitops_repository_owner     value:   vmware-tanzu</pre>                 |
| <code>gitops_repository_name</code>    | The name of the repository.                                                                           | <pre>- name:   gitops_repository_name     value:   cartographer</pre>                  |
| <code>gitops_server_kind</code>        | The kind of Git provider                                                                              | <pre>- name:   gitops_server_kind     value:   gitlab</pre>                            |

| Parameter name                                       | Meaning                                                                                                                                                                                                                                     | Example                                                                            |
|------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------|
| <code>carvel_package<br/>_gitops_subpat<br/>h</code> | Specifies the subpath to which Carvel Packages should be written.                                                                                                                                                                           | <pre>- name:   carvel_package   _gitops_subpat   h   value:   path/to/my/dir</pre> |
| <code>carvel_package<br/>_name_suffix</code>         | Specifies the suffix to append to the Carvel Package name. The format is <code>WORKLOAD_NAME.WORKLOAD_NAMESPACE.carvel_package_name_suffix</code> . The full Carvel Package name must be a valid DNS subdomain name as defined in RFC 1123. | <pre>- name:   carvel_package   _name_suffix   value:   vmware.com</pre>           |

## More information

See [Gitops vs RegistryOps](#) for more information about the operation of this template and of the [package-config-writer-template \(experimental\)](#).

## ClusterRunTemplate reference

All the objects referenced in this topic are [Cartographer ClusterRunTemplates](#) packaged in [Out of the Box Templates](#). This topic describes the one or more objects they create, the supply chains that include them, and the parameters they use.

## tekton-source-pipelinerun

### Purpose

Tests source code.

### Used by

- [testing-pipeline](#)

### Creates

This ClusterRunTemplate creates a [Tekton PipelineRun](#) referring to the user's Tekton Pipeline.

### Inputs

## ClusterRunTemplate reference

| Input name    | Meaning                                          | Example                                                                                                                                                                                                                                        |
|---------------|--------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| tekton-params | Set of parameters to pass to the Tekton Pipeline | <pre> - name: source-url   value: https://github.com/vmware-tanzu/cartographer.git - name: source-revision   value: e4a53f49a92fc913d26f8cc23d59102a51a5e635 - name: verbose   value: true - name: foo   value: bar                     </pre> |

## More information

For information about the runnable created in the OOTB Testing and OOTB Testing and Scanning, see [testing-pipeline](#).

For information about the Tekton Pipeline that the user must create, see [Tekton/Pipeline](#).

## tekton-taskrun

### Purpose

Generic template for creating a Tekton TaskRun.

### Used by

- [config-writer-template](#)

### Creates

A Tekton TaskRun.

### Inputs

| Input name     | Meaning                                                               | Example                                                                                                     |
|----------------|-----------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------|
| serviceAccount | Service Account with permissions necessary for the Tekton Task        | default                                                                                                     |
| taskRef        | Reference to the Tekton Task to which the TaskRun provides parameters | <pre> kind: ClusterTask name: git-writer                     </pre>                                         |
| params         | Parameters which are provided to the Tekton Task                      | <pre> - name: git_branch   value: main - name: git_username   value: "Some Name"                     </pre> |

## commit-and-pr-pipelinerun

## Purpose

Commit configuration to a Git repository and open a pull request for review.

## Used by

- [config-writer-and-pull-requester-template](#)

## Creates

Creates a Tekton TaskRun referring to the `commit-and-pr` Tekton Task.

## Inputs

| Input name                      | Meaning                                                                                    | Example                                                                                                |
|---------------------------------|--------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|
| <code>serviceAccount</code>     | Service Account with credentials for the Git repository                                    | default                                                                                                |
| <code>git_server_kind</code>    | Type of Git provider                                                                       | github                                                                                                 |
| <code>git_server_address</code> | Server URL                                                                                 | https://github.com                                                                                     |
| <code>repository_owner</code>   | Owner or Organization in which the repository resides                                      | vmware-tanzu                                                                                           |
| <code>repository_name</code>    | Name of the repository                                                                     | cartographer                                                                                           |
| <code>commit_branch</code>      | Name of the commit branch.<br>Recommended value is an empty string.                        | ""                                                                                                     |
| <code>pull_request_title</code> | Title of the PR to be opened                                                               | "Update"                                                                                               |
| <code>pull_request_body</code>  | Body of the PR to be opened                                                                | "Ready for review"                                                                                     |
| <code>base_branch</code>        | Branch into which the PR is merged                                                         | main                                                                                                   |
| <code>git_username</code>       | User name associated with the commit                                                       | Waciuma Rasheed                                                                                        |
| <code>git_user_email</code>     | User email associated with the commit                                                      | Sam@todd.com                                                                                           |
| <code>git_commit_message</code> | Message on commit                                                                          | "App update"                                                                                           |
| <code>git_files</code>          | Base64 encoded JSON file where keys equal the filename and the value is the file contents. | "eyJkZWxpdmVyeS55bWwiOiJhcGlWZXJzaW9uOiBzZXJ2aW5nLmtuYXRpdmUuZGV2L3YxXG5raW5kOiBTZXJ2aW5kOjZlXG4ifQ==" |

---

`sub_path` The directory location in the repository in which to write the files.

---

## More information

For information about the template creating the related runnable, see [config-writer-and-pull-requester-template](#).

For information about gitops, see [GitOps versus RegistryOps](#).

## Delivery reference

Tanzu Application Platform delivery package installs a single [ClusterDelivery](#).

The delivery provides some parameters to the templates. The parameters provided by the deliverable might override some of the delivery parameters in this topic. See [parameters](#) in the Cartographer documentation.

## delivery-basic

### Purpose

- Fetches Kubernetes configuration created by a supply chain.
- Deploys the configuration on the cluster.

### Resources

The following resources describe the templates.

#### source-provider

Refers to [delivery-source-template](#).

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by deliverable.
- `gitImplementation` from tap-value `git_implementation`. Not overridable by deliverable.

#### Deployer

Refers to [app-deploy](#) template.

Parameter provided:

- `serviceAccount` from tap-value `service_account`. Overridable by deliverable.

### Package

Refers to [Out of the Box Delivery Basic](#).

## More information

For information about setting `tap-values.yaml` at installation time, see [Install Out of the Box Delivery Basic](#).

## Git

The out of the box supply chains and delivery use Git in 3 ways:

- To fetch the developers source code, using the [template](#).
- To store complete Kubernetes configuration, the “write” side of gitops, use [template 1](#) and [template 2](#).
- To fetch stored Kubernetes configuration, the read side of gitops, from either the same or a different Kubernetes cluster, use the [template](#).

## Supported Git Repositories

Tanzu Application Platform supports two Git providers:

- GitHub
- GitLab

## Related Articles

[Git Authentication](#): walks through the objects, such as secrets and service accounts, to create on cluster to allow supply chain Git operations to succeed.

[GitOps versus RegistryOps](#): discusses the two methods of storing built Kubernetes configuration, either in a git repository or an image registry, and walks through the parameters that must be provided for each.

## GitOps versus 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 a container image registry.



### Note

For more information about providing source code either from a local directory or Git repository, see [Building from Source](#).

## GitOps

The GitOps approach differs from local iteration in that GitOps configures the supply chains to push the Kubernetes configuration to a remote Git repository. This allows users to compare configuration changes and promote those changes through environments by using GitOps principles.

Typically associated with an outerloop workflow, the GitOps approach is only activated if a collection of parameters are set:

- `gitops.server_address` during the Out of the Box Supply Chains package installation or `gitops_server_address` configured as a workload parameter.
- `gitops.repository_owner` during the Out of the Box Supply Chains package installation or `gitops_repository_owner` configured as a workload parameter.
- `gitops.repository_name` during the Out of the Box Supply Chains package installation or `gitops_repository_name` configured as a workload parameter.

With all three values set, Kubernetes configuration is written to the specified repository. If a value is set at installation and the corresponding workload parameter is also set, the value of the workload parameter is respected.

In the repository, files are located in the `./config/{workload-namespace}/{workload-name}` directory. This allows multiple workloads to commit configuration to the same repository.

## Examples

---

`tap-values.yaml`

```
gitops:
  server_address:
  repository_owner:
  repository_name:
```

`workload`

```
name: incrediApp
namespace: awesomeTeam
params:
  - name: gitops_server_address
    value: https://github.com/
  - name: gitops_repository_owner
    value: vmware-tanzu
  - name: gitops_repository_name
    value: cartographer
```

Resulting gitops repository: <https://github.com/vmware-tanzu/cartographer>

Directory containing configuration: `./config/awesomeTeam/incrediApp`

---

`tap-values.yaml`

```
gitops:
  server_address: https://github.com/
  repository_owner: vmware-tanzu
  repository_name: cartographer
```

`workload`

```
name: superApp
namespace: awesomeTeam
```

Resulting gitops repository: <https://github.com/vmware-tanzu/cartographer>

Directory containing configuration: `./config/awesomeTeam/superApp`

---

`tap-values.yaml`

```
gitops:
  server_address: https://github.com/
  repository_owner: vmware-tanzu
```

#### workload

```
name: superApp
namespace: awesomeTeam
params:
  - name: gitops_repository_owner
    value: pivotal
  - name: gitops_repository_name
    value: kpack
```

Resulting gitops repository: <https://github.com/pivotal/kpack>

Directory containing configuration: `./config/awesomeTeam/superApp`

---

#### tap-values.yaml

```
gitops:
  server_address:
  repository_owner:
  repository_name:
```

#### workload

```
name: superApp
namespace: awesomeTeam
params:
  - name: gitops_repository_owner
    value: pivotal
  - name: gitops_repository_name
    value: kpack
```

Resulting gitops repository: Fails to resolve as some, but not all, of the three required values are provided.

---

## Deprecated parameters

The following parameters are deprecated and no longer recommended for specifying gitops repositories:

- `gitops.repository_prefix`: configured during the Out of the Box Supply Chains package installation.
- `gitops_repository`: configured as a workload parameter.

For example, assuming the installation of the supply chain packages through Tanzu Application Platform profiles and a `tap-values.yaml`:

```
ootb_supply_chain_basic:
  registry:
    server: REGISTRY-SERVER
    repository: REGISTRY-REPOSITORY

gitops:
  repository_prefix: https://github.com/my-org/
```

Workloads in the cluster with the Kubernetes configuration produced throughout the supply chain are pushed to the repository whose name is formed by concatenating `gitops.repository_prefix`



with the name of the workload. In this case, for example, [https://github.com/my-org/\\${workload.metadata.name}.git](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 ${GIT_PROJECT_URL} \
  --git-branch ${GIT_BRANCH} \
  --param gitops_ssh_secret=GIT-SECRET-NAME \
  --param gitops_repository=https://github.com/my-org/config-repo
```

In this case, at the end of the supply chain, the configuration for this workload is published to the repository provided under the `gitops_repository` parameter.

If you use deprecated parameters, Kubernetes configuration is committed to the `./config` directory in the repository. This can lead to collisions if two workloads specify the same repository, or two workloads in different namespaces have the same name and the `gitops.repository_prefix` is set in `tap-values.yaml`.

If the deprecated values are set and any of the suggested gitops values are set, the deprecated values are ignored.

## Examples

`tap-values.yaml`

```
gitops:
  repository_prefix: https://github.com/vmware-tanzu
```

`workload`

```
name: superApp
namespace: awesomeTeam
```

Resulting gitops repository: <https://github.com/vmware-tanzu/incrediApp>

Directory containing configuration: `./config`

`tap-values.yaml`

```
gitops:
  server_address: https://github.com/
```

```
repository_owner: vmware-tanzu
repository_name: cartographer
```

#### workload

```
name: superApp
namespace: awesomeTeam
params:
  - name: gitops_repository
    value: https://github.com/pivotal/kpack
```

Resulting gitops repository: <https://github.com/vmware-tanzu/cartographer> (The deprecated param `gitops_repository` is ignored.)

Directory containing configuration: `./config/awesomeTeam/superApp`

---

#### tap-values.yaml

```
gitops:
  repository_prefix: https://github.com/vmware-tanzu
```

#### workload

```
name: superApp
namespace: awesomeTeam
params:
  - name: gitops_repository_owner
    value: pivotal
  - name: gitops_repository_name
    value: kpack
```

Resulting gitops repository: Fails to resolve as some, but not all, of the three gitops values are provided. (The deprecated value `repository_prefix` is ignored because suggested values are present)

---

## Pull requests

In the standard `git-ops` approach, configuration is pushed to a repository and is immediately applied to a cluster by any deliverable watching that repository. Operators might want to manually review configuration before applying it to the cluster. To do this, operators must specify a `pull_request` commit strategy. You can use this strategy with the following Git providers:

- GitHub
- GitLab

### Authentication

The pull request approach requires HTTP(S) authentication with a token.

The pull request function is not a part of the Git specification, but most Git server providers include it. You must authenticate with those providers using a token.

In the [Kubernetes secret](#) that holds the Git credentials, the password text box must contain a token. When generating a token, ensure that it has the proper scope:

- On GitHub, the token must have a [Repo scope](#).
- On GitLab, the token must have an [API scope](#).

To use the `pull_request` commit strategy, set the following parameters:

- `commit_strategy == pull_request` configured during the Out of the Box Supply Chains package installation.
- `gitops.pull_request.server_kind` configured during the Out of the Box Supply Chains package installation or `gitops_server_kind` configured as a workload parameter. Supported values are `github` and `gitlab`.
- `gitops.pull_request.commit_branch` configured during the Out of the Box Supply Chains package installation or `gitops_commit_branch` configured as a workload parameter.
- `gitops.pull_request.pull_request_title` configured during the Out of the Box Supply Chains package installation or `gitops_pull_request_title` configured as a workload parameter.
- `gitops.pull_request.pull_request_body` configured during the Out of the Box Supply Chains package installation or `gitops_pull_request_body` configured as a workload parameter.

If a value is set at both installation and in a workload parameter, the workload parameter is respected.

The recommended value for `commit_branch` is an empty string. This generates a new branch for each commit based on a hash of the time when the commit is created. This prevents collisions between multiple workloads using a single Git repository.

For example, using the following Tanzu Application Platform values:

```
ootb_supply_chain_basic:
  gitops:
    server_address: https://github.com/
    repository_owner: vmware-tanzu
    repository_name: cartographer
    branch: main
    commit_strategy: pull_request
    pull_request:
      server_kind: github
      commit_branch: ""
      pull_request_title: ready for review
      pull_request_body: generated by supply chain
```

In a workload with the name `app` in the `dev` namespace, you find:

A commit to the `https://github.com/vmware-tanzu/cartographer` repository on a branch with a random name. For example, `MTYlMTYxMzE0NQo=`. There is a pull request open to merge this branch into the base branch `main`.

## Authentication

Regardless of how the supply chains are configured, if the repository prefix or repository name is configured to push to Git, you must provide credentials for the remote provider by using a Kubernetes secret in the same namespace as the workload attached to the workload `ServiceAccount`.

Because the operation of pushing requires elevated permissions, credentials are required by both public and private repositories.

### HTTP(S) Basic-auth or Token-based authentication

If the repository at which configuration is published uses `https://` or `http://` as the URL scheme, the Kubernetes secret must provide the credentials for that repository as follows:

```

apiVersion: v1
kind: Secret
metadata:
  name: GIT-SECRET-NAME # `git-ssh` is the default name.
                        # - operators can change such default by using the
                        #   `gitops.ssh_secret` property in `tap-values.yaml`
                        # - developers can override by using the workload parameter
                        #   named `gitops_ssh_secret`.
  annotations:
    tekton.dev/git-0: GIT-SERVER # ! required
type: kubernetes.io/basic-auth # ! required
stringData:
  username: GIT-USERNAME
  password: GIT-PASSWORD

```

Both the Tekton annotation and the `basic-auth` secret type must be set. `GIT-SERVER` must be prefixed with the appropriate URL scheme and the Git server. For example, for <https://github.com/vmware-tanzu/cartographer>, <https://github.com> must be provided as the `GIT-SERVER`.

To use the pull request approach, the password text box must contain a token. See [Pull Requests](#).

After the `Secret` is created, attach it to the `ServiceAccount` used by the workload. For example:

```

apiVersion: v1
kind: ServiceAccount
metadata:
  name: default
secrets:
  - name: registry-credentials
  - name: tap-registry
  - name: GIT-SECRET-NAME
imagePullSecrets:
  - name: registry-credentials
  - name: tap-registry

```

For more information about the credentials and setting up the Kubernetes secret, see [Git Authentication's HTTP section](#).

## SSH

If the repository to which configuration is published uses `https://` or `http://` as the URL scheme, the Kubernetes secret must provide the credentials for that repository as follows:

```

apiVersion: v1
kind: Secret
metadata:
  name: GIT-SECRET-NAME # `git-ssh` is the default name.
                        # - operators can change such default through the
                        #   `gitops.ssh_secret` property in `tap-values.yaml`
                        # - developers can override by using the workload parameter
                        #   named `gitops_ssh_secret`.
  annotations:
    tekton.dev/git-0: GIT-SERVER
type: kubernetes.io/ssh-auth
stringData:
  ssh-privatekey: SSH-PRIVATE-KEY # private key with push-permissions
  identity: SSH-PRIVATE-KEY # private key with pull permissions
  identity.pub: SSH-PUBLIC-KEY # public of the `identity` private key
  known_hosts: GIT-SERVER-PUBLIC-KEYS # git server public keys

```

After the `Secret` is created, attach it to the `ServiceAccount` used by the workload. For example:

```

apiVersion: v1
kind: ServiceAccount
metadata:
  name: default
secrets:
  - name: registry-credentials
  - name: tap-registry
  - name: GIT-SECRET-NAME
imagePullSecrets:
  - name: registry-credentials
  - name: tap-registry

```

For information about the credentials and setting up the Kubernetes secret, see [Git Authentication's SSH section](#).

## GitOps workload parameters

While installing `ootb-*`, operators can configure `gitops.repository_prefix` to indicate what prefix the supply chain must use when forming the name of the repository to push to the Kubernetes configurations produced by the supply chains.

To change the behavior to use GitOps, set the source of the source code to a Git repository. As the supply chain progresses, configuration is pushed to a repository named

`$(gitops.repository_prefix) + $(workload.name)`.

For example, configure `gitops.repository_prefix` to `git@github.com/foo/` and create a workload as follows:

```

tanzu apps workload create tanzu-java-web-app \
  --git-repo ${GIT_PROJECT_URL} \
  --git-branch ${GIT_BRANCH} \
  --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/my/tanzu-project.git

```

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`

## Read more on Git

See [Git Reference](#)

## RegistryOps

RegistryOps is typically used for inner loop flows where configuration is treated as an artifact from quick iterations by developers. In this scenario, at the end of the supply chain, configuration is pushed to a container image registry in the form of an [imgpkg bundle](#). You can think of it as a container image whose sole purpose is to carry arbitrary files.

To enable this mode of operation, the supply chains must be configured **without** the following parameters being configured during the installation of the `ootb-` packages or overwritten by the workload by using the following parameters:

- `gitops_repository_prefix`
- `gitops_repository`

If none of the parameters are set, the configuration is pushed to the same container image registry as the application image. That is, to the registry configured under the `registry: {}` section of the `ootb-` values.

For example, assuming the installation of Tanzu Application Platform by using profiles, configure the `ootb-supply-chain*` package as follows:

```
ootb_supply_chain_basic:
  registry:
    server: REGISTRY-SERVER
    repository: REGISTRY_REPOSITORY
```

The Kubernetes configuration produced by the supply chain is pushed to an image named after `REGISTRY-SERVER/REGISTRY-REPOSITORY` including the workload name.

In this scenario, no extra credentials must be set up, because the secret containing the credentials for the container image registry were already configured during the setup of the workload namespace.

## Overview of Supply Chain Security Tools - Scan

This topic describes overview information, such as, use cases, features, and CVEs for 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.

## Language support

For information about the languages and frameworks that are supported by Tanzu Application Platform components, see the [Language and framework support in Tanzu Application Platform](#) table.

## Use cases

The following use cases apply to Supply Chain Security Tools - Scan:

- Use your scanner as a plug-in to scan source code repositories and images for known Common Vulnerabilities and Exposures (CVEs) before deploying to a cluster.
- Identify CVEs by continuously scanning each new code commit or each new image built.
- Analyze scan results against user-defined policies by using Open Policy Agent.
- Produce vulnerability scan results and post them to the 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 Gype.
- CRD for policy enforcement.
- Enhanced scanning coverage by analyzing the Cloud Native Buildpack SBoMs that Tanzu Build Service images provide.

## A Note on Vulnerability Scanners

Although vulnerability scanning is an important practice in DevSecOps and the benefits of it are widely recognized and accepted, it is important to remember that there are limitations present that impact its efficacy. The following examples illustrate the limitations that are prevalent in most scanners today:

### Missed CVEs

One limitation of all vulnerability scanners is that there is no one tool that can find 100% of all CVEs, which means there is always a risk that a missed CVE can be exploited. Some reasons for missed CVEs include:

- The scanner does not detect the vulnerability because it is just discovered and the CVE databases that the scanner checks against are not updated yet.
- Scanners verify different CVE sources based on the detected package type and OS.
- The scanner might not fully support a particular programming language, packaging system or manifest format.
- The scanner might not implement binary analysis or fingerprinting.

- The detected component does not always include a canonical name and vendor, requiring the scanner to infer and attempt fuzzy matching.
- When vendors register impacted software with NVD, the provided information might not exactly match the values in the release artifacts.

### False positives

Vulnerability scanners cannot always access the information to accurately identify whether a CVE exists. This often leads to an influx of false positives where the tool mistakenly flags something as a vulnerability when it isn't. Unless a user is specialized in security or is deeply familiar with what is deemed to be a vulnerable component by the scanner, assessing and determining false positives becomes a challenging and time-consuming activity. Some reasons for a false positive flag include:

- A component might be misidentified due to similar names.
- A subcomponent might be identified as the parent component.
- A component is correctly identified but the impacted function is not on a reachable code path.
- A component's impacted function is on a reachable code path but is not a concern due to the specific environment or configuration.
- The version of a component might be incorrectly flagged as impacted.
- The detected component does not always include a canonical name and vendor, requiring the scanner to infer and attempt fuzzy matching.

So what can you do to protect yourselves and your software?

Although vulnerability scanning is not a perfect solution, it is an essential part of the process for keeping your organization secure. You can take the following measures to maximize the benefits while minimizing the impact of the limitations:

- Scan more continuously and comprehensively to identify and remediate zero-day vulnerabilities quicker. Comprehensive scanning can be achieved by:
  - scanning earlier in the development cycle to ensure issues can be addressed more efficiently and do not delay a release. Tanzu Application Platform includes security practices such as source and container image vulnerability scanning earlier in the path to production for application teams.
  - scanning any base images in use. Tanzu Application Platform image scanning includes the ability to recognize and scan the OS packages from a base image.
  - scanning running software in test, stage, and production environments at a regular cadence.
  - generating accurate provenance at any level so that scanners have a complete picture of the dependencies to scan. This is where a software bill of materials (SBoM) comes into play. To help you automate this process, VMware Tanzu Build Service, leveraging Cloud Native Buildpacks, generates an SBoM for buildpack-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.

## Overview of Supply Chain Security Tools - Scan

This topic describes overview information, such as, use cases, features, and CVEs for 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.

### Language support

For information about the languages and frameworks that are supported by Tanzu Application Platform components, see the [Language and framework support in Tanzu Application Platform](#) table.

### Use cases

The following use cases apply to Supply Chain Security Tools - Scan:

- Use your scanner as a plug-in to scan source code repositories and images for known Common Vulnerabilities and Exposures (CVEs) before deploying to a cluster.
- Identify CVEs by continuously scanning each new code commit or each new image built.
- Analyze scan results against user-defined policies by using Open Policy Agent.
- Produce vulnerability scan results and post them to the 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 cannot always access the information to accurately identify whether a CVE exists. This often leads to an influx of false positives where the tool mistakenly flags something as a vulnerability when it isn't. Unless a user is specialized in security or is deeply familiar with what is deemed to be a vulnerable component by the scanner, assessing and determining false positives becomes a challenging and time-consuming activity. Some reasons for a false positive flag include:

- A component might be misidentified due to similar names.
- A subcomponent might be identified as the parent component.
- A component is correctly identified but the impacted function is not on a reachable code path.
- A component's impacted function is on a reachable code path but is not a concern due to the specific environment or configuration.
- The version of a component might be incorrectly flagged as impacted.
- The detected component does not always include a canonical name and vendor, requiring the scanner to infer and attempt fuzzy matching.

So what can you do to protect yourselves and your software?

Although vulnerability scanning is not a perfect solution, it is an essential part of the process for keeping your organization secure. You can take the following measures to maximize the benefits while minimizing the impact of the limitations:

- Scan more continuously and comprehensively to identify and remediate zero-day vulnerabilities quicker. Comprehensive scanning can be achieved by:
  - scanning earlier in the development cycle to ensure issues can be addressed more efficiently and do not delay a release. Tanzu Application Platform includes security practices such as source and container image vulnerability scanning earlier in the path to production for application teams.
  - scanning any base images in use. Tanzu Application Platform image scanning includes the ability to recognize and scan the OS packages from a base image.

- scanning running software in test, stage, and production environments at a regular cadence.
- generating accurate provenance at any level so that scanners have a complete picture of the dependencies to scan. This is where a software bill of materials (SBoM) comes into play. To help you automate this process, VMware Tanzu Build Service, leveraging Cloud Native Buildpacks, generates an SBoM for buildpack-based projects. Since this SBoM is generated during the image building stage, it is more accurate and complete than one generated earlier or later in the release life cycle. This is because it can highlight dependencies introduced at the time of build that might introduce potential for compromise.
- Scan by using multiple scanners to maximize CVE coverage.
- Practice keeping your dependencies up-to-date.
- Reduce overall surface area of attack by:
  - using smaller dependencies.
  - reducing the amount of third party dependencies when possible.
  - using distroless base images when possible.
- Maintain a central record of false positives to ease CVE triaging and remediation efforts.

## Install Supply Chain Security Tools - Scan

This topic describes how you can install Supply Chain Security Tools - Scan from the Tanzu Application Platform package repository.



### Note

Follow the steps in this topic if you do not want to use a profile to install Supply Chain Security Tools - Scan. For more information about profiles, see [Components and installation profiles](#).

## Prerequisites

Before installing SCST - Scan:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see [Prerequisites](#).
- Install [Supply Chain Security Tools - Store](#) for scan results to persist. The integration with SCST - Store are handled in:
  - **Single Cluster:** The SCST - Store is present in the same cluster where SCST - Scan and the `ScanTemplates` are present.
  - **Multi-Cluster:** The SCST - Store is present in a different cluster (e.g.: view cluster) where the SCST - Scan and `ScanTemplates` are present.
  - **Integration Deactivated:** The SCST - Scan deployment is not required to communicate with SCST - Store.

For information about SCST - Store, see [Using the Supply Chain Security Tools - Store](#).

## Configure properties

When you install the SCST - Scan (Scan controller), you can configure the following optional properties:

| Key                                          | Default                                                          | Type            | Description                                                                                                       | ScanTemplate Version |
|----------------------------------------------|------------------------------------------------------------------|-----------------|-------------------------------------------------------------------------------------------------------------------|----------------------|
| resources.limits.cpu                         | 250m                                                             | integer /string | Limits describes the maximum amount of CPU resources allowed.                                                     | <i>n/a</i>           |
| resources.limits.memory                      | 256Mi                                                            | integer /string | Limits describes the maximum amount of memory resources allowed.                                                  | <i>n/a</i>           |
| resources.requests.cpu                       | 100m                                                             | integer /string | Requests describes the minimum amount of CPU resources required.                                                  | <i>n/a</i>           |
| resources.requests.memory                    | 128Mi                                                            | integer /string | Requests describes the minimum amount of memory resources required.                                               | <i>n/a</i>           |
| namespace                                    | scan-link-system                                                 | string          | Deployment namespace for the Scan Controller                                                                      | <i>n/a</i>           |
| metadataStore.caSecret.importFromNamespace   | metadata-store                                                   | string          | Namespace from which you import the Insight Metadata Store CA Cert                                                | earlier than v1.2.0  |
| metadataStore.caSecret.name                  | app-tls-cert                                                     | string          | Name of deployed secret with key ca.crt holding the CA Cert of the Insight Metadata Store                         | earlier than v1.2.0  |
| metadataStore.clusterRole                    | metadata-store-read-write                                        | string          | Name of the deployed ClusterRole for read/write access to the Insight Metadata Store deployed in the same cluster | earlier than v1.2.0  |
| metadataStore.url                            | https://metadata-store-app.metadata-store.svc.cluster.local:8443 | string          | URL of the Insight Metadata Store                                                                                 | earlier than v1.2.0  |
| metadataStore.authSecret.importFromNamespace | <i>n/a</i>                                                       | string          | Namespace from which to import the Insight Metadata Store auth_token                                              | earlier than v1.2.0  |
| metadataStore.authSecret.name                | <i>n/a</i>                                                       | string          | Name of deployed secret with key auth_token                                                                       | earlier than v1.2.0  |
| retryScanJobsSecondsAfterError               | 60                                                               | integer         | Seconds to wait before retrying errored scans                                                                     | v1.3.1 and later     |
| caCertData                                   | ""                                                               | string          | The custom certificates trusted by the scans' connections.                                                        | v1.4.0 and later     |

When you install the SCST - Scan (Grype scanner), you can configure the following optional properties:

| Key                       | Default       | Type            | Description                                                         | ScanTemplate Version |
|---------------------------|---------------|-----------------|---------------------------------------------------------------------|----------------------|
| resources.requests.cpu    | 250m          | integer /string | Requests describes the minimum amount of CPU resources required.    |                      |
| resources.requests.memory | 128Mi         | integer /string | Requests describes the minimum amount of memory resources required. |                      |
| scanner.serviceAccount    | grype-scanner | string          | Name of scan pod's service ServiceAccount                           |                      |

| Key                                          | Default                                                          | Type   | Description                                                                                                       | ScanTemplate Version |
|----------------------------------------------|------------------------------------------------------------------|--------|-------------------------------------------------------------------------------------------------------------------|----------------------|
| scanner.serviceAccountAnnotations            | nil                                                              | object | Annotations added to ServiceAccount                                                                               |                      |
| targetImagePullSecret                        | <i>n/a</i>                                                       | string | Reference to the secret used for pulling images from private registry                                             |                      |
| targetSourceSshSecret                        | <i>n/a</i>                                                       | string | Reference to the secret containing SSH credentials for cloning private repositories                               |                      |
| namespace                                    | default                                                          | string | Deployment namespace for the Scan Templates                                                                       | <i>n/a</i>           |
| metadataStore.url                            | https://metadata-store-app.metadata-store.svc.cluster.local:8443 | string | URL of the Insight Metadata Store                                                                                 | v1.2.0 and earlier   |
| metadataStore.authSecret.name                | <i>n/a</i>                                                       | string | Name of deployed secret with key auth_token                                                                       | v1.2.0 and earlier   |
| metadataStore.authSecret.importFromNamespace | <i>n/a</i>                                                       | string | Namespace from which to import the Insight Metadata Store auth_token                                              | v1.2.0 and earlier   |
| metadataStore.caSecret.importFromNamespace   | metadata-store                                                   | string | Namespace from which to import the Insight Metadata Store CA Cert                                                 | v1.2.0 and earlier   |
| metadataStore.caSecret.name                  | app-tls-cert                                                     | string | Name of deployed secret with key ca.crt holding the CA Cert of the Insight Metadata Store                         | v1.2.0 and earlier   |
| metadataStore.clusterRole                    | metadata-store-read-write                                        | string | Name of the deployed ClusterRole for read/write access to the Insight Metadata Store deployed in the same cluster | v1.2.0               |

## Install

There are two options for installing Supply Chain Security Tools – Scan

### Option 1: Install to multiple namespaces with the Namespace Provisioner

The Namespace Provisioner enables operators to securely automate the provisioning of multiple developer namespaces in a shared cluster. To install Supply Chain Security Tools – Scan by using the Namespace Provisioner, see [Tutorial: Provisioning new developer namespaces](#).

The Namespace Provisioner can also create scan policies across multiple developer namespaces. See [Add the resources required by the Out of the Box Testing and Scanning Supply Chain](#) for configuration steps.

### Option 2: Install manually to each individual namespace

The installation for Supply Chain Security Tools – Scan involves installing two packages:

- Scan controller
- Gype scanner

The Scan controller enables you to use a scanner, in this case, the Grype scanner. Ensure that both the Grype scanner and the Scan controller are installed.

To install SCST - Scan (Scan controller):

1. List version information for the package by running:

```
tanzu package available list scanning.apps.tanzu.vmware.com --namespace tap-ins
tall
```

For example:

```
$ tanzu package available list scanning.apps.tanzu.vmware.com --namespace tap-i
ninstall
/ Retrieving package versions for scanning.apps.tanzu.vmware.com...
NAME                                VERSION      RELEASED-AT
scanning.apps.tanzu.vmware.com      1.1.0
```

2. (Optional) Make changes to the default installation settings:

If you are using Grype Scanner [v1.5.0 and later](#) or other supported scanners included with Tanzu Application Platform [v1.5 and later](#) and do not want to use the default SCST - Store integration, explicitly deactivate the integration by appending the following field to the `values.yaml` file:

```
---
metadataStore: {} # Deactivate Supply Chain Security Tools - Store integration
```

If you are using Grype Scanner [v1.2.0 and earlier](#), or the Snyk Scanner, the following scanning configuration deactivates the embedded SCST - Store integration with a `scan-values.yaml` file.

```
---
metadataStore:
  url: ""
```

If you're using the Grype Scanner [earlier than 1.2.0](#), the scanning configuration must configure the store parameters. See the [v1.1 docs](#) for reference.

You can retrieve any other configurable setting using the following command, and appending the key-value pair to the previous `scan-values.yaml` file:

```
tanzu package available get scanning.apps.tanzu.vmware.com/VERSION --values-sch
ema -n tap-install
```

Where `VERSION` is your package version number. For example, `1.1.0`.

3. Install the package by running:

```
tanzu package install scan-controller \
--package-name scanning.apps.tanzu.vmware.com \
--version VERSION \
--namespace tap-install \
--values-file scan-values.yaml
```

Where `VERSION` is your package version number. For example, `1.1.0`.

To install SCST - Scan (Grype scanner):



#### Note

You can use [Namespace Provisioner](#) to install Grype in multiple namespaces.

1. 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
```

2. (Optional) Make changes to the default installation settings:

To define the configuration for the SCST - Store integration in the `grype-values.yaml` file for the Grype Scanner:

```
---
namespace: "DEV-NAMESPACE" # The developer namespace where the ScanTemplates are gonna be deployed
metadataStore:
  url: "METADATA-STORE-URL" # The base URL where the Store deployment can be reached
  caSecret:
    name: "CA-SECRET-NAME" # The name of the secret containing the ca.crt
    importFromNamespace: "SECRET-NAMESPACE" # The namespace where Store is deployed (if single cluster) or where the connection secrets were created (if multi-cluster)
  authSecret:
    name: "TOKEN-SECRET-NAME" # The name of the secret containing the auth token to connect to Store
    importFromNamespace: "SECRET-NAMESPACE" # The namespace where the connection secrets were created (if multi-cluster)
```

**Note** In a single cluster, the connection between the scanning pod and the metadata store happens inside the cluster and does not pass through ingress. This is automatically configured. You do not need to provide an ingress connection to the store. For information about troubleshooting issues with scanner to metadata store connection configuration, see [Troubleshooting Scanner to MetadataStore Configuration](#).



#### Important

You must either define both the `METADATA-STORE-URL` and `CA-SECRET-NAME`, or not define them as they depend on each other.

Where:

- `DEV-NAMESPACE` is the namespace where you want to deploy the `ScanTemplates`. This is the namespace where the scanning feature runs.
- `METADATA-STORE-URL` is the base URL where the Supply Chain Security Tools (SCST) - Store deployment is reached, for example, `https://metadata-store-app.metadata-store.svc.cluster.local:8443`.
- `CA-SECRET-NAME` is the name of the secret containing the `ca.crt` to connect to the SCST - Store deployment.

- `SECRET-NAMESPACE` is the namespace where SCST - Store is deployed, if you are using a single cluster. If you are using multicluster, it is where the connection secrets were created.
- `TOKEN-SECRET-NAME` is the name of the secret containing the authentication token to connect to the SCST - Store deployment when installed in a different cluster, if you are using multicluster. If built images are pushed to the same registry as the Tanzu Application Platform images, this can reuse the `tap-registry` secret created in [Add the Tanzu Application Platform package repository](#) as described earlier.

You can retrieve any other configurable setting using the following command, and appending the key-value pair to the previous `grype-values.yaml` file:

```
tanzu package available get grype.scanning.apps.tanzu.vmware.com/VERSION --values-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 --values-schema -n tap-install
| Retrieving package details for grype.scanning.apps.tanzu.vmware.com/1.1.0...
  KEY                                DEFAULT  TYPE    DESCRIPTION
  namespace                           default  string  Deployment namespace for the Scan
  Templates
  resources.limits.cpu                 1000m   <nil>   Limits describes the maximum amou
  nt of cpu resources allowed.
  resources.requests.cpu               250m    <nil>   Requests describes the minimum am
  ount of cpu resources required.
  resources.requests.memory            128Mi   <nil>   Requests describes the minimum am
  ount of memory resources required.
  targetImagePullSecret                <EMPTY> string  Reference to the secret used for
  pulling images from private registry.
  targetSourceSshSecret                <EMPTY> string  Reference to the secret containin
  g SSH credentials for cloning private repositories.
```



### Important

If `targetSourceSshSecret` is not set, the private source scan template is not installed.

### 3. Install the package by running:

```
tanzu package install grype-scanner \
  --package-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'

```

## Upgrade Supply Chain Security Tools - Scan

This topic describes how you can upgrade Supply Chain Security Tools - Scan from the Tanzu Application Platform package repository.

You can perform a fresh install of SCST - Scan by following the instructions in [Install Supply Chain Security Tools - Scan](#).

## Prerequisites

Before you upgrade SCST - Scan, upgrade the Tanzu Application Platform by following the instructions in [Upgrading Tanzu Application Platform](#).

## General Upgrades for SCST - Scan

When you're upgrading to any version of SCST - Scan these are some factors to accomplish this task:

1. Inspect the [Release Notes](#) for the version you're upgrading to. There you can find any breaking changes for the installation.
2. Get the values schema for the package version you're upgrading to by running:

```

tanzu package available get scanning.apps.tanzu.vmware.com/$VERSION --values-schema -n tap-install

```

Where `$VERSION` is the new version. This gives you insights on the values you can configure in your `tap-values.yaml` for the new version.

## Upgrading a scanner in all namespaces

This section describes how to upgrade a supported scanner in all namespaces. The procedure is different depending on the installation method:

1. Installation by using Namespace Provisioner
2. Manual installation

### Installation by using Namespace Provisioner

All scanners installed by the Namespace Provisioner in all managed namespaces are upgraded automatically. For example, if you upgrade your installation of Tanzu Application Platform and the version of Grype is updated, all Grype scanners installed by the Namespace Provisioner for all managed namespaces are automatically upgraded.

### Manual installation

1. If a scanner, such as Grype Scanner, was installed as part of Tanzu Application Platform by using the [full profile](#), run to upgrade:

```
tanzu package installed update tap -p tap.tanzu.vmware.com -v VERSION --values-file tap-values.yaml -n tap-install
```

Where `VERSION` is your Tanzu Application Platform version.

2. If a scanner, such as Grype Scanner, was installed by using [component installation](#) you must manually run:

```
tanzu package installed update grype -p grype.scanning.apps.tanzu.vmware.com -v GRYPE-VERSION --values-file grype-values.yaml -n NAMESPACE
```

Where:

- `GRYPE-VERSION` is the version of Grype that you are upgrading to.
- `NAMESPACE` is the namespace in which Grype is installed in.

## Upgrade to Version v1.2.0

To upgrade from a previous version of SCST - Scan to the version `v1.2.0`:

1. Change the `SecretExports` from SCST - Store.

SCST - Scan needs information to connect to the SCST - Store deployment, you must change where these secrets are exported to enable the connection with the version `v1.2.0` of SCST - Scan.

- **For a single cluster deployment:**

1. Edit the `tap-values.yaml` file you used to deploy SCST - Store to export the CA certificate to your developer namespace.

```
metadata_store:
  ns_for_export_app_cert: "DEV-NAMESPACE"
```



### Note

The `ns_for_export_app_cert` supports one namespace at a time. If you have multiple namespaces you can replace this value with a `*`, but this exports the CA certificate to all namespaces. Consider whether this increased visibility presents a risk.

2. Update Tanzu Application Platform to apply the changes:

```
tanzu package installed update tap -f tap-values.yaml -n tap-install
```

- **For a multi-cluster deployment:**

You must reapply the `SecretExport` by changing the `toNamespace: scan-link-system` to `toNamespace: DEV-NAMESPACE`:

```
---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretExport
metadata:
  name: store-ca-cert
  namespace: metadata-store-secrets
spec:
```

```

toNamespace: "DEV-NAMESPACE"
---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretExport
metadata:
  name: store-auth-token
  namespace: metadata-store-secrets
spec:
  toNamespace: "DEV-NAMESPACE"

```

2. Update your `tap-values.yaml` file.

The installation of the SCST - Scan and the Grype scanner have some changes. The connection to the SCST - Store component have moved to the Grype scanner package. To deactivate the connection from the SCST - Scan, which is still present for backwards compatibility, but is deprecated and is removed in `v1.3.0`.

```

# Deactivate scan controller embedded Supply Chain Security Tools - Store integ
ration
scanning:
  metadataStore:
    url: ""

# Install Grype Scanner v1.2.0
grype:
  namespace: "DEV-NAMESPACE" # The developer namespace where the ScanTemplates
are gonna be deployed
  metadataStore:
    url: "METADATA-STORE-URL" # The base URL where the Store deployment can be
reached
  caSecret:
    name: "CA-SECRET-NAME" # The name of the secret containing the ca.crt
    importFromNamespace: "SECRET-NAMESPACE" # The namespace where Store is de
ployed (if single cluster) or where the connection secrets were created (if mul
ti-cluster)
  authSecret:
    name: "TOKEN-SECRET-NAME" # The name of the secret containing the auth to
ken to connect to Store
    importFromNamespace: "SECRET-NAMESPACE" # The namespace where the connect
ion secrets were created (if multi-cluster)

```

For more insights on how to install Grype, see [Install Supply Chain Security Tools - Scan \(Grype Scanner\)](#).



#### Note

If a mix of Grype templates, such as earlier than v1.2.0 and v1.2.0 and later, are used, both `scanning` and `grype` must configure the parameters. The secret must also export to both `scan-link-system` and the developer namespace. Do this by exporting to `*` or by defining multiple secrets and exports. If Grype is installed to multiple namespaces there must be corresponding exports. See [Install Supply Chain Security Tools - Scan \(Grype Scanner\)](#).

3. Update Tanzu Application Platform to apply the changes:

```
tanzu package installed update tap -f tap-values.yaml -n tap-install
```

4. Update the `ScanPolicy` to include the latest structure changes for `v1.2.0`.

To update to the latest valid Rego File in the [ScanPolicy](#), [Enforce compliance policy using Open Policy Agent](#). [v1.2.0](#) introduced some breaking changes in the Rego File structure used for the [ScanPolicies](#). For more information, see the [Release Notes](#).

## 5. Verify the upgrade.

You can run any [ImageScan](#) or [SourceScan](#) in your `DEV-NAMESPACE` where the Grype Scanner was installed, and it finishes. Here is a sample you can try to run to detect if everything upgraded.

1. Create the `verify-upgrade.yaml` file in your system with the following content:

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: scan-policy
  labels:
    'app.kubernetes.io/part-of': 'enable-in-gui'
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible",
    "UnknownSeverity"
    notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]
    ignoreCves := []

    contains(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
      severities := { e | e := match.ratings.rating.severity } | { e | e
:= match.ratings.rating[_].severity }
      some i
      fails := contains(notAllowedSeverities, severities[i])
      not fails
    }

    isSafe(match) {
      ignore := contains(ignoreCves, match.id)
      ignore
    }

    deny[msg] {
      comps := { e | e := input.bom.components.component } | { e | e := i
nput.bom.components.component[_] }
      some i
      comp := comps[i]
      vulns := { e | e := comp.vulnerabilities.vulnerability } | { e | e
:= comp.vulnerabilities.vulnerability[_] }
      some j
      vuln := vulns[j]
      ratings := { e | e := vuln.ratings.rating.severity } | { e | e := v
uln.ratings.rating[_].severity }
      not isSafe(vuln)
      msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
    }
  ---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ImageScan
metadata:
  name: sample-public-image-scan
spec:
```

```
registry:
  image: "nginx:1.16"
scanTemplate: public-image-scan-template
scanPolicy: scan-policy
```

2. Deploy the resources:

```
kubectl apply -f verify-upgrade.yaml -n DEV-NAMESPACE
```

3. View the scan results:

```
kubectl describe imagescan sample-public-image-scan -n DEV-NAMESPACE
```

If it is successful, the `ImageScan` goes to the `Failed` phase and shows the results of the scan in the `Status`.

## Install another scanner for Supply Chain Security Tools - Scan

This topic describes how you can install scanners to work with Supply Chain Security Tools - Scan from the Tanzu Application Platform package repository.

Follow the instructions in this topic to install a scanner other than the out of the box Grype Scanner with SCST - Scan.

### Prerequisites

Before installing a new scanner, install [Supply Chain Security Tools - Scan](#). It must be present on the same cluster. The prerequisites for Scan are also required.

### Install

To install a new scanner, follow these steps:

1. Complete scanner specific prerequisites for the scanner you're trying to install. For example, creating an API token to connect to the scanner.
  - [Snyk Scanner \(Beta\)](#) is available for image scanning.
  - [Carbon Black Scanner \(Beta\)](#) is available for image scanning.
2. List the available packages to discover what scanners you can use by running:

```
tanzu package available list --namespace tap-install
```

For example:

```
$ tanzu package available list --namespace tap-install
/ Retrieving available packages...
NAME                                DISPLAY-NAME
SHORT-DESCRIPTION
grype.scanning.apps.tanzu.vmware.com  Grype Scanner for Supply
Chain Security Tools - Scan          Default scan templates using A
nchore Grype
snyk.scanning.apps.tanzu.vmware.com   Snyk for Supply Chain Se
curity Tools - Scan                 Default scan templates using
Snyk
carbonblack.scanning.apps.tanzu.vmware.com  Carbon Black Scanner for
Supply Chain Security Tools - Scan  Default scan templates using C
arbon Black
```

- List version information for the scanner package by running:

```
tanzu package available list SCANNER-NAME --namespace tap-install
```

For example:

```
$ tanzu package available list snyk.scanning.apps.tanzu.vmware.com --namespace
tap-install
/ Retrieving package versions for snyk.scanning.apps.tanzu.vmware.com...
NAME                                VERSION                                RELEASED-AT
snyk.scanning.apps.tanzu.vmware.com 1.0.0-beta.2
```

- (Optional) Confirm that the secret created in Step 1 for scanner specific prerequisites is created.
- Create a `values.yaml` to apply custom configurations to the scanner:



#### Note

This step might be required for some scanners but optional for others.

To list the values you can configure for any scanner, run:

```
tanzu package available get SCANNER-NAME/VERSION --values-schema -n tap-install
```

Where:

- `SCANNER-NAME` is the name of the scanner package you retrieved earlier.
- `VERSION` is your package version number. For example, `snyk.scanning.apps.tanzu.vmware.com/1.0.0-beta.2`.

For example:

```
$ tanzu package available get snyk.scanning.apps.tanzu.vmware.com/1.0.0-beta.2
--values-schema -n tap-install

KEY                                DEFAULT
TYPE    DESCRIPTION
metadataStore.authSecret.name
string  Name of deployed Secret with key auth_token
metadataStore.authSecret.importFromNamespace
string  Namespace from which to import the Insight Metadata Store auth_token
metadataStore.caSecret.importFromNamespace    metadata-store
string  Namespace from which to import the Insight Metadata Store CA Cert
metadataStore.caSecret.name                    app-tls-cert
string  Name of deployed Secret with key ca.crt holding the CA Cert of the Insi
ght Metadata Store
metadataStore.clusterRole                      metadata-store-read-write
string  Name of the deployed ClusterRole for read/write access to the Insight M
etadata Store deployed in the same cluster
metadataStore.url                              https://metadata-store-app.metada
ta-store.svc.cluster.local:8443 string  Url of the Insight Metadata Store
namespace                                     default
string  Deployment namespace for the Scan Templates
resources.requests.cpu                          250m
<nil>    Requests describes the minimum amount of cpu resources required.
resources.requests.memory                       128Mi
<nil>    Requests describes the minimum amount of memory resources required.
resources.limits.cpu                            1000m
<nil>    Limits describes the maximum amount of cpu resources allowed.
snyk.tokenSecret.name
```

```
string Reference to the secret containing a Snyk API Token as snyk_token.
targetImagePullSecret
string Reference to the secret used for pulling images from private registry.
```

6. Define the `--values-file` flag to customize the default configuration:

The `values.yaml` file you created earlier is referenced with the `--values-file` flag when running your Tanzu install command:

```
tanzu package install REFERENCE-NAME \
  --package-name SCANNER-NAME \
  --version VERSION \
  --namespace tap-install \
  --values-file PATH-TO-VALUES-YAML
```

Where:

- `REFERENCE-NAME` is the name referenced by the installed package. For example, `grype-scanner`, `snyk-scanner`.
- `SCANNER-NAME` is the name of the scanner package you retrieved earlier. For example, `snyk.scanning.apps.tanzu.vmware.com`.
- `VERSION` is your package version number. For example, `1.0.0-beta.2`.
- `PATH-TO-VALUES-YAML` is the path that points to the `values.yaml` file created earlier.

For example:

```
$ tanzu package install snyk-scanner \
  --package-name snyk.scanning.apps.tanzu.vmware.com \
  --version 1.1.0 \
  --namespace tap-install \
  --values-file values.yaml
/ Installing package 'snyk.scanning.apps.tanzu.vmware.com'
| Getting namespace 'tap-install'
| Getting package metadata for 'snyk.scanning.apps.tanzu.vmware.com'
| Creating service account 'snyk-scanner-tap-install-sa'
| Creating cluster admin role 'snyk-scanner-tap-install-cluster-role'
| Creating cluster role binding 'snyk-scanner-tap-install-cluster-rolebinding'
/ Creating package resource
- Package install status: Reconciling

Added installed package 'snyk-scanner' in namespace 'tap-install'
```

## Verify Installation

To verify the installation create an `ImageScan` or `SourceScan` referencing one of the newly added `ScanTemplates` for the scanner.

1. (Optional) Create a `ScanPolicy` formatted for the output specific to the scanner you are installing, to reference in the `ImageScan` or `SourceScan`.

```
kubectl apply -n $DEV_NAMESPACE -f SCAN-POLICY-YAML
```



### Note

As vulnerability scanners output different formats, the `ScanPolicies` can vary. For information about policies and samples, see [Enforce compliance policy using Open Policy Agent](#).

- Retrieve available `ScanTemplates` from the namespace where the scanner is installed:

```
kubectl get scantemplates -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

For example:

```
$ kubectl get scantemplates
NAME                                AGE
blob-source-scan-template           10d
private-image-scan-template         10d
public-image-scan-template          10d
public-source-scan-template         10d
snyk-private-image-scan-template    10d
snyk-public-image-scan-template     10d
```

- Create the following ImageScan YAML:



#### Note

Some scanners do not support both `ImageScan` and `SourceScan`.

```
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ImageScan
metadata:
  name: sample-scanner-public-image-scan
spec:
  registry:
    image: "nginx:1.16"
  scanTemplate: SCAN-TEMPLATE
  scanPolicy: SCAN-POLICY # Optional
```

Where:

- `SCAN-TEMPLATE` is the name of the installed `ScanTemplate` in the `DEV-NAMESPACE` you retrieved earlier.
- `SCAN-POLICY` it's an optional reference to an existing `ScanPolicy` in the same `DEV-NAMESPACE`.

For example:

```
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ImageScan
metadata:
  name: sample-snyk-public-image-scan
spec:
  registry:
    image: "nginx:1.16"
  scanTemplate: snyk-public-image-scan-template
  scanPolicy: snyk-scan-policy
```

- Create the following SourceScan YAML:



#### Note

Some scanners do not support both `ImageScan` and `SourceScan`.



```

apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: SourceScan
metadata:
  name: sample-scanner-public-source-scan
spec:
  git:
    url: "https://github.com/houndci/hound.git"
    revision: "5805c650"
  scanTemplate: SCAN-TEMPLATE
  scanPolicy: SCAN-POLICY # Optional

```

Where:

- `SCAN-TEMPLATE` is the name of the installed `ScanTemplate` in the `DEV-NAMESPACE` you retrieved earlier.
- `SCAN-POLICY` is an optional reference to an existing `ScanPolicy` in the same `DEV-NAMESPACE`.

For example:

```

apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: SourceScan
metadata:
  name: sample-grype-public-source-scan
spec:
  git:
    url: "https://github.com/houndci/hound.git"
    revision: "5805c650"
  scanTemplate: public-source-scan-template
  scanPolicy: scan-policy

```

## 5. Apply the ImageScan and SourceScan YAMLs:

To run the scans, apply them to the cluster by running these commands:

**ImageScan:**

```
kubectl apply -f PATH-TO-IMAGE-SCAN-YAML -n DEV-NAMESPACE
```

Where `PATH-TO-IMAGE-SCAN-YAML` is the path to the YAML file created earlier.

**SourceScan:**

```
kubectl apply -f PATH-TO-SOURCE-SCAN-YAML -n DEV-NAMESPACE
```

Where `PATH-TO-SOURCE-SCAN-YAML` is the path to the YAML file created earlier.

For example:

```

$ kubectl apply -f imagescan.yaml -n my-apps
imagescan.scanning.apps.tanzu.vmware.com/sample-snyk-public-image-scan created

$ kubectl apply -f sourcescan.yaml -n my-apps
sourcescan.scanning.apps.tanzu.vmware.com/sample-grype-public-source-scan creat
ed

```

## 6. To verify the integration, get the scan to see if it completed by running:

For **ImageScan**:

```
kubectl get imagescan IMAGE-SCAN-NAME -n DEV-NAMESPACE
```

Where `IMAGE-SCAN-NAME` is the name of the `ImageScan` as defined in the YAML file created earlier.

For `SourceScan`:

```
kubectl get sourcescan SOURCE-SCAN-NAME -n DEV-NAMESPACE
```

Where `SOURCE-SCAN-NAME` is the name of the `SourceScan` as defined in the YAML file created earlier.

For example:

```
$ kubectl get imagescan sample-snyk-public-image-scan -n my-apps
NAME                                PHASE      SCANNEDIMAGE  AGE    CRITICAL  HIG
H  MEDIUM  LOW  UNKNOWN  CVETOTAL
sample-snyk-public-image-scan      Completed  nginx:1.16    26h   0         114
58          314   0        486

$ kubectl get sourcescan sample-grype-public-source-scan -n my-apps
NAME                                PHASE
SCANNEDREVISION  SCANNEDREPOSITORY  AGE    CRITICAL  HIG
H  MEDIUM  LOW  UNKNOWN  CVETOTAL
sourcescan.scanning.apps.tanzu.vmware.com/grypesourcescan-sample-public  Compl
eted  5805c650          https://github.com/houndci/hound.git  8m34s  21
121   112    9     0        263
```



#### Note

If you define a `ScanPolicy` for the scans and the evaluation finds a violation, the `Phase` is `Failed` instead of `Completed`. In both cases the scan finished.

#### 7. Clean up:

```
kubectl delete -f PATH-TO-SCAN-YAML -n DEV-NAMESPACE
```

Where `PATH-TO-SCAN-YAML` is the path to the YAML file created earlier.

## Install scanner to multiple namespaces

To install a Scanner to multiple namespaces, VMware recommends using a namespace provisioner. See [Namespace Provisioner](#)

## Configure Tanzu Application Platform Supply Chain to use new scanner

In order to scan your images with the new scanner installed in the [Out of the Box Supply Chain with Testing and Scanning](#), you must update your Tanzu Application Platform installation.

Add the `ootb_supply_chain_testing_scanning.scanning` section to your `tap-values.yaml` and perform a [Tanzu Application Platform update](#).

You can define which `ScanTemplates` is used for both `SourceScan` and `ImageScan`. The default values are the Grype Scanner `ScanTemplates`, but they are overwritten by any other `ScanTemplate` present in your `DEV-NAMESPACE`. The same applies to the `ScanPolicies` applied to each kind of scan.

```
ootb_supply_chain_testing_scanning:
  scanning:
    image:
```

```

template: IMAGE-SCAN-TEMPLATE
policy: IMAGE-SCAN-POLICY
source:
  template: SOURCE-SCAN-TEMPLATE
  policy: SOURCE-SCAN-POLICY

```



### Note

For the Supply Chain to work properly, the `SOURCE-SCAN-TEMPLATE` must support blob files and the `IMAGE-SCAN-TEMPLATE` must support private images.

For example:

```

ootb_supply_chain_testing_scanning:
  scanning:
    image:
      template: snyk-private-image-scan-template
      policy: snyk-scan-policy
    source:
      template: blob-source-scan-template
      policy: scan-policy

```

## Uninstall Scanner

To replace the scanner in the Supply Chain, follow the steps mentioned in [Configure TAP Supply Chain to Use New Scanner](#). After the scanner is no longer required by the Supply Chain, you can remove the package by running:

```

tanzu package installed delete REFERENCE-NAME \
  --namespace tap-install

```

Where `REFERENCE-NAME` is the name you identified the package with, when installing in the [Install](#) section. For example, `grype-scanner`, `snyk-scanner`.

For example:

```

$ tanzu package installed delete snyk-scanner \
  --namespace tap-install

```

## Other Available Scanner Integrations

In addition to providing the above supported integrations, VMware encourages the broader community to support VMware in our goal of integrating with customers' preferred CVE scanners.

Additional integrations:

- [Prisma Scanner \(Alpha\)](#) is available for source and image scanning.



### Note

This integration is in Alpha, which means that it is still in active development by the Tanzu Practices Global Tech Team and may be subject to change at any point. Users may encounter unexpected behavior.

## Prerequisites for Snyk Scanner for Supply Chain Security Tools - Scan (Beta)

This topic describes the prerequisites you must complete to install Supply Chain Security Tools - Scan (Snyk Scanner) from the Tanzu Application Platform package repository.



### Important

Snyk's image scanning capability is in beta. Snyk might only return a partial list of CVEs when scanning Buildpack images.

## Prepare the Snyk Scanner configuration

1. Obtain a Snyk API Token from the [Snyk documentation](#).
2. Create a Snyk secret YAML file and insert the base64 encoded Snyk API token into the `snyk_token`:

```
apiVersion: v1
kind: Secret
metadata:
  name: snyk-token-secret
  namespace: my-apps
data:
  snyk_token: BASE64-SNYK-API-TOKEN
```

Where `BASE64-SNYK-API-TOKEN` is the Snyk API Token obtained earlier.

3. Apply the Snyk secret YAML file by running:

```
kubectl apply -f YAML-FILE
```

Where `YAML-FILE` is the name of the Snyk secret YAML file you created.

4. Define the `--values-file` flag to customize the default configuration. You must define the following fields in the `values.yaml` file for the Snyk Scanner configuration. You can add fields as needed to activate or deactivate behaviors. You can append the values to this file as shown later in this topic. Create a `values.yaml` file by using the following configuration:

```
---
namespace: DEV-NAMESPACE
targetImagePullSecret: TARGET-REGISTRY-CREDENTIALS-SECRET
snyk:
  tokenSecret:
    name: SNYK-TOKEN-SECRET
```

Where:

- `DEV-NAMESPACE` is your developer namespace.



### Note

To use a namespace other than the default namespace, ensure that the namespace exists before you install. If the namespace does not exist, the scanner installation fails.

- `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the secret that contains the credentials to pull an image from a private registry for scanning.
- `SNYK-TOKEN-SECRET` is the name of the secret you created that contains the `snyk_token` to connect to the [Snyk API](#). This field is required.

The Snyk Scanner integration can work with or without the SCST - Store integration. The `values.yaml` file is slightly different for each configuration.

## Supply Chain Security Tools - Store integration

**Using Supply Chain Security Tools - Store Integration:** To persist the results found by the Snyk Scanner, you can enable the SCST - Store integration by appending the fields to the `values.yaml` file.

The Grype and Snyk Scanner Integrations both enable the Metadata Store. To prevent conflicts, the configuration values are slightly different based on whether the Grype Scanner Integration is installed or not. If Tanzu Application Platform is installed using the Full Profile, the Grype Scanner Integration is installed, unless it is explicitly excluded.

- If the Grype Scanner Integration is installed in the same `dev-namespace` Snyk Scanner is installed:

```
#! ...
metadataStore:
  #! The url where the Store deployment is accessible.
  #! Default value is: "https://metadata-store-app.metadata-store.svc.cluster.local:8443"
  url: "STORE-URL"
  caSecret:
    #! The name of the secret that contains the ca.crt to connect to the Store Deployment.
    #! Default value is: "app-tls-cert"
    name: "CA-SECRET-NAME"
    importFromNamespace: "" #! since both Snyk and Grype both enable store, one must leave importFromNamespace blank
  #! authSecret is for multicluster configurations.
  authSecret:
    #! The name of the secret that contains the auth token to authenticate to the Store Deployment.
    name: "AUTH-SECRET-NAME"
    importFromNamespace: "" #! since both Snyk and Grype both enable store, one must leave importFromNamespace blank
```

- If the Grype Scanner Integration is not installed in the same `dev-namespace` Snyk Scanner is installed:

```
#! ...
metadataStore:
  #! The url where the Store deployment is accessible.
  #! Default value is: "https://metadata-store-app.metadata-store.svc.cluster.local:8443"
  url: "STORE-URL"
  caSecret:
    #! The name of the secret that contains the ca.crt to connect to the Store Deployment.
    #! Default value is: "app-tls-cert"
    name: "CA-SECRET-NAME"
    #! The namespace where the secrets for the Store Deployment live.
    #! Default value is: "metadata-store"
    importFromNamespace: "STORE-SECRETS-NAMESPACE"
  #! authSecret is for multicluster configurations.
  authSecret:
```

```

#! The name of the secret that contains the auth token to authenticate to the
Store Deployment.
name: "AUTH-SECRET-NAME"
#! The namespace where the secrets for the Store Deployment live.
importFromNamespace: "STORE-SECRETS-NAMESPACE"

```

**Without SCST - Store Integration:** The SCST - Store integration is enabled by default. If you don't want to use this integration, explicitly deactivate the integration by appending the following field to the `values.yaml` file:

```

# ...
metadataStore:
  url: "" # Configuration is moved, so set this string to empty.

```

## Sample ScanPolicy for Snyk in SPDX JSON format

1. Create a ScanPolicy YAML with a Rego file for scanner output in the SPDX JSON format. Here is a sample scan policy resource:

```

apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: snyk-scan-policy
  labels:
    'app.kubernetes.io/part-of': 'enable-in-gui'
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "UnknownSeverity"
    notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]
    ignoreCves := []

    contains(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
      fails := contains(notAllowedSeverities, match.relationships[_].ratedBy.rating[_].severity)
      not fails
    }

    isSafe(match) {
      ignore := contains(ignoreCves, match.id)
      ignore
    }

    deny[msg] {
      vuln := input.vulnerabilities[_]
      ratings := vuln.relationships[_].ratedBy.rating[_].severity
      comp := vuln.relationships[_].affect.to[_]
      not isSafe(vuln)
      msg = sprintf("CVE %s %s %s", [comp, vuln.id, ratings])
    }

```

2. Apply the YAML file by running:

```
kubectl apply -n $DEV_NAMESPACE -f SCAN-POLICY-YAML
```

**Note**

The Snyk Scanner integration is only available for an image scan, not a source scan.

After all prerequisites are completed, follow the steps in [Install another scanner for Supply Chain Security Tools - Scan](#) to install the Snyk Scanner.

## Prerequisites for Carbon Black Scanner for Supply Chain Security Tools - Scan(Beta)

This topic describes prerequisites you must complete to install Supply Chain Security Tools - Scan (Carbon Black Scanner) from the Tanzu Application Platform package repository. The Carbon Black Scanner integration is only available for an image scan, not a source scan.

**Important**

Carbon Black's image scanning capability is in beta. Carbon Black might only return a partial list of CVEs when scanning Buildpack images.

## Prepare the Carbon Black Scanner configuration

To prepare the Carbon Black Scanner configuration before you install any scanners:

1. Obtain a Carbon Black API Token from Carbon Black Cloud.
2. Create a Carbon Black secret YAML file and insert the Carbon Black API configuration key. Obtain all values from your CBC console.

```
apiVersion: v1
kind: Secret
metadata:
  name: CARBONBLACK-CONFIG-SECRET
  namespace: my-apps
stringData:
  cbc_api_id: CBC-API-ID
  cbc_api_key: CBC-API-KEY
  cbc_org_key: CBC-ORG-KEY
  cbc_saas_url: CBC-SAAS-URL
```

Where:

- o **CBC-API-ID** is the API ID obtained from CBC.
- o **CBC-API-KEY** is the API Key obtained from CBC.
- o **CBC-ORG-KEY** is the Org Key of your CBC organization.
- o **CBC-SAAS-URL** is the CBC Backend URL.

3. Apply the Carbon Black secret YAML file by running:

```
kubectl apply -f YAML-FILE
```

Where **YAML-FILE** is the name of the Carbon Black secret YAML file you created.

4. Define the **--values-file** flag to customize the default configuration. Create a **values.yaml** file by using the following configuration:

You must define the following fields in the `values.yaml` file for the Carbon Black Scanner configuration. You can add fields as needed to enable or deactivate behaviors. You can append the values to this file as shown later in this topic.

```
---
namespace: DEV-NAMESPACE
targetImagePullSecret: TARGET-REGISTRY-CREDENTIALS-SECRET
carbonBlack:
  configSecret:
    name: CARBONBLACK-CONFIG-SECRET
```

- `DEV-NAMESPACE` is your developer namespace.



### Important

To use a namespace other than the default namespace, ensure that the namespace exists before you install. If the namespace does not exist, the scanner installation fails.

- `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the secret that contains the credentials to pull an image from a private registry for scanning.
- `CARBONBLACK-CONFIG-SECRET` is the name of the secret you created that contains the Carbon Black configuration to connect to CBC. This field is required.

The Carbon Black Scanner integration can work with or without the SCST - Store integration. The `values.yaml` file is slightly different for each configuration.

## Supply Chain Security Tools - Store integration

To Integrate:

1. Do one of the following procedures:
  - [Use the Supply Chain Security Tools - Store](#)
  - [Without using the Supply Chain Security Tools - Store](#)
2. Apply the YAML.

### Using Supply Chain Security Tools - Store Integration

To persist the results found by the Carbon Black Scanner, you can enable the SCST - Store integration by appending the fields to the `values.yaml` file.

The Grype and Carbon Black Scanner Integrations both enable the Metadata Store. To prevent conflicts, the configuration values are slightly different based on whether the Grype Scanner Integration is installed or not. If Tanzu Application Platform was installed using the Full Profile, the Grype Scanner Integration was installed, unless it was explicitly excluded.

- If the Grype Scanner Integration is installed in the same `dev-namespace` Carbon Black Scanner is installed:

```
#! ...
metadataStore:
  #! The url where the Store deployment is accessible.
  #! Default value is: "https://metadata-store-app.metadata-store.svc.cluster.l
ocal:8443"
  url: "STORE-URL"
  caSecret:
    #! The name of the secret that contains the ca.crt to connect to the Store
```



```
Deployment.
  #! Default value is: "app-tls-cert"
  name: "CA-SECRET-NAME"
  importFromNamespace: "" #! since both Carbon Black and Grype both enable store, one must leave importFromNamespace blank
  #! authSecret is for multicluster configurations.
  authSecret:
    #! The name of the secret that contains the auth token to authenticate to the Store Deployment.
    name: "AUTH-SECRET-NAME"
    importFromNamespace: "" #! since both Carbon Black and Grype both enable store, one must leave importFromNamespace blank
```

- If the Grype Scanner Integration is not installed in the same `dev-namespace` Carbon Black Scanner is installed:

```
#! ...
metadataStore:
  #! The url where the Store deployment is accessible.
  #! Default value is: "https://metadata-store-app.metadata-store.svc.cluster.local:8443"
  url: "STORE-URL"
  caSecret:
    #! The name of the secret that contains the ca.crt to connect to the Store Deployment.
    #! Default value is: "app-tls-cert"
    name: "CA-SECRET-NAME"
    #! The namespace where the secrets for the Store Deployment live.
    #! Default value is: "metadata-store"
    importFromNamespace: "STORE-SECRETS-NAMESPACE"
  #! authSecret is for multicluster configurations.
  authSecret:
    #! The name of the secret that contains the auth token to authenticate to the Store Deployment.
    name: "AUTH-SECRET-NAME"
    #! The namespace where the secrets for the Store Deployment live.
    importFromNamespace: "STORE-SECRETS-NAMESPACE"
```

## Without Supply Chain Security Tools - Store Integration

If you don't want to enable the SCST - Store integration, explicitly deactivate the integration by appending the next field to the `values.yaml` file, because it's enabled by default:

```
# ...
metadataStore:
  url: "" # Deactivate Supply Chain Security Tools - Store integration
```

## Sample ScanPolicy in CycloneDX format

1. Create a ScanPolicy YAML with a Rego file for scanner output in the CycloneDX format. For example:

```
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: scan-policy
  labels:
    'app.kubernetes.io/part-of': 'enable-in-gui'
spec:
  regoFile: |
    package main
```

```

# Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "UnknownSeverity"
notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]
ignoreCves := []

contains(array, elem) = true {
  array[_] = elem
} else = false { true }

isSafe(match) {
  severities := { e | e := match.ratings.rating.severity } | { e | e := match.ratings.rating[_].severity }
  some i
  fails := contains(notAllowedSeverities, severities[i])
  not fails
}

isSafe(match) {
  ignore := contains(ignoreCves, match.id)
  ignore
}

deny[msg] {
  comps := { e | e := input.bom.components.component } | { e | e := input.bom.components.component[_] }
  some i
  comp := comps[i]
  vulns := { e | e := comp.vulnerabilities.vulnerability } | { e | e := comp.vulnerabilities.vulnerability[_] }
  some j
  vuln := vulns[j]
  ratings := { e | e := vuln.ratings.rating.severity } | { e | e := vuln.ratings.rating[_].severity }
  not isSafe(vuln)
  msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
}

```

## 2. Apply the YAML:

```
kubectl apply -n $DEV_NAMESPACE -f SCAN-POLICY-YAML
```

After all prerequisites are completed, follow the steps in [Install another scanner for Supply Chain Security Tools - Scan](#) to install the Carbon Black Scanner.

## Prerequisites for Prisma Scanner for Supply Chain Security Tools - Scan (Alpha)

This topic describes prerequisites you must complete to install SCST - Scan (Prisma) from the VMware package repository.



### Important

This integration is in Alpha, which means that it is still in active development by the Tanzu Practices Global Tech Team and might be subject to change at any point. Users might encounter unexpected behavior.

## Verify the latest alpha package version

Run this command to output a list of available tags.

```
imgpkg tag list -i projects.registry.vmware.com/tanzu_practice/tap-scanners-package/prisma-repo-scanning-bundle | grep -v sha | sort -V
```

Use the latest version returned in place of the sample version in this topic, such as `0.1.4-alpha.12` in the following output.

```
imgpkg tag list -i projects.registry.vmware.com/tanzu_practice/tap-scanners-package/prisma-repo-scanning-bundle | grep -v sha | sort -V

0.1.4-alpha.1
0.1.4-alpha.6
0.1.4-alpha.11
0.1.4-alpha.12
```

## Relocate images to a registry

VMware recommends relocating the images from VMware Tanzu Network registry to your own container image registry before installing. The Prisma Scanner is in the Alpha development phase, and not packaged as part of Tanzu Application Platform. It is hosted on the VMware Project Repository instead of VMware Tanzu Network. If you relocated the Tanzu Application Platform images, you can also relocate the Prisma Scanner package. If you don't relocate the images, the Prisma Scanner installation depends on VMware Tanzu Network for continued operation, and VMware Tanzu Network offers no uptime guarantees. The option to skip relocation is documented for evaluation and proof-of-concept only.

For information about supported registries, see each registry's documentation.

To relocate images from the VMware Project Registry to your registry:

1. Install Docker if it is not already installed.
2. Log in to your container image registry by running:

```
docker login MY-REGISTRY
```

Where `MY-REGISTRY` is your own registry.

3. Log in to the VMware Tanzu Network registry with your VMware Tanzu Network credentials by running:

```
docker login projects.registry.vmware.com
```

4. Set up environment variables for installation by running:

```
export INSTALL_REGISTRY_USERNAME=MY-REGISTRY-USER
export INSTALL_REGISTRY_PASSWORD=MY-REGISTRY-PASSWORD
export INSTALL_REGISTRY_HOSTNAME=MY-REGISTRY
export VERSION=VERSION-NUMBER
export INSTALL_REPO=TARGET-REPOSITORY
```

Where:

- `MY-REGISTRY-USER` is the user with write access to `MY-REGISTRY`.
- `MY-REGISTRY-PASSWORD` is the password for `MY-REGISTRY-USER`.
- `MY-REGISTRY` is your own registry.
- `VERSION` is your Prisma Scanner version. For example, `0.1.4-alpha.12`.
- `TARGET-REPOSITORY` is your target repository, a directory or repository on `MY-REGISTRY` that serves as the location for the installation files for Prisma Scanner.

5. Install the Carvel tool `imgpkg` CLI. See [Deploying Cluster Essentials](#).
6. Relocate images with the `imgpkg` CLI by running:

```
imgpkg copy -b projects.registry.vmware.com/tanzu_practice/tap-scanners-package/prisma-repo-scanning-bundle:${VERSION} --to-repo ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/prisma-repo-scanning-bundle
```



#### Note

The VMware project repository does not require authentication, so there is no need to perform a docker login.

## Add the Prisma Scanner package repository

Tanzu CLI packages are available on repositories. Adding the Prisma Scanning package repository makes the Prisma Scanning bundle and its packages available for installation.



#### Note

VMware recommends, but does not require, relocating images to a registry for installation. This section assumes that you relocated images to a registry. See the earlier section to fill in the variables.

VMware recommends installing the Prisma Scanner objects in the existing `tap-install` namespace to keep the Prisma Scanner grouped logically with the other Tanzu Application Platform components.

1. Add the Prisma Scanner package repository to the cluster by running:

```
tanzu package repository add prisma-scanner-repository \
  --url ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/prisma-repo-scanning-bundle:${VERSION} \
  --namespace tap-install
```

2. Get the status of the Prisma Scanner package repository, and ensure that the status updates to `Reconcile succeeded` by running:

```
tanzu package repository get prisma-scanner-repository --namespace tap-install
```

For example:

```
$ tanzu package repository get prisma-scanning-repository --namespace tap-install
- Retrieving repository prisma-scanner-repository...
NAME:          prisma-scanner-repository
VERSION:       71091125
REPOSITORY:    projects.registry.vmware.com/tanzu_practice/tap-scanners-package/prisma-repo-scanning-bundle
TAG:           0.1.4-alpha.12
STATUS:        Reconcile succeeded
REASON:
```

3. List the available packages by running:

```
tanzu package available list --namespace tap-install
```

For example:

```
$ tanzu package available list --namespace tap-install
/ Retrieving available packages...
NAME                                DISPLAY-NAME
SHORT-DESCRIPTION
  prisma.scanning.apps.tanzu.vmware.com      Prisma for Supply Chain
Security Tools - Scan                       Default scan templates using
Prisma
```

## Prepare the Prisma Scanner configuration

Before installing the Prisma scanner, you must create the configuration and a Kubernetes secret that contains credentials to access Prisma Cloud.

### Obtain Console url and Access Keys/Token

The Prisma Scanner supports two methods of authentication:

- 1) Basic Authentication with API Key and Secret
- 2) Token Based Authentication

The steps to configure both are outlined to allow you to decide which option you use.



#### Note

The token method issued by Prisma Cloud has a expiration of 1 hour, so it requires frequent refreshing.

To obtain your Prisma Compute Console URL and Access Keys and Token. See [Access keys](#) in the Palo Alto Networks documentation.

- 1) Basic Authentication with API Key and Secret
- 2) Token Based Authentication

### Access key and secret authentication

To create a Prisma secret, use the following instructions.

1. Create a Prisma secret YAML file and insert the base64 encoded Prisma API token into the `prisma_token`:

```
apiVersion: v1
kind: Secret
metadata:
  name: PRISMA-ACCESS-KEY-SECRET
  namespace: APP-NAME
data:
  username: BASE64-PRISMA-ACCESS-KEY-ID
  password: BASE64-PRISMA-ACCESS-KEY-PASSWORD
```

Where:

- `PRISMA-ACCESS-KEY-SECRET` is the name of your Prisma token secret.
- `APP-NAME` is the namespace you want to use.
- `BASE64-PRISMA-ACCESS-KEY-ID` is your base64 encoded Prisma Access Key ID.
- `BASE64-PRISMA-ACCESS-KEY-PASSWORD` is your base64 encoded Prisma Access Key Password.

2. Apply the Prisma secret YAML file by running:

```
kubectl apply -f YAML-FILE
```

Where `YAML-FILE` is the name of the Prisma secret YAML file you created.

3. Define the `--values-file` flag to customize the default configuration. You must define the following fields in the `values.yaml` file for the Prisma Scanner configuration. You can add fields to activate or deactivate behaviors. You can append the values to this file as shown later in this topic. Create a `values.yaml` file by using the following configuration:

```
---
namespace: DEV-NAMESPACE
targetImagePullSecret: TARGET-REGISTRY-CREDENTIALS-SECRET
prisma:
  url: PRISMA-URL
  basicAuth:
    name: PRISMA-ACCESS-KEY-SECRET
```

Where:

- `DEV-NAMESPACE` is your developer namespace.



#### Note

To use a namespace other than the default namespace, ensure that the namespace exists before you install. If the namespace does not exist, the scanner installation fails.

- `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the secret that contains the credentials to pull an image from a private registry for scanning.
- `PRISMA-URL` is the FQDN of your Twistlock server.
- `PRISMA-CONFIG-SECRET` is the name of the secret you created that contains the Prisma configuration to connect to Prisma. This field is required.

The Prisma integration can work with or without the SCST - Store integration. The `values.yaml` file is slightly different for each configuration.

### Access Token Authentication

1. Create a Prisma secret YAML file and insert the base64 encoded Prisma API token into the `prisma_token`:

## Create Prisma Secret

To create a Prisma secret, follow the instructions in the sections below.

### Access Token Authentication

1. Create a Prisma secret YAML file and insert the base64 encoded Prisma API token into the `prisma_token`:

```
apiVersion: v1
kind: Secret
metadata:
  name: PRISMA-TOKEN-SECRET
  namespace: APP-NAME
data:
  prisma_token: BASE64-PRISMA-API-TOKEN
```

Where:

- `PRISMA-TOKEN-SECRET` is the name of your Prisma token secret.
- `APP-NAME` is the namespace you want to use.
- `BASE64-PRISMA-API-TOKEN` is the name of your base64 encoded Prisma API token.

2. Apply the Prisma secret YAML file by running:

```
kubectl apply -f YAML-FILE
```

Where `YAML-FILE` is the name of the Prisma secret YAML file you created.

3. Define the `--values-file` flag to customize the default configuration. You must define the following fields in the `values.yaml` file for the Prisma Scanner configuration. You can add fields to activate or deactivate behaviors. You can append the values to this file as shown later in this topic. Create a `values.yaml` file by using the following configuration:

```
---
namespace: DEV-NAMESPACE
targetImagePullSecret: TARGET-REGISTRY-CREDENTIALS-SECRET
prisma:
  url: PRISMA-URL
  tokenSecret:
    name: PRISMA-CONFIG-SECRET
```

Where:

- `DEV-NAMESPACE` is your developer namespace.



#### Note

To use a namespace other than the default namespace, ensure that the namespace exists before you install. If the namespace does not exist, the scanner installation fails.

- `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the secret that contains the credentials to pull an image from a private registry for scanning.
- `PRISMA-URL` is the FQDN of your Twistlock server.
- `PRISMA-CONFIG-SECRET` is the name of the secret you created that contains the Prisma configuration to connect to Prisma. This field is required.

## Access Key Authentication

1. Create a Prisma secret YAML file and insert the base64 encoded Prisma API token into the `prisma_token`:

```
apiVersion: v1
kind: Secret
metadata:
  name: PRISMA-ACCESS-KEY-SECRET
  namespace: APP-NAME
data:
  username: BASE64-PRISMA-ACCESS-KEY-ID
  password: BASE64-PRISMA-ACCESS-KEY-PASSWORD
```

Where:

- `PRISMA-ACCESS-KEY-SECRET` is the name of your Prisma token secret.

- `APP-NAME` is the namespace you want to use.
- `BASE64-PRISMA-ACCESS-KEY-ID` is your base64 encoded Prisma Access Key ID.
- `BASE64-PRISMA-ACCESS-KEY-PASSWORD` is your base64 encoded Prisma Access Key Password.

2. Apply the Prisma secret YAML file by running:

```
kubectl apply -f YAML-FILE
```

Where `YAML-FILE` is the name of the Prisma secret YAML file you created.

3. Define the `--values-file` flag to customize the default configuration. You must define the following fields in the `values.yaml` file for the Prisma Scanner configuration. You can add fields as needed to activate or deactivate behaviors. You can append the values to this file as shown later in this topic. Create a `values.yaml` file by using the following configuration:

```
---
namespace: DEV-NAMESPACE
targetImagePullSecret: TARGET-REGISTRY-CREDENTIALS-SECRET
prisma:
  url: PRISMA-URL
  basicAuth:
    name: PRISMA-ACCESS-KEY-SECRET
```

Where:

- `DEV-NAMESPACE` is your developer namespace.



#### Note

To use a namespace other than the default namespace, ensure that the namespace exists before you install. If the namespace does not exist, the scanner installation fails.

- `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the secret that contains the credentials to pull an image from a private registry for scanning.
- `PRISMA-URL` is the FQDN of your Twistlock server.
- `PRISMA-CONFIG-SECRET` is the name of the secret you created that contains the Prisma configuration to connect to Prisma. This field is required.

The Prisma integration can work with or without the SCST - Store integration. The `values.yaml` file is slightly different for each configuration.

## Supply Chain Security Tools - Store integration

The Prisma Scanner integration can work with or without the SCST - Store integration. The `values.yaml` file is slightly different for each configuration.

When using SCST - Store integration, to persist the results found by the Prisma Scanner, you can enable the SCST - Store integration by appending the fields to the `values.yaml` file.

The Grype, Snyk, and Prisma Scanner Integrations enable the Metadata Store. To prevent conflicts, the configuration values are slightly different based on whether the Grype Scanner Integration is installed or not. If Tanzu Application Platform is installed using the Full Profile, the Grype Scanner Integration is installed unless it is explicitly excluded.



## Multiple Scanners installed

In order to find your CA secret name and authentication token secret name as needed for your `values.yaml` when installing Prisma Scanner you must look at the configuration of a prior installed scanner in the same namespace as it already exists.

For information about how the scanner was likely initially created, see [Multicluster Setup](#)

An example `values.yaml` when there are other scanners already installed in the same `dev-namespace` where the Prisma Scanner is installed:

```
#! ...
metadataStore:
  #! The url where the Store deployment is accessible.
  #! Default value is: "https://metadata-store-app.metadata-store.svc.cluster.local:8443"
  url: "STORE-URL"
  caSecret:
    #! The name of the secret that contains the ca.crt to connect to the Store Deployment.
    #! Default value is: "app-tls-cert"
    name: "CA-SECRET-NAME"
    importFromNamespace: "" #! since both Prisma and Grype/Snyk both enable store, one must leave importFromNamespace blank
  #! authSecret is for multicluster configurations.
  authSecret:
    #! The name of the secret that contains the auth token to authenticate to the Store Deployment.
    name: "AUTH-SECRET-NAME"
    importFromNamespace: "" #! since both Prisma and Grype/Snyk both enable store, one must leave importFromNamespace blank
```

Where:

- `STORE-URL` is the URL where the Store deployment is accessible.
- `CA-SECRET-NAME` is the name of the secret that contains the ca.crt to connect to the Store Deployment. Default is `app-tls-cert`.
- `AUTH-SECRET-NAME` is the name of the secret that contains the authentication token to authenticate to the Store Deployment.

## Prisma Only Scanner Installed

For information about creating and exporting secrets for the Metadata Store CA and authentication token referenced in the data values when installing Prisma Scanner, see [Multicluster Setup](#).

An example `values.yaml` when no other scanner integrations installed in the same `dev-namespace` where the Prisma Scanner is installed:

```
#! ...
metadataStore:
  #! The url where the Store deployment is accessible.
  #! Default value is: "https://metadata-store-app.metadata-store.svc.cluster.local:8443"
  url: "STORE-URL"
  caSecret:
    #! The name of the secret that contains the ca.crt to connect to the Store Deployment.
    #! Default value is: "app-tls-cert"
    name: "CA-SECRET-NAME"
    #! The namespace where the secrets for the Store Deployment live.
    #! Default value is: "metadata-store"
    importFromNamespace: "STORE-SECRETS-NAMESPACE"
```

```

#! authSecret is for multicluster configurations.
authSecret:
  #! The name of the secret that contains the auth token to authenticate to the Store Deployment.
  name: "AUTH-SECRET-NAME"
  #! The namespace where the secrets for the Store Deployment live.
  importFromNamespace: "STORE-SECRETS-NAMESPACE"

```

Where:

- `STORE-URL` is the URL where the Store deployment is accessible.
- `CA-SECRET-NAME` is the name of the secret that contains the `ca.crt` to connect to the Store Deployment. Default is `app-tls-cert`.
- `STORE-SECRETS-NAMESPACE` is the namespace where the secrets for the Store Deployment live. Default is `metadata-store`.
- `AUTH-SECRET-NAME` is the name of the secret that contains the authentication token to authenticate to the Store Deployment.

## No Store Integration

If you do not want to enable the SCST - Store integration, explicitly deactivate the integration by appending the following fields to the `values.yaml` file that is enabled by default:

```

# ...
metadataStore:
  url: "" # Configuration is moved, so set this string to empty

```

## Prepare the ScanPolicy

To prepare the ScanPolicy, use the instructions in the following sections.

### Sample ScanPolicy using Prisma Policies

The following sample ScanPolicy allows you to control whether the SupplyChain passes or fails based on the compliance and vulnerability rules configured in the Prisma Compute Console.

The policy reads the `complianceScanPassed` and `vulnerabilityScanPassed` fields returned from Prisma scanner output to control the results of the scan.

```

apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: prisma-scan-policy
  labels:
    'app.kubernetes.io/part-of': 'enable-in-gui'
spec:
  regoFile: |
    package main

    import future.keywords.in
    import future.keywords.if

    failedPrismaComplianceOrVulnerabilityChecks(metadata) {
      x := false in cast_set(metadata.properties.property)
      x
    }

    deny[msg] {
      failedPrismaComplianceOrVulnerabilityChecks(input.bom.metadata)
      vulnerabilityMessages := { message |

```

```

        components := {e | e := input.bom.components.component} | {e | e := input.bom.
components.component[_]}
        some component in components
        vulnerabilities := {e | e := component.vulnerabilities.vulnerability} | {e | e
:= component.vulnerabilities.vulnerability[_]}
        some vulnerability in vulnerabilities
        ratings := {e | e := vulnerability.ratings.rating.severity} | {e | e := vulner
ability.ratings.rating[_].severity}
        formattedRatings := concat(" ", ratings)
        message := sprintf("Vulnerability - Component: %s CVE: %s Severity: %s", [comp
onent.name, vulnerability.id, formattedRatings])
    }
    complianceMessages := { message |
        compliances := {e | e := input.bom.metadata.component.compliances.compliance}
| {e | e := input.bom.metadata.component.compliances.compliance[_]}
        some compliance in compliances
        message := sprintf("Compliance - %s \nId: %s Severity: %s Category: %s", [com
pliance.title, compliance.id, compliance.severity, compliance.category])
    }
    combinedMessages := complianceMessages | vulnerabilityMessages
    some message in combinedMessages
    msg := message
}

```

## Sample ScanPolicy using Local Policies

The following sample ScanPolicy allows you to control whether the SupplyChain passes or fails based on the Prisma Scanner CycloneDX vulnerability results returned from the Prisma Scanner.

```

apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: prisma-scan-policy
  labels:
    'app.kubernetes.io/part-of': 'enable-in-gui'
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "UnknownSeve
rity"
    notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]
    ignoreCves := []

    contains(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
      severities := { e | e := match.ratings.rating.severity } | { e | e := match.rati
ngs.rating[_].severity }
      some i
      fails := contains(notAllowedSeverities, severities[i])
      not fails
    }

    isSafe(match) {
      ignore := contains(ignoreCves, match.id)
      ignore
    }

    deny[msg] {
      comps := { e | e := input.bom.components.component } | { e | e := input.bom.comp
onents.component[_] }

```

```

    some i
      comp := comps[i]
      vulns := { e | e := comp.vulnerabilities.vulnerability } | { e | e := comp.vulnerabilities.vulnerability[_] }
      some j
        vuln := vulns[j]
        ratings := { e | e := vuln.ratings.rating.severity } | { e | e := vuln.ratings.rating[_].severity }
        not isSafe(vuln)
        msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
      }

```

Apply the YAML:

```
kubect1 apply -n $DEV-NAMESPACE -f SCAN-POLICY-YAML
```

Where:

- `DEV-NAMESPACE` is the name of the developer namespace you want to use.
- `SCAN-POLICY-YAML` is the name of your SCST - Scan YAML.

## Install Prisma Scanner

After all prerequisites are completed, install the Prisma Scanner. See [Install another scanner for Supply Chain Security Tools - Scan](#).

## Self-Signed Registry Certificate

When attempting to pull an image from a registry with a self-signed certificate during image scans additional configuration is necessary.

## Tanzu Application Platform Values Shared CA

If your `tap-values.yaml` used during install has the following shared section filled out, Prisma Scanner uses this and enable it to connect to your registry without additional configuration.

```

shared:
  ca_cert_data: | # To be passed if using custom certificates.
    -----BEGIN CERTIFICATE-----
    MIIFXzCCA0egAwIBAgIJAJYm37SFocj1MA0GCSqGSIb3DQEEDQUAMEY...
    -----END CERTIFICATE-----

```

## Secret within Developer Namespace

1. Create a secret that holds the registry's CA certificate data.

An example of the secret:

```

apiVersion: v1
kind: Secret
metadata:
  name: prisma-registry-cert
  namespace: dev
type: Opaque
data:
  ca_cert_data: BASE64_CERT

```

2. Update your Prisma Scanner install values.yaml.

Add `caCertSecret` to the root of your `prisma-values.yaml` when installing Prisma Scanner

Example:

```
namespace: dev
targetImagePullSecret: tap-registry
caCertSecret: prisma-registry-cert
```

## Known Limits

- OpenShift is not supported

## Spec reference

This topic describes the specifications and custom resources you can use with Supply Chain Security Tools - Scan.

With the Scan Controller and Grype Scanner installed the following Custom Resource Definitions (CRDs) are now available:

```
$ kubectl get crds | grep scanning.apps.tanzu.vmware.com
imagescans.scanning.apps.tanzu.vmware.com      2021-09-09T15:22:07Z
scanpolicies.scanning.apps.tanzu.vmware.com    2021-09-09T15:22:07Z
scantemplates.scanning.apps.tanzu.vmware.com   2021-09-09T15:22:07Z
sourcescans.scanning.apps.tanzu.vmware.com     2021-09-09T15:22:07Z
```

For more information about installing SCST - Scan, see [Installing Individual Packages](#).

## About source and image scans

Both SourceScan ([sourcescans.scanning.apps.tanzu.vmware.com](#)) and ImageScan ([imagescans.scanning.apps.tanzu.vmware.com](#)) define what will be scanned, and ScanTemplate ([scantemplates.scanning.apps.tanzu.vmware.com](#)) will define how to run a scan. We have provided five custom resources (CRs) pre-installed for use. You can either use them as-is or as samples to create your own.

To view the pre-installed Scan Template CRs, run:

```
kubectl get scantemplates
```

You will see the following scan templates:

| CR Name                                      | Use Case                                                                                                                                                |
|----------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">public-source-scan-template</a>  | Clones and scans source code from a public repository.                                                                                                  |
| <a href="#">private-source-scan-template</a> | Connects with SSH credentials to clone and scan source code from a private repository.                                                                  |
| <a href="#">public-image-scan-template</a>   | Pulls and scans images from a public registry.                                                                                                          |
| <a href="#">private-image-scan-template</a>  | Connects with the registry credentials to pull and scan images from a private registry.                                                                 |
| <a href="#">blob-source-scan-template</a>    | 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](https://scanpolicies.scanning.apps.tanzu.vmware.com)) allows scan results to be validated for company policy compliance and can prevent source code from being built or images from being deployed.

For more information, see [Configuring Policy Enforcement using Open Policy Agent \(OPA\)](#).

## Scan samples for Supply Chain Security Tools - Scan

This section provides samples on multiple use cases for SCST - Scan that you can copy to your cluster for testing purposes.

- [Running a sample public image scan with compliance check](#)
- [Running a sample public source scan with compliance check](#)
- [Running a sample private image scan](#)
- [Running a sample private source scan](#)
- [Running a sample public source scan of a blob/tar file](#)

## Scan samples for Supply Chain Security Tools - Scan

This section provides samples on multiple use cases for SCST - Scan that you can copy to your cluster for testing purposes.

- [Running a sample public image scan with compliance check](#)
- [Running a sample public source scan with compliance check](#)
- [Running a sample private image scan](#)
- [Running a sample private source scan](#)
- [Running a sample public source scan of a blob/tar file](#)

## Sample public image scan with compliance check for Supply Chain Security Tools - Scan

This topic includes an example public image scan with compliance check for SCST - Scan.

### Public image scan

The following example performs an image scan on an image in a public registry. This image revision has 223 known vulnerabilities (CVEs), spanning a number of severities. ImageScan uses the ScanPolicy to run a compliance check against the CVEs.

The policy in this example is set to only consider **Critical** severity CVEs as a violation, which returns 21 Critical Severity Vulnerabilities.



### Note

This example ScanPolicy is deliberately constructed to showcase the features available and must not be considered an acceptable base policy.

In this example, the scan does the following:

- Finds all 223 of the CVEs
- Ignores any CVEs with severities that are not critical
- Indicates in the `Status.Conditions` that 21 CVEs have violated policy compliance

## Define the ScanPolicy and ImageScan

Create `sample-public-image-scan-with-compliance-check.yaml`:

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: sample-scan-policy
  labels:
    'app.kubernetes.io/part-of': 'enable-in-gui'
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "UnknownSeverity"
    notAllowedSeverities := ["Critical"]
    ignoreCves := []

    contains(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
      severities := { e | e := match.ratings.rating.severity } | { e | e := match.ratings.rating[_].severity }
      some i
      fails := contains(notAllowedSeverities, severities[i])
      not fails
    }

    isSafe(match) {
      ignore := contains(ignoreCves, match.id)
      ignore
    }

    deny[msg] {
      comps := { e | e := input.bom.components.component } | { e | e := input.bom.components.component[_] }
      some i
      comp := comps[i]
      vulns := { e | e := comp.vulnerabilities.vulnerability } | { e | e := comp.vulnerabilities.vulnerability[_] }
      some j
      vuln := vulns[j]
      ratings := { e | e := vuln.ratings.rating.severity } | { e | e := vuln.ratings.r
```

```

ating[_].severity }
    not isSafe(vuln)
    msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
}

---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ImageScan
metadata:
  name: sample-public-image-scan-with-compliance-check
spec:
  registry:
    image: "nginx:1.16"
  scanTemplate: public-image-scan-template
  scanPolicy: sample-scan-policy

```

## (Optional) Set up a watch

Before deploying the resources to a user specified namespace, set up a watch in another terminal to view the progression:

```

watch kubectl get sourcescans,imagescans,pods,taskruns,scantemplates,scanpolicies -n DEV-NAMESPACE

```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

For more information about setting up a watch, see [Observing and Troubleshooting](#).

## Deploy the resources

```

kubectl apply -f sample-public-image-scan-with-compliance-check.yaml -n DEV-NAMESPACE

```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

## View the scan results

```

kubectl describe imagescan sample-public-image-scan-with-compliance-check -n DEV-NAMESPACE

```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.



### Note

The `Status.Conditions` includes a `Reason: EvaluationFailed` and `Message: Policy violated because of 21 CVEs`.

For more information about scan status conditions, see [Viewing and Understanding Scan Status Conditions](#).

## Edit the ScanPolicy

To edit the Scan Policy, see [Step 5: Sample Public Source Code Scan with Compliance Check](#).

## Clean up

To clean up, run:

```

kubectl delete -f sample-public-image-scan-with-compliance-check.yaml -n DEV-NAMESPACE

```



Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

## Sample public source code scan with compliance check for Supply Chain Security Tools - Scan

This topic includes an example public source code scan with compliance check for SCST - Scan.

### Public source scan

This example performs a source scan on a public repository. The source revision has 192 known Common Vulnerabilities and Exposures (CVEs), spanning several severities. SourceScan uses the ScanPolicy to run a compliance check against the CVEs.

The example policy is set to only consider `Critical` severity CVEs as violations, which returns 7 Critical Severity Vulnerabilities.



#### Note

This example ScanPolicy is deliberately constructed to showcase the features available and must not be considered an acceptable base policy.

For this example, the scan (at the time of writing):

- Finds all 192 of the CVEs.
- Ignores any CVEs that have severities that are not critical.
- Indicates in the `Status.Conditions` that 7 CVEs have violated policy compliance.

### Run an example public source scan

To perform an example source scan on a public repository:

1. Create `sample-public-source-scan-with-compliance-check.yaml` to define the ScanPolicy and SourceScan:

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: sample-scan-policy
  labels:
    'app.kubernetes.io/part-of': 'enable-in-gui'
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "UnknownSeverity"
    notAllowedSeverities := ["Critical"]
    ignoreCves := []

    contains(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
      severities := { e | e := match.ratings.rating.severity } | { e | e := match.ratings.rating[_].severity }
      some i
      fails := contains(notAllowedSeverities, severities[i])
```

```

    not fails
  }

  isSafe(match) {
    ignore := contains(ignoreCves, match.id)
    ignore
  }

  deny[msg] {
    comps := { e | e := input.bom.components.component } | { e | e := input.bom.components.component[_] }
    some i
    comp := comps[i]
    vulns := { e | e := comp.vulnerabilities.vulnerability } | { e | e := comp.vulnerabilities.vulnerability[_] }
    some j
    vuln := vulns[j]
    ratings := { e | e := vuln.ratings.rating.severity } | { e | e := vuln.ratings.rating[_].severity }
    not isSafe(vuln)
    msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
  }

---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: SourceScan
metadata:
  name: sample-public-source-scan-with-compliance-check
spec:
  git:
    url: "https://github.com/houndci/hound.git"
    revision: "5805c650"
  scanTemplate: public-source-scan-template
  scanPolicy: sample-scan-policy

```

- (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](#).

- 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.

- 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](#).

- If the failing CVEs are acceptable or the build must be deployed regardless of these CVEs, the app is patched to remove the vulnerabilities. Update the `ignoreCVEs` array in the `ScanPolicy` to include the CVEs to ignore:

```

...
spec:
  regoFile: |
    package policies

    default isCompliant = false

    # Accepted Values: "UnknownSeverity", "Critical", "High", "Medium", "Low",
    "Negligible"
    violatingSeverities := ["Critical"]
    # Adding the failing CVEs to the ignore array
    ignoreCVEs := ["CVE-2018-14643", "GHSA-f2jv-r9rf-7988", "GHSA-w457-6q6x-cgp
    9", "CVE-2021-23369", "CVE-2021-23383", "CVE-2020-15256", "CVE-2021-29940"]
    ...

```

6. The changes applied to the new ScanPolicy trigger the scan to run again. Reapply the resources by running:

```
kubectl apply -f sample-public-source-scan-with-compliance-check.yaml -n DEV-NAME-SPACE
```

7. Re-describe the SourceScan CR by running:

```
kubectl describe sourcescan sample-public-source-scan-with-compliance-check -n DEV-NAME-SPACE
```

8. Ensure that `Status.Conditions` now includes a `Reason: EvaluationPassed` and `No CVEs were found that violated the policy`. You can update the `violatingSeverities` array in the ScanPolicy if you want. For reference, the Grype scan returns the following Severity spread of vulnerabilities:

- o Critical: 7
- o High: 88
- o Medium: 92
- o Low: 5
- o Negligible: 0
- o UnknownSeverity: 0

9. Clean up by running:

```
kubectl delete -f sample-public-source-scan-with-compliance-check.yaml -n DEV-NAME-SPACE
```

## Sample private image scan for Supply Chain Security Tools - Scan

This example describes how you can perform a scan against an image located in a private registry for SCST - Scan.

### Define the resources

#### Set up target image pull secret

1. Confirm that target image secret is configured. This is completed during Tanzu Application Platform installation. If the target image secret exists, see [Create the private image scan](#).

2. If the target image secret was not configured, create a secret containing the credentials used to pull the target image you want to scan. For information about secret creation, see the [Kubernetes documentation](#).

```
kubectl create secret docker-registry TARGET-REGISTRY-CREDENTIALS-SECRET \
--docker-server=<your-registry-server> \
--docker-username=<your-name> \
--docker-password=<your-password> \
--docker-email=<your-email> \
-n DEV-NAMESPACE
```

Where:

- `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the secret that is created.
  - `DEV-NAMESPACE` is the developer namespace where the scanner is installed.
3. Update the `tap-values.yaml` file to include the name of secret created earlier.

```
grype:
  namespace: "MY-DEV-NAMESPACE"
  targetImagePullSecret: "TARGET-REGISTRY-CREDENTIALS-SECRET"
```

4. Upgrade Tanzu Application Platform with the modified `tap-values.yaml` file.

```
tanzu package installed update tap -p tap.tanzu.vmware.com -v ${TAP-VERSION} -
-values-file tap-values.yaml -n tap-install
```

Where `TAP-VERSION` is the Tanzu Application Platform version.

## Create the private image scan

Create `sample-private-image-scan.yaml`:

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ImageScan
metadata:
  name: sample-private-image-scan
spec:
  registry:
    image: IMAGE-URL
  scanTemplate: private-image-scan-template
```

Where `IMAGE-URL` is the URL of an image in a private registry.

## (Optional) Set up a watch

Before deploying the resources to a user specified namespace, set up a watch in another terminal to view the progression:

```
watch kubectl get sourcescans,imagescans,pods,taskruns,scantemplates,scanpolicies -n D
EV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

For more information, see [Observing and Troubleshooting](#).

## Deploy the resources

```
kubectl apply -f sample-private-image-scan.yaml -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

## View the scan results

When the scan completes, run:

```
kubectl describe imagescan sample-private-image-scan -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.



### Note

The `Status.Conditions` includes a `Reason: JobFinished` and `Message: The scan job finished`. See [Viewing and Understanding Scan Status Conditions](#).

## Clean up

```
kubectl delete -f sample-private-image-scan.yaml -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

## View vulnerability reports

After completing the scans, [query the Supply Chain Security Tools - Store](#) to view your vulnerability results.

## Sample private source scan for Supply Chain Security Tools - Scan

This example shows how you can perform a private source scan for SCST - Scan.

## Define the resources

1. Create a Kubernetes secret with an SSH key for cloning a Git repository. See the [Kubernetes documentation](#).

```
cat <<EOF | kubectl create -f -
apiVersion: v1
kind: Secret
metadata:
name: SECRET-SSH-AUTH
namespace: DEV-NAMESPACE
annotations:
  tekton.dev/git-0: https://github.com
  tekton.dev/git-1: https://gitlab.com
type: kubernetes.io/ssh-auth
stringData:
ssh-privatekey: |
  -----BEGIN 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+PXYPcPy6rbTrTtw7PHkccKrrp0yVhp5HdEIcKr6pLlVDBfOLX9QUusyC
OV0wzFjIjN1GEYsd1LJizHhbn2mUjvSAHQqZETYP81eFzLQnNPHt4EvvUUh7VfDESU8
4KezmD5QlWpXlMvU31/yMf+Se8xhHTvKSCZIFImWwoG6mbUoWf9nzpIoaSjB+weqqU
UmpaaasXVal72J+UX2B+2RPW3RcT0eOzQgq1JL3RkrTJvdsjE3JEAvgq3lGHSZxy28
G3skua2SmVi/w4yCE6gbODqnTWlg7+wC604ydGXA8VJiS5ap43JXiUFFAaQ==
github.com ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAA
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/CzuAeQNV4fZVf2pLxpGH1e15zKpX0osckequUDxoq
  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
DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

See [Observing and Troubleshooting](#).

## Deploy the resources

```
kubectl apply -f sample-private-source-scan.yaml -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

## View the scan status

After the scan has completed, run:

```
kubectl describe sourcescan sample-private-source-scan -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

Notice the `Status.Conditions` includes a `Reason: JobFinished` and `Message: The scan job finished`. See [Viewing and Understanding Scan Status Conditions](#).

## Clean up

```
kubectl delete -f sample-private-source-scan.yaml -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

## View vulnerability reports

After completing the scans, [query the Supply Chain Security Tools - Store](#) to view your vulnerability results.

## Sample public source scan of a blob for Supply Chain Security Tools - Scan

You can do a public source scan of a blob for SCST - Scan. This example performs a scan against source code in a `.tar.gz` file. This is helpful in a Supply Chain, where there is a `GitRepository` step that handles cloning a repository and outputting the source code as a compressed archive.

## Define the resources

Create `public-blob-source-example.yaml`:

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: SourceScan
metadata:
  name: public-blob-source-example
spec:
  blob:
    url: "https://gitlab.com/nina-data/ckan/-/archive/master/ckan-master.tar.gz"
  scanTemplate: blob-source-scan-template
```

## (Optional) Set up a watch

Before deploying the resources to a user specified namespace, set up a watch in another terminal to view the progression:

```
watch kubectl get sourcescans,imagescans,pods,taskruns,scantemplates,scanpolicies -n 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.



## Using Grype in air-gapped (offline) environments for Supply Chain Security Tools - Scan

This topic tells you how to use Grype in air-gapped (offline) environments for Supply Chain Security Tools (SCST) - Scan.

The `grype` CLI attempts to perform two over the Internet calls:

- One to verify for later versions of the CLI.
- One to update the vulnerability database before scanning.

For the `grype` CLI to function in an offline or air-gapped environment, the vulnerability database must be hosted within the environment. You must configure the `grype` CLI with the internal URL.

The `grype` CLI accepts environment variables to satisfy these needs.

For information about setting up an offline vulnerability database, see the [Anchore Grype README](#) in GitHub.

### To enable Grype in offline air-gapped environments

1. Add the following to your `tap-values.yaml` file:

```
grype:
  db:
    dbUpdateUrl: INTERNAL-VULN-DB-URL
```

Where `INTERNAL-VULN-DB-URL` is the URL that points to the internal file server.

2. Update Tanzu Application Platform:

```
tanzu package installed update tap -f tap-values.yaml -n tap-install
```



#### Note

If you are using the Namespace Provisioner to provision a new developer namespace and want to apply a package overlay for Grype, you must complete additional configuration steps. See [Grype package overlays are not applied to scantemplates created by Namespace Provisioner](#).

## Troubleshooting

### ERROR failed to fetch latest cli version



#### Note

This message is a warning and the Grype scan still runs with this message.

The Grype CLI checks for later versions of the CLI by contacting the anchore endpoint over the Internet.

```
ERROR failed to fetch latest version: Get "https://toolbox-data.anchore.io/grype/releases/latest/VERSION": dial tcp: lookup toolbox-data.anchore.io on [::1]:53: read udp [::1]:65010->[::1]:53: read: connection refused
```

## Solution

To deactivate this check, set the environment variable `GRYPE_CHECK_FOR_APP_UPDATE` to `false` by using a package overlay with the following steps:

1. Create a secret that contains the ytt overlay to add the Grype environment variable to the ScanTemplates.

```
apiVersion: v1
kind: Secret
metadata:
  name: grype-airgap-deactivate-cli-check-overlay
  namespace: tap-install #! namespace where tap is installed
stringData:
  patch.yaml: |
    #@ load("@ytt:overlay", "overlay")

    #@overlay/match by=overlay.subset({"kind":"ScanTemplate"}), expects="1+"
    ---
    spec:
      template:
        initContainers:
          #@overlay/match by=overlay.subset({"name": "scan-plugin"}), expects
          ="1+"
          - name: scan-plugin
            #@overlay/match missing_ok=True
            env:
              #@overlay/append
              - name: GRYPE_CHECK_FOR_APP_UPDATE
                value: "false"
```

2. Configure `tap-values.yaml` to use `package_overlays`. Add the following to your `tap-values.yaml` file:

```
package_overlays:
  - name: "grype"
  secrets:
    - name: "grype-airgap-deactivate-cli-check-overlay"
```

3. Update Tanzu Application Platform:

```
tanzu package installed update tap -f tap-values.yaml -n tap-install
```

## Database is too old

```
1 error occurred:
 * db could not be loaded: the vulnerability database was built N days/weeks ago (max
allowed age is 5 days)
```

Grype needs up-to-date vulnerability information to provide accurate matches. By default, it fails to run if the local database was not built in the last 5 days.

## Solution

Two options to resolve this:

1. Stale databases weaken your security posture. VMware recommends updating the database daily as the first recommended solution.
2. If updating the database daily is not an option, the data staleness check is configurable by using the environment variable `GRYPE_DB_MAX_ALLOWED_BUILT_AGE` and is addressed using a

package overlay with the following steps:

1. Create a secret that contains the ytt overlay to add the Grype environment variable to the ScanTemplates.

```

apiVersion: v1
kind: Secret
metadata:
  name: grype-airgap-override-stale-db-overlay
  namespace: tap-install #! namespace where tap is installed
stringData:
  patch.yaml: |
    #@ load("@ytt:overlay", "overlay")

    #@overlay/match by=overlay.subset({"kind":"ScanTemplate"}), expects="1+"
    ---
    spec:
      template:
        initContainers:
          #@overlay/match by=overlay.subset({"name": "scan-plugin"}), expects="1+"
          - name: scan-plugin
            #@overlay/match missing_ok=True
            env:
              #@overlay/append
              - name: GRYPE_DB_MAX_ALLOWED_BUILT_AGE #! see note on best practices
                value: "120h"
  
```



#### Note

The default maximum allowed built age of Grype's vulnerability database is 5 days. This means that scanning with a 6 day old database causes the scan to fail. You can use the `GRYPE_DB_MAX_ALLOWED_BUILT_AGE` parameter to override the default in accordance with your security posture.

2. Configure `tap-values.yaml` to use `package_overlays`. Add the following to your `tap-values.yaml` file:

```

package_overlays:
  - name: "grype"
    secrets:
      - name: "grype-airgap-override-stale-db-overlay"
  
```

3. Update Tanzu Application Platform:

```
tanzu package installed update tap -f tap-values.yaml -n tap-install
```

## Grype package overlays are not applied to scantemplates created by Namespace Provisioner

If you used the Namespace Provisioner to provision a new developer namespace and want to apply a package overlay for Grype, see [Import overlay secrets](#).

```

scan-pod[scan-plugin] 1 error occurred:
scan-pod[scan-plugin] * failed to load vulnerability db: vulnerability database is in
valid (run db update to correct): database metadata not found: /.cache/grype/db/5
  
```

## Solution

Examine the `listing.json` file you created. This matches the format of the listing file. The listing file is located at Anchore Grype's public endpoint. See the [Grype README.md](#) in GitHub.

Here is an example of a properly formatted `listing.json`:

```
#@ load("@ytt:overlay", "overlay")

#@ def matchGrypeScanners(index, left, right):
  #@ if left["apiVersion"] != "packaging.carvel.dev/v1alpha1" or left["kind"] != "PackageInstall":
    #@ return False
  #@ end

  #@ return left["metadata"]["name"].startswith("grype-scanner")
#@ end

#@ def customize():

#@overlay/match by=matchGrypeScanners, expects="0+"
---
metadata:
  annotations:
    #@overlay/match missing_ok=True
    ext.packaging.carvel.dev/ytt-paths-from-secret-name.0: SECRET-NAME
#@ end
```

Where:

- `5` refers to the Grype's vulnerability database schema.
- `built` is the build timestamp in the format `yyyy-MM-ddTHH:mm:ssZ`.
- `url` is the download URL for the tarball containing the database. This points at your internal endpoint. The tarball contains the following files:
  - `vulnerability.db` is an SQLite file that is Grype's vulnerability database. Each time the data shape of the vulnerability database changes, a new schema is created. Different Grype versions require specific database schema versions. For example, Grype `v0.54.0` requires database schema version `v5`.
  - `metadata.json` file
- `checksum` is the SHA used to verify the database's integrity.

Verify these possible reasons why the vulnerability database is not valid:

1. The database schema is invalid. Confirm that the required database schema for the installed Grype version is used. Confirm that the top level version key matches the nested `version`. For example, the top level version `1` in the following snippet does not match the nested `version: 5`.

```
{
  "available": {
    "1": [{
      "built": "2023-02-08T08_17_20Z",
      "version": 5,
      "url": "https://INTERNAL-ENDPOINT/PATH-TO-TARBALL/vulnerability-db_v5_2023-02-08T08_17_20Z_6ef73016d160043c630f.tar.gz",
      "checksum": "sha256:aab8d369933c845878ef1b53bb5c26ee49b91ddc5cd87c9eb57ffb203a88a72f"
    }]
  }
}
```

Where `PATH-TO-TARBALL` is the path to the tarball containing the vulnerability database.

As stale databases weaken your security posture, VMware recommends using the newest entry of the relevant schema version in the `listing.json` file. See Anchore's [grype-db](#) in GitHub.

2. The `built` parameters in the `listing.json` file are incorrectly formatted. The proper format is `yyyy-MM-ddTHH:mm:ssZ`.
3. The `url` that you modified to point at an internal endpoint is not reachable from within the cluster. For information about verifying connectivity, see [Debug Grype database in a cluster](#).

## Debug Grype database in a cluster

1. Describe the failed source scan or image scan to verify the name of the ScanTemplate being used.
  - o For `sourcescan`, run:

```
kubectl describe sourcescan SCAN-NAME -n DEV-NAMESPACE
```

- o For `imagescan`, run:

```
kubectl describe imagescan SCAN-NAME -n DEV-NAMESPACE
```

Where `SCAN-NAME` is the name of the source or image scan that failed.

2. Edit the ScanTemplate's `scan-plugin` container to include a "sleep" entrypoint which allows you to troubleshoot inside the container:

```
- name: scan-plugin
  volumeMounts:
    ...
  image: #@ data.values.scanner.image
  imagePullPolicy: IfNotPresent
  env:
    ...
  command: ["/bin/bash"]
  args:
    - "sleep 1800" # insert 30 min sleep here
```

3. Re-run the scan.
4. Get the name of the `scan-plugin` pod.

```
kubectl get pods -n DEV-NAMESPACE
```

5. Get a shell to the container. See the [Kubernetes documentation](#):

```
kubectl exec --stdin --tty SCAN-PLUGIN-POD -c step-scan-plugin -- /bin/bash
```

Where `SCAN-PLUGIN-POD` is the name of the `scan-plugin` pod.

6. Inside the container, run Grype CLI commands to report database status and verify connectivity from cluster to mirror. See the [Grype documentation](#) in GitHub.
  - o Report current status of Grype's database (location, build date, and checksum):

```
grype db status
```

7. Ensure that the built parameters in the listing.json has timestamps in this proper format  
`yyyy-MM-ddTHH:mm:ssZ`.

## Triage and Remediate CVEs for Supply Chain Security Tools - Scan

This topic explains how you can triage and remediate CVEs related to SCST - Scan.

### Confirm that Supply Chain stopped due to failed policy enforcement

To confirm that Supply Chain failure is related to policy enforcement:

1. Verify that the status of the workload is `MissingValueAtPath` due to waiting on a `.status.compliantArtifact` from either the SourceScan or ImageScan:

```
kubectl describe workload WORKLOAD-NAME -n DEVELOPER-NAMESPACE
```

2. Describe the SourceScan or ImageScan to determine what CVE(s) violated the ScanPolicy:

```
kubectl describe sourcescan NAME -n DEVELOPER-NAMESPACE
kubectl describe imagescan NAME -n DEVELOPER-NAMESPACE
```

## Triage

The goal of triage is to analyze and prioritize the reported vulnerability data to discover the appropriate course of action to take at the remediation step. To remediate efficiently and appropriately, you need context on the vulnerabilities that are blocking your supply chain, the packages that are affected, and the impact they can have.

During triage, review which packages are impacted by the CVEs that violated your scan policy. Enabling CVE scan causes Supply Chain Choreographer by using Tanzu Application Platform GUI to visualize your supply chain, including the scans, scan policy, and CVEs. See [Enable CVE scan results](#). You can also use the Tanzu Insight plug-in to query packages and CVEs using a CLI. See [Tanzu Insight plug-in](#).

During this stage, VMware recommends reviewing information pertaining to the CVEs from sources such as the [National Vulnerability Database](#) or the release page of a package.

## Remediation

After triage is complete, the next step is to remediate the blocking vulnerabilities quickly. Some common methods for CVE remediation are as follows:

- Updating the affected component to remove the CVE
- Amending the scan policy with an exception if you decide to accept the CVE and unblock your supply chain

### Updating the affected component

Vulnerabilities that occur in older versions of a package might be resolved in later versions. Apply a patch by upgrading to a later version. You can further adopt security best practices by using your project's package manager tools, such as `go mod graph` for projects in Go, to identify transitive or indirect dependencies that can affect CVEs.

## Amending the scan policy

If you decide to proceed without remediating the CVE, for example, when a CVE is evaluated to be a false positive or when a fix is not available, you can amend the ScanPolicy to ignore one or more CVEs. For information about common scanner limitations, see [Note on Vulnerability Scanners](#). For information about templates, see [Writing Policy Templates](#).

Under RBAC, users with the `app-operator-scanning` role that is part of the `app-operator` aggregate role, have permission to edit the ScanPolicy. See [Detailed role permissions breakdown](#).

## Observe Supply Chain Security Tools - Scan

This topic outlines observability and troubleshooting methods and issues you can use with SCST - Scan components.

### Observability

The scans run inside a Kubernetes Job where the Job creates a pod. Both the Job and pod are cleaned up after completion.

Before applying a new scan, you can set a watch on the TaskRuns, Pods, SourceScans, and Imagescans to observe their progression:

```
watch kubectl get sourcescans,imagescans,pods,taskruns,scantemplates,scanpolicies -n D
DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

## Troubleshoot Supply Chain Security Tools - Scan

This topic describes troubleshooting methods you can use with SCST - Scan.

### Debugging commands

Run these commands to get more logs and details about the errors around scanning. The Jobs and pods persist for a predefined amount of seconds before getting deleted.

(`deleteScanJobsSecondsAfterFinished` is the tap pkg variable that defines this)

### Debugging Scan pods

Run the following to get error logs from a pod when scan pods are in a failing state:

```
kubectl logs scan-pod-name -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the name of the developer namespace you want to use.

See [here](#) for more details about debugging Kubernetes pods.

The following is an example of a successful scan run output:

```
scan:
  cveCount:
    critical: 20
    high: 120
    medium: 114
    low: 9
    unknown: 0
  scanner:
    name: Grype
```

```

vendor: Anchore
version: v0.37.0
reports:
- /workspace/scan.xml
eval:
violations:
- CVE node-fetch GHSA-w7rc-rwvf-8q5r Low
store:
locations:
- https://metadata-store-app.metadata-store.svc.cluster.local:8443/api/sources?repo=hound&sha=5805c6502976c10f5529e7f7aeb0af0c370c0354&org=houndci

```

A scan run that has an error means that one of the init containers: `scan-plugin`, `metadata-store-plugin`, `compliance-plugin`, `summary`, or any other additional containers had a failure.

To inspect for a specific init container in a pod:

```
kubectl logs scan-pod-name -n DEV-NAMESPACE -c init-container-name
```

Where `DEV-NAMESPACE` is the name of the developer namespace you want to use.

See [Debug Init Containers](#) in the Kubernetes documentation for debug init container tips.

## Debugging SourceScan and ImageScan

To retrieve status conditions of an SourceScan and ImageScan, run:

```
kubectl describe sourcescan SOURCE-SCAN -n DEV-NAMESPACE
```

Where:

- `DEV-NAMESPACE` is the name of the developer namespace you want to use.
- `SOURCE-SCAN` is the name of the SourceScan you want to use.

```
kubectl describe imagescan IMAGE-SCAN -n DEV-NAMESPACE
```

Where:

- `DEV-NAMESPACE` is the name of the developer namespace you want to use.
- `IMAGE-SCAN` is the name of the ImageScan you want to use.

Under `Status.Conditions`, for a condition look at the “Reason”, “Type”, “Message” values that use the keyword “Error” to investigate issues.

## Debugging Scanning within a SupplyChain

See [here](#) for Tanzu workload commands for tailing build and runtime logs and getting workload status and details.

## Viewing the Scan-Controller manager logs

To retrieve scan-controller manager logs:

```
kubectl -n scan-link-system logs -f deployment/scan-link-controller-manager -c manager
```

## Restarting Deployment

If you encounter an issue with the scan-link controller not starting, run the following to restart the deployment to see if it’s reproducible or flaking upon starting:



```
kubectl rollout restart deployment scan-link-controller-manager -n scan-link-system
```

## Troubleshooting scanner to MetadataStore configuration

### Insight CLI failed to post scan results to metadata store due to failed certificate verification

If you encounter this issue:

```
✘ Error: Post "https://metadata-store.tap.tanzu.example.com/api/sourceReport?": tls: failed to verify certificate: x509: certificate signed by unknown authority
```

To ensure that the `caSecret` from the scanner `DEV-NAMESPACE` matches the `caSecret` from the `METADASTORE-NAMESPACE` namespace:

1. In a single cluster, the connection between the scanning pod and the metadata store happens inside the cluster and does not pass through ingress. This is automatically configured. You do not need to provide an ingress connection to the store. If you provided an ingress connection to the store, delete it.
2. Get the `caSecret.name` depending if your setup is single or multicluster.
  1. If you are using a single cluster setup, the default value for `grype.metadataStore.caSecret.name` is `app-tls-cert`. See [Install Supply Chain Security Tools - Scan](#).
  2. If you are using a multicluster setup, retrieve `grype.metadataStore.caSecret.name` from the Grype config:

```
grype:
  metadataStore:
    caSecret:
      name: store-ca-cert
      importFromNamespace: metadata-store-secrets
```

**Note** `caSecret.name` is set to `store-ca-cert`. See [Multicluster setup](#).

3. Verify that the `CA-SECRET` secret exists in the `DEV-NAMESPACE`.

```
kubectl get secret CA-SECRET -n DEV-NAMESPACE
```

4. If the secret `CA-SECRET` doesn't exist in your `DEV-NAMESPACE`, verify that the `CA-SECRET` exists in the `METADASTORE-NAMESPACE` namespace:

```
kubectl get secret CA-SECRET -n METADASTORE-NAMESPACE
```

Where `METADASTORE-NAMESPACE` is the namespace that contains the secret `CA-SECRET`. If you are using a single cluster, it is configured using the `metadata-store` namespace. If multicluster, it is configured using the `metadata-store-secrets`.

- If `CA-SECRET` doesn't exist in the metadata store namespace, configure the certificate. See [Custom certificate configuration](#).
5. Check if the `secretexports` and `secretimports` exist and are reconciling successfully:

```
kubectl get secretexports.secretgen.carvel.dev -n `METADASTORE-NAMESPACE`
kubectl get secretimports.secretgen.carvel.dev -n `DEV-NAMESPACE`
```

- SCST - Store creates the single cluster secretexport by default. See [Deployment details and configuration](#).
  - For information about creating the multicluster secretexport, see [Multicluster setup](#).
6. Verify that the `ca.crt` field in both secrets from `METADASTORE-NAMESPACE` and `DEV-NAMESPACE` match, or that the `ca.crt` field of the secret in the `METADASTORE-NAMESPACE` includes the `ca.crt` field of the `DEV-NAMESPACE` secret.

You can confirm this by base64 decoding both secrets and seeing if there is a match:

```
kubectl get secret CA-SECRET -n DEV-NAMESPACE -o json | jq -r '.data."ca.crt"' | base64 -d
kubectl get secret CA-SECRET -n METADASTORE-NAMESPACE -o json | jq -r '.data."ca.crt"' | base64 -d
```

The certificates in the `METADASTORE-NAMESPACE` and `DEV-NAMESPACE` must have a match for the scanner to connect to the metadata-store.

## Troubleshooting issues

### Missing target SSH secret

Scanning source code from a private source repository requires an SSH secret present in the namespace and referenced as `grype.targetSourceSshSecret` in `tap-values.yaml`. See [Installing the Tanzu Application Platform Package and Profiles](#).

If a private source scan is triggered and the secret cannot be found, the scan pod includes a `FailedMount` warning in Events with the message `MountVolume.SetUp failed for volume "ssh-secret" : secret "secret-ssh-auth" not found`, where `secret-ssh-auth` is the value specified in `grype.targetSourceSshSecret`.

### Missing target image pull secret

Scanning an image from a private registry requires an image pull secret to exist in the Scan CRs namespace and be referenced as `grype.targetImagePullSecret` in `tap-values.yaml`. See [Installing the Tanzu Application Platform Package and Profiles](#).

If a private image scan is triggered and the secret is not configured, the scan job fails 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"]
```

### Deactivate Supply Chain Security Tools (SCST) - Store

SCST - Store is required to install SCST - Scan. If you install without the SCST - Store, you must edit the configurations to deactivate the Store:

```
---
metadataStore:
  url: ""
```

Install the package with the edited configurations by running:

```
tanzu package install scan-controller \
  --package-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 [Install Supply Chain Security Tools - Scan](#). Although this change bypasses the check on Syft Schema Version, it does not resolve the incompatibility issue and produces a partial scanning result.

```
syft:
  failOnSchemaErrors: false
```

## Resolving incompatible scan policy

If your scan policy appears to not be enforced, it might be because the Rego file defined in the scan policy is incompatible with the scanner that is being used. For example, the Grype Scanner outputs in the CycloneDX XML format while the Snyk Scanner outputs SPDX JSON.

See [Sample ScanPolicy for Snyk in SPDX JSON format](#) for an example of a ScanPolicy formatted for SPDX JSON.

## Could not find CA in secret

If you encounter the following issue, it might be due to not exporting `app-tls-cert` to the correct namespace:

```
{"level":"error","ts":"2022-06-08T15:20:48.43237873Z","logger":"setup","msg":"Could not find CA in Secret","err":"unable to set up connection to Supply Chain Security Tools - Store"}
```

Configure `ns_for_export_app_cert` in your `tap-values.yaml`.

```
metadata_store:
  ns_for_export_app_cert: "DEV-NAMESPACE"
```

Where `DEV-NAMESPACE` is the name of the developer namespace you want to use.

If there are multiple developer namespaces, use `ns_for_export_app_cert: "**"`.

## Blob Source Scan is reporting wrong source URL

A Source Scan for a blob artifact can cause reporting in the `status.artifact` and `status.compliantArtifact` the wrong URL for the resource, passing the remote SSH URL instead of the cluster local fluxcd one. One symptom of this issue is the `image-builder` failing with a `ssh:// is an unsupported protocol` error message.

You can confirm you're having this problem by running `kubectl describe` in the affected resource and comparing the `spec.blob.url` value against the `status.artifact.blob.url`. The problem occurs if they are different URLs. For example:

```
kubectl describe sourcescan SOURCE-SCAN-NAME -n DEV-NAMESPACE
```

Where:

- `SOURCE-SCAN-NAME` is the name of the source scan you want to configure.
- `DEV-NAMESPACE` is the name of the developer namespace you want to use. And compare the output:

```
...
spec:
  blob:
    ...
    url: http://source-controller.flux-system.svc.cluster.local./gitrepository/sample/
repo/8d4cea98b0fa9e0112d58414099d0229f190f7f1.tar.gz
    ...
status:
  artifact:
    blob:
      ...
      url: ssh://git@github.com:sample/repo.git
  compliantArtifact:
    blob:
      ...
      url: ssh://git@github.com:sample/repo.git
```

**Workaround:** This problem happens in SCST - Scan `v1.2.0` when you use a Grype Scanner ScanTemplates earlier than `v1.2.0`, because this is a deprecated path. To fix this problem, upgrade your Grype Scanner deployment to `v1.2.0` or later. See [Upgrading Supply Chain Security Tools - Scan](#) for step-by-step instructions.

## Resolving failing scans that block a Supply Chain

If the Supply Chain is not progressing due to CVEs found in either the SourceScan or ImageScan, see the CVE triage workflow in [Triaging and Remediating CVEs](#).

## Policy not defined in the Tanzu Application Platform GUI

If you encounter `No policy has been defined`, it might be because the Tanzu Application Platform GUI is unable to view the Scan Policy resource.

Confirm that the Scan Policy associated with a SourceScan or ImageScan exists. For example, the `scanPolicy` in the scan matches the name of the Scan Policy.

```
kubectl describe sourcescan NAME -n DEV-NAMESPACE
kubectl describe imagescan NAME -n DEV-NAMESPACE
kubectl get scanpolicy NAME -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the name of the developer namespace you want to use.

Add the `app.kubernetes.io/part-of` label to the Scan Policy. See [Enable Tanzu Application Platform GUI to view ScanPolicy Resource](#).

## Lookup error when connecting to SCST - Store

If your scan pod is failing, you might see the following connection error in the logs:

```
dial tcp: lookup metadata-store-app.metadata-store.svc.cluster.local on 10.100.0.10:53: no such host
```

A connection error while attempting to connect to the local cluster URL causes this error. If this is a multicluster deployment, set the `grype.metadataStore.url` property in your Build profile `values.yaml`. You must set the ingress domain of SCST - Store which is deployed in the View cluster. For information about this configuration, see [Install Build profile](#).

## Sourcscan error with SCST - Store endpoint without a prefix

If your Source Scan resource is failing, the status might show this error:

```
Error: endpoint require 'http://' or 'https://' prefix
```

This is because the `grype.metadataStore.url` value in the Tanzu Application Platform profile `values.yaml` was not configured with the correct prefix. Verify that the URL starts with either `http://` or `https://`.

## Deprecated pre-v1.2 templates

If the scan phase is in `Error` and the status condition message is:

```
Summary logs could not be retrieved: . error opening stream pod logs reader: container summary is not valid for pod scan-grypeimagescan-sample-public-zmj2g-hqv5g
```

This error might be a consequence of using Grype Scanner ScanTemplates shipped with SCST - Scan v1.1 or earlier. These ScanTemplates are deprecated and are not supported in Tanzu Application Platform v1.4.0 and later.

There are two options to resolve this issue:

- Option 1: Upgrade to the latest Grype Scanner version. This automatically replaces the old ScanTemplates with the upgraded ScanTemplates.
- Option 2: Create a ScanTemplate. Follow the steps in [Create a scan template](#).

## Incorrectly configured self-signed certificate

The following error in the pod logs indicate that the self-signed certificate might be incorrectly configured:

```
x509: certificate signed by unknown authority
```

To resolve this issue, ensure that `shared.ca_cert_data` contains the required certificate. For an example of setting up the shared self-signed certificate, see [Build profile](#).

For information about `shared.ca_cert_data`, see [View possible configuration settings for your package](#).

## Unable to pull scan controller and scanner images from a specified registry

The `docker` field and related sub-fields by SCST - Scan Controller, Grype Scanner, or Snyk Scanner are deprecated in Tanzu Application Platform v1.4.0. Previously these text boxes might be used to populate the `registry-credentials` secret. You might encounter the following error during installation:

```
UNAUTHORIZED: unauthorized to access repository
```

The recommended migration path for users setting up their namespaces manually is to add registry credentials to both the developer namespace and the `scan-link-system` namespace, using these [instructions](#).



### Important

This step does not apply to users who used `--export-to-all-namespaces` when setting up the Tanzu Application Platform package repository.

## Grype database not available

Prior to running a scan, the Grype scanner downloads a copy of its database. If the database fails to download, the following log message might appear.

```
Vulnerability DB [no update available] New version of grype is available: 0.50.2 [000
0] WARN unable to check for vulnerability database update 1 error occurred: * failed t
o load vulnerability db: vulnerability database is corrupt (run db update to correct):
database metadata not found: ~/Library/Caches/grype/db/3
```

To resolve this issue, ensure that Grype has access to its vulnerability database:

- If you have set up a [mirror](#) of the vulnerability database, verify that it is populated and reachable.
- If you did not set up a mirror, Grype manages its database behind the scenes. Verify that the cluster has access to <https://anchore.com/>.

This issue is unrelated to Supply Chain Security Tools for Tanzu – Store.

## Configure code repositories and image artifacts for Supply Chain Security Tools - Scan

This topic describes how you can configure code repositories and image artifacts for SCST - Scan.

### Prerequisite

Both the source and image scans require you to define a `ScanTemplate`. Run `kubectl get scantemplates` for the ScanTemplates provided with the scanner installation. For information about how to reference these ScanTemplates, see [How to create a ScanTemplate](#).

### Deploy scan custom resources

The scan controller defines two custom resources to create scanning jobs:

- SourceScan
- ImageScan

#### SourceScan

The `SourceScan` custom resource helps you define and trigger a scan for a given repository. You can deploy `SourceScan` with source code existing in a public repository or a private one:

1. Create the `SourceScan` custom resource.

Example:

```

apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: SourceScan
metadata:
  # set the name of the source scan CR
  name: sample-source-scan
spec:
  # At least one of these fields (blob or git) must be defined.
  blob:
    # location to a file with the source code compressed (supported files: .tar.gz)
    url:
  git:
    # A multiline string defining the known hosts that are going to be used for the SSH client on the container
    knownHosts:
    # Branch, tag, or commit digest
    revision:
    # The name of the kubernetes secret containing the private SSH key information.
    sshKeySecret:
    # A string containing the repository URL.
    url:
    # The username needed to SSH connection. Default value is "git"
    username:

  # A string defining the name of an existing ScanTemplate custom resource.
  scanTemplate: my-scan-template

  # A string defining the name of an existing ScanPolicy custom resource. See "Enforcement Policies (OPA)" section.
  scanPolicy: my-scan-policy

```

2. Deploy the `SourceScan` custom resource to the desired namespace on cluster by running:

```
kubectl apply -f <path_to_the_cr>/<custom_resource_filename>.yaml -n <desired_namespace>
```

After the scanning completes, the following fields appear in the custom resource and are filled by the scanner:

```

# These fields are populated from the source scan results
status:
  # The source code information as provided in the CycloneDX `bom>metadata>component>` fields
  artifact:
    blob:
      url:
    git:
      url:
      revision:

  # An array populated with information about the scanning status
  # and the policy validation. These conditions might change in the lifecycle
  # of the scan, refer to the "View Scan Status and Understanding Conditions" section to learn more.
  conditions: []

  # The URL of the vulnerability scan results in the Metadata Store integration.
  # Only available when the integration is configured.
  metadataUrl:

  # When the CRD is updated to point at new revisions, this lets you know
  # if the status reflects the latest one or not

```

```

observedGeneration: 1
observedPolicyGeneration: 1
observedTemplateGeneration: 1

# The latest datetime when the scanning was successfully finished.
scannedAt:
# Information about the scanner that was used for the latest image scan.
# This information reflects what's in the CycloneDX `bom>metadata>tools>tool>
*` fields.
scannedBy:
  scanner:
    # The name of the scanner that was used.
    name: my-image-scanner

    # The name of the scanner's development company or team
    vendor: my-image-scanner-provider

    # The version of the scanner used.
    version: 1.0.0

```

## ImageScan

The `ImageScan` custom resource helps you define and trigger a scan for a given image. You can deploy `ImageScan` with an image existing in a public or private registry:

1. Create the `ImageScan` custom resource.

Example:

```

apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ImageScan
metadata:
  # set the name of the image scan CR
  name: sample-image-scan
spec:
  registry:
    # Required. A string containing the image name can additionally add its tag
    or its digest
    image: nginx:1.16

    # A string containing the secret needed to pull the image from a private re
    gistry.
    # The secret needs to be deployed in the same namespace as the ImageScan
    imagePullSecret: my-image-pull-secret

    # A string defining the name of an existing ScanTemplate custom resource. See
    "How To Create a ScanTemplate" section.
    scanTemplate: my-scan-template

    # A string defining the name of an existing ScanPolicy custom resource. See
    "Enforcement Policies (OPA)" section.
    scanPolicy: my-scan-policy

```

2. Deploy the `ImageScan` custom resource to the desired namespace on cluster by running:

```
kubectl apply -f <path_to_the_cr>/<custom_resource_filename>.yaml -n <desired_n
amespace>
```

After the scanning completes, the following fields appear in the custom resource and are filled by the scanner:

```

# These fields are populated from the image scan results
status:
  artifact:

```



```

registry:
  # The image name with its digest as provided in the CycloneDX `bom>metada
  ta>component>*\` fields
  image:
    imagePullSecret:

  # An array that is populated with information about the scanning status
  # and the policy validation. These conditions might change in the lifecycle
  # of the scan, refer to the "View Scan Status and Understanding Conditions" s
  ection to learn more.
  conditions: []

  # The URL of the vulnerability scan results in the Metadata Store integratio
  n.
  # Only available when the integration is configured.
  metadataUrl:

  # When the CRD is updated to point at new revisions, this lets you know
  # whether the status reflects the latest one
  observedGeneration: 1
  observedPolicyGeneration: 1
  observedTemplateGeneration: 1

  # The latest datetime when the scanning was successfully finished.
  scannedAt:
  # Information about the scanner used for the latest image scan.
  # This information reflects what's in the CycloneDX `bom>metadata>tools>tool>
  *\` fields.
  scannedBy:
    scanner:
      # The name of the scanner that was used.
      name: my-image-scanner

      # The name of the scanner's development company or team
      vendor: my-image-scanner-provider

      # The version of the scanner used.
      version: 1.0.0

```

## Configure code repositories and image artifacts for Supply Chain Security Tools - Scan

This topic describes how you can configure code repositories and image artifacts for SCST - Scan.

### Prerequisite

Both the source and image scans require you to define a [ScanTemplate](#). Run `kubectl get scantemplates` for the ScanTemplates provided with the scanner installation. For information about how to reference these ScanTemplates, see [How to create a ScanTemplate](#).

### Deploy scan custom resources

The scan controller defines two custom resources to create scanning jobs:

- SourceScan
- ImageScan

### SourceScan

The `SourceScan` custom resource helps you define and trigger a scan for a given repository. You can deploy `SourceScan` with source code existing in a public repository or a private one:

1. Create the `SourceScan` custom resource.

Example:

```
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: SourceScan
metadata:
  # set the name of the source scan CR
  name: sample-source-scan
spec:
  # At least one of these fields (blob or git) must be defined.
  blob:
    # location to a file with the source code compressed (supported files: .tar.gz)
    url:
  git:
    # A multiline string defining the known hosts that are going to be used for
    the SSH client on the container
    knownHosts:
    # Branch, tag, or commit digest
    revision:
    # The name of the kubernetes secret containing the private SSH key information.
    sshKeySecret:
    # A string containing the repository URL.
    url:
    # The username needed to SSH connection. Default value is "git"
    username:

  # A string defining the name of an existing ScanTemplate custom resource.
  scanTemplate: my-scan-template

  # A string defining the name of an existing ScanPolicy custom resource. See
  "Enforcement Policies (OPA)" section.
  scanPolicy: my-scan-policy
```

2. Deploy the `SourceScan` custom resource to the desired namespace on cluster by running:

```
kubectl apply -f <path_to_the_cr>/<custom_resource_filename>.yaml -n <desired_namespace>
```

After the scanning completes, the following fields appear in the custom resource and are filled by the scanner:

```
# These fields are populated from the source scan results
status:
  # The source code information as provided in the CycloneDX `bom>metadata>component>*\` fields
  artifact:
    blob:
      url:
    git:
      url:
      revision:

  # An array populated with information about the scanning status
  # and the policy validation. These conditions might change in the lifecycle
  # of the scan, refer to the "View Scan Status and Understanding Conditions" section to learn more.
  conditions: []

  # The URL of the vulnerability scan results in the Metadata Store integration
```

```
n.
# Only available when the integration is configured.
metadataUrl:

# When the CRD is updated to point at new revisions, this lets you know
# if the status reflects the latest one or not
observedGeneration: 1
observedPolicyGeneration: 1
observedTemplateGeneration: 1

# The latest datetime when the scanning was successfully finished.
scannedAt:
# Information about the scanner that was used for the latest image scan.
# This information reflects what's in the CycloneDX `bom>metadata>tools>tool>
*` fields.
scannedBy:
  scanner:
    # The name of the scanner that was used.
    name: my-image-scanner

    # The name of the scanner's development company or team
    vendor: my-image-scanner-provider

    # The version of the scanner used.
    version: 1.0.0
```

## ImageScan

The `ImageScan` custom resource helps you define and trigger a scan for a given image. You can deploy `ImageScan` with an image existing in a public or private registry:

1. Create the `ImageScan` custom resource.

Example:

```
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ImageScan
metadata:
  # set the name of the image scan CR
  name: sample-image-scan
spec:
  registry:
    # Required. A string containing the image name can additionally add its tag
    # or its digest
    image: nginx:1.16

    # A string containing the secret needed to pull the image from a private re
    # gistry.
    # The secret needs to be deployed in the same namespace as the ImageScan
    imagePullSecret: my-image-pull-secret

    # A string defining the name of an existing ScanTemplate custom resource. See
    # "How To Create a ScanTemplate" section.
    scanTemplate: my-scan-template

    # A string defining the name of an existing ScanPolicy custom resource. See
    # "Enforcement Policies (OPA)" section.
    scanPolicy: my-scan-policy
```

2. Deploy the `ImageScan` custom resource to the desired namespace on cluster by running:

```
kubectl apply -f <path_to_the_cr>/<custom_resource_filename>.yaml -n <desired_n
amespace>
```

After the scanning completes, the following fields appear in the custom resource and are filled by the scanner:

```
# These fields are populated from the image scan results
status:
  artifact:
    registry:
      # The image name with its digest as provided in the CycloneDX `bom>metada
ta>component>` fields
      image:
        imagePullSecret:

    # An array that is populated with information about the scanning status
    # and the policy validation. These conditions might change in the lifecycle
    # of the scan, refer to the "View Scan Status and Understanding Conditions" s
ection to learn more.
    conditions: []

    # The URL of the vulnerability scan results in the Metadata Store integratio
n.
    # Only available when the integration is configured.
    metadataUrl:

    # When the CRD is updated to point at new revisions, this lets you know
    # whether the status reflects the latest one
    observedGeneration: 1
    observedPolicyGeneration: 1
    observedTemplateGeneration: 1

    # The latest datetime when the scanning was successfully finished.
    scannedAt:
    # Information about the scanner used for the latest image scan.
    # This information reflects what's in the CycloneDX `bom>metadata>tools>tool>
*` fields.
    scannedBy:
      scanner:
        # The name of the scanner that was used.
        name: my-image-scanner

        # The name of the scanner's development company or team
        vendor: my-image-scanner-provider

        # The version of the scanner used.
        version: 1.0.0
```

## Enforce compliance policy using Open Policy Agent

This topic describes how you can use Open Policy Agent to enforce compliance policy for Supply Chain Security Tools - Scan.

### Writing a policy template

The Scan Policy custom resource (CR) allows you to define a Rego file for policy enforcement that you can reuse across image scan and source scan CRs.

The Scan Controller supports policy enforcement by using an Open Policy Agent (OPA) engine with Rego files. This allows you to validate scan results for company policy compliance and can prevent source code from being built or images from being deployed.

### Rego file contract

To define a Rego file for an image scan or source scan, you must comply with the requirements defined for every Rego file for the policy verification to work. For information about how to write Rego, see [Open Policy Agent documentation](#).

- **Package main:** The Rego file must define a package in its body called `main`. The system looks for this package to verify the scan results compliance.
- **Input match:** The Rego file evaluates one vulnerability match at a time, iterating as many times as the Rego file finds vulnerabilities in the scan. The match structure is accessed in the `input.currentVulnerability` object inside the Rego file and has the [CycloneDX](#) format.
- **deny rule:** The Rego file must define a `deny` rule inside its body. `deny` is a set of error messages that are returned to the user. Each rule you write adds to that set of error messages. If the conditions in the body of the `deny` statement are true then the user is handed an error message. If false, the vulnerability is allowed in the Source or Image scan.

## Define a Rego file for policy enforcement

Follow these steps to define a Rego file for policy enforcement that you can reuse across image scan and source scan CRs that output in the CycloneDX XML format.



### Note

The Snyk Scanner outputs SPDX JSON. For an example of a ScanPolicy formatted for SPDX JSON output, see [Sample ScanPolicy for Snyk in SPDX JSON format](#).

1. Create a scan policy with a Rego file. The following is an example scan policy resource:

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: scan-policy
  labels:
    app.kubernetes.io/part-of: enable-in-gui
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "Unkn
ownSeverity"
    notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]
    ignoreCves := []

    contains(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
      severities := { e | e := match.ratings.rating.severity } | { e | e := mat
ch.ratings.rating[_].severity }
      some i
      fails := contains(notAllowedSeverities, severities[i])
      not fails
    }

    isSafe(match) {
      ignore := contains(ignoreCves, match.id)
      ignore
    }
  }
```

```
deny[msg] {
  comps := { e | e := input.bom.components.component } | { e | e := input.bom.components.component[_] }
  some i
  comp := comps[i]
  vulns := { e | e := comp.vulnerabilities.vulnerability } | { e | e := comp.vulnerabilities.vulnerability[_] }
  some j
  vuln := vulns[j]
  ratings := { e | e := vuln.ratings.rating.severity } | { e | e := vuln.ratings.rating[_].severity }
  not isSafe(vuln)
  msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
}
```

You can edit the following text boxes of the Rego file as part of the [CVE triage workflow](#):

- `notAllowedSeverities` contains the categories of CVEs that cause the SourceScan or ImageScan failing policy enforcement. The following example shows an `app-operator` blocking only `Critical`, `High` and `UnknownSeverity` CVEs.

```
...
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible",
    "UnknownSeverity"
    notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]
    ignoreCves := []
  ...
```

- `ignoreCves` contains individual ignored CVEs when determining policy enforcement. In the following example, an `app-operator` ignores `CVE-2018-14643` and `GHSA-f2jv-r9rf-7988` if they are false positives. See [A Note on Vulnerability Scanners](#).

```
...
spec:
  regoFile: |
    package main

    notAllowedSeverities := []
    ignoreCves := ["CVE-2018-14643", "GHSA-f2jv-r9rf-7988"]
  ...
```

2. Deploy the scan policy to the cluster:

```
kubectl apply -f <path_to_scan_policy>/<scan_policy_filename>.yaml -n <desired_namespace>
```

For information about how scan policies are used in the CVE triage workflow, see [Triaging and Remediating CVEs](#).

## Further refine the Scan Policy for use

The scan policy earlier demonstrates how vulnerabilities are ignored during a compliance check. It is not possible to audit why a vulnerability is ignored. You might want to allow an exception, where a build with a failing vulnerability is allowed to progress through a supply chain. You can allow this exception for a certain period of time, requiring an expiration date. Vulnerability Exploitability Exchange (VEX) documents are gaining popularity to capture security advisory information pertaining to vulnerabilities. You can use Rego for these use cases.

For example, the following scan policy includes an additional text box to capture comments regarding why the scan ignores a vulnerability. The `notAllowedSeverities` array remains an array of strings, but the `ignoreCves` array updates from an array of strings to an array of objects. This causes a change to the `contains` function, splitting it into separate functions for each array.

```

---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: scan-policy
  labels:
    app.kubernetes.io/part-of: enable-in-gui
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "UnknownSeverity"
    notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]

    # List of known vulnerabilities to ignore when deciding whether to fail compliance. Example:
    # ignoreCves := [
    #   {
    #     "id": "CVE-2018-14643",
    #     "detail": "Determined affected code is not in the execution path."
    #   }
    # ]
    ignoreCves := []

    containsSeverity(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
      severities := { e | e := match.ratings.rating.severity } | { e | e := match.ratings.rating[_].severity }
      some i
      fails := containsSeverity(notAllowedSeverities, severities[i])
      not fails
    }

    containsCve(array, elem) = true {
      array[_].id = elem
    } else = false { true }

    isSafe(match) {
      ignore := containsCve(ignoreCves, match.id)
      ignore
    }

    deny[msg] {
      comps := { e | e := input.bom.components.component } | { e | e := input.bom.components.component[_] }
      some i
      comp := comps[i]
      vulns := { e | e := comp.vulnerabilities.vulnerability } | { e | e := comp.vulnerabilities.vulnerability[_] }
      some j
      vuln := vulns[j]
      ratings := { e | e := vuln.ratings.rating.severity } | { e | e := vuln.ratings.rating[_].severity }
      not isSafe(vuln)
  }

```

```

    msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
  }

```

The following example includes an expiration text box and only allows the vulnerability to be ignored for a period of time:

```

---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: scan-policy
  labels:
    app.kubernetes.io/part-of: enable-in-gui
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "UnknownSeverity"
    notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]

    # List of known vulnerabilities to ignore when deciding whether to fail compliance. Example:
    # ignoreCves := [
    #   {
    #     "id": "CVE-2018-14643",
    #     "detail": "Determined affected code is not in the execution path.",
    #     "expiration": "2022-Dec-31"
    #   }
    # ]
    ignoreCves := []

    containsSeverity(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
      severities := { e | e := match.ratings.rating.severity } | { e | e := match.ratings.rating[_].severity }
      some i
      fails := containsSeverity(notAllowedSeverities, severities[i])
      not fails
    }

    containsCve(array, elem) = true {
      array[_].id = elem
      curr_time := time.now_ns()
      date_format := "2006-Jan-02"
      expire_time := time.parse_ns(date_format, array[_].expiration)
      curr_time < expire_time
    } else = false { true }

    isSafe(match) {
      ignore := containsCve(ignoreCves, match.id)
      ignore
    }

    deny[msg] {
      comps := { e | e := input.bom.components.component } | { e | e := input.bom.components.component[_] }
      some i
      comp := comps[i]
      vulns := { e | e := comp.vulnerabilities.vulnerability } | { e | e := comp.vulnerabilities.vulnerability[_] }
      some j

```



```

vuln := vulns[j]
ratings := { e | e := vuln.ratings.rating.severity } | { e | e := vuln.ratings.r
ating[_].severity }
not isSafe(vuln)
msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
}

```

## Enable Tanzu Application Platform GUI to view ScanPolicy Resource

For the Tanzu Application Platform GUI to view the ScanPolicy resource, it must have a matching `kubernetes-label-selector` with a `part-of` prefix.

The following example is portion of a ScanPolicy that is viewable by the Tanzu Application Platform GUI:

```

---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: scan-policy
  labels:
    app.kubernetes.io/part-of: enable-in-gui
spec:
  regoFile: |
    ...

```



### Note

The value for the label can be anything. The Tanzu Application Platform GUI is looking for the existence of the `part-of` prefix string and doesn't match for anything else specific.

## Deprecated Rego file Definition

Before Scan Controller v1.2.0, you must use the following format where the rego file differences are:

- The package name must be `package policies` instead of `package main`.
- The deny rule is a Boolean `isCompliant` instead of `deny[msg]`.
  - **isCompliant rule:** The Rego file must define inside its body an `isCompliant` rule. This must be a Boolean type containing the result whether the vulnerability violates the security policy or not. If `isCompliant` is `true`, the vulnerability is allowed in the Source or Image scan. Otherwise, `false` is considered. Any scan that finds at least one vulnerability that evaluates to `isCompliant=false` makes the `PolicySucceeded` condition set to false.

The following is an example scan policy resource:

```

apiVersion: scanning.apps.tanzu.vmware.com/v1alpha1
kind: ScanPolicy
metadata:
  name: v1alpha1-scan-policy
  labels:
    app.kubernetes.io/part-of: enable-in-gui
spec:
  regoFile: |

```

```

package policies

default isCompliant = false

ignoreSeverities := ["Critical", "High"]

contains(array, elem) = true {
  array[_] = elem
} else = false { true }

isCompliant {
  ignore := contains(ignoreSeverities, input.currentVulnerability.Ratings.Rating
[_].Severity)
  ignore
}

```

## Create a ScanTemplate with Supply Chain Security Tools - Scan

This topic describes how to create a ScanTemplate with Supply Chain Security Tools - Scan.

### Overview

The `ScanTemplate` custom resource (CR) defines how the scan Pod fulfills the task of vulnerability scanning. There are default `ScanTemplates` provided out of the box using the Tanzu Application Platform default scanner, `Anchore Grype`. One or more `initContainers` run to complete the scan and must save results to a shared `volume`. After the `initContainers` completes, a single container on the scan Pod called `summary` combines the result of the `initContainers` so that the `Scan CR` status is updated.

A customized ScanTemplate is created by editing or replacing `initContainer` definitions and reusing the `summary` container from the `grype` package. A container can read the `out.yaml` from an earlier step to locate relevant inputs.

### Output Model

Each `initContainer` can create a subdirectory in `/workspace` to use as a scratch space. Before terminating the container must create an `out.yaml` file in the subdirectory containing the relevant subset of fields from the output model:

```

fetch:
  git:
    url:
    revision:
    path:
  blob:
    url:
    revision:
    path:
  image:
    url:
    revision:
    path:
sbom:
  packageCount:
  reports: []
scan:
  cveCount:
  critical:
  high:

```

```

medium:
  low:
  unknown:
scanner:
  name:
  vendor:
  version:
  db:
    version:
reports: []
eval:
  violations: []
store:
  locations: []

```

The `scan` portion of the earlier output is required and if missing the scan controller fails to properly update the final status of the `Scan CR`. Other portions of the output, including those of `store` and `policy evaluation`, are optional and can be omitted if not applicable in a custom supply chain setup.

## ScanTemplate Structure

```

apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanTemplate
spec:
  template: # a core/v1 PodSpec
    # Here are list volumes mounted for writing to or
    # reading from during different stages of the scan
    volumes:
      # required the results of different scan stages
      # should be saved in files digestible by the scan
      # controller in this volume
      - name: workspace
        emptyDir: { }
      # different steps required for a scanning can be staged
      # in sequential stages through initContainers.
      initContainers:
        # Summary container will take results of initContainers
        # and will let Controller to update Scan CR status.
      containers:
        - name: summary

```

## Sample Outputs

```

# example for a typical git clone (source scan fetch stage)
# saved at: /workspace/git-clone/out.yaml
fetch:
  git:
    url: github.com/my/repo
    revision: aee9f8
    path: /workspace/git-clone/cloned-repository

```

```

# an example of typical scan stage
# saved at: /workspace/grype-scan/out.yaml
scan:
  cveCount:
    critical: 0
    high: 1
    medium: 3
    low: 25
    unknown: 0

```

```

scanner:
  name: grype
  vendor: Anchore
  version: 0.33.0
  db:
    version: 2022-04-13
  reports:
  - /workspace/grype-scan/repo.cyclonedx.xml
  - /workspace/grype-scan/app.cyclonedx.xml
  - /workspace/grype-scan/base.cyclonedx.xml

```

```

# example of a typical evaluation stage
# saved at: /workspace/policy-eval/out.yaml
eval:
  violations:
  - banned package log4j
  - critical CVE 2022-01-01-3333
  - number of critical CVEs over threshold

```

```

# example of a typical upload to store stage
# saved at: /workspace/upload-to-store/out.yaml
store:
  locations:
  - http://metadata-store.cluster.local:8080/reports/3

```

## View scan status conditions for Supply Chain Security Tools - Scan

This topic explains how you can view scan status conditions for Supply Chain Security Tools - Scan.

### Viewing scan status

You can view the scan status by using `kubectl describe` on a `SourceScan` or `ImageScan`. You can see information about the scan status under the Status field for each scan CR.

### Understanding conditions

The `Status.Conditions` array is populated with the scan status information during and after scanning execution, and the policy validation (if defined for the scan) after the results are available.

### Condition types for the scans

#### Scanning

The Condition with type `Scanning` indicates the execution of the scanning job. The Status field indicates whether the scan is still running or has already finished (i.e., if `Status: True`, the scan job is still running; if `Status: False`, the scan is done).

The Reason field is `JobStarted` while the scanning is running and `JobFinished` when it is done.

The Message field can either be `The scan job is running` or `The scan job terminated` depending on the current Status and Reason.

#### Succeeded

The Condition with type `Succeeded` indicates the scanning job result. The Status field indicates whether the scan finished successfully or if it encountered an error (i.e., the status is `Status: True`

if it completed successfully or `Status: False` otherwise).

The Reason field is `JobFinished` if the scanning was successful or `Error` if otherwise.

The Message and Error fields have more information about the last seen status of the scan job.

### SendingResults

The condition with type `SendingResults` indicates sending the scan results to the metadata store. In addition to a successful process of sending the results, the condition may also indicate that the metadata store integration has not been configured or that there was an error sending. An error would usually be a misconfigured metadata store url or that the metadata store is inaccessible. Check the installation steps to ensure the configuration is correct regarding secrets being set within the `scan-link-system` namespace.

### PolicySucceeded

The Condition with type `PolicySucceeded` indicates the compliance of the scanning results against the defined policies (see [Code Compliance Policy Enforcement using Open Policy Agent \(OPA\)](#)). The Status field indicates whether the results are compliant or not (`Status: True` or `Status: False` respectively) or `Status: Unknown` in case an error occurred during the policy verification.

The Reason field is `EvaluationPassed` if the scan complies with the defined policies. The Reason field is `EvaluationFailed` if the scan is not compliant, or `Error` if something went wrong.

The Message and Error fields are populated with `An error has occurred` and an error message if something went wrong during policy verification. Otherwise, the Message field displays `No CVEs were found that violated the policy` if there are no non-compliant vulnerabilities found or `Policy violated because of X CVEs` indicating the count of unique vulnerabilities found.

## Understanding CVECount

The `status.CVECount` is populated with the number of CVEs in each category (CRITICAL, HIGH, MEDIUM, LOW, UNKNOWN) and the total (CVETOTAL).



#### Note

You can also view scan CVE summary in print columns with `kubectl get` on a `SourceScan` or `ImageScan`.

## Understanding MetadataURL

The `status.metadataURL` is populated with the url of the vulnerability scan results in the metadata store integration. This is only available when the integration is configured.

## Understanding Phase

The `status.phase` field is populated with the current phase of the scan. The phases are: Pending, Scanning, Completed, Failed, and Error.

- `Pending`: initial phase of the scan.
- `Scanning`: execution of the scan job is running.
- `Completed`: scan completed and no CVEs were found that violated the scan policy.
- `Failed`: scan completed but CVEs were found that violated the scan policy.

- `Error`: indication of an error (e.g., an invalid scantemplate or scan policy).



#### Note

The PHASE print column also shows this with `kubectl get` on a `SourceScan` or `ImageScan`.

## 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.

## Overview of Supply Chain Security Tools for VMware Tanzu - Policy Controller

Supply Chain Security Tools - Policy Controller is a security tool that helps you ensure that the container images in their registry have not been tampered with. Policy Controller is a Kubernetes Admission Controller that allows you to apply policies to verify signatures on container images before being admitted to a cluster.

The Policy Controller:

- Verifies signatures on container images used by Kubernetes resources
- Enforces policies to allow or deny images being admitted a cluster
- Allows operators to define multiple policies in the cluster
- Allows operators to select which `namespaces` to enforce policies against
- Supports `cosign` signatures and keyless signing
- Supports storing public keys in a KMS

It enforces its policies against all resources that create `Pods` as part of their life cycle:

- `Pod`
- `ReplicaSet`
- `Deployment`
- `Job`
- `StatefulSet`
- `DaemonSet`
- `CronJob`



#### Note

This component is the successor to `Supply Chain Security Tools - Sign`, which is deprecated. Support and maintenance for `Supply Chain Security Tools - Sign` continues. Monitor Release Notes for updates.

Supply Chain Security Tools - Policy Controller is based on Sigstore's Policy Controller and is compatible only with `cosign` signatures. See [Cosign](#) and [Policy Controller](#) in GitHub. For information about image signing and verification, see [Sigstore](#) open source community and the [cosign](#) project in GitHub.

The Policy Controller component is a policy enforcement tool only. It does not sign images. Operators can configure image signing for their containers in several ways, including:

- By using [Tanzu Build Service](#)
- By using [kpack](#)
- By integrating [cosign](#) into their build pipelines

Image signatures generated by `cosign` are stored in the same registry location as the image itself unless configured with the `COSIGN_REPOSITORY` environment variable. Policy Controller uses registry credentials provided in the admission request, Service Account, or `signaturePullSecrets` defined in the policy to connect to the registry to verify a signature.



#### Important

This component does not work with insecure registries.

To Install Supply Chain Security Tools - Policy Controller, see [Install Supply Chain Security Tools - Policy Controller](#)

## Overview of Supply Chain Security Tools for VMware Tanzu - Policy Controller

Supply Chain Security Tools - Policy Controller is a security tool that helps you ensure that the container images in their registry have not been tampered with. Policy Controller is a Kubernetes Admission Controller that allows you to apply policies to verify signatures on container images before being admitted to a cluster.

The Policy Controller:

- Verifies signatures on container images used by Kubernetes resources
- Enforces policies to allow or deny images being admitted a cluster
- Allows operators to define multiple policies in the cluster
- Allows operators to select which `namespaces` to enforce policies against
- Supports `cosign` signatures and keyless signing
- Supports storing public keys in a KMS

It enforces its policies against all resources that create `Pods` as part of their life cycle:

- `Pod`
- `ReplicaSet`
- `Deployment`
- `Job`
- `StatefulSet`
- `DaemonSet`
- `CronJob`

**Note**

This component is the successor to [Supply Chain Security Tools - Sign](#), which is deprecated. Support and maintenance for [Supply Chain Security Tools - Sign](#) continues. Monitor Release Notes for updates.

Supply Chain Security Tools - Policy Controller is based on Sigstore's Policy Controller and is compatible only with `cosign` signatures. See [Cosign](#) and [Policy Controller](#) in GitHub. For information about image signing and verification, see [Sigstore](#) open source community and the `cosign` project in GitHub.

The Policy Controller component is a policy enforcement tool only. It does not sign images. Operators can configure image signing for their containers in several ways, including:

- By using [Tanzu Build Service](#)
- By using `kpack`
- By integrating `cosign` into their build pipelines

Image signatures generated by `cosign` are stored in the same registry location as the image itself unless configured with the `COSIGN_REPOSITORY` environment variable. Policy Controller uses registry credentials provided in the admission request, Service Account, or `signaturePullSecrets` defined in the policy to connect to the registry to verify a signature.

**Important**

This component does not work with insecure registries.

To Install Supply Chain Security Tools - Policy Controller, see [Install Supply Chain Security Tools - Policy Controller](#)

## Install Supply Chain Security Tools - Policy Controller

You install Supply Chain Security Tools - Policy Controller as part of Tanzu Application Platform's Full, Iterate, and Run profiles. You can use the instructions in this topic to manually install SCST - Policy Controller.

**Note**

Follow the steps in this topic if you do not want to use a profile to install Supply Chain Security Tools - Policy Controller. For more information about profiles, see [Components and installation profiles](#).

## Prerequisites

- Complete all prerequisites to install Tanzu Application Platform. For more information, see [Prerequisites](#).
- A container image registry that supports TLS connections.

**Important**



This component does not work with not secure registries.

- For keyless authorities support, you must set `policy.tuf_enabled: true`. By default, the public official Sigstore The Update Framework (TUF) server is used. To target an alternative Sigstore stack, specify `policy.tuf_mirror` and `policy.tuf_root`.
- If you are installing in an air-gapped environment and require keyless authorities, you must deploy a Sigstore Stack on the cluster or be accessible from the air-gapped environment. For information, see [Install Sigstore Stack](#).
- During configuration, you provide a cosign public key to validate signed images. The Policy Controller only supports ECDSA public keys. An example cosign public key is provided that can validate an image from the public cosign registry. To provide your own key and images, follow the [Cosign Quick Start Guide](#) in GitHub.



### Caution

This component rejects `Pods` if they are not correctly configured. Test your configuration in a test environment before applying policies to your production cluster.

## Install

To install Supply Chain Security Tools - Policy Controller:

1. List version information for the package by running:

```
tanzu package available list policy.apps.tanzu.vmware.com --namespace tap-install
```

For example:

```
$ tanzu package available list policy.apps.tanzu.vmware.com --namespace tap-install
- Retrieving package versions for policy.apps.tanzu.vmware.com...
NAME                                VERSION      RELEASED-AT
policy.apps.tanzu.vmware.com        1.2.0        2023-10-01 20:00:00 -0400 EDT
```

2. (Optional) Make changes to the default installation settings by running:

```
tanzu package available get policy.apps.tanzu.vmware.com/VERSION --values-schema --namespace tap-install
```

Where `VERSION` is the version number you discovered. For example, `1.2.0`.

For example:

```
$ tanzu package available get policy.apps.tanzu.vmware.com/1.2.0 --values-schema --namespace tap-install
| Retrieving package details for policy.apps.tanzu.vmware.com/1.2.0...

KEY                                DEFAULT      TYPE      DESCRIPTION
custom_cas                          <nil>        array     List of custom CA contents 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                 cosign-system string      Deployment namespace specifies the namespace where this component should be deployed to.
```

|                           |       |         |                                                                                                                                             |
|---------------------------|-------|---------|---------------------------------------------------------------------------------------------------------------------------------------------|
|                           |       |         | If not specified, "cosign-sy                                                                                                                |
| stem"                     |       |         | stem" is assumed.                                                                                                                           |
| fail_on_empty_authorities | true  | boolean | Configure if a ClusterImageP                                                                                                                |
| olicy                     |       |         | olicy will fail or allow empty authorities                                                                                                  |
| limits_cpu                | 200m  | string  | The CPU limit defines a hard                                                                                                                |
| ceiling                   |       |         | ceiling on how much CPU time                                                                                                                |
|                           |       |         | that the Policy Controller m                                                                                                                |
| anager                    |       |         | anager container can use.                                                                                                                   |
|                           |       |         | <a href="https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/#meaning-of-cpu">https://kubernetes.io/docs/c</a>    |
| oncepts/                  |       |         | oncepts/configuration/manage-resources-containers/#meaning-of-cpu                                                                           |
| no_match_policy           | deny  | string  | The action when no policy ma                                                                                                                |
| tches                     |       |         | tches the admitting image digest. Valid values are "warn", "allow", or "deny".                                                              |
| quota.pod_number          | 6     | string  | The maximum number of Policy                                                                                                                |
| Controller                |       |         | Controller Pods allowed to be created with the priority class                                                                               |
|                           |       |         | system-cluster-critical. Thi                                                                                                                |
| s                         |       |         | s value must be enclosed in quotes (""). If this value is not                                                                               |
|                           |       |         | specified then a default val                                                                                                                |
| ue                        |       |         | ue of 6 is used.                                                                                                                            |
| replicas                  | 1     | integer | The number of replicas to be                                                                                                                |
| created                   |       |         | created for the Policy Controller. This value must not be enclosed                                                                          |
|                           |       |         | in quotes. If this value is                                                                                                                 |
| not                       |       |         | not specified then a default value of 1 is used.                                                                                            |
| requests_memory           | 20Mi  | string  | The memory request defines t                                                                                                                |
| he                        |       |         | he minium memory amount for the Policy Controller manager.                                                                                  |
|                           |       |         | <a href="https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/#meaning-of-memory">https://kubernetes.io/docs/c</a> |
| oncepts/                  |       |         | oncepts/configuration/manage-resources-containers/#meaning-of-memory                                                                        |
| tuf_root                  |       | string  | The root.json file content o                                                                                                                |
| f                         |       |         | f the TUF mirror                                                                                                                            |
| custom_ca_secrets         | <nil> | array   | List of custom CA secrets th                                                                                                                |
| at                        |       |         | at should be included in the application container for registry communication.                                                              |
|                           |       |         | An array of secret reference                                                                                                                |
| s                         |       |         | s each containing a secret_name field with the secret name to be referenced                                                                 |
|                           |       |         | and a namespace field with t                                                                                                                |
| he                        |       |         | he name of the namespace where the referred secret resides.                                                                                 |
| limits_memory             | 200Mi | string  | The memory limit defines a h                                                                                                                |
| ard                       |       |         | ard ceiling on how much memory                                                                                                              |
|                           |       |         | that the Policy Controller m                                                                                                                |
| anager                    |       |         | anager container can use.                                                                                                                   |
|                           |       |         | <a href="https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/#meaning-of-memory">https://kubernetes.io/docs/c</a> |
| oncepts/                  |       |         | oncepts/configuration/manage-resources-containers/#meaning-of-memory                                                                        |
| requests_cpu              | 20m   | string  | The CPU request defines the                                                                                                                 |
| minimum                   |       |         | minimum CPU time for the Policy                                                                                                             |
|                           |       |         | Controller manager. During C                                                                                                                |
| PU                        |       |         | PU contention, CPU request is used as                                                                                                       |
|                           |       |         | a weighting where higher CPU                                                                                                                |
| requests                  |       |         | requests are allocated more CPU time.                                                                                                       |
|                           |       |         | <a href="https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/#meaning-of-cpu">https://kubernetes.io/docs/c</a>    |
| oncepts/                  |       |         | oncepts/configuration/manage-resources-containers/#meaning-of-cpu                                                                           |
| tuf_mirror                |       | string  | TUF mirror address                                                                                                                          |

### 3. Create a file named `scst-policy-values.yaml` and add the settings you want to customize:

- `custom_ca_secrets`: If your container registries are secured by self-signed certificates, this setting controls which secrets are added to the application container as custom certificate authorities (CAs). `custom_ca_secrets` consists of an array of items. Each item contains two text boxes: the `secret_name` text box defines the name of the secret, and the `namespace` text box defines the name of the namespace where said secret is stored.

For example:

```

custom_ca_secrets:
- secret_name: first-ca
  namespace: ca-namespace
- secret_name: second-ca
  namespace: ca-namespace

```



#### Note

This setting is allowed even if `custom_cas` is defined.

- `custom_cas`: This setting enables adding certificate content in PEM format. The certificate content is added to the application container as custom certificate authorities (CAs) to communicate with registries deployed with self-signed certificates. `custom_cas` consists of an array of items. Each item contains a single text box named `ca_content`. The value of this text box must be a PEM-formatted certificate authority. The certificate content must be defined as a YAML block, preceded by the literal indicator (`|`) to preserve line breaks and ensure that the certificates are interpreted correctly.

For example:

```

custom_cas:
- ca_content: |
    ----- BEGIN CERTIFICATE -----
    first certificate content here...
    ----- END CERTIFICATE -----
- ca_content: |
    ----- BEGIN CERTIFICATE -----
    second certificate content here...
    ----- END CERTIFICATE -----

```



#### Note

This setting is allowed even if `custom_ca_secrets` is defined.

- `deployment_namespace`: This setting controls the namespace to which this component is deployed. When not specified, the namespace `cosign-system` is assumed. This component creates the specified namespace to deploy required resources. Select a namespace that is not used by any other components.
- `limits_cpu`: This setting controls the maximum CPU resource allocated to the Policy admission controller. The default value is “200m”. See [Kubernetes documentation](#).
- `limits_memory`: This setting controls the maximum memory resource allocated to the Policy admission controller. The default value is “200Mi”. See [Kubernetes documentation](#).
- `quota.pod_number`: This setting controls the maximum number of pods that are allowed in the deployment namespace with the `system-cluster-critical` priority class. This priority class is added to the pods to prevent preemption of this component’s pods in case of node pressure.

The default value for this text box is `6`. If your use requires more than 6 pods, change this value to allow the number of replicas you intend to deploy.



#### Note

VMware recommends to run this component with a critical priority level to prevent the cluster from rejecting all admission requests if the component's `Pods` are evicted due to resource limits.

- `replicas`: This setting controls the default amount of replicas deployed by this component. The default value is `1`.

**For production environments:** VMware recommends you increase the number of replicas to `3` to ensure that the availability of the component and better admission performance.

- `requests_cpu`: This setting controls the minimum CPU resource allocated to the Policy admission controller. During CPU contention, this value is used as a weighting where higher values indicate more CPU time is allocated. The default value is `20m`. See [CPU resource units](#) in the Kubernetes documentation.
- `requests_memory`: This setting controls the minimum memory resource allocated to the Policy admission controller. The default value is `20Mi`. See [Memory resource units](#) in the Kubernetes documentation.
- `tuf_enabled`: This setting defines whether the TUF initialization is done on startup. It is required for keyless verification support. The default value is `false`, which means that keyless authorities of `ClusterImagePolicy` are not supported. Also, policy-controller does not have an external dependency on setup.
- `tuf_root`: The root.json file content of the TUF mirror.
- `tuf_mirror`: This setting defines the TUF mirror address which is used for doing the initialization.
- `no_match_policy`: The action when no policy matches the admitting image digest. Valid values are `"warn"`, `"allow"`, or `"deny"`. Default value is `"deny"`
- `fail_on_empty_authorities`: Failing or allowing empty authorities when adding a new `ClusterImagePolicy`. Default value is `true`.

#### 4. Install the package:

```
tanzu package install policy-controller \
  --package-name policy.apps.tanzu.vmware.com \
  --version VERSION \
  --namespace tap-install \
  --values-file scst-policy-values.yaml
```

Where `VERSION` is the version number you discovered earlier. For example, `1.2.0`.

For example:

```
$ tanzu package install policy-controller \
  --package-name policy.apps.tanzu.vmware.com \
  --version 1.2.0 \
  --namespace tap-install \
  --values-file scst-policy-values.yaml

Installing package 'policy.apps.tanzu.vmware.com'
Getting package metadata for 'policy.apps.tanzu.vmware.com'
Creating service account 'policy-controller-tap-install-sa'
Creating cluster admin role 'policy-controller-tap-install-cluster-role'
Creating cluster role binding 'policy-controller-tap-install-cluster-rolebinding'
Creating package resource
Waiting for 'PackageInstall' reconciliation for 'policy-controller'
'PackageInstall' resource install status: Reconciling
```

```
'PackageInstall' resource install status: ReconcileSucceeded
'PackageInstall' resource successfully reconciled

Added installed package 'policy-controller'
```

After you run the commands earlier the policy controller is running.

Policy Controller is now installed, but it does not enforce any policies by default. Policies must be explicitly configured on the cluster. To configure signature verification policies, see [Configuring Supply Chain Security Tools - Policy](#).

## Install Sigstore Stack

**Note:** VMware does not support Sigstore stack deployments. However, a sample version 0.4.8 deployment that uses keyless signing and verification in air-gapped environments is shown in this section.

[Sigstore/scaffolding](#) is used for bringing up the Sigstore Stack.

The Sigstore Stack consists of:

- [Trillian](#)
- [Rekor](#)
- [Fulcio](#)
- [Certificate Transparency Log \(CTLog\)](#)
- [TheUpdateFramework \(TUF\)](#)

For information about air-gapped installation, see [Install Tanzu Application Platform in an air-gapped environment](#).

If a Sigstore Stack TUF is already deployed and accessible in the air-gapped environment, proceed to [Update Policy Controller with TUF Mirror and Root](#).

## Download Stack Release Files

Download the release files of all the Sigstore Stack components from [Sigstore/scaffolding](#):

```
RELEASE_VERSION="v0.4.8"

TRILLIAN_URL="https://github.com/sigstore/scaffolding/releases/download/${RELEASE_VERSION}/release-trillian.yaml"
REKOR_URL="https://github.com/sigstore/scaffolding/releases/download/${RELEASE_VERSION}/release-rekor.yaml"
FULCIO_URL="https://github.com/sigstore/scaffolding/releases/download/${RELEASE_VERSION}/release-fulcio.yaml"
CTLOG_URL="https://github.com/sigstore/scaffolding/releases/download/${RELEASE_VERSION}/release-ctlog.yaml"
TUF_URL="https://github.com/sigstore/scaffolding/releases/download/${RELEASE_VERSION}/release-tuf.yaml"

curl -sL "${TRILLIAN_URL}" -o "release-trillian.yaml"
curl -sL "${REKOR_URL}" -o "release-rekor.yaml"
curl -sL "${FULCIO_URL}" -o "release-fulcio.yaml"
curl -sL "${CTLOG_URL}" -o "release-ctlog.yaml"
curl -sL "${TUF_URL}" -o "release-tuf.yaml"
```

## Migrate Images onto Internal Registry

For air-gapped environments, you must migrate the images from the `release-*.yaml` to the internal air-gapped registry and update the corresponding image references.

The following example shows this migration:

```
TARGET_REGISTRY=TARGET-REGISTRY

Where `TARGET-REGISTRY` is the name of the registry you want to migrate to.

# Use yq to find all "image" keys from the release-*.yaml downloaded
found_images=$(yq eval '.. | select(has("image")) | .image' release-*.yaml | grep --invert-match -- '---')

# Loop through each found image
# Pull, retag, push the images
# Update the found image references in all the release-*.yaml
for image in "${found_images[@]"; do
  if echo "${image}" | grep -q '@'; then
    # If image is a digest reference
    image_ref=$(echo "${image}" | cut -d'@' -f1)
    image_sha=$(echo "${image}" | cut -d'@' -f2)
    image_path=$(echo "${image_ref}" | cut -d'/' -f2-)

    docker pull "${image}"
    docker tag "${image}" "${TARGET_REGISTRY}/${image_path}"
    # Obtain the new sha256 from the `docker push` output
    new_sha=$(docker push "${TARGET_REGISTRY}/${image_path}" | tail -n1 | cut -d' ' -f3)

    new_reference="${TARGET_REGISTRY}/${image_path}@${new_sha}"
  else
    # If image is a tag reference
    image_path=$(echo ${image} | cut -d'/' -f2-)

    docker pull ${image}
    docker tag ${image} ${TARGET_REGISTRY}/${image_path}
    docker push ${TARGET_REGISTRY}/${image_path}

    new_reference="${TARGET_REGISTRY}/${image_path}"
  fi

  # Replace the image reference with the new reference in all the release-*.yaml
  sed -i.bak -E "s:image: ${image}#image: ${new_reference}#" release-*.yaml
done
```

During Sigstore Stack deployment, a sidecar image such as `queue-proxy`, can require additional credentials. You can achieve this by adding a `secretgen` annotated placeholder secret to the target namespace and patching the corresponding service account. The placeholder imports the `tap-registry` secret to the targeted namespace.

```
# <SERVICE> includes "trillian", "rekor", "fulcio", "ctlog", and "tuf"
echo "Create tap-registry secret import"
cat <<EOF | kubectl apply -f -
---
apiVersion: v1
kind: Secret
metadata:
  name: tap-registry
  namespace: SERVICE-system
annotations:
  secretgen.carvel.dev/image-pull-secret: ""
stringData:
  .dockerconfigjson: "{}"
type: kubernetes.io/dockerconfigjson
```

```
EOF

echo "Patch SERVICE service account"
kubectl -n SERVICE-system patch serviceaccount SERVICE -p '{"imagePullSecrets": [{"name": "tap-registry"}]}'
```

Where `SERVICE` is the name of the service you want to configure with your target namespace.

## Copy Release Files to Cluster Accessible Machine

With the images migrated and accessible, copy the `release-*.yaml` files onto the cluster accessible machine that is installing the Sigstore Stack with Kubernetes cluster access.

## Prepare Patching Fulcio Release File

The default `release-fulcio.yaml` has a `fulcio-config` resource. This config specifies the `OIDCIssuer`. By default, there are issuers for:

- `Kubernetes API ServiceAccount token`
- `Google Accounts`
- `Sigstore OAuth2`
- `Github Action Token`

To add other OIDC Issuers, configure `fulcio-config` further.

Apart from `Kubernetes API ServiceAccount token`, the other `OIDCIssuers` require access to external services. Your cluster must have an OIDC issuer enabled to configure `OIDCIssuers` correctly. If you don't need keyless signatures, you can remove the `OIDCIssuers` entry. In an air-gapped environment, you must remove these `OIDCIssuers`.

You can add the correct `Metalssuers` for your IaaS environment.

A `config_json` is constructed and then applied to the `release-fulcio.yaml`.

## OIDCIssuer

A `config_json` containing the `Kubernetes API ServiceAccount token` issuer:

```
config_json='{
  "OIDCIssuers": {
    "https://kubernetes.default.svc": {
      "IssuerURL": "https://kubernetes.default.svc",
      "ClientID": "sigstore",
      "Type": "kubernetes"
    }
  },
  "MetaIssuers": {
    "https://kubernetes.*.svc": {
      "ClientID": "sigstore",
      "Type": "kubernetes"
    }
  }
}'
```

Set the `IssuerURL` to the OIDC issuer configured in your cluster. You can discover your the URL by using `kubectl proxy -p 8001` and running:

```
curl localhost:8001/.well-known/openid-configuration | jq .issuer
```

Then set the `OIDCIssuer` to the value returned in the last command.

Other sample `OIDCIssuers`:

```
config_json='{
  "OIDCIssuers": {
    "https://accounts.google.com": {
      "IssuerURL": "https://accounts.google.com",
      "ClientID": "sigstore",
      "Type": "email"
    },
    "https://allow.pub": {
      "IssuerURL": "https://allow.pub",
      "ClientID": "sigstore",
      "Type": "spiffe",
      "SPIFFETrustDomain": "allow.pub"
    },
    "https://oauth2.sigstore.dev/auth": {
      "IssuerURL": "https://oauth2.sigstore.dev/auth",
      "ClientID": "sigstore",
      "Type": "email",
      "IssuerClaim": "$.federated_claims.connector_id"
    },
    "https://token.actions.githubusercontent.com": {
      "IssuerURL": "https://token.actions.githubusercontent.com",
      "ClientID": "sigstore",
      "Type": "github-workflow"
    }
  }
}'
```

## MetaIssuers

If installing on EKS, update the `config_json` to include this `MetaIssuer`:

```
config_json='{
  "MetaIssuers": {
    ...

    "https://oidc.eks.*.amazonaws.com/id/*": {
      "ClientID": "sigstore",
      "Type": "kubernetes"
    }
  }
}'
```

If installing on GCP, update the `config_json` to include this `MetaIssuer`:

```
config_json='{
  "MetaIssuers": {
    ...

    "https://container.googleapis.com/v1/projects/*/locations/*/clusters/*": {
      "ClientID": "sigstore",
      "Type": "kubernetes"
    }
  }
}'
```

If installing on AKS, update the `config_json` to include this `MetaIssuer`:

```
config_json='{
  "MetaIssuers": {
    ...
```



```
"https://oidc.prod-aks.azure.com/*": {
  "ClientID": "sigstore",
  "Type": "kubernetes"
}
}'
```

## Applying the patch for Fulcio release file

After configuring the required `config_json`, you can apply it by manually editing the `release-fulcio.yaml` file or by running:

```
# Use `yq` to find the correct fulcio-config resource
# Update the `data.config.json` property with the new config JSON string
config_json="${config_json}" \
yq e '. |
  select(.metadata.name == "fulcio-config") as $config |
  select(.metadata.name != "fulcio-config") as $other |
  $config.data["config.json"] = strenv(config_json) |
  ($other, $config)' -i release-fulcio.yaml
```

## Patch Knative-Serving

Knative Serving might already be deployed, depending on the selected profile, during the first attempt of installing Tanzu Application Platform. Knative Serving is required to continue deploying the Sigstore Stack. If Knative is not present, install it. See [Install Cloud Native Runtimes](#).

With the Sigstore Stack deployment, you must update Knative Serving's `configmap/config-features` to enable required features. Run:

```
kubectl patch configmap/config-features \
--namespace knative-serving \
--type merge \
--patch '{"data":{"kubernetes.podspec-fieldref":"enabled", "kubernetes.podspec-volumes-emptydir":"enabled", "multicontainer":"enabled"}}'
```

## Create OIDC Reviewer Binding

To fetch public keys and validate the JWT tokens from the [Discovery Document](#), you must allow unauthenticated requests.

```
kubectl create clusterrolebinding oidc-reviewer \
--clusterrole=system:service-account-issuer-discovery \
--group=system:unauthenticated
```

For more information, see [Service Account Issuer Discovery](#) in the Kubernetes documentation.

## Install Trillian

To install Trillian:

1. `kubectl apply` the `release-trillian.yaml`.
2. Add the `secretgen` placeholder for `secretgen` to import `tap-registry` secret to the namespace for `queue-proxy`.
3. Patch the service account to use the imported `tap-registry` secret.
4. Wait for the jobs and services to be `Complete` or be `Ready`.

```

echo 'Install Trillian'
kubectl apply -f "release-trillian.yaml"

echo "Create tap-registry secret import"
cat <<EOF | kubectl apply -f -
---
apiVersion: v1
kind: Secret
metadata:
  name: tap-registry
  namespace: trillian-system
  annotations:
    secretgen.carvel.dev/image-pull-secret: ""
stringData:
  .dockerconfigjson: "{}"
type: kubernetes.io/dockerconfigjson
EOF

echo "Patch trillian service account"
kubectl -n trillian-system patch serviceaccount trillian -p '{"imagePullSecrets": [{"name": "tap-registry"}]}'

echo 'Restart trillian deployment if tap-registry secret was required'
kubectl -n trillian-system rollout restart deployment/log-server-00001-deployment
kubectl -n trillian-system rollout restart deployment/log-signer-00001-deployment

echo 'Wait for Trillian ready'
kubectl wait --timeout 2m -n trillian-system --for=condition=Ready ksvc log-server
kubectl wait --timeout 2m -n trillian-system --for=condition=Ready ksvc log-signer

```

## Install Rekor

To install Rekor:

1. `kubectl apply` the `release-rekor.yaml`.
2. Add the `secretgen` placeholder for `secretgen` to import `tap-registry` secret to the namespace for `queue-proxy`.
3. Patch the service account to use the imported `tap-registry` secret.
4. Wait for the jobs and services to be `Complete` or be `Ready`.

```

echo 'Install Rekor'
kubectl apply -f "release-rekor.yaml"

echo "Create tap-registry secret import"
cat <<EOF | kubectl apply -f -
---
apiVersion: v1
kind: Secret
metadata:
  name: tap-registry
  namespace: rekor-system
  annotations:
    secretgen.carvel.dev/image-pull-secret: ""
stringData:
  .dockerconfigjson: "{}"
type: kubernetes.io/dockerconfigjson
EOF

echo "Patch rekor service account"
kubectl -n rekor-system patch serviceaccount rekor -p '{"imagePullSecrets": [{"name": "tap-registry"}]}'

```

```

echo 'Restart rekor deployment if tap-registry secret was required'
kubectl -n rekor-system rollout restart deployment/rekor-00001-deployment

echo 'Wait for Rekor ready'
kubectl wait --timeout 5m -n rekor-system --for=condition=Complete jobs --all
kubectl wait --timeout 2m -n rekor-system --for=condition=Ready ksvc rekor

```

## Install Fulcio

To install Fulcio:

1. `kubectl apply` the `release-fulcio.yaml`.
2. Add the `secretgen` placeholder for `secretgen` to import `tap-registry` secret to the namespace for `queue-proxy`.
3. Patch the service account to use the imported `tap-registry` secret.
4. Wait for the jobs and services to be `Complete` or be `Ready`.

The Sigstore Scaffolding `release-fulcio.yaml` downloaded can have an empty YAML document at the end of the file separated by `---` and followed by no elements. This results in:

```

error: error validating "release-fulcio.yaml": error validating data: [apiVersion not set, kind not set]; if you choose to ignore these errors, turn validation off with --validate=false

```

This is a known issue and you can ignore it.

```

echo 'Install Fulcio'
kubectl apply -f "release-fulcio.yaml"

echo "Create tap-registry secret import"
cat <<EOF | kubectl apply -f -
---
apiVersion: v1
kind: Secret
metadata:
  name: tap-registry
  namespace: fulcio-system
  annotations:
    secretgen.carvel.dev/image-pull-secret: ""
stringData:
  .dockerconfigjson: "{}"
type: kubernetes.io/dockerconfigjson
EOF

echo "Patch fulcio service account"
kubectl -n fulcio-system patch serviceaccount fulcio -p '{"imagePullSecrets": [{"name": "tap-registry"}]}'

echo 'Restart fulcio deployment if tap-registry secret was required'
kubectl -n fulcio-system rollout restart deployment/fulcio-00001-deployment

echo 'Wait for Fulcio ready'
kubectl wait --timeout 5m -n fulcio-system --for=condition=Complete jobs --all
kubectl wait --timeout 5m -n fulcio-system --for=condition=Ready ksvc fulcio

```

## Install Certificate Transparency Log (CTLog)

To install CTLog:

1. `kubectl apply` the `release-ctlog.yaml`.
2. Add the `secretgen` placeholder for `secretgen` to import `tap-registry` secret to the namespace for `queue-proxy`.
3. Patch the service account to use the imported `tap-registry` secret.
4. Wait for the jobs and services to be `Complete` or be `Ready`.

```

echo 'Install CTLog'
kubectl apply -f "release-ctlog.yaml"

echo "Create tap-registry secret import"
cat <<EOF | kubectl apply -f -
---
apiVersion: v1
kind: Secret
metadata:
  name: tap-registry
  namespace: ctlog-system
  annotations:
    secretgen.carvel.dev/image-pull-secret: ""
stringData:
  .dockerconfigjson: "{}"
type: kubernetes.io/dockerconfigjson
EOF

echo "Patch ctlog service account"
kubectl -n ctlog-system patch serviceaccount ctlog -p '{"imagePullSecrets": [{"name": "tap-registry"}]}'

echo 'Restart ctlog deployment if tap-registry secret was required'
kubectl -n ctlog-system rollout restart deployment/ctlog-00001-deployment

echo 'Wait for CTLog ready'
kubectl wait --timeout 5m -n ctlog-system --for=condition=Complete jobs --all
kubectl wait --timeout 2m -n ctlog-system --for=condition=Ready ksvc ctlog

```

## Install TUF

To install TUF:

1. If you are using OpenShift, add a `RoleBinding`.
2. `kubectl apply` the `release-tuf.yaml`.
3. Add the `secretgen` placeholder for `secretgen` to import `tap-registry` secret to the namespace for `queue-proxy`.
4. Patch the service account to use the imported `tap-registry` secret.
5. Copy the public keys from the previous deployment of CTLog, Fulcio, and Rekor to the TUF namespace.
6. Wait for the jobs and services to be `Complete` or be `Ready`.

If you are using OpenShift, you must set the correct Security Context Constraints so the TUF server can write to the root file system. This is done by adding the `anyuid` Security Context Constraint through a `RoleBinding`:

```

cat <<EOF >> release-tuf.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: tuf-os-scc-role-binding

```

```

  namespace: tuf-system
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:openshift:scc:anyuid
subjects:
- kind: ServiceAccount
  namespace: tuf-system
  name: tuf
EOF

```

Now proceed to install TUF:

```

echo 'Install TUF'
kubect1 apply -f "release-tuf.yaml"

echo "Create tap-registry secret import"
cat <<EOF | kubect1 apply -f -
---
apiVersion: v1
kind: Secret
metadata:
  name: tap-registry
  namespace: tuf-system
  annotations:
    secretgen.carvel.dev/image-pull-secret: ""
stringData:
  .dockerconfigjson: "{}"
type: kubernetes.io/dockerconfigjson
EOF

echo "Patch tuf service account"
kubect1 -n tuf-system patch serviceaccount tuf -p '{"imagePullSecrets": [{"name": "tap-registry"}]}'

# Then copy the secrets (even though it's all public stuff, certs, public keys)
# to the tuf-system namespace so that we can construct a tuf root out of it.
kubect1 -n ctlog-system get secrets ctlog-public-key -oyaml | sed 's/namespace: .*/namespace: tuf-system/' | kubect1 apply -f -
kubect1 -n fulcio-system get secrets fulcio-pub-key -oyaml | sed 's/namespace: .*/namespace: tuf-system/' | kubect1 apply -f -
kubect1 -n rekor-system get secrets rekor-pub-key -oyaml | sed 's/namespace: .*/namespace: tuf-system/' | kubect1 apply -f -

echo 'Wait for TUF ready'
kubect1 wait --timeout 4m -n tuf-system --for=condition=Complete jobs --all
kubect1 wait --timeout 2m -n tuf-system --for=condition=Ready ksvc tuf

```

## Update Policy Controller with TUF Mirror and Root

Obtain the `root.json` file from the `tuf-system` namespace. Run:

```

kubect1 -n tuf-system get secrets tuf-root -o jsonpath='{.data.root}' | base64 -d > root.json

```

Update the `tap-values` that are used for installation of Tanzu Application Platform.

If the internally deployed TUF is used, `tuf_mirror` is `http://tuf.tuf-system.svc`. If the mirror is hosted elsewhere, provide the correct mirror URL. The default public TUF instance mirror URL is `https://sigstore-tuf-root.storage.googleapis.com`.

The `tuf_root` is the contents of the obtained `root.json` from the `tuf-root` secret in the `tuf-system` namespace. The public TUF instance's `root.json`.

If Policy Controller was installed through Tanzu Application Profiles, update the values file with:

```
policy:
  tuf_mirror: http://tuf.tuf-system.svc
  tuf_root: |
    MULTI-LINE-ROOT-JSON
```

Where `MULTI-LINE-ROOT-JSON` is a multi-line string content of from your `root.json` file.

When updating the current Tanzu Application Platform installed through profiles with the updated values file, the previously failing Tanzu Application Platform `PackageInstall` has the following error:

```
tanzu package installed update tap --values-file tap-values-updated.yaml -n tap-install
1
Updating installed 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'

Error: resource reconciliation failed: kapp: Error: waiting on reconcile packageinstall
1/policy-controller (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
```

Although the command fails, the values file is updated in the installation secrets. During the next reconciliation cycle, the package attempts to reconcile and sync with the expected configuration. At that point, Policy Controller updates and reconciles with the latest values.

If Policy Controller was installed standalone or updated manually, update the values file with:

```
tuf_mirror: http://tuf.tuf-system.svc
tuf_root: |
  MULTI-LINE-ROOT-JSON
```

Where `MULTI-LINE-ROOT-JSON` is a multi-line string content of from your `root.json` file.

Run with the values file configured for Policy Controller only:

```
tanzu package installed update policy-controller --values-file tap-values-standalone.y
aml -n tap-install
Updating installed package 'policy-controller'
Getting package install for 'policy-controller'
Getting package metadata for 'policy.apps.tanzu.vmware.com'
Creating secret 'policy-controller-tap-install-values'
Updating package install for 'policy-controller'
Waiting for 'PackageInstall' reconciliation for 'policy-controller'
'PackageInstall' resource install status: Reconciling
'PackageInstall' resource install status: ReconcileSucceeded
'PackageInstall' resource successfully reconciled
Updated installed package 'policy-controller' in namespace 'tap-install'
```

This updates the policy-controller only. It is important that if Policy Controller was installed through the Tanzu Application Platform package with profiles, the `update` command to update the Tanzu Application Platform installation is still required, as it updates the values file. If only the Policy Controller package is updated with new values and not the Tanzu Application Platform package's values, the Tanzu Application Platform package's values overwrite the Policy Controller's values.

For more information about profiles, see [Package Profiles](#). For more information about Policy Controller, see [Install Supply Chain Security Tools - Policy Controller](#) documentation.

## Uninstall Sigstore Stack

To uninstall Sigstore Stack, run:

```
kubectl delete -f "release-tuf.yaml"
kubectl delete -f "release-ctlog.yaml"
kubectl delete -f "release-fulcio.yaml"
kubectl delete -f "release-rekor.yaml"
kubectl delete -f "release-trillian.yaml"
```

## Migration From Supply Chain Security Tools - Sign

This topic explains how you can migrate from Supply Chain Security Tools - Sign to Supply Chain Security Tools - Policy.

In Tanzu Application Platform v1.4, the Image Policy Webhook is removed. If the Image Policy Webhook was used with the previous Tanzu Application Platform versions in your cluster, you must migrate the `ClusterImagePolicy` resource from Image Policy Webhook to Policy Controller. For information about additional features introduced in SCST - Policy, see [Configuring Supply Chain Security Tools - Policy](#).

## Enable Policy Controller on Namespaces

Policy Controller works with an opt-in system. Operators must update namespaces with the label `policy.sigstore.dev/include: "true"` to the namespace resource to enable Policy Controller verification.

```
kubectl label namespace my-secure-namespace policy.sigstore.dev/include=true
```



### Caution

Without a Policy Controller `ClusterImagePolicy` applied, there are fallback behaviors where images are validated against the public Sigstore Rekor and Fulcio servers by using a keyless authority flow. Therefore, if the deploying image is signed publicly by a third-party using the keyless authority flow, the image are admitted as it can validate against the public Rekor and Fulcio. To avoid this behavior, develop, and apply a `ClusterImagePolicy` that applies to the images being deployed in the namespace.

## Policy Controller ClusterImagePolicy

The Policy Controller `ClusterImagePolicy` does not have a name. Image Policy Controller required that the `ClusterImagePolicy` be named `image-policy` and that there be only one `ClusterImagePolicy`. Multiple Policy Controller `ClusterImagePolicies` are applied. During validation, all `ClusterImagePolicy` that have an image `glob` pattern that matches the deploying image is evaluated. All matched `ClusterImagePolicies` must be valid. For a `ClusterImagePolicy` to be valid, at least one authority in the policy must validate the signature of the deploying image.

## Excluding Namespaces

The namespaces listed in `spec.verification.exclude.resources.namespaces[]` must have `policy.sigstore.dev/include` set to `false` or not be set. Therefore, they are exempted from Policy Controller validation.

#### Image Policy Webhook:

```
---
apiVersion: signing.apps.tanzu.vmware.com/v1beta1
kind: ClusterImagePolicy
metadata:
  name: image-policy
spec:
  verification:
    ...

  exclude:
    resources:
      namespaces:
      - image-policy-system
      - kube-system
      - cert-manager

    ...
```

## Specifying Public Keys

`spec.verification.keys[].publicKey` from Image Policy Webhook is mapped to `spec.authorities[].key.data` for Policy Controller.

The `name` associated with each `key` is no longer required. Image Policy Webhook has direct association between `key` name and `imagePattern`. For Policy Controller, multiple `ClusterImagePolicy` resources are defined to create direct association between image patterns and key authorities.

Image patterns and keys are scoped to each `ClusterImagePolicy` resource.

Therefore, to have direct association be isolated between `key` and `imagePattern`, multiple Policy Controller `ClusterImagePolicy` must be created. Each `ClusterImagePolicy` has the image glob pattern defined and the associated key authorities defined.

#### Image Policy Webhook:

```
---
apiVersion: signing.apps.tanzu.vmware.com/v1beta1
kind: ClusterImagePolicy
metadata:
  name: image-policy
spec:
  verification:
    ...

  keys:
  - name: official-cosign-key
    publicKey: |
      -----BEGIN PUBLIC KEY-----
      MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEhyQCx0E9wQWSFI9ULGwy3BuRklnt
      IqozONbbdbqz11hlRJy9c7SG+hdCF19jE9uE/dwtuwU2MqU9T/cN0YkWww==
      -----END PUBLIC KEY-----

    ...
```

#### Policy Controller:



```

---
apiVersion: policy.sigstore.dev/v1beta1
kind: ClusterImagePolicy
metadata:
  name: POLICY-NAME
spec:
  authorities:
    ...

  - key:
    data: |
      -----BEGIN PUBLIC KEY-----
      MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEhyQCx0E9wQWSFI9ULGwy3BuRk1nt
      IqozONbdblqz11hlRJy9c7SG+hdcF19jE9uE/dwtuwU2MqU9T/cN0YkWww==
      -----END PUBLIC KEY-----

    ...

```

Where `POLICY-NAME` is the name of the cluster image policy you want to use.

## Specifying Image Matching

`spec.verification.images[].namePattern` from Image Policy Webhook maps to `spec.images[].glob` for Policy Controller.

Policy Controller follows more closely to `glob` matching. For the Image Policy Webhook, `registry.com/*` wildcards all projects and images under the registry. However, `glob` matching uses `/` separator delimiting. Therefore, the `glob` wildcard matching equivalent is `registry.com/**/*`. The `**` allows for recursive project path matching while the trailing `*` images found in the terminating project path.

If only one level of pathing is required, the `glob` pattern is `registry.com/*/*`.

Policy Controller has defaults defined. If `*` is specified, the `glob` matching behavior is `index.docker.io/library/*`. If `**/*` is specified, the `glob` matching behavior is `index.docker.io/**/*`. With these defaults, the `glob` pattern `**` matches against all images.

### Image Policy Webhook:

```

---
apiVersion: signing.apps.tanzu.vmware.com/v1beta1
kind: ClusterImagePolicy
metadata:
  name: image-policy
spec:
  verification:
    ...

  images:
  - namePattern: gcr.io/projectsigstore/cosign*
    keys:
    - name: official-cosign-key
      secretRef:
        name: your-secret
        namespace: your-namespace

    ...

```

### Policy Controller:

```

---
apiVersion: policy.sigstore.dev/v1beta1
kind: ClusterImagePolicy

```

```

metadata:
  name: POLICY-NAME
spec:
  images:
  - glob: gcr.io/projectsigstore/cosign*

```

Where `POLICY-NAME` is the name of the cluster image policy you want to use.

## Specifying policy mode

If `AllowUnmatchedImages` is set to `true` in the Image Policy Webhook deployment, create the following policy in the cluster:

```

---
apiVersion: policy.sigstore.dev/v1beta1
kind: ClusterImagePolicy
metadata:
  name: allow-unmatched-image-policy
spec:
  images:
  - glob: "*"
    authorities:
    - static:
        action: pass

```

## Configuring Supply Chain Security Tools - Policy

This topic describes how you can configure Supply Chain Security Tools - Policy. SCST - Policy requires extra configuration steps to verify your container images.

## Admission of Images

An image is admitted after it is validated against all policies with matching image patterns, and where at least one valid signature is obtained from the authorities provided in the matched [ClusterImagePolicy](#) later in the topic. Within a single policy, every signature must be valid. When more than one policy has a matching image pattern, the image must match at least one signature from each [ClusterImagePolicy](#).

## Including Namespaces

The Policy Controller only validates resources in namespaces that have chosen to opt-in. This is done by adding the label `policy.sigstore.dev/include: "true"` to the namespace resource.

```
kubectl label namespace my-secure-namespace policy.sigstore.dev/include=true
```



### Caution

Without a Policy Controller [ClusterImagePolicy](#) applied, there are fallback behaviors where images are validated against the public Sigstore Rekor and Fulcio servers by using a keyless authority flow. Therefore, if the deploying image is signed publicly by a third-party using the keyless authority flow, the image is admitted as it can validate against the public Rekor and Fulcio. To avoid this behavior, develop, and apply a [ClusterImagePolicy](#) that applies to the images being deployed in the namespace.

## Create a `ClusterImagePolicy` resource

The cluster image policy is a custom resource containing the following properties:

- `images`: The images block defines the patterns of images that must be subject to the `ClusterImagePolicy`. If multiple policies match a particular image, *ALL* of those policies must be satisfied for the image to be admitted.

Policy Controller by default defines if the following globs are specified:

- If `*` is specified, the `glob` matching behavior is `index.docker.io/library/*`.
- If `*/*` is specified, the `glob` matching behavior is `index.docker.io/*/*`. With these defaults, you require the `glob` pattern `**` to match against all images. If your image is hosted on Docker Hub, include `index.docker.io` as the host for the glob.
- `authorities`: The authorities block defines the rules for discovering and validating signatures. Discovery is done by using the `sources` text box, and is specified on any entry. Signatures are cryptographically verified using one of the `key` or `keyless` text boxes.

When a policy is selected to be evaluated against the matched image, the authorities are used to validate signatures. If at least one authority is satisfied and a signature is validated, the policy is validated.

### mode

In a `ClusterImagePolicy`, `spec.mode` specifies the action of a policy:

- `enforce`: The default behavior. If the policy fails to validate the image, the policy fails.
- `warn`: If the policy fails to validate the image, validation error messages are converted to Warnings and the policy passes.

A sample of a `ClusterImagePolicy` which has `warn` mode configured.

```
---
apiVersion: policy.sigstore.dev/v1beta1
kind: ClusterImagePolicy
metadata:
  name: POLICY-NAME
spec:
  mode: warn
```

Where `POLICY-NAME` is the name of the policy you want to configure your `ClusterImagePolicy` with.

When `enforce` mode rejects an image, the image is not admitted.

Sample output message:

```
error: failed to patch: admission webhook "policy.sigstore.dev" denied the request: va
lida
tion failed: failed policy: POLICY-NAME: spec.template.spec.containers[0].image
IMAGE-REFERENCE signature key validation failed for authority authority-0 for IMAGE-RE
FERENCE: GET IMAGE-SIGNATURE-REFERENCE: DENIED: denied; denied
failed policy: POLICY-NAME: spec.template.spec.containers[1].image
IMAGE-REFERENCE signature key validation failed for authority authority-0 for IMAGE-RE
FERENCE: GET IMAGE-SIGNATURE-REFERENCE: DENIED: denied; denied
```

When `warn` mode rejects an image, the image is admitted.

Sample output message:

```
Warning: failed policy: POLICY-NAME: spec.template.spec.containers[0].image
Warning: IMAGE-REFERENCE signature key validation failed for authority authority-0 for
IMAGE-REFERENCE: GET IMAGE-SIGNATURE-REFERENCE: DENIED: denied; denied
```

```
Warning: failed policy: POLICY-NAME: spec.template.spec.containers[1].image
Warning: IMAGE-REFERENCE signature key validation failed for authority authority-0 for
IMAGE-REFERENCE: GET IMAGE-SIGNATURE-REFERENCE: DENIED: denied; denied
```

If a namespace contains both signed and unsigned images, utilizing two ClusterImagePolicies can address this. You can configure one policy with `enforce` for images that are signed and configure the other policy with `warn` to allow expected unsigned images.

For example, allowing unsigned `tap-packages` images required for the platform through a `warn` policy. However, the signed images produced from Tanzu Build Service are verified with an `enforce` policy.

If `Warning` is undesirable, you might configure a `static.action pass` authority to allow expected unsigned images. For information about static action authorities, see the [Static Action](#) documentation.

## images

In a ClusterImagePolicy, `spec.images` specifies a list of glob matching patterns. These patterns are matched against the image digest in `PodSpec` for resources attempting deployment.

Policy Controller defines the following globs by default:

- If `*` is specified, the `glob` matching behavior is `index.docker.io/library/*`.
- If `*/*` is specified, the `glob` matching behavior is `index.docker.io/*/*`.

With these defaults, you require the `glob` pattern `**` to match against all images. If your image is hosted on Docker Hub, include `index.docker.io` as the host for the glob.

A sample of a ClusterImagePolicy which matches against all images using glob:

```
apiVersion: policy.sigstore.dev/v1beta1
kind: ClusterImagePolicy
metadata:
  name: image-policy
spec:
  images:
  - glob: "**"
```

## match

You can use `match` to filter resources using group, version, kind, or labels in a selected namespace to enforce the defined policy. If the list of matching resources is empty, all core resources are used by default.

For example, you can filter all `v1 cronjobs` with the label `app: tap` in a namespace that is labeled for policy enforcement:

```
spec:
  match:
  - group: batch
    resource: cronjobs
    version: v1
    selector:
      matchLabels:
        app: tap
```

## authorities

Authorities listed in the `authorities` block of the ClusterImagePolicy are `key` or `keyless` specifications.

**Note**

Keyless support is deactivated by default. For more information, see [Install Supply Chain Security Tools - Policy Controller](#).

Each `key` authority can contain a PEM-encoded ECDSA public key, a `secretRef`, or a `kms` path.

**Important**

Only ECDSA public keys are supported.

```
spec:
  authorities:
    - key:
      data: |
        -----BEGIN PUBLIC KEY-----
        ...
        -----END PUBLIC KEY-----
    - key:
      secretRef:
        name: secretName
    - key:
      kms: KMSPATH
```

Where `KMSPATH` is the name of the KMS path you want to configure in your key authority.

**Note**

The secret referenced in `key.secretRef.name` must be created in the `cosign-system` namespace or the namespace where the Policy Controller is installed. Such secret must only contain one `data` entry with the public key.

Each keyless authority can contain a Fulcio URL, a Rekor URL, a certificate, or an array of identities.

Identities are represented with a combination of `issuer` or `issuerRegExp` with `subject` or `subjectRegExp`.

- `issuer`: Defines the issuer for this identity.
- `issuerRegExp`: Specifies a regular expression to match the issuer for this identity.
- `subject`: Defines the subject for this identity.
- `subjectRegExp`: Specifies a regular expression to match the subject for this identity.

An example of keyless authority structure:

```
spec:
  authorities:
    - keyless:
      url: https://fulcio.example.com
      ca-cert:
        data: Certificate Data
      identities:
        - issuer: https://accounts.google.com
          subjectRegExp: .*@example.com
        - issuer: https://token.actions.githubusercontent.com
          subject: https://github.com/mycompany/*/.github/workflows/*@*
      ctlog:
```

```

url: https://rekor.example.com
- keyless:
url: https://fulcio.example.com
ca-cert:
secretRef:
name: secretName
identities:
- issuerRegExp: .*kubernetes.default.*
subjectRegExp: .*kubernetes.io/namespaces/default/serviceaccounts/default

```

The authorities are evaluated using the `any of` operator to admit container images. For each pod, the Policy Controller iterates over the list of containers and init containers. For every policy that matches against the images, they must each have at least one valid signature obtained using the authorities specified. If an image does not match any policy, the Policy Controller does not admit the image.

`static.action`

ClusterImagePolicy authorities are configured to always `pass` or `fail` with `static.action`.

Sample ClusterImagePolicy with static action `fail`.

```

apiVersion: policy.sigstore.dev/v1beta1
kind: ClusterImagePolicy
metadata:
name: POLICY-NAME
spec:
authorities:
- static:
action: fail

```

Where `POLICY-NAME` is the name of the policy you want to configure your ClusterImagePolicy with.

A sample output of static action `fail`:

```

error: failed to patch: admission webhook "policy.sigstore.dev" denied the request: validation failed: failed policy: POLICY-NAME: spec.template.spec.containers[0].image IMAGE-REFERENCE disallowed by static policy
failed policy: POLICY-NAME: spec.template.spec.containers[1].image IMAGE-REFERENCE disallowed by static policy

```

Images that are unsigned in a namespace with validation enabled are admitted with an authority with static action `pass`.

A scenario where this applies is configuring a policy with `static.action pass` for `tap-packages` images. Another policy is then configured to validate signed images produced by Tanzu Build Service. This allows images from `tap-packages`, which are unsigned and required by the platform, to be admitted while still validating signed built images from Tanzu Build Service. See [Configure your supply chain to sign and verify your image builds](#) for an example.

If `Warning` messages are desirable for admitted images where validation failed, you can configure a policy with `warn` mode and valid authorities. For information about ClusterImagePolicy modes, see the [Mode](#) documentation.

## Provide credentials for the package

There are three ways the package reads credentials to authenticate to registries protected by authentication:

1. Reading `imagePullSecrets` directly from the resource being admitted. See [Container image pull secrets](#) in the Kubernetes documentation.

2. Reading `imagePullSecrets` from the service account the resource is running as. See [Arranging for imagePullSecrets to be automatically attached](#) in the Kubernetes documentation.
3. Reading a `secretRef` from the `ClusterImagePolicy` resource's `signaturePullSecrets` when specifying the cosign signature source.

Authentication can fail for the following scenarios:

- A not valid credential is specified in the `imagePullSecrets` of the resource or in the service account the resource runs as.
- A not valid credential is specified in the `ClusterImagePolicy signaturePullSecrets` text box.

## Provide secrets for authentication in your policy

You can provide secrets for authentication as part of the policy configuration. The `oci` location is the image location or a remote location where signatures are configured to be stored during signing. The `signaturePullSecrets` is available in the `cosign-system` namespace or the namespace where the Policy Controller is installed.

By default, `imagePullSecrets` from the resource or service account is used while the default `oci` location is the image location.

See the following example:

```
spec:
  authorities:
    - key:
        data: |
          -----BEGIN PUBLIC KEY-----
          ...
          -----END PUBLIC KEY-----
        source:
          - oci: registry.example.com/project/signature-location
            signaturePullSecrets:
              - name: MY-SECRET
    - keyless:
        url: https://fulcio.example.com
        source:
          - oci: registry.example.com/project/signature-location
            signaturePullSecrets:
              - name: MY-SECRET
```

Where `MY-SECRET` is the name of the secret you want to use with your credentials.

VMware recommends using a set of credentials with the least amount of privilege that allows reading the signature stored in your registry.

## Verify your configuration

A sample policy:

```
apiVersion: policy.sigstore.dev/v1beta1
kind: ClusterImagePolicy
metadata:
  name: image-policy
spec:
  images:
    - glob: "gcr.io/projectsigstore/cosign*"
  authorities:
    - name: official-cosign-key
```

```
key:
  data: |
    -----BEGIN PUBLIC KEY-----
    MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEhyQCx0E9wQWSFI9ULGwy3BuRk1nt
    IqozONbbdbqz11h1Rjy9c7SG+hdcF19jE9uE/dwtuwU2MqU9T/cN0YkWww==
    -----END PUBLIC KEY-----
```

When using the sample policy, run these commands to verify your configuration:

1. Verify that the Policy Controller admits the signed image that validates with the configured public key. Run:

```
kubectl run cosign \
  --image=gcr.io/projectsigstore/cosign:v1.2.1 \
  --dry-run=server
```

For example:

```
$ kubectl run cosign \
  --image=gcr.io/projectsigstore/cosign:v1.2.1 \
  --dry-run=server
pod/cosign created (server dry run)
```

2. Verify that the Policy Controller rejects the unmatched image. Run:

```
kubectl run busybox --image=busybox --dry-run=server
```

For example:

```
$ kubectl run busybox --image=busybox --dry-run=server
Error from server (BadRequest): admission webhook "policy.sigstore.dev" denied the request: validation failed: no matching policies: spec.containers[0].image
index.docker.io/library/busybox@sha256:3614ca5eac0a3a1bcc361c939202a974b4902
b9334ff36eb29ffe9011aaad83
```

In the output, it did not specify which authorities were used as there was no policy found that matched the image. Therefore, the image fails to validate for a signature and fails to deploy.

3. Verify that the Policy Controller rejects a matched image signed with a different key than the one configured. Run:

```
kubectl run cosign-fail \
  --image=gcr.io/projectsigstore/cosign:v0.3.0 \
  --dry-run=server
```

For example:

```
$ kubectl run cosign-fail \
  --image=gcr.io/projectsigstore/cosign:v0.3.0 \
  --dry-run=server
Error from server (BadRequest): admission webhook "policy.sigstore.dev" denied the request: validation failed: failed policy: image-policy: spec.containers
[0].image
gcr.io/projectsigstore/cosign@sha256:135d8c5e27bdc917f04b415fc947d7d5b1137f99
bb8fa00bffc3ecal856e9c52 failed to validate public keys with authority official
-cosign-key for gcr.io/projectsigstore/cosign@sha256:135d8c5e27bdc917f04b415fc9
47d7d5b1137f99bb8fa00bffc3ecal856e9c52: no matching signatures:
```

In the output, it specifies which authorities were used for validation when a policy was found that matched the image. In this case, the authority used was `official-cosign-key`. If



no name is specified, it is defaulted to `authority-#`.

## Overview of Supply Chain Security Tools for VMware Tanzu - Sign

This component is removed in Tanzu Application Platform v1.4 in favor of [Supply Chain Security Tools - Policy Controller](#).

To migrate from Supply Chain Security Tools - Sign to Supply Chain Security Tools - Policy Controller, see [Migration From Supply Chain Security Tools - Sign](#)

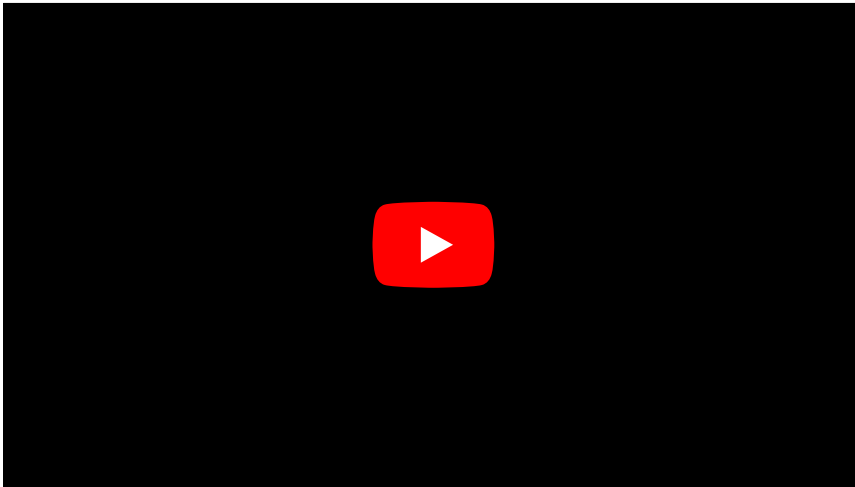
## Overview of Supply Chain Security Tools for Tanzu – Store

This topic gives you an overview of Supply Chain Security Tools (SCST) – Store.

### Overview

Supply Chain Security Tools - Store saves software bills of materials (SBOMs) to a database and allows you to query for image, source code, package, and vulnerability relationships. It integrates with [Supply Chain Security Tools - Scan](#) to automatically store the resulting source code and image vulnerability reports. It accepts CycloneDX input and outputs in both human-readable and machine-readable formats, including JSON, text, and CycloneDX.

The following is a quick demo of configuring the tanzu insight plug-in and querying the metadata store for CVEs and scan results.



### Using the Tanzu Insight CLI plug-in

the Tanzu Insight CLI plug-in is the primary way to view results from the Supply Chain Security Tools - Scan of source code and image files. Use it to query by source code commit, image digest, and CVE identifier to understand security risks.

See [Tanzu Insight plug-in overview](#) to install, configure, and use `tanzu insight`.

### Multicluster configuration

See [Multicluster setup](#) for information about how to set up SCST - Store in a multicluster setup.

### Integrating with Tanzu Application Platform GUI

Using the Supply Chain Choreographer in Tanzu Application Platform GUI, you can visualize your supply chain. It uses SCST - Store to show the packages and vulnerabilities in your source code and images.

To enable this feature, see [Supply Chain Choreographer in Tanzu Application Platform GUI - Enable CVE scan results](#).

## Additional documentation

[Additional documentation](#) includes information about the API, deployment details and configuration, AWS RDS configuration, other database backup recommendations, known issues, and other topics.

## Overview of Supply Chain Security Tools for Tanzu – Store

This topic gives you an overview of Supply Chain Security Tools (SCST) – Store.

### Overview

Supply Chain Security Tools - Store saves software bills of materials (SBOMs) to a database and allows you to query for image, source code, package, and vulnerability relationships. It integrates with [Supply Chain Security Tools - Scan](#) to automatically store the resulting source code and image vulnerability reports. It accepts CycloneDX input and outputs in both human-readable and machine-readable formats, including JSON, text, and CycloneDX.

The following is a quick demo of configuring the tanzu insight plug-in and querying the metadata store for CVEs and scan results.



### Using the Tanzu Insight CLI plug-in

the Tanzu Insight CLI plug-in is the primary way to view results from the Supply Chain Security Tools - Scan of source code and image files. Use it to query by source code commit, image digest, and CVE identifier to understand security risks.

See [Tanzu Insight plug-in overview](#) to install, configure, and use `tanzu insight`.

### Multicluster configuration

See [Multicluster setup](#) for information about how to set up SCST - Store in a multicluster setup.

## Integrating with Tanzu Application Platform GUI

Using the Supply Chain Choreographer in Tanzu Application Platform GUI, you can visualize your supply chain. It uses SCST - Store to show the packages and vulnerabilities in your source code and images.

To enable this feature, see [Supply Chain Choreographer in Tanzu Application Platform GUI - Enable CVE scan results](#).

### Additional documentation

[Additional documentation](#) includes information about the API, deployment details and configuration, AWS RDS configuration, other database backup recommendations, known issues, and other topics.

## Configure your target endpoint and certificate for Supply Chain Security Tools - Store

This topic describes how you can configure your target endpoint and certificate for Supply Chain Security Tools (SCST) - Store.

### Overview

The connection to Supply Chain Security Tools - Store requires TLS encryption, and the configuration depends on the kind of installation.

For a production environment, VMware recommends that SCST - Store is installed with ingress enabled. The following instructions help set up the TLS connection, assuming that you deployed with ingress enabled.

### Using Ingress

When using an [Ingress setup](#), SCST - Store creates a specific TLS Certificate for HTTPS communications under the `metadata-store` namespace.

Set the endpoint host to `metadata-store.INGRESS-DOMAIN`, such as `metadata-store.example.domain.com`. Where `INGRESS-DOMAIN` is the value of the `ingress_domain` property in your deployment yaml.

**Note** In a multi-cluster setup, a DNS record is **required** for the domain. The below instructions for single cluster setup do not apply, skip to Set Target section.

### Single Cluster setup

In a single-cluster setup, a DNS record is still recommended. However, if no accessible DNS record exists for the domain, edit the `/etc/hosts` file to add a local record:

```
ENVOY_IP=$(kubectl get svc envoy -n tanzu-system-ingress -o jsonpath="{.status.loadBalancer.ingress[0].ip}")

# Replace with your domain
METADATA_STORE_DOMAIN="metadata-store.example.domain.com"

# Delete any previously added entry
sudo sed -i ' ' "/$METADATA_STORE_DOMAIN/d" /etc/hosts

echo "$ENVOY_IP $METADATA_STORE_DOMAIN" | sudo tee -a /etc/hosts > /dev/null
```

## Set Target

To get the certificate, run:

```
kubectl get secret ingress-cert -n metadata-store -o json | jq -r '.data."ca.crt"' | base64 -d > insight-ca.crt
```

Set the target by running:

```
tanzu insight config set-target https://$METADATA_STORE_DOMAIN --ca-cert insight-ca.crt
```



### Important

The `tanzu insight config set-target` does not initiate a test connection. Use `tanzu insight health` to test connecting using the configured endpoint and CA certificate. Neither commands test whether the access token is correct. For that you must use the plug-in to [add data](#) and [query data](#).

## Next Step

- [Configure access token](#)

## Additional Resources

For information about deploying SCST - Store **without** Ingress, see:

- [Using LoadBalancer](#)
- [Using NodePort](#)

## Configure your access tokens for Supply Chain Security Tools - Store

This topic describes how to configure your access tokens for Supply Chain Security Tools - Store.

The access token is a `Bearer` token used in the http request header `Authorization`. For example, `Authorization: Bearer eyJhbGciOiJSUzI1NiIsImtpZCI6IjhmV0...`

Service accounts are required to have associated access tokens. Before Kubernetes 1.24, service accounts generated access tokens automatically. Since Kubernetes 1.24, a secret must be applied manually.

By default, Supply Chain Security Tools - Store includes a `read-write` service account installed with an access token generated. This service account is cluster-wide. If you want to create your own service accounts, see [Create Service Accounts](#).

## Setting the Access Token

When using the `insight` plug-in, you must set the `METADATA_STORE_ACCESS_TOKEN` environment variable, or use the `--access-token` flag. VMware discourages using the `--access-token` flag as the token appears in your shell history.

The following command retrieves the access token from the default `metadata-store-read-write-client` service account and stores it in `METADATA_STORE_ACCESS_TOKEN`:

```
export METADATA_STORE_ACCESS_TOKEN=$(kubectl get secrets metadata-store-read-write-client -n metadata-store -o jsonpath="{.data.token}" | base64 -d)
```

## Additional Resources

- [Retrieve access tokens](#)
- [Create service accounts](#)
- [Create a service account with a custom cluster role](#)

## Security details for Supply Chain Security Tools - Store

This topic describes the security details for Supply Chain Security Tools (SCST) - Store.

### Application security

#### TLS encryption

Supply Chain Security Tools - Store requires TLS connection. If certificates are not provided, the application does not start. It supports TLS v1.2 and TLS v1.3. It does not support TLS 1.0, so a downgrade attack cannot happen. TLS 1.0 is prohibited under Payment Card Industry Data Security Standard (PCI DSS).

#### Cryptographic algorithms

Elliptic Curve:

```
CurveP521
CurveP384
CurveP256
```

Cipher Suites:

```
TLS_AES_128_GCM_SHA256
TLS_AES_256_GCM_SHA384
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
```

### Access controls

SCST - Store uses [kube-rbac-proxy](#) as the only entry point to its API. Authentication and Authorization must be completed by using the [kube-rbac-proxy](#) before its API is accessible.

#### Authentication

The [kube-rbac-proxy](#) uses [Token Review](#) to verify that the token is valid. [Token Review](#) is a Kubernetes API to ensure that a trusted vendor issued the access token provided by the user. To issue an access token using Kubernetes, the user can create a Kubernetes Service Account and retrieve the corresponding generated secret for the access token.

To create a service account and use its access token, see the [Create Service Account Docs](#).

#### Authorization

The `kube-rbac-proxy` uses [Subject Access Review](#) to ensure that users access certain operations. [Subject Access Review](#) is a Kubernetes API that uses [Kubernetes RBAC](#) to verify that the user can perform specific actions. See [Create Service Account Doc](#).

There are two supported roles:

- `Read Only` cluster role
- `Read and Write` cluster role

These cluster roles are deployed by default. Additionally, a service account is created and bound to the `Read and Write` cluster role by default. If you do not want this service account, set the `add_default_rw_service_account` property to `false` in the `metadata-store-values.yaml` file during deployment. See [Install SCST - Store](#).

There is no default service account bound to the `Read Only` cluster role. You must create your service account and cluster role binding to bind to the `Read Only` role.



#### Important

There is no support for roles with access to only specific types of resources. For example, images, packages, and vulnerabilities.

## Container security

### Non-root user

All containers shipped do not use root user accounts or accounts with root access. Using Kubernetes Security Context ensures that applications do not run with root users.

Security Context for the API server:

```
allowPrivilegeEscalation: false
runAsUser: 65532
fsGroup: 65532
```

Security Context for the PostgreSQL database pod:

```
allowPrivilegeEscalation: false
runAsUser: 999
fsGroup: 999
```



#### Note

`65532` is the UUID for the nobody user. `999` is the UUID for the PostgreSQL user.

## Security scanning

There are two types of security scans that are performed before every release.

### Static Application Security Testing (SAST)

A Coverity Scan is run on the source code of the API server, CLI, and all their dependencies. There are no high or critical items outstanding at the time of release.

### Software Composition Analysis (SCA)

A Black Duck scan is run on the compiled binary to check for vulnerabilities and license data. There are no high or critical items outstanding at the time of release.

A Gype scan is run against the source code and the compiled container for dependencies vulnerabilities. There are no high or critical items outstanding at the time of release.

## Additional documentation for Supply Chain Security Tools - Store

This topic describes additional documentation you can use with Supply Chain Security Tools - Store.

### Use and operate

- [Multicluster setup](#)
- [Developer namespace setup](#)
- [API details](#)
- [API walkthrough](#)
- [Failover, redundancy, and backups](#)

### Troubleshooting and logging

- [Troubleshooting upgrading](#)
- [Log configuration and usage](#)
- [Connecting to the Postgres Database](#)

### Configuration

- [Deployment details and configuration](#)

### Access control

- [Retrieve access tokens](#)
- [Create service accounts](#)
- [Create a service account with a custom cluster role](#)

### Certificates

- [Ingress support](#)
- [Using LoadBalancer](#)
- [Using NodePort](#)
- [Custom certificate configuration](#)
- [TLS configuration](#)
- [Certificate rotation](#)

### Database

- [Use external postgres database](#)
- [AWS RDS postgres configuration](#)
- [Database backup recommendations](#)

## Other

- [Install SCST - Store independent from TAP profiles](#)

## Additional documentation for Supply Chain Security Tools - Store

This topic describes additional documentation you can use with Supply Chain Security Tools - Store.

### Use and operate

- [Multicluster setup](#)
- [Developer namespace setup](#)
- [API details](#)
- [API walkthrough](#)
- [Failover, redundancy, and backups](#)

### Troubleshooting and logging

- [Troubleshooting upgrading](#)
- [Log configuration and usage](#)
- [Connecting to the Postgres Database](#)

### Configuration

- [Deployment details and configuration](#)

### Access control

- [Retrieve access tokens](#)
- [Create service accounts](#)
- [Create a service account with a custom cluster role](#)

### Certificates

- [Ingress support](#)
- [Using LoadBalancer](#)
- [Using NodePort](#)
- [Custom certificate configuration](#)
- [TLS configuration](#)
- [Certificate rotation](#)

### Database

- [Use external postgres database](#)
- [AWS RDS postgres configuration](#)
- [Database backup recommendations](#)



## Other

- [Install SCST - Store independent from TAP profiles](#)

## API reference for Supply Chain Security Tools - Store

This topic contains API reference information for Supply Chain Security Tools - Store. See [API walkthrough](#) for an SCST - Store example.

## Information

### Version

1.4.1

## Content negotiation

### URI Schemes

- http
- https

### Consumes

- application/json

### Produces

- application/json

## All endpoints

### images

| Method | URI                                 | Name                                     | Summary                                                                           |
|--------|-------------------------------------|------------------------------------------|-----------------------------------------------------------------------------------|
| POST   | /api/imageReport                    | <a href="#">create image report</a>      | Create a new image report. Related packages and vulnerabilities are also created. |
| GET    | /api/images                         | <a href="#">get images</a>               | Search image by id, name or digest .                                              |
| GET    | /api/packages/{IDorName}/images     | <a href="#">get package images</a>       | List the images that contain the given package.                                   |
| GET    | /api/vulnerabilities/{CVEID}/images | <a href="#">get vulnerability images</a> | List the images that contain the given vulnerability.                             |

## Operations

| Method | URI         | Name                         | Summary |
|--------|-------------|------------------------------|---------|
| GET    | /api/health | <a href="#">health check</a> |         |

## Packages

| Method | URI                                   | Name                                       | Summary                                      |
|--------|---------------------------------------|--------------------------------------------|----------------------------------------------|
| GET    | /api/images/{IDorDigest}/packages     | <a href="#">get image packages</a>         | List the packages in an image.               |
| GET    | /api/images/packages                  | <a href="#">get image packages query</a>   | List packages of the given image.            |
| GET    | /api/packages                         | <a href="#">get packages</a>               | Search packages by id, name and/or version.  |
| GET    | /api/sources/{IDorRepoorSha}/packages | <a href="#">get source packages</a>        |                                              |
| GET    | /api/sources/packages                 | <a href="#">get source packages query</a>  | List packages of the given source.           |
| GET    | /api/vulnerabilities/{CVEID}/packages | <a href="#">get vulnerability packages</a> | List packages that contain the given CVE id. |

## Sources

| Method | URI                                  | Name                                      | Summary                                                                            |
|--------|--------------------------------------|-------------------------------------------|------------------------------------------------------------------------------------|
| POST   | /api/sourceReport                    | <a href="#">create source report</a>      | Create a new source report. Related packages and vulnerabilities are also created. |
| GET    | /api/packages/{IDorName}/sources     | <a href="#">get package sources</a>       | List the sources containing the given package.                                     |
| GET    | /api/sources                         | <a href="#">get sources</a>               | Search for sources by ID, repository, commit sha and/or organization.              |
| GET    | /api/vulnerabilities/{CVEID}/sources | <a href="#">get vulnerability sources</a> | List sources that contain the given vulnerability.                                 |

## v1artifact\_groups

| Method | URI                                             | Name                                                   | Summary                                                                                                                                                                                                                                                                                                       |
|--------|-------------------------------------------------|--------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| POST   | /api/v1/artifact-groups                         | <a href="#">create artifact group</a>                  | Create an artifact group with specified labels and entity                                                                                                                                                                                                                                                     |
| POST   | /api/v1/artifact-groups/_search                 | <a href="#">search artifact groups</a>                 | Query for a list of artifact group that contains image(s) with specified digests, and or source(s) with specified shas. At least one image digest or source sha must be provided. This query can be further refined by matching images and sources with a specific combination of package name and/or cve id. |
| POST   | /api/v1/artifact-groups/vulnerabilities/_reach  | <a href="#">search artifact groups vuln reach</a>      | Search for how many artifact groups are affected by vulnerabilities associated with the specified image(s) digests, and/or source(s) shas. At least one image digest or source sha must be provided.                                                                                                          |
| POST   | /api/v1/artifact-groups/vulnerabilities/_search | <a href="#">search artifact groups vulnerabilities</a> | Search for all vulnerabilities associated with an artifact group that contains image(s) with specified digests, and/or source(s) with specified shas. At least one image digest or source sha must be provided.                                                                                               |

## v1images

| Method | URI                 | Name                            | Summary                                                                             |
|--------|---------------------|---------------------------------|-------------------------------------------------------------------------------------|
| GET    | /api/v1/images/{ID} | <a href="#">get image by ID</a> | Search image by ID                                                                  |
| GET    | /api/v1/images      | <a href="#">v1 get images</a>   | Query for images. If no parameters are given, this endpoint will return all images. |

## v1packages

| Method | URI                      | Name                                    | Summary                                                                                                                           |
|--------|--------------------------|-----------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| GET    | /api/v1/packages/{ID}    | <a href="#">get package by ID</a>       | Search package by ID                                                                                                              |
| GET    | /api/v1/images/packages  | <a href="#">v1 get images packages</a>  | Query for packages with images parameters. If no parameters are given, this endpoint will return all packages related to images.  |
| GET    | /api/v1/packages         | <a href="#">v1 get packages</a>         | Query for packages. If no parameters are given, this endpoint will return all packages.                                           |
| GET    | /api/v1/sources/packages | <a href="#">v1 get sources packages</a> | Query for packages with source parameters. If no parameters are given, this endpoint will return all packages related to sources. |

## v1sources

| Method | URI                             | Name                                           | Summary                                                                                                                      |
|--------|---------------------------------|------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------|
| GET    | /api/v1/sources/{ID}            | <a href="#">get source by ID</a>               | Search source by ID                                                                                                          |
| GET    | /api/v1/sources                 | <a href="#">v1 get sources</a>                 | Query for sources. If no parameters are given, this endpoint will return all sources.                                        |
| GET    | /api/v1/sources/vulnerabilities | <a href="#">v1 get sources vulnerabilities</a> | Query for vulnerabilities with source parameters. If no parameters are given, this endpoint will return all vulnerabilities. |

## v1vulnerabilities

| Method | URI                            | Name                                          | Summary                                                                                                                    |
|--------|--------------------------------|-----------------------------------------------|----------------------------------------------------------------------------------------------------------------------------|
| GET    | /api/v1/vulnerabilities/{ID}   | <a href="#">get vulnerability by ID</a>       | Search vulnerability by ID                                                                                                 |
| GET    | /api/v1/images/vulnerabilities | <a href="#">v1 get images vulnerabilities</a> | Query for vulnerabilities with image parameters. If no parameters are give, this endpoint will return all vulnerabilities. |

## vulnerabilities

| Method | URI                                          | Name                                             | Summary                                      |
|--------|----------------------------------------------|--------------------------------------------------|----------------------------------------------|
| GET    | /api/images/{IDorDigest}/vulnerabilities     | <a href="#">get image vulnerabilities</a>        | List vulnerabilities from the given image.   |
| GET    | /api/packages/{IDorName}/vulnerabilities     | <a href="#">get package vulnerabilities</a>      | List vulnerabilities from the given package. |
| GET    | /api/sources/{IDorRepoorSha}/vulnerabilities | <a href="#">get source vulnerabilities</a>       |                                              |
| GET    | /api/sources/vulnerabilities                 | <a href="#">get source vulnerabilities query</a> | List vulnerabilities of the given source.    |
| GET    | /api/vulnerabilities                         | <a href="#">get vulnerabilities</a>              | Search for vulnerabilities by CVE id.        |

## Paths

### Create an artifact group with specified labels and entity (*CreateArtifactGroup*)

```
POST /api/v1/artifact-groups
```

#### Parameters

| Name                     | Source | Type                     | Go type                         | Separator | Required | Default | Description |
|--------------------------|--------|--------------------------|---------------------------------|-----------|----------|---------|-------------|
| ArtifactGroupPostRequest | body   | ArtifactGroupPostRequest | models.ArtifactGroupPostRequest |           | ✓        |         |             |

#### All responses

| Code    | Status      | Description               | Has headers | Schema                 |
|---------|-------------|---------------------------|-------------|------------------------|
| 201     | Created     | ArtifactGroupPostResponse |             | <a href="#">schema</a> |
| 400     | Bad Request | ErrorMessage              |             | <a href="#">schema</a> |
| default |             | ErrorMessage              |             | <a href="#">schema</a> |

#### Responses

##### 201 - ArtifactGroupPostResponse

Status: Created

Schema

[ArtifactGroupPostResponse](#)

##### 400 - ErrorMessage

Status: Bad Request

Schema

[ErrorMessage](#)

Default Response

ErrorMessage

Schema

[ErrorMessage](#)

### 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        |             | <a href="#">schema</a> |
| default |        | ErrorMessage |             | <a href="#">schema</a> |

### 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       |             | <a href="#">schema</a> |
| default |        | ErrorMessage |             | <a href="#">schema</a> |

### Responses

200 - Source

Status: OK

Schema

[Source](#)

Default Response

ErrorMessage

Schema

[ErrorMessage](#)

## Search image by ID (*GetImageByID*)

```
GET /api/v1/images/{ID}
```

### Parameters

| Name | Source               | Type                       | Go type                | Separator | Required | Default | Description |
|------|----------------------|----------------------------|------------------------|-----------|----------|---------|-------------|
| ID   | <a href="#">path</a> | uint64 (formatted integer) | <a href="#">uint64</a> |           | ✓        |         |             |

### All responses

| Code                    | Status    | Description  | Has headers | Schema                 |
|-------------------------|-----------|--------------|-------------|------------------------|
| 200                     | OK        | Image        |             | <a href="#">schema</a> |
| 404                     | Not Found | ErrorMessage |             | <a href="#">schema</a> |
| <a href="#">default</a> |           | ErrorMessage |             | <a href="#">schema</a> |

### Responses

#### 200 - Image

Status: OK

Schema

[Image](#)

#### 404 - ErrorMessage

Status: Not Found

Schema

[ErrorMessage](#)

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](#)

Default Response

[ErrorMessage](#)

Schema

[ErrorMessage](#)

## List packages of the given image. (*GetImagePackagesQuery*)

```
GET /api/images/packages
```

### Parameters

| Name   | Source | Type                      | Go type | Separator | Required | Default | Description |
|--------|--------|---------------------------|---------|-----------|----------|---------|-------------|
| digest | query  | string                    | string  |           |          |         |             |
| id     | query  | int64 (formatted integer) | int64   |           |          |         |             |
| name   | query  | string                    | string  |           |          |         |             |

### All responses

| Code    | Status | Description  | Has headers | Schema                 |
|---------|--------|--------------|-------------|------------------------|
| 200     | OK     | Package      |             | <a href="#">schema</a> |
| default |        | ErrorMessage |             | <a href="#">schema</a> |

### Responses

200 - Package

Status: OK

Schema

[\[\]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  |           | ✓        |         |                                                                                                              |
| Severity   | query  | string | string  |           |          |         | Case insensitive vulnerabilities severity filter. Possible values are: low, medium, high, critical, unknown. |

### All responses

| Code    | Status | Description   | Has headers | Schema                 |
|---------|--------|---------------|-------------|------------------------|
| 200     | OK     | Vulnerability |             | <a href="#">schema</a> |
| default |        | ErrorMessage  |             | <a href="#">schema</a> |

### Responses

200 - Vulnerability

Status: OK

Schema



[Vulnerability](#)

Default Response

ErrorMessage

Schema

[ErrorMessage](#)

### Search image by id, name or digest . (*GetImages*)

```
GET /api/images
```

#### All responses

| Code    | Status | Description  | Has headers | Schema                 |
|---------|--------|--------------|-------------|------------------------|
| 200     | OK     | Image        |             | <a href="#">schema</a> |
| default |        | ErrorMessage |             | <a href="#">schema</a> |

#### Responses

200 - Image

Status: OK

Schema

[Image](#)

Default Response

ErrorMessage

Schema

[ErrorMessage](#)

### Search package by ID (*GetPackageByID*)

```
GET /api/v1/packages/{ID}
```

#### Parameters

| Name | Source               | Type                       | Go type                | Separator | Required | Default | Description |
|------|----------------------|----------------------------|------------------------|-----------|----------|---------|-------------|
| ID   | <a href="#">path</a> | uint64 (formatted integer) | <a href="#">uint64</a> |           | ✓        |         |             |

#### All responses

| Code    | Status    | Description  | Has headers | Schema                 |
|---------|-----------|--------------|-------------|------------------------|
| 200     | OK        | Package      |             | <a href="#">schema</a> |
| 404     | Not Found | ErrorMessage |             | <a href="#">schema</a> |
| default |           | ErrorMessage |             | <a href="#">schema</a> |

## Responses

### 200 - Package

Status: OK

Schema

[Package](#)

### 404 - ErrorMessage

Status: Not Found

Schema

[ErrorMessage](#)

Default Response

ErrorMessage

Schema

[ErrorMessage](#)

## List the images that contain the given package. (*GetPackageImages*)

```
GET /api/packages/{IDorName}/images
```

### Parameters

| Name     | Source | Type   | Go type | Separator | Required | Default | Description |
|----------|--------|--------|---------|-----------|----------|---------|-------------|
| IDorName | path   | string | string  |           | ✓        |         |             |

### All responses

| Code    | Status | Description  | Has headers | Schema                 |
|---------|--------|--------------|-------------|------------------------|
| 200     | OK     | Image        |             | <a href="#">schema</a> |
| default |        | ErrorMessage |             | <a href="#">schema</a> |

## Responses

### 200 - Image

Status: OK

Schema

[\[\]Image](#)

Default Response

ErrorMessage

Schema

[ErrorMessage](#)

### List the sources containing the given package. (*GetPackageSources*)

```
GET /api/packages/{IDorName}/sources
```

#### Parameters

| Name     | Source | Type   | Go type | Separator | Required | Default | Description |
|----------|--------|--------|---------|-----------|----------|---------|-------------|
| IDorName | path   | string | string  |           | ✓        |         |             |

#### All responses

| Code    | Status | Description  | Has headers | Schema |
|---------|--------|--------------|-------------|--------|
| 200     | OK     | Source       |             | schema |
| default |        | ErrorMessage |             | schema |

#### Responses

200 - Source

Status: OK

Schema

[\[\]Source](#)

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  |           | ✓        |         |                                                                                                              |
| Severity | query  | string | string  |           |          |         | Case insensitive vulnerabilities severity filter. Possible values are: low, medium, high, critical, unknown. |

### All responses

| Code    | Status | Description   | Has headers | Schema |
|---------|--------|---------------|-------------|--------|
| 200     | OK     | Vulnerability |             | schema |
| default |        | ErrorMessage  |             | schema |

### Responses

200 - Vulnerability

Status: OK

Schema

[\[\]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      |             | <a href="#">schema</a> |
| default |        | ErrorMessage |             | <a href="#">schema</a> |

### Responses

200 - Package

Status: OK

Schema

[\[\]Package](#)

Default Response

ErrorMessage

Schema

[ErrorMessage](#)

## Search source by ID (*GetSourceByID*)

```
GET /api/v1/sources/{ID}
```

### Parameters

| Name | Source               | Type                       | Go type                | Separator | Required | Default | Description |
|------|----------------------|----------------------------|------------------------|-----------|----------|---------|-------------|
| ID   | <a href="#">path</a> | uint64 (formatted integer) | <a href="#">uint64</a> |           | ✓        |         |             |

### All responses

| Code    | Status    | Description  | Has headers | Schema                 |
|---------|-----------|--------------|-------------|------------------------|
| 200     | OK        | Source       |             | <a href="#">schema</a> |
| 404     | Not Found | ErrorMessage |             | <a href="#">schema</a> |
| default |           | ErrorMessage |             | <a href="#">schema</a> |

### Responses

200 - Source

Status: OK

Schema

[Source](#)

404 - ErrorMessage

Status: Not Found

Schema

[ErrorMessage](#)

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 | <code>path</code> | string | <code>string</code> |           | ✓        |         |             |

### All responses

| Code    | Status | Description  | Has headers | Schema                 |
|---------|--------|--------------|-------------|------------------------|
| 200     | OK     | Package      |             | <a href="#">schema</a> |
| default |        | ErrorMessage |             | <a href="#">schema</a> |

### Responses

200 - Package

Status: OK

Schema

[\[\]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](#)

Default Response

ErrorMessage

Schema

[ErrorMessage](#)

## get source vulnerabilities (*GetSourceVulnerabilities*)

```
GET /api/sources/{IDorReporSha}/vulnerabilities
```

### Parameters

| Name         | Source | Type   | Go type | Separator | Required | Default | Description |
|--------------|--------|--------|---------|-----------|----------|---------|-------------|
| IDorReporSha | 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](#)

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                                                                                                  |
|----------|--------|----------------------------|---------|-----------|----------|---------|--------------------------------------------------------------------------------------------------------------|
| Severity | query  | string                     | string  |           |          |         | Case insensitive vulnerabilities severity filter. Possible values are: low, medium, high, critical, unknown. |
| 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](#)



Default Response

ErrorMessage

Schema

[ErrorMessage](#)

## Search for sources by ID, repository, commit sha and/or organization. (*GetSources*)

```
GET /api/sources
```

### Parameters

| Name | Source                | Type                      | Go type             | Separator | Required | Default | Description |
|------|-----------------------|---------------------------|---------------------|-----------|----------|---------|-------------|
| id   | <a href="#">query</a> | int64 (formatted integer) | <code>int64</code>  |           |          |         |             |
| org  | <a href="#">query</a> | string                    | <code>string</code> |           |          |         |             |
| repo | <a href="#">query</a> | string                    | <code>string</code> |           |          |         |             |
| sha  | <a href="#">query</a> | string                    | <code>string</code> |           |          |         |             |

### All responses

| Code    | Status | Description  | Has headers | Schema                 |
|---------|--------|--------------|-------------|------------------------|
| 200     | OK     | Source       |             | <a href="#">schema</a> |
| default |        | ErrorMessage |             | <a href="#">schema</a> |

### Responses

200 - Source

Status: OK

Schema

[\[\]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  |           | ✓        |         |                                                                                                              |
| Severity | query  | string | string  |           |          |         | Case insensitive vulnerabilities severity filter. Possible values are: low, medium, high, critical, unknown. |

### All responses

| Code    | Status | Description   | Has headers | Schema |
|---------|--------|---------------|-------------|--------|
| 200     | OK     | Vulnerability |             | schema |
| default |        | ErrorMessage  |             | schema |

### Responses

200 - Vulnerability

Status: OK

Schema

[\[\]Vulnerability](#)

Default Response

ErrorMessage

Schema

[ErrorMessage](#)

### Search vulnerability by ID (*GetVulnerabilityByID*)

```
GET /api/v1/vulnerabilities/{ID}
```

### Parameters

| Name | Source | Type                       | Go type | Separator | Required | Default | Description |
|------|--------|----------------------------|---------|-----------|----------|---------|-------------|
| ID   | path   | uint64 (formatted integer) | uint64  |           | ✓        |         |             |

### All responses

| Code    | Status    | Description   | Has headers | Schema |
|---------|-----------|---------------|-------------|--------|
| 200     | OK        | Vulnerability |             | schema |
| 404     | Not Found | ErrorMessage  |             | schema |
| default |           | ErrorMessage  |             | schema |

## Responses

200 - Vulnerability

Status: OK

Schema

[Vulnerability](#)

404 - ErrorMessage

Status: Not Found

Schema

[ErrorMessage](#)

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        |             | <a href="#">schema</a> |
| default |        | ErrorMessage |             | <a href="#">schema</a> |

## Responses

200 - Image

Status: OK

Schema

[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 | <a href="#">path</a> | string | <a href="#">string</a> |           | ✓        |         |             |

### All responses

| Code                    | Status | Description  | Has headers | Schema                 |
|-------------------------|--------|--------------|-------------|------------------------|
| 200                     | OK     | Package      |             | <a href="#">schema</a> |
| <a href="#">default</a> |        | ErrorMessage |             | <a href="#">schema</a> |

### Responses

200 - Package

Status: OK

Schema

[\[\]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 | <a href="#">path</a> | string | <a href="#">string</a> |           | ✓        |         |             |

All responses

| Code                    | Status | Description  | Has headers | Schema                 |
|-------------------------|--------|--------------|-------------|------------------------|
| 200                     | OK     | Source       |             | <a href="#">schema</a> |
| <a href="#">default</a> |        | ErrorMessage |             | <a href="#">schema</a> |

Responses

200 - Source

Status: OK

Schema

[\[\]Source](#)

Default Response

ErrorMessage

Schema

[ErrorMessage](#)

health check (*HealthCheck*)

```
GET /api/health
```

All responses

| Code                    | Status | Description  | Has headers | Schema                 |
|-------------------------|--------|--------------|-------------|------------------------|
| 200                     | OK     |              |             | <a href="#">schema</a> |
| <a href="#">default</a> |        | ErrorMessage |             | <a href="#">schema</a> |

Responses

200

Status: OK

Schema

Default Response

ErrorMessage

Schema

[ErrorMessage](#)

Query for a list of artifact group that contains image(s) with specified digests, and or source(s) with specified shas. At least one image digest or source sha must be provided. This query can be further refined by matching images and sources with a specific combination of package name and/or cve id. (*SearchArtifactGroups*)

```
POST /api/v1/artifact-groups/_search
```

Query for a list of artifact group that contains image(s) with specified digests, and or source(s) with specified shas.

Parameters

| Name                                | Source | Type                           | Go type                               | Separator | Required | Default | Description |
|-------------------------------------|--------|--------------------------------|---------------------------------------|-----------|----------|---------|-------------|
| ArtifactGroupFilter<br>sPostRequest | body   | ArtifactGroupS<br>earchFilters | models.ArtifactGro<br>upSearchFilters |           | ✓        |         |             |

All responses

| Code    | Status      | Description                    | Has headers | Schema |
|---------|-------------|--------------------------------|-------------|--------|
| 200     | OK          | PaginatedArtifactGroupResponse |             | schema |
| 400     | Bad Request | ErrorMessage                   |             | schema |
| default |             | ErrorMessage                   |             | schema |

Responses

200 - PaginatedArtifactGroupResponse

Status: OK

Schema

[PaginatedArtifactGroupResponse](#)

400 - ErrorMessage

Status: Bad Request

Schema

[ErrorMessage](#)

Default Response

ErrorMessage

Schema

[ErrorMessage](#)

Search for how many artifact groups are affected by vulnerabilities associated with the specified image(s) digests, and/or source(s) shas. At least one image digest or source sha must be provided. (*SearchArtifactGroupsVulnReach*)

```
POST /api/v1/artifact-groups/vulnerabilities/_reach
```

Parameters

| Name                                         | Source | Type                                         | Go type                                             | Separator | Required | Default | Description |
|----------------------------------------------|--------|----------------------------------------------|-----------------------------------------------------|-----------|----------|---------|-------------|
| ArtifactGroupVulnReachFiltersPositionRequest | body   | ArtifactGroupVulnReachFiltersPositionRequest | models.ArtifactGroupVulnReachFiltersPositionRequest |           | ✓        |         |             |

All responses

| Code    | Status      | Description                             | Has headers | Schema                 |
|---------|-------------|-----------------------------------------|-------------|------------------------|
| 200     | OK          | PaginatedArtifactGroupVulnReachResponse |             | <a href="#">schema</a> |
| 400     | Bad Request | ErrorMessage                            |             | <a href="#">schema</a> |
| default |             | ErrorMessage                            |             | <a href="#">schema</a> |

Responses

200 - PaginatedArtifactGroupVulnReachResponse

Status: OK

Schema

[PaginatedArtifactGroupVulnReachResponse](#)

400 - ErrorMessage

Status: Bad Request

Schema

[ErrorMessage](#)

Default Response

ErrorMessage

Schema

[ErrorMessage](#)

Search for all vulnerabilities associated with an artifact group that contains image(s) with specified digests, and/or source(s) with

specified shas. At least one image digest or source sha must be provided. (*SearchArtifactGroupsVulnerabilities*)

```
POST /api/v1/artifact-groups/vulnerabilities/_search
```

The result can be further refined by matching the images and sources with a package name and/or an artifact group UID

### Parameters

| Name                                               | Source | Type                                    | Go type                                        | Separator | Required | Default | Description |
|----------------------------------------------------|--------|-----------------------------------------|------------------------------------------------|-----------|----------|---------|-------------|
| ArtifactGroupVulnerabilitySearchFiltersPostRequest | body   | ArtifactGroupVulnerabilitySearchFilters | models.ArtifactGroupVulnerabilitySearchFilters |           | ✓        |         |             |

### All responses

| Code    | Status      | Description                                 | Has headers | Schema                 |
|---------|-------------|---------------------------------------------|-------------|------------------------|
| 200     | OK          | PaginatedArtifactGroupVulnerabilityResponse |             | <a href="#">schema</a> |
| 400     | Bad Request | ErrorMessage                                |             | <a href="#">schema</a> |
| default |             | ErrorMessage                                |             | <a href="#">schema</a> |

### Responses

200 - PaginatedArtifactGroupVulnerabilityResponse

Status: OK

Schema

[PaginatedArtifactGroupVulnerabilityResponse](#)

400 - ErrorMessage

Status: Bad Request

Schema

[ErrorMessage](#)

Default Response

ErrorMessage

Schema

[ErrorMessage](#)

Query for images. If no parameters are given, this endpoint will return all images. (*V1GetImages*)



GET /api/v1/images

## Parameters

| Name      | Source | Type                         | Go type | Separator | Required | Default | Description                                                                                   |
|-----------|--------|------------------------------|---------|-----------|----------|---------|-----------------------------------------------------------------------------------------------|
| all       | query  | boolean                      | bool    |           |          |         | If no pagination parameters are provided, defaults to true and returns all available results. |
| digest    | query  | string                       | string  |           |          |         |                                                                                               |
| name      | query  | string                       | string  |           |          |         |                                                                                               |
| page      | query  | int64<br>(formatted integer) | int64   |           |          | 1       |                                                                                               |
| page_size | query  | int64<br>(formatted integer) | int64   |           |          | 20      |                                                                                               |
| registry  | query  | string                       | string  |           |          |         |                                                                                               |

## All responses

| Code    | Status    | Description            | Has headers | Schema                 |
|---------|-----------|------------------------|-------------|------------------------|
| 200     | OK        | PaginatedImageResponse |             | <a href="#">schema</a> |
| 404     | Not Found | ErrorMessage           |             | <a href="#">schema</a> |
| default |           | ErrorMessage           |             | <a href="#">schema</a> |

## Responses

### 200 - PaginatedImageResponse

Status: OK

Schema

### [PaginatedImageResponse](#)

### 404 - ErrorMessage

Status: Not Found

Schema

### [ErrorMessage](#)

### Default Response

ErrorMessage

Schema

ErrorMessage

Query for packages with images parameters. If no parameters are given, this endpoint will return all packages related to images. (V1GetImagesPackages)

```
GET /api/v1/images/packages
```

Parameters

| Name         | Source | Type                      | Go type | Separator | Required | Default | Description                                                                                   |
|--------------|--------|---------------------------|---------|-----------|----------|---------|-----------------------------------------------------------------------------------------------|
| all          | query  | boolean                   | bool    |           |          |         | If no pagination parameters are provided, defaults to true and returns all available results. |
| digest       | query  | string                    | string  |           |          |         |                                                                                               |
| name         | query  | string                    | string  |           |          |         |                                                                                               |
| package_name | query  | string                    | string  |           |          |         | Substring package name filter. For example, setting name=cur would match curl and libcurl.    |
| page         | query  | int64 (formatted integer) | int64   |           |          | 1       |                                                                                               |
| page_size    | query  | int64 (formatted integer) | int64   |           |          | 20      |                                                                                               |
| registry     | query  | string                    | string  |           |          |         |                                                                                               |

All responses

| Code    | Status    | Description              | Has headers | Schema |
|---------|-----------|--------------------------|-------------|--------|
| 200     | OK        | PaginatedPackageResponse |             | schema |
| 404     | Not Found | ErrorMessage             |             | schema |
| default |           | ErrorMessage             |             | schema |

Responses

200 - PaginatedPackageResponse

Status: OK

Schema

PaginatedPackageResponse

404 - ErrorMessage

Status: Not Found

Schema

[ErrorMessage](#)

Default Response

ErrorMessage

Schema

[ErrorMessage](#)

Query for vulnerabilities with image parameters. If no parameters are give, this endpoint will return all vulnerabilities. *(V1GetImagesVulnerabilities)*

```
GET /api/v1/images/vulnerabilities
```

Parameters

| Name      | Source | Type                      | Go type | Separator | Required | Default | Description                                                                                                  |
|-----------|--------|---------------------------|---------|-----------|----------|---------|--------------------------------------------------------------------------------------------------------------|
| Severity  | query  | string                    | string  |           |          |         | Case insensitive vulnerabilities severity filter. Possible values are: low, medium, high, critical, unknown. |
| all       | query  | boolean                   | bool    |           |          |         | If no pagination parameters are provided, defaults to true and returns all available results.                |
| digest    | query  | string                    | string  |           |          |         |                                                                                                              |
| name      | query  | string                    | string  |           |          |         |                                                                                                              |
| page      | query  | int64 (formatted integer) | int64   |           |          | 1       |                                                                                                              |
| page_size | query  | int64 (formatted integer) | int64   |           |          | 20      |                                                                                                              |
| registry  | query  | string                    | string  |           |          |         |                                                                                                              |

All responses

| Code | Status    | Description                    | Has headers | Schema                 |
|------|-----------|--------------------------------|-------------|------------------------|
| 200  | OK        | PaginatedVulnerabilityResponse |             | <a href="#">schema</a> |
| 404  | Not Found | ErrorMessage                   |             | <a href="#">schema</a> |

| Code                    | Status | Description  | Has headers | Schema                 |
|-------------------------|--------|--------------|-------------|------------------------|
| <a href="#">default</a> |        | ErrorMessage |             | <a href="#">schema</a> |

## Responses

### 200 - PaginatedVulnerabilityResponse

Status: OK

Schema

#### [PaginatedVulnerabilityResponse](#)

### 404 - ErrorMessage

Status: Not Found

Schema

#### [ErrorMessage](#)

Default Response

ErrorMessage

Schema

#### [ErrorMessage](#)

Query for packages. If no parameters are given, this endpoint will return all packages. (*V1GetPackages*)

```
GET /api/v1/packages
```

## Parameters

| Name            | Source | Type                      | Go type | Separator | Required | Default | Description                                                                                                                                                     |
|-----------------|--------|---------------------------|---------|-----------|----------|---------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| all             | query  | boolean                   | bool    |           |          |         | If no pagination parameters are provided, defaults to true and returns all available results.                                                                   |
| name            | query  | string                    | string  |           |          |         | Name filter works as a substring match on the package name. For example, setting <code>name=cur</code> would match <code>curl</code> and <code>libcurl</code> . |
| package_manager | query  | string                    | string  |           |          |         |                                                                                                                                                                 |
| page            | query  | int64 (formatted integer) | int64   |           |          | 1       |                                                                                                                                                                 |

| Name      | Source | Type                         | Go type | Separator | Required | Default | Description |
|-----------|--------|------------------------------|---------|-----------|----------|---------|-------------|
| page_size | query  | int64<br>(formatted integer) | int64   |           |          | 20      |             |
| version   | query  | string                       | string  |           |          |         |             |

### All responses

| Code    | Status    | Description              | Has headers | Schema                 |
|---------|-----------|--------------------------|-------------|------------------------|
| 200     | OK        | PaginatedPackageResponse |             | <a href="#">schema</a> |
| 404     | Not Found | ErrorMessage             |             | <a href="#">schema</a> |
| default |           | ErrorMessage             |             | <a href="#">schema</a> |

### Responses

#### 200 - PaginatedPackageResponse

Status: OK

Schema

[PaginatedPackageResponse](#)

#### 404 - ErrorMessage

Status: Not Found

Schema

[ErrorMessage](#)

#### Default Response

ErrorMessage

Schema

[ErrorMessage](#)

**Query for sources. If no parameters are given, this endpoint will return all sources. (*V1GetSources*)**

```
GET /api/v1/sources
```

### Parameters

| Name      | Source                | Type                         | Go type             | Separator | Required | Default | Description                                                                                   |
|-----------|-----------------------|------------------------------|---------------------|-----------|----------|---------|-----------------------------------------------------------------------------------------------|
| all       | <a href="#">query</a> | boolean                      | <code>bool</code>   |           |          |         | If no pagination parameters are provided, defaults to true and returns all available results. |
| org       | <a href="#">query</a> | string                       | <code>string</code> |           |          |         |                                                                                               |
| page      | <a href="#">query</a> | int64<br>(formatted integer) | <code>int64</code>  |           |          | 1       |                                                                                               |
| page_size | <a href="#">query</a> | int64<br>(formatted integer) | <code>int64</code>  |           |          | 20      |                                                                                               |
| repo      | <a href="#">query</a> | string                       | <code>string</code> |           |          |         |                                                                                               |
| sha       | <a href="#">query</a> | string                       | <code>string</code> |           |          |         |                                                                                               |

## All responses

| Code    | Status    | Description             | Has headers | Schema                 |
|---------|-----------|-------------------------|-------------|------------------------|
| 200     | OK        | PaginatedSourceResponse |             | <a href="#">schema</a> |
| 404     | Not Found | ErrorMessage            |             | <a href="#">schema</a> |
| default |           | ErrorMessage            |             | <a href="#">schema</a> |

## Responses

### 200 - PaginatedSourceResponse

Status: OK

Schema

[PaginatedSourceResponse](#)

### 404 - ErrorMessage

Status: Not Found

Schema

[ErrorMessage](#)

### Default Response

ErrorMessage

Schema

[ErrorMessage](#)

Query for packages with source parameters. If no parameters are given, this endpoint will return all packages related to sources.  
(*V1GetSourcesPackages*)

```
GET /api/v1/sources/packages
```

#### All responses

| Code    | Status    | Description              | Has headers | Schema                 |
|---------|-----------|--------------------------|-------------|------------------------|
| 200     | OK        | PaginatedPackageResponse |             | <a href="#">schema</a> |
| 404     | Not Found | ErrorMessage             |             | <a href="#">schema</a> |
| default |           | ErrorMessage             |             | <a href="#">schema</a> |

#### Responses

##### 200 - PaginatedPackageResponse

Status: OK

Schema

[PaginatedPackageResponse](#)

##### 404 - ErrorMessage

Status: Not Found

Schema

[ErrorMessage](#)

##### Default Response

ErrorMessage

Schema

[ErrorMessage](#)

Query for vulnerabilities with source parameters. If no parameters are given, this endpoint will return all vulnerabilities.  
(*V1GetSourcesVulnerabilities*)

```
GET /api/v1/sources/vulnerabilities
```

#### Parameters

| Name      | Source | Type                      | Go type | Separator | Required | Default | Description                                                                                                  |
|-----------|--------|---------------------------|---------|-----------|----------|---------|--------------------------------------------------------------------------------------------------------------|
| Severity  | query  | string                    | string  |           |          |         | Case insensitive vulnerabilities severity filter. Possible values are: low, medium, high, critical, unknown. |
| all       | query  | boolean                   | bool    |           |          |         | If no pagination parameters are provided, defaults to true and returns all available results.                |
| org       | query  | string                    | string  |           |          |         |                                                                                                              |
| page      | query  | int64 (formatted integer) | int64   |           |          | 1       |                                                                                                              |
| page_size | query  | int64 (formatted integer) | int64   |           |          | 20      |                                                                                                              |
| repo      | query  | string                    | string  |           |          |         |                                                                                                              |
| sha       | query  | string                    | string  |           |          |         |                                                                                                              |

## All responses

| Code    | Status    | Description                    | Has headers | Schema                 |
|---------|-----------|--------------------------------|-------------|------------------------|
| 200     | OK        | PaginatedVulnerabilityResponse |             | <a href="#">schema</a> |
| 404     | Not Found | ErrorMessage                   |             | <a href="#">schema</a> |
| default |           | ErrorMessage                   |             | <a href="#">schema</a> |

## Responses

### 200 - PaginatedVulnerabilityResponse

Status: OK

Schema

[PaginatedVulnerabilityResponse](#)

### 404 - ErrorMessage

Status: Not Found

Schema

[ErrorMessage](#)

### Default Response

[ErrorMessage](#)



Schema

[ErrorMessage](#)

## Models

### ArtifactGroupPostRequest

#### Properties

| Name     | Type                       | Go type                        | Required | Default | Description                                                                                | Example                                                    |
|----------|----------------------------|--------------------------------|----------|---------|--------------------------------------------------------------------------------------------|------------------------------------------------------------|
| EntityID | uint64 (formatted integer) | <code>uint64</code>            |          |         | The database ID of the source or image being associated with this artifact group           | <code>24</code>                                            |
| Labels   | map of string              | <code>map[string]string</code> |          |         | Key-Value pair of labels associated with the Artifact Group                                | <code>{"env": "production", "namespace": "default"}</code> |
| Type     | string                     | <code>string</code>            |          |         | The entity type being associated with this artifact group. Allowable values: image, source | <code>image</code>                                         |
| UID      | string                     | <code>string</code>            | ✓        |         | Unique identifier for the Artifact Group such as workload UID                              | <code>8b1cc5da-fabe-45a6-ab8c-49260bbeef99</code>          |

### ArtifactGroupResponse

#### Properties

| Name     | Type                  | Go type                        | Required | Default | Description                                                   | Example                                                    |
|----------|-----------------------|--------------------------------|----------|---------|---------------------------------------------------------------|------------------------------------------------------------|
| Entities | <code>[]Entity</code> | <code>[]*Entity</code>         |          |         | Entities associated with the Artifact Group                   |                                                            |
| Labels   | map of string         | <code>map[string]string</code> |          |         | Key-Value pair of labels associated with the Artifact Group   | <code>{"env": "production", "namespace": "default"}</code> |
| UID      | string                | <code>string</code>            |          |         | Unique identifier for the Artifact Group such as workload UID | <code>8b1cc5da-fabe-45a6-ab8c-49260bbeef99</code>          |

### ArtifactGroupSearchFilters

#### Properties

| Name  | Type    | Go type             | Required | Default | Description                                                                                                                                                                                                                                                                            | Example                    |
|-------|---------|---------------------|----------|---------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------|
| All   | boolean | <code>bool</code>   |          |         | If no pagination parameters are provided, defaults to true and returns all available results.                                                                                                                                                                                          |                            |
| CVEID | string  | <code>string</code> |          |         | An optional CVE ID that the image and source must contain. Only packages, and their images and sources, with this CVE ID will be returned. If both package name and CVE ID are provided, then only the images and sources with the specified package name and CVE ID will be returned. | <code>CVE-7467-2020</code> |

| Name        | Type                      | Go type  | Required | Default | Description                                                                                                                                                                                                                                                                                | Example                                                                                                                                              |
|-------------|---------------------------|----------|----------|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| Digests     | []string                  | []string |          |         | A list of image digests. At least one image digest or source sha must be provided.                                                                                                                                                                                                         | ["9n38274ods897fmay487gsdyfga678wr82","7n38274ods897fmay487gsdyfga678wr82"]                                                                          |
| PackageName | string                    | string   |          |         | An optional package name that the image and source must contain. Only packages, and their images and sources, with this name will be returned. If both package name and CVE ID are provided, then only the images and sources with the specified package name and CVE ID will be returned. | package1                                                                                                                                             |
| Page        | int64 (formatted integer) | int64    |          | 1       |                                                                                                                                                                                                                                                                                            |                                                                                                                                                      |
| PageSize    | int64 (formatted integer) | int64    |          | 20      |                                                                                                                                                                                                                                                                                            |                                                                                                                                                      |
| Shas        | []string                  | []string |          |         | A list of source shas. At least one image digest or source sha must be provided.                                                                                                                                                                                                           | ["sha256:2c11624a8d9c9071996a886a4acaf09939ef3386e4c07735c6a2532f02eed4ea","sha256:04baf0d8df23ec342edb72acc3fb02f61c418bc6e8d7093149956a9aad2d12a"] |

## ArtifactGroupVulnReachFiltersPostRequest

### Properties

| Name     | Type                      | Go type  | Required | Default | Description                                                                                   | Example                                                                                                                                              |
|----------|---------------------------|----------|----------|---------|-----------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| All      | boolean                   | bool     |          |         | If no pagination parameters are provided, defaults to true and returns all available results. |                                                                                                                                                      |
| Digests  | []string                  | []string |          |         | A list of image digests.                                                                      | ["sha256:2c11624a8d9c9071996a886a4acaf09939ef3386e4c07735c6a2532f02eed4ea","sha256:04baf0d8df23ec342edb72acc3fb02f61c418bc6e8d7093149956a9aad2d12a"] |
| Page     | int64 (formatted integer) | int64    |          | 1       |                                                                                               |                                                                                                                                                      |
| PageSize | int64 (formatted integer) | int64    |          | 20      |                                                                                               |                                                                                                                                                      |

| Name | Type     | Go type  | Required | Default | Description            | Example                                                                      |
|------|----------|----------|----------|---------|------------------------|------------------------------------------------------------------------------|
| Shas | []string | []string |          |         | A list of source shas. | ["9n38274ods897fmay487gsdyfga678wr82", "7n38274ods897fmay487gsdyfga678wr82"] |

## ArtifactGroupVulnReachPostResponse

### Properties

| Name          | Type                       | Go type      | Required | Default | Description                                             | Example |
|---------------|----------------------------|--------------|----------|---------|---------------------------------------------------------|---------|
| AgCount       | uint64 (formatted integer) | uint64       |          |         | Number of artifact groups affected by the vulnerability | 5       |
| Vulnerability | VulnResponse               | VulnResponse |          |         |                                                         |         |

## ArtifactGroupVulnSearchFilters

### Properties

| Name             | Type                      | Go type  | Required | Default | Description                                                                                                                                                                                                                                                                                                                                   | Example                                                                      |
|------------------|---------------------------|----------|----------|---------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------|
| All              | boolean                   | bool     |          |         | If no pagination parameters are provided, defaults to true and returns all available results.                                                                                                                                                                                                                                                 |                                                                              |
| ArtifactGroupUID | string                    | string   |          |         | An optional artifact group UID that the image and source must contain. Only artifact groups, and their images and sources, with this artifact group UID will be returned. If both package name and artifact group UID are provided, then only the images and sources with the specified package name and artifact group UID will be returned. | 9aa3548e-5fae-11ed-9b6a-0242ac120002                                         |
| Digests          | []string                  | []string |          |         | A list of image digests. At least one image digest or source sha must be provided.                                                                                                                                                                                                                                                            | ["9n38274ods897fmay487gsdyfga678wr82", "7n38274ods897fmay487gsdyfga678wr82"] |
| PackageName      | string                    | string   |          |         | An optional package name that the image and source must contain. Only packages, and their images and sources, with this name will be returned. If both package name and artifact group UID are provided, then only the images and sources with the specified package name and artifact group UID will be returned.                            | package1                                                                     |
| Page             | int64 (formatted integer) | int64    |          | 1       |                                                                                                                                                                                                                                                                                                                                               |                                                                              |
| PageSize         | int64 (formatted integer) | int64    |          | 20      |                                                                                                                                                                                                                                                                                                                                               |                                                                              |

| Name | Type     | Go type  | Required | Default | Description                                                                      | Example                                                                                                                                                          |
|------|----------|----------|----------|---------|----------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Shas | []string | []string |          |         | A list of source shas. At least one image digest or source sha must be provided. | <pre>["sha256:2c11624a8d9c9071996a886a4acaf09939ef3386e4c07735c6a2532f02eed4ea", "sha256:04baf0d8df23ec342edb72acc3fb02f61c418bc6e8d7093149956a9aad2d12a"]</pre> |

## DeletedAt

- composed type [NullTime](#)

## Entity

### Properties

| Name     | Type                       | Go type   | Required | Default | Description                                                                                            | Example                                                                            |
|----------|----------------------------|-----------|----------|---------|--------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------|
| Digest   | string                     | string    |          |         | The digest of the image entity. Only visible if the entity is of image type                            | <pre>sha256:f7de1564f13da1ef7e5720ebce14006793242c0d8d7d60c343632bcf3bc5306d</pre> |
| Host     | string                     | string    |          |         | The dns name where the source entity is hosted on. Only visible if the entity type is of source type   | <pre>gitlab.com</pre>                                                              |
| ID       | uint64 (formatted integer) | uint64    | ✓        |         | The database ID of the source or image                                                                 | <pre>24</pre>                                                                      |
| Name     | string                     | string    |          |         | The name of the image entity. Only visible if the entity is of image type.                             | <pre>checkr/flagr</pre>                                                            |
| Org      | string                     | string    |          |         | The organization name of the source entity. Only visible if the entity type is of source type          | <pre>my-organization</pre>                                                         |
| Packages | []Package                  | []Package |          |         |                                                                                                        |                                                                                    |
| Registry | string                     | string    |          |         | The DNS name of the registry that stores the image entity. Only visible if the entity is of image type | <pre>docker.io</pre>                                                               |
| Repo     | string                     | string    |          |         | The repository name of the source entity. Only visible if the entity type is of source type            | <pre>my-sample-repo</pre>                                                          |
| Sha      | string                     | string    |          |         | The commit sha of the source entity. Only visible if the entity type is of source type                 | <pre>d6cd1e2bd19e03a81132a23bd2025920577f84e37</pre>                               |
| Type     | string                     | string    | ✓        |         | The entity Type of scan that is stored. This is set to either "image" or "source".                     | <pre>image</pre>                                                                   |

## ErrorMessage

ErrorMessage wraps an error message in a struct so responses are properly marshalled as a JSON object.

### Properties

| Name    | Type   | Go type | Required | Default | Description | Example              |
|---------|--------|---------|----------|---------|-------------|----------------------|
| Message | string | string  |          |         | in: body    | something went wrong |

## Image

### Properties

| Name     | Type                       | Go type    | Required | Default | Description | Example                            |
|----------|----------------------------|------------|----------|---------|-------------|------------------------------------|
| Digest   | string                     | string     | ✓        |         |             | 9n38274ods897fmay487gsdyfga678wr82 |
| ID       | uint64 (formatted integer) | uint64     |          |         |             |                                    |
| Name     | string                     | string     | ✓        |         |             | myorg/application                  |
| Package  | <a href="#">[]Package</a>  | []*Package |          |         |             |                                    |
| Registry | string                     | string     | ✓        |         |             | docker.io                          |
| Sources  | <a href="#">[]Source</a>   | []*Source  |          |         |             |                                    |

## MethodType

### Properties

| Name      | Type                         | Go type         | Required | Default | Description | Example |
|-----------|------------------------------|-----------------|----------|---------|-------------|---------|
| CreatedAt | date-time (formatted string) | strfmt.DateTime |          |         |             |         |
| DeletedAt | <a href="#">DeletedAt</a>    | DeletedAt       |          |         |             |         |
| ID        | uint64 (formatted integer)   | uint64          |          |         |             |         |
| Name      | string                       | string          |          |         |             |         |
| Rating    | <a href="#">[]Rating</a>     | []*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 own 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 | <a href="#">DeletedAt</a>    | DeletedAt       |          |         |             |         |
| ID        | uint64 (formatted integer)   | uint64          |          |         |             |         |

| Name      | Type                         | Go type                      | Required | Default | Description | Example |
|-----------|------------------------------|------------------------------|----------|---------|-------------|---------|
| UpdatedAt | date-time (formatted string) | <code>strfmt.DateTime</code> |          |         |             |         |

## NullTime

NullTime implements the Scanner interface so it can be used as a scan destination, similar to NullString.

### Properties

| Name  | Type                         | Go type                      | Required | Default | Description | Example |
|-------|------------------------------|------------------------------|----------|---------|-------------|---------|
| Time  | date-time (formatted string) | <code>strfmt.DateTime</code> |          |         |             |         |
| Valid | boolean                      | <code>bool</code>            |          |         |             |         |

## Package

### Properties

| Name            | Type                         | Go type                       | Required | Default | Description | Example |
|-----------------|------------------------------|-------------------------------|----------|---------|-------------|---------|
| Homepage        | string                       | <code>string</code>           |          |         |             |         |
| ID              | uint64 (formatted integer)   | <code>uint64</code>           |          |         |             |         |
| Images          | <code>[]Image</code>         | <code>[]*Image</code>         |          |         |             |         |
| Name            | string                       | <code>string</code>           |          |         |             |         |
| PackageManager  | string                       | <code>string</code>           |          |         |             |         |
| Sources         | <code>[]Source</code>        | <code>[]*Source</code>        |          |         |             |         |
| Version         | string                       | <code>string</code>           |          |         |             |         |
| Vulnerabilities | <code>[]Vulnerability</code> | <code>[]*Vulnerability</code> |          |         |             |         |

## PaginatedArtifactGroupVulnReachResponse

### Properties

| Name        | Type                                              | Go type                                            | Required | Default | Description | Example |
|-------------|---------------------------------------------------|----------------------------------------------------|----------|---------|-------------|---------|
| Count       | int64 (formatted integer)                         | <code>int64</code>                                 |          |         |             | 10      |
| CurrentPage | int64 (formatted integer)                         | <code>int64</code>                                 |          |         |             | 1       |
| LastPage    | int64 (formatted integer)                         | <code>int64</code>                                 |          |         |             | 2       |
| PageSize    | int64 (formatted integer)                         | <code>int64</code>                                 |          |         |             | 20      |
| Results     | <code>[]ArtifactGroupVulnReachPostResponse</code> | <code>[]*ArtifactGroupVulnReachPostResponse</code> |          |         |             |         |

## PaginatedResponse

### Properties

| Name  | Type                      | Go type            | Required | Default | Description | Example |
|-------|---------------------------|--------------------|----------|---------|-------------|---------|
| Count | int64 (formatted integer) | <code>int64</code> |          |         |             | 10      |

| Name        | Type                       | Go type                    | Required | Default | Description | Example |
|-------------|----------------------------|----------------------------|----------|---------|-------------|---------|
| CurrentPage | int64 (formatted integer)  | <code>int64</code>         |          |         |             | 1       |
| LastPage    | int64 (formatted integer)  | <code>int64</code>         |          |         |             | 2       |
| PageSize    | int64 (formatted integer)  | <code>int64</code>         |          |         |             | 20      |
| Results     | <code>[]interface{}</code> | <code>[]interface{}</code> |          |         |             |         |

## Rating

### Properties

| Name         | Type                       | Go type                 | Required | Default | Description | Example |
|--------------|----------------------------|-------------------------|----------|---------|-------------|---------|
| ID           | uint64 (formatted integer) | <code>uint64</code>     |          |         |             |         |
| MethodType   | <code>MethodType</code>    | <code>MethodType</code> |          |         |             |         |
| MethodTypeID | uint64 (formatted integer) | <code>uint64</code>     |          |         |             |         |
| Score        | double (formatted number)  | <code>float64</code>    |          |         |             |         |
| Severity     | string                     | <code>string</code>     |          |         |             |         |
| Vector       | string                     | <code>string</code>     |          |         |             |         |

## RatingResponse

### Properties

| Name     | Type                       | Go type              | Required | Default | Description                   | Example                    |
|----------|----------------------------|----------------------|----------|---------|-------------------------------|----------------------------|
| ID       | uint64 (formatted integer) | <code>uint64</code>  |          |         | Rating ID                     | 3                          |
| Score    | double (formatted number)  | <code>float64</code> |          |         | CVSS score                    | 9.7                        |
| Severity | string                     | <code>string</code>  |          |         | Threat level of vulnerability | High                       |
| Vector   | string                     | <code>string</code>  |          |         | CVSS score in vector format   | AV:L/AC:L/Au:N/C:C/I:C/A:C |

## Source

### Properties

| Name         | Type                       | Go type                 | Required | Default | Description | Example    |
|--------------|----------------------------|-------------------------|----------|---------|-------------|------------|
| DeletedAt    | <code>DeletedAt</code>     | <code>DeletedAt</code>  |          |         |             |            |
| Host         | string                     | <code>string</code>     |          |         |             | gitlab.com |
| ID           | uint64 (formatted integer) | <code>uint64</code>     |          |         |             |            |
| Images       | <code>[]Image</code>       | <code>[]*Image</code>   |          |         |             |            |
| Organization | string                     | <code>string</code>     |          |         |             | vmware     |
| Packages     | <code>[]Package</code>     | <code>[]*Package</code> |          |         |             |            |
| Repository   | string                     | <code>string</code>     | ✓        |         |             | myproject  |

| Name | Type   | Go type | Required | Default | Description | Example  |
|------|--------|---------|----------|---------|-------------|----------|
| Sha  | string | string  | ✓        |         |             | 0eb5fcd1 |

## StringArray

[]string

## VulnResponse

### Properties

| Name        | Type                       | Go type          | Required | Default | Description                      | Example                                                                                                                                                                                                                             |
|-------------|----------------------------|------------------|----------|---------|----------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CNA         | string                     | string           |          |         | CVE Numbering Authority          | GitHub, Inc.                                                                                                                                                                                                                        |
| CVEID       | string                     | string           |          |         | CVE ID of the vulnerability      | CVE-7467-2020                                                                                                                                                                                                                       |
| Description | string                     | string           |          |         | Description of the vulnerability | IBM Datapower Gateway 10.0.2.0 through 10.0.4.0, 10.0.1.0 through 10.0.1.5, and 2018.4.1.0 through 2018.4.1.18 could allow unauthorized viewing of logs and files due to insufficient authorization checks. IBM X-Force ID: 218856. |
| ID          | uint64 (formatted integer) | uint64           |          |         | Vulnerability ID                 | 12                                                                                                                                                                                                                                  |
| Rating      | []RatingResponse           | []RatingResponse |          |         | Rating information               |                                                                                                                                                                                                                                     |
| References  | []string                   | []string         |          |         | Additional external links        | https://github.com/example/repo/issues/11                                                                                                                                                                                           |
| URL         | string                     | string           |          |         | Related url to the vulnerability | https://nvd.nist.gov/vuln/detail/CVE-7467-2020                                                                                                                                                                                      |

## Vulnerability

### Properties

| Name        | Type                       | Go type   | Required | Default | Description | Example                        |
|-------------|----------------------------|-----------|----------|---------|-------------|--------------------------------|
| CNA         | string                     | string    |          |         |             | GitHub, Inc.                   |
| CVEID       | string                     | string    | ✓        |         |             | CVE-7467-2020                  |
| Description | string                     | string    |          |         |             | A description of CVE-7467-2020 |
| ID          | uint64 (formatted integer) | uint64    |          |         |             |                                |
| Package     | []Package                  | []Package |          |         |             |                                |



| Name       | Type                     | Go type                  | Required | Default | Description | Example                                                     |
|------------|--------------------------|--------------------------|----------|---------|-------------|-------------------------------------------------------------|
| Ratings    | <code>[]Rating</code>    | <code>[]*Rating</code>   |          |         |             |                                                             |
| References | <code>StringArray</code> | <code>StringArray</code> |          |         |             |                                                             |
| URL        | <code>string</code>      | <code>string</code>      |          |         |             | <code>https://nvd.nist.gov/vuln/detail/CVE-7467-2020</code> |

## artifactGroupPostEntity

### Properties

| Name | Type                                       | Go type             | Required | Default | Description                                                                        | Example            |
|------|--------------------------------------------|---------------------|----------|---------|------------------------------------------------------------------------------------|--------------------|
| ID   | <code>uint64</code><br>(formatted integer) | <code>uint64</code> | ✓        |         | The database ID of the source or image                                             | <code>24</code>    |
| Type | <code>string</code>                        | <code>string</code> | ✓        |         | The entity Type of scan that is stored. This is set to either "image" or "source". | <code>image</code> |

## artifactGroupPostResponse

### Properties

| Name     | Type                                   | Go type                                 | Required | Default | Description                                                   | Example                                                    |
|----------|----------------------------------------|-----------------------------------------|----------|---------|---------------------------------------------------------------|------------------------------------------------------------|
| Entities | <code>[]ArtifactGroupPostEntity</code> | <code>[]*ArtifactGroupPostEntity</code> |          |         | Entities associated with the Artifact Group                   |                                                            |
| Labels   | <code>map[string]string</code>         | <code>map[string]string</code>          |          |         | Key-Value pair of labels associated with the Artifact Group   | <code>{"env": "production", "namespace": "default"}</code> |
| UID      | <code>string</code>                    | <code>string</code>                     |          |         | Unique identifier for the Artifact Group such as workload UID | <code>8b1cc5da-fabe-45a6-ab8c-49260bbeef99</code>          |

## artifactGroupVulnArtifactGroup

### Properties

| Name     | Type                                   | Go type                                 | Required | Default | Description                                                   | Example                                                    |
|----------|----------------------------------------|-----------------------------------------|----------|---------|---------------------------------------------------------------|------------------------------------------------------------|
| Entities | <code>[]ArtifactGroupVulnEntity</code> | <code>[]*ArtifactGroupVulnEntity</code> |          |         | Entities associated with the Artifact Group                   |                                                            |
| Labels   | <code>map[string]string</code>         | <code>map[string]string</code>          |          |         | Key-Value pair of labels associated with the Artifact Group   | <code>{"env": "production", "namespace": "default"}</code> |
| UID      | <code>string</code>                    | <code>string</code>                     |          |         | Unique identifier for the Artifact Group such as workload UID | <code>8b1cc5da-fabe-45a6-ab8c-49260bbeef99</code>          |

## artifactGroupVulnEntity

### Properties

| Name     | Type                       | Go type | Required | Default | Description                                                                                            | Example                                                                 |
|----------|----------------------------|---------|----------|---------|--------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------|
| Digest   | string                     | string  |          |         | The digest of the image entity. Only visible if the entity is of image type                            | sha256:f7de1564f13da1ef7e5720ebce14006793242c0d8d7d60c343632bcf3bc5306d |
| Host     | string                     | string  |          |         | The dns name where the source entity is hosted on. Only visible if the entity type is of source type   | gitlab.com                                                              |
| ID       | uint64 (formatted integer) | uint64  | ✓        |         | The database ID of the source or image                                                                 | 24                                                                      |
| Name     | string                     | string  |          |         | The name of the image entity. Only visible if the entity is of image type.                             | checkr/flagr                                                            |
| Org      | string                     | string  |          |         | The organization name of the source entity. Only visible if the entity type is of source type          | my-organization                                                         |
| Registry | string                     | string  |          |         | The DNS name of the registry that stores the image entity. Only visible if the entity is of image type | docker.io                                                               |
| Repo     | string                     | string  |          |         | The repository name of the source entity. Only visible if the entity type is of source type            | my-sample-repo                                                          |
| Sha      | string                     | string  |          |         | The commit sha of the source entity. Only visible if the entity type is of source type                 | d6cd1e2bd19e03a81132a23b2025920577f84e37                                |
| Type     | string                     | string  | ✓        |         | The entity Type of scan that is stored. This is set to either "image" or "source".                     | image                                                                   |

## artifactGroupVulnPackage

### Properties

| Name            | Type                       | Go type          | Required | Default | Description                     | Example |
|-----------------|----------------------------|------------------|----------|---------|---------------------------------|---------|
| Homepage        | string                     | string           |          |         |                                 |         |
| ID              | uint64 (formatted integer) | uint64           |          |         |                                 |         |
| Images          | []Image                    | []*Image         |          |         | This field will always be empty | []      |
| Name            | string                     | string           |          |         |                                 |         |
| PackageManager  | string                     | string           |          |         |                                 |         |
| Sources         | []Source                   | []*Source        |          |         | This field will always be empty | []      |
| Version         | string                     | string           |          |         |                                 |         |
| Vulnerabilities | []Vulnerability            | []*Vulnerability |          |         | This field will always be empty | []      |

## artifactGroupVulnResult

## Properties

| Name           | Type                                          | Go type                                        | Required | Default | Description | Example                                                                                                     |
|----------------|-----------------------------------------------|------------------------------------------------|----------|---------|-------------|-------------------------------------------------------------------------------------------------------------|
| ArtifactGroups | <code>[]ArtifactGroupVulnArtifactGroup</code> | <code>[]*ArtifactGroupVulnArtifactGroup</code> |          |         |             |                                                                                                             |
| CNA            | string                                        | string                                         |          |         |             | GitHub, Inc.                                                                                                |
| CVEID          | string                                        | string                                         | ✓        |         |             | CVE-7467-2020                                                                                               |
| Description    | string                                        | string                                         |          |         |             | A description of CVE-7467-2020                                                                              |
| ID             | uint64 (formatted integer)                    | uint64                                         |          |         |             |                                                                                                             |
| Packages       | <code>[]ArtifactGroupVulnPackage</code>       | <code>[]*ArtifactGroupVulnPackage</code>       |          |         |             |                                                                                                             |
| Ratings        | <code>[]Rating</code>                         | <code>[]*Rating</code>                         |          |         |             |                                                                                                             |
| References     | StringArray                                   | StringArray                                    |          |         |             |                                                                                                             |
| URL            | string                                        | string                                         |          |         |             | <a href="https://nvd.nist.gov/vuln/detail/CVE-7467-2020">https://nvd.nist.gov/vuln/detail/CVE-7467-2020</a> |

## paginatedArtifactGroupResponse

### Properties

| Name        | Type                                 | Go type                               | Required | Default | Description | Example |
|-------------|--------------------------------------|---------------------------------------|----------|---------|-------------|---------|
| Count       | int64 (formatted integer)            | int64                                 |          |         |             | 10      |
| CurrentPage | int64 (formatted integer)            | int64                                 |          |         |             | 1       |
| LastPage    | int64 (formatted integer)            | int64                                 |          |         |             | 2       |
| PageSize    | int64 (formatted integer)            | int64                                 |          |         |             | 20      |
| Results     | <code>[]ArtifactGroupResponse</code> | <code>[]*ArtifactGroupResponse</code> |          |         |             |         |

## paginatedArtifactGroupVulnerabilityResponse

### Properties

| Name        | Type                                   | Go type                                 | Required | Default | Description | Example |
|-------------|----------------------------------------|-----------------------------------------|----------|---------|-------------|---------|
| Count       | int64 (formatted integer)              | int64                                   |          |         |             | 10      |
| CurrentPage | int64 (formatted integer)              | int64                                   |          |         |             | 1       |
| LastPage    | int64 (formatted integer)              | int64                                   |          |         |             | 2       |
| PageSize    | int64 (formatted integer)              | int64                                   |          |         |             | 20      |
| Results     | <code>[]ArtifactGroupVulnResult</code> | <code>[]*ArtifactGroupVulnResult</code> |          |         |             |         |

## paginatedImageResponse

### Properties

| Name        | Type                            | Go type                       | Required | Default | Description | Example |
|-------------|---------------------------------|-------------------------------|----------|---------|-------------|---------|
| Count       | int64 (formatted integer)       | <code>int64</code>            |          |         |             | 10      |
| CurrentPage | int64 (formatted integer)       | <code>int64</code>            |          |         |             | 1       |
| LastPage    | int64 (formatted integer)       | <code>int64</code>            |          |         |             | 2       |
| PageSize    | int64 (formatted integer)       | <code>int64</code>            |          |         |             | 20      |
| Results     | <a href="#">[]ResponseImage</a> | <code>[]*ResponseImage</code> |          |         |             |         |

## paginatedPackageResponse

### Properties

| Name        | Type                              | Go type                         | Required | Default | Description | Example |
|-------------|-----------------------------------|---------------------------------|----------|---------|-------------|---------|
| Count       | int64 (formatted integer)         | <code>int64</code>              |          |         |             | 10      |
| CurrentPage | int64 (formatted integer)         | <code>int64</code>              |          |         |             | 1       |
| LastPage    | int64 (formatted integer)         | <code>int64</code>              |          |         |             | 2       |
| PageSize    | int64 (formatted integer)         | <code>int64</code>              |          |         |             | 20      |
| Results     | <a href="#">[]ResponsePackage</a> | <code>[]*ResponsePackage</code> |          |         |             |         |

## paginatedSourceResponse

### Properties

| Name        | Type                             | Go type                        | Required | Default | Description | Example |
|-------------|----------------------------------|--------------------------------|----------|---------|-------------|---------|
| Count       | int64 (formatted integer)        | <code>int64</code>             |          |         |             | 10      |
| CurrentPage | int64 (formatted integer)        | <code>int64</code>             |          |         |             | 1       |
| LastPage    | int64 (formatted integer)        | <code>int64</code>             |          |         |             | 2       |
| PageSize    | int64 (formatted integer)        | <code>int64</code>             |          |         |             | 20      |
| Results     | <a href="#">[]ResponseSource</a> | <code>[]*ResponseSource</code> |          |         |             |         |

## paginatedVulnerabilityResponse

### Properties

| Name        | Type                                    | Go type                               | Required | Default | Description | Example |
|-------------|-----------------------------------------|---------------------------------------|----------|---------|-------------|---------|
| Count       | int64 (formatted integer)               | <code>int64</code>                    |          |         |             | 10      |
| CurrentPage | int64 (formatted integer)               | <code>int64</code>                    |          |         |             | 1       |
| LastPage    | int64 (formatted integer)               | <code>int64</code>                    |          |         |             | 2       |
| PageSize    | int64 (formatted integer)               | <code>int64</code>                    |          |         |             | 20      |
| Results     | <a href="#">[]ResponseVulnerability</a> | <code>[]*ResponseVulnerability</code> |          |         |             |         |

## responseImage

### Properties

| Name      | Type                         | Go type                           | Required | Default | Description | Example                                              |
|-----------|------------------------------|-----------------------------------|----------|---------|-------------|------------------------------------------------------|
| CreatedAt | date-time (formatted string) | <code>strfmt.Date<br/>Time</code> |          |         |             |                                                      |
| Digest    | string                       | <code>string</code>               | ✓        |         |             | <code>9n38274ods897fmay487gsdyfg<br/>a678wr82</code> |
| ID        | uint64 (formatted integer)   | <code>uint64</code>               |          |         |             |                                                      |
| Name      | string                       | <code>string</code>               | ✓        |         |             | <code>myorg/application</code>                       |
| Packages  | <a href="#">[]Package</a>    | <code>[]*Package</code>           |          |         |             |                                                      |
| Registry  | string                       | <code>string</code>               | ✓        |         |             | <code>docker.io</code>                               |
| Sources   | <a href="#">[]Source</a>     | <code>[]*Source</code>            |          |         |             |                                                      |
| UpdatedAt | date-time (formatted string) | <code>strfmt.Date<br/>Time</code> |          |         |             |                                                      |

## responsePackage

### Properties

| Name            | Type                            | Go type                       | Required | Default | Description | Example |
|-----------------|---------------------------------|-------------------------------|----------|---------|-------------|---------|
| CreatedAt       | date-time (formatted string)    | <code>strfmt.DateTime</code>  |          |         |             |         |
| Homepage        | string                          | <code>string</code>           |          |         |             |         |
| ID              | uint64 (formatted integer)      | <code>uint64</code>           |          |         |             |         |
| Images          | <a href="#">[]Image</a>         | <code>[]*Image</code>         |          |         |             |         |
| Name            | string                          | <code>string</code>           |          |         |             |         |
| PackageManager  | string                          | <code>string</code>           |          |         |             |         |
| Sources         | <a href="#">[]Source</a>        | <code>[]*Source</code>        |          |         |             |         |
| UpdatedAt       | date-time (formatted string)    | <code>strfmt.DateTime</code>  |          |         |             |         |
| Version         | string                          | <code>string</code>           |          |         |             |         |
| Vulnerabilities | <a href="#">[]Vulnerability</a> | <code>[]*Vulnerability</code> |          |         |             |         |

## responseSource

### Properties

| Name         | Type                         | Go type                      | Required | Default | Description | Example                 |
|--------------|------------------------------|------------------------------|----------|---------|-------------|-------------------------|
| CreatedAt    | date-time (formatted string) | <code>strfmt.DateTime</code> |          |         |             |                         |
| DeletedAt    | <a href="#">DeletedAt</a>    | <code>DeletedAt</code>       |          |         |             |                         |
| Host         | string                       | <code>string</code>          |          |         |             | <code>gitlab.com</code> |
| ID           | uint64 (formatted integer)   | <code>uint64</code>          |          |         |             |                         |
| Images       | <a href="#">[]Image</a>      | <code>[]*Image</code>        |          |         |             |                         |
| Organization | string                       | <code>string</code>          |          |         |             | <code>vmware</code>     |
| Packages     | <a href="#">[]Package</a>    | <code>[]*Package</code>      |          |         |             |                         |
| Repository   | string                       | <code>string</code>          | ✓        |         |             | <code>myproject</code>  |

| Name      | Type                         | Go type                      | Required | Default | Description | Example               |
|-----------|------------------------------|------------------------------|----------|---------|-------------|-----------------------|
| Sha       | string                       | <code>string</code>          | ✓        |         |             | <code>0eb5fcd1</code> |
| UpdatedAt | date-time (formatted string) | <code>strfmt.DateTime</code> |          |         |             |                       |

## responseVulnerability

### Properties

| Name        | Type                         | Go type                      | Required | Default | Description | Example                                                     |
|-------------|------------------------------|------------------------------|----------|---------|-------------|-------------------------------------------------------------|
| CNA         | string                       | <code>string</code>          |          |         |             | <code>GitHub, Inc.</code>                                   |
| CVEID       | string                       | <code>string</code>          | ✓        |         |             | <code>CVE-7467-2020</code>                                  |
| Created At  | date-time (formatted string) | <code>strfmt.DateTime</code> |          |         |             |                                                             |
| Description | string                       | <code>string</code>          |          |         |             | <code>A description of CVE-7467-2020</code>                 |
| ID          | uint64 (formatted integer)   | <code>uint64</code>          |          |         |             |                                                             |
| Packages    | <code>[]Package</code>       | <code>[]*Package</code>      |          |         |             |                                                             |
| Ratings     | <code>[]Rating</code>        | <code>[]*Rating</code>       |          |         |             |                                                             |
| References  | <code>StringArray</code>     | <code>StringArray</code>     |          |         |             |                                                             |
| URL         | string                       | <code>string</code>          |          |         |             | <code>https://nvd.nist.gov/vuln/detail/CVE-7467-2020</code> |
| UpdatedAt   | date-time (formatted string) | <code>strfmt.DateTime</code> |          |         |             |                                                             |

## API walkthrough for Supply Chain Security Tools - Store

This topic tells you how to make an API call that you can use with Supply Chain Security Tools - Store. For information about using the SCST - Store API, see [API reference for Supply Chain Security Tools - Store](#).

### Using curl to post an image report

1. Switch to the `kubectl` context or `kubeconfig` to target the View cluster.
2. Retrieve the CA certificate and store it locally. Run:

```
kubectl get secret ingress-cert -n metadata-store -o json | jq -r '.data."ca.crt"' | base64 -d > /tmp/ca.crt
```

3. Using the `health` endpoint as an example, run:

```
curl -i https://metadata-store.INGRESS-DOMAIN/api/health \
--cacert /tmp/ca.crt
```

For example:

```
$ curl -i https://metadata-store.example.com/api/health \
--cacert /tmp/ca.crt
```

```
HTTP/2 200
content-length: 0
date: Tue, 23 Jan 2024 22:50:57 GMT
x-envoy-upstream-service-time: 0
server: envoy
```

- To make a request to an authenticated endpoint an access token is required. To retrieve the `metadata-store-read-write-client` access token, run:

```
export METADATA_STORE_ACCESS_TOKEN=$(kubectl get secrets metadata-store-read-wr
ite-client -n metadata-store -o jsonpath="{.data.token}" | base64 -d)
```

For more information, see [Retrieve access tokens for Supply Chain Security Tools - Store](#).

- Using the `api/imageReport` endpoint as an example, create a post request:

```
curl https://metadata-store.INGRESS-DOMAIN/api/imageReport \
--cacert /tmp/ca.crt \
-H "Authorization: Bearer ${METADATA_STORE_ACCESS_TOKEN}" \
-H "Content-Type: application/json" \
-X POST \
--data "@ABSOLUTE-PATH-TO-THE-POST-BODY"
```

Where `ABSOLUTE-PATH-TO-THE-POST-BODY` is the absolute filepath of the API JSON for an image report.

For example, the following is a sample post body of an image report API JSON:

```
{
  "Name" : "burger-image-2",
  "Registry" : "test-registry",
  "Digest" : "test-digest@45asd61asasssdfsdfddssghjkdfsdasdfsasdsasdasdfghjdd
asfddfsadfadfgfshdasdfsdfsdsdasdsdfsdfadssdfdasdfaasdsdfsdfdsasgsasddffdgf
dasddfgdfssdfakasdadasdsdasddasdsd23",
  "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.com",
        "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" : [""]
      }
    ]
  }
]

```



#### Note

If Tanzu Application Platform is deployed without Ingress, see [Use your NodePort with Supply Chain Security Tools - Store](#) and [Use your LoadBalancer with Supply Chain Security Tools - Store](#).

## Connect to the PostgreSQL database

You can use a PostgreSQL database with Supply Chain Security Tools - Store. To connect to the PostgreSQL database, you need the following values:

- database name
- user name
- password
- database host
- database port
- database CA certificate

Connect to the PostgreSQL database:

1. Obtain the database name, user name, password, and CA certificate. Run:

```

db_name=$(kubectl get secret postgres-db-secret -n metadata-store -o json | jq
-r '.data.POSTGRES_DB' | base64 -d)
db_username=$(kubectl get secret postgres-db-secret -n metadata-store -o json |
jq -r '.data.POSTGRES_USER' | base64 -d)
db_password=$(kubectl get secret postgres-db-secret -n metadata-store -o json |
jq -r '.data.POSTGRES_PASSWORD' | base64 -d)

db_ca_dir=$(mktemp -d -t ca-cert-XXXX)
db_ca_path="$db_ca_dir/ca.crt"
kubectl get secrets postgres-db-tls-cert -n metadata-store -o json | jq -r '.da
ta."ca.crt"' | base64 -d > $db_ca_path

```



If the password was auto-generated, the `password` command returns an empty string. Run:

```
db_password=$(kubectl get secret postgres-db-password -n metadata-store -o json
| jq -r '.data.DB_PASSWORD' | base64 -d)
```

2. In a separate terminal, run:

```
kubectl port-forward service/metadata-store-db 5432:5432 -n metadata-store
```

3. Set the database host and port values on the first terminal:

```
db_host="localhost"
db_port=5432
```

4. To port forward to a different local port number, use the following command template:

```
kubectl port-forward service/metadata-store-db <LOCAL_PORT>:5432 -n metadata-store
```

Where `LOCAL-PORT` is the port number for the database you want to use.

You can now connect to the database and make queries. For example:

```
psql "host=$db_host port=$db_port user=$db_username dbname=$db_name sslmode=verify-ca
sslrootcert=$db_ca_path" -c "SELECT * FROM images"
```

You can use GUI clients such as [Postico](#) or [DBeaver](#) to interact with the database.

## Deployment details and configuration for Supply Chain Security Tools - Store

This topic describes how you can deploy and configure your Kubernetes cluster for Supply Chain Security Tools (SCST) - Store.

### What is deployed

The installation creates the following in your Kubernetes cluster:

- Two components — an API back end and a database. Each component includes:
  - service
  - deployment
  - replicaset
  - Pod
- Persistent volume claim
- External IP address (based on a deployment configuration set to use [LoadBalancer](#)).
- A Kubernetes secret to allow pulling SCST - Store images from a registry.
- A namespace called `metadata-store`.
- A service account with read-write privileges named `metadata-store-read-write-client`, and a corresponding secret for the service account. It's bound to a ClusterRole named `metadata-store-read-write`.
- A read-only ClusterRole named `metadata-store-read-only` that isn't bound to a service account. See [Service Accounts](#).

- (Optional) An HTTPProxy object for ingress support.

## Deployment configuration

All configurations are nested inside of `metadata_store` in your tap values deployment YAML.

## Supported Network Configurations

The following connection methods are recommended based on Tanzu Application Platform setup:

- Single or multicluster with Contour = `Ingress`
- Single cluster without Contour and with `LoadBalancer` support = `LoadBalancer`
- Single cluster without Contour and without `LoadBalancer` = `NodePort`
- Multicluster without Contour = Not supported

For a production environment, VMware recommends that you install SCST - Store with ingress enabled.

### App service type

Supported values include `LoadBalancer`, `ClusterIP`, `NodePort`. The `app_service_type` is set to `LoadBalancer` by default. If your environment does not support `LoadBalancer`, and you want to use `ClusterIP`, configure the `app_service_type` property in your deployment YAML:

```
app_service_type: "ClusterIP"
```

If you set the `ingress_enabled` to `"true"`, VMware recommends setting the `app_service_type` property to `"ClusterIP"`.

### Ingress support

SCST - Store's values file allows you to enable ingress support and to configure a custom domain name to use Contour to provide external access to SCST - Store's API. For example:

```
ingress_enabled: "true"
ingress_domain: "example.com"
app_service_type: "ClusterIP" # recommended setting
```

An HTTPProxy object is installed with `metadata-store.example.com` as the fully qualified domain name. See [Ingress](#).



#### Note

The `ingress_enabled` property expects a string value of `"true"` or `"false"`, not a Boolean value.

## Database configuration

The default database included with the deployment is meant to get users started using the metadata store. The default database deployment does not support many enterprise production requirements, including scaling, redundancy, or failover. However, it is a secure deployment.

### Using AWS RDS PostgreSQL database

Users can also configure the deployment to use their own RDS database instead of the default. See [AWS RDS Postgres Configuration](#).

### Using external PostgreSQL database

Users can configure the deployment to use any other PostgreSQL database. See [Use external postgres database](#).

### Custom database password

By default, a database password is generated upon deployment. To configure a custom password, use the `db_password` property in the deployment YAML.

```
db_password: "PASSWORD-0123"
```

Where `PASSWORD-0123` is the same password used between deployments.



#### Important

There is a known issue related to changing database passwords [Persistent Volume Retains Data](#).

## Service accounts

By default, a service account with read-write privileges to the metadata store app is installed. This service account is a cluster-wide account that uses ClusterRole. If you don't want the service account and role, set the `add_default_rw_service_account` property to `"false"`. To create a custom service account, see [Create Service Account](#).

The store creates a read-only cluster role, which is bound to a service account by using `ClusterRoleBinding`. To create service accounts to bind to this cluster role, see [Create Service Account](#).

## Exporting certificates

SCST - Store creates a [Secret Export](#) for exporting certificates to `Supply Chain Security Tools - Scan` to securely post scan results. These certificates are exported to the namespace where `Supply Chain Security Tools - Scan` is installed.

## Configure your AWS RDS PostgreSQL configuration

This topic describes how you can configure your AWS RDS PostgreSQL configuration for Supply Chain Security Tools (SCST) - Store.

### Prerequisites

- AWS Account

### Setup certificate and configuration

1. Create an Amazon RDS Postgres using the [Amazon RDS Getting Started Guide](#)
2. Once the database instance starts, retrieve the following information:
  1. DB Instance Endpoint

2. Master Username
3. Master Password
4. Database Name

**Note**

If the database name is - in the AWS RDS UI, the value is likely `postgres`.

3. Create a security group to allow inbound connections from the cluster to the Postgres DB
4. Retrieve the corresponding CA Certificate that signed the Postgres TLS Certificate using the following [link](#)
5. In the `metadata-store-values.yaml` fill the following settings:

```
db_host: "<DB Instance Endpoint>"
db_user: "<Master Username>"
db_password: "<Master Password>"
db_name: "<Database Name>"
db_port: "5432"
db_sslmode: "verify-full"
db_max_open_conns: 10
db_max_idle_conns: 100
db_conn_max_lifetime: 60
db_ca_certificate: |
  <Corresponding CA Certification>
  ...
  ...
  ...
deploy_internal_db: "false"
```

**Note**

If `deploy_internal_db` is set to `false`, an instance of Postgres will not be deployed in the cluster.

## Use external PostgreSQL database for Supply Chain Security Tools - Store

This topic describes how you can configure and use your external PostgreSQL database for Supply Chain Security Tools (SCST) - Store.

### Prerequisites

- Set up your external PostgreSQL database. After the database instance starts, retrieve the following information:
  1. Database Instance Endpoint
  2. Main User name
  3. Main Password
  4. Database Name

### Set up certificate and configuration

1. Create a security group to allow inbound connections from the cluster to the PostgreSQL database.
2. Retrieve the corresponding CA Certificate that signed the PostgreSQL TLS Certificate.
3. In the `metadata-store-values.yaml` fill the following settings:

```
db_host: "<DB Instance Endpoint>"
db_user: "<Master Username>"
db_password: "<Master Password>"
db_name: "<Database Name>"
db_port: "5432"
db_sslmode: "verify-full"
db_max_open_conns: 10
db_max_idle_conns: 100
db_conn_max_lifetime: 60
db_ca_certificate: |
  <Corresponding CA Certification>
  ...
  ...
  ...
deploy_internal_db: "false"
```



#### Note

If `deploy_internal_db` is set to `false`, an instance of PostgreSQL is not deployed in the cluster.

## Validation

Verification was done using bitnami PostgreSQL. You can get more information from the [bitnami documentation](#).

## Database backup recommendations for Supply Chain Security Tools - Store

This topic describes database backup recommendations for Supply Chain Security Tools - Store.

By default, the metadata store uses a `PersistentVolume` mounted on a Postgres instance, making it a stateful component of Tanzu Application Platform. VMware recommends implementing a regular backup strategy as part of your disaster recovery plan when using the provided Postgres instance.

## Backup

You can use [Velero](#) to create regular backups.



#### Note

Backup support for `PersistentVolume` depends on the used `StorageClass` and existing provider plug-ins. See the officially [supported plug-ins here](#).

```
velero install --provider PROVIDER --bucket BUCKET-NAME --plugins PLUGIN-IMAGE-LOCATIO
N --secret-file SECRET-FILE
```

Where:

- `PROVIDER` is the name of the provider you want to use.
- `BUCKET-NAME` is the name of the bucket you want to use.
- `PLUGIN-IMAGE-LOCATION` is the location of the plug ins you want to use.
- `SECRET-FILE` is the file where the secret is located.

Velero CLI can then be used to create a backup of all the resources in the `metadata-store` namespace, including `PersistentVolumeClaim` and `PersistentVolume`.

```
velero backup create metadata-store-$(date +%s) --include-namespaces=metadata-store
```

## Restore

Velero CLI can restore the Store in the same or a different cluster. The same namespace can be used to restore, but may collide with other Supply Chain Security Tools – Store installations. Furthermore, restoring into the same namespace restores a fully functional instance of Supply Chain Security Tools – Store; however, this instance is not managed by Tanzu Application Platform and can cause conflicts with future installations.

```
velero restore create restore-metadata-store-$timestamp --from-backup metadata-store-$timestamp --namespace-mappings metadata-store:metadata-store
```

Alternatively, a different namespace can be used to restore Supply Chain Security Tools – Store. In this case, Supply Chain Security Tools – Store API is not available due to conflicting definitions in the RBAC proxy configuration, causing all requests to fail with an `Unauthorized` error. In this scenario, the postgres instance is still accessible, and tools such as `pg_dump` can be used to retrieve table contents and restore in a new live installation of Supply Chain Security Tools – Store.

```
velero restore create restore-metadata-store-$timestamp --from-backup metadata-store-$timestamp --namespace-mappings metadata-store:restored-metadata-store
```

Currently, mounting an existing `PersistentVolume` or `PersistentVolumeClaim` during installation is not supported.

The minimum suggested resources for backups are `PersistentVolume`, `PersistentVolumeClaim` and `Secret`. The database password `Secret` is needed to set up a Postgres instance with the correct password to properly read data from the restored volume.

## Log configuration and usage for Supply Chain Security Tools - Store

This topic describes how you can configure Supply Chain Security Tools (SCST) - Store to output and interpret detailed log information.

### Verbosity levels

There are six verbosity levels that the Supply Chain Security Tools - Store supports.

| Level | Description                             |
|-------|-----------------------------------------|
| Trace | Output extended debugging logs.         |
| Debug | Output standard debugging logs.         |
| More  | Output more verbose informational logs. |

| Level   | Description                                  |
|---------|----------------------------------------------|
| Default | Output standard informational logs.          |
| Less    | Outputs less verbose informational logs.     |
| Minimum | Outputs a minimal set of informational logs. |

When the Store is deployed at a specific verbosity level, all logs of that level and lower are outputted to the console. For example, setting the verbosity level to `More` outputs logs from `Minimal` to `More`, while `Debug` and `Trace` logs are muted.

Currently, the application logs output at these levels:

- **Minimum** does not output any logs.
- **Less** outputs a single log line indicating the current verbosity level the Metadata Store is configured to when the application starts.
- **Default** outputs API endpoint access information.
- **Debug** outputs API endpoint payload information, both for requests and responses.
- **Trace** outputs verbose debug information about the actual SQL queries for the database.

Other log levels do not output any additional log information and are present for future extensibility.

If no verbosity level is specified when the Store is installed, the level is set to `default`.

## Error Logs

Error logs are always outputted regardless of the verbosity level, even when set to `minimum`.

## Obtaining logs

Kubernetes pods emit logs. The deployment has two pods: one for the database and one for the API back end.

Use `kubectl get pods` to obtain the names of the pods by running:

```
kubectl get pods -n metadata-store
```

For example:

```
$ kubectl get pods -n metadata-store
NAME                                READY   STATUS    RESTARTS   AGE
metadata-store-app-67659bbc66-2rc6k  2/2     Running   0           4d3h
metadata-store-db-64d5b88587-8dns7    1/1     Running   0           4d3h
```

The database pod has prefix `metadata-store-db-` and the API backend pod has the prefix `metadata-store-app-`. Use `kubectl logs` to get the logs from the pod you're interested in. For example, to see the logs of the database pod, run:

```
$ kubectl logs metadata-store-db-64d5b88587-8dns7 -n metadata-store
The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.
...
```

The API backend pod has two containers, one for `kube-rbac-proxy`, and the other for the API server. Use the `--all-containers` flag to see logs from both containers. For example:

```
$ kubectl logs metadata-store-app-67659bbc66-2rc6k --all-containers -n metadata-store
I1206 18:34:17.686135      1 main.go:150] Reading config file: /etc/kube-rbac-proxy/c
onfig-file.yaml
I1206 18:34:17.784900      1 main.go:180] Valid token audiences:
...
{"level":"info","ts":"2022-05-27T13:47:52.54099339Z","logger":"MetadataStore","msg":"L
og settings","hostname":"metadata-store-app-5c9d6bccdb-kcrt2","LOG_LEVEL":"default"}
{"level":"info","ts":"2022-05-27T13:47:52.541133699Z","logger":"MetadataStore","ms
g":"Server Settings","hostname":"metadata-store-app-5c9d6bccdb-kcrt2","bindingaddres
s":"localhost:9443"}
{"level":"info","ts":"2022-05-27T13:47:52.541150096Z","logger":"MetadataStore","ms
g":"Database Settings","hostname":"metadata-store-app-5c9d6bccdb-kcrt2","maxopenconnec
tion":10,"maxidleconnection":100,"connectionmaxlifetime":60}
```



### Note

The `kube-rbac-proxy` container uses a different log format than the Store. For information about the proxy's container log format, see [Logging Formats](#) in Github.

## API endpoint log output

When an API endpoint handles a request, the Store generates two and five log lines. They are:

1. When the endpoint receives a request, it outputs a `Processing request` line. This logline is shown at the `default` verbosity level.
2. If the endpoint includes query or path parameters, it outputs a `Request parameters` line. This line logs the parameters passed in the request. This line is shown at the `default` verbosity level.
3. If the endpoint takes in a request body, it outputs a `Request body` line. This line outputs the entire request body as a string. This line is shown at the `debug` verbosity level.
4. When the endpoint returns a response, it outputs a `Request response` line. This line is shown at the `default` verbosity level.
5. If the endpoint returns a response body, it outputs a second `Request response` line with an extra key `payload`, and its value is set to the entire response body. This line is shown at the `debug` verbosity level.

## Format

The logs use JSON output format.

When the Store handles a request, it outputs some API endpoint access information in the following format:

```
{"level":"info","ts":"2022-05-27T15:41:36.051991749Z","logger":"MetadataStore","ms
g":"Processing request","hostname":"metadata-store-app-c7c8648f7-8dmd1","method":"GE
T","endpoint":"/api/images?digest=sha256%3A20521f76ff3d27f436e03dc666cc97a511bbe71e8e8
495f851d0f4bf57b0bab6"}
```

### Key-value pairs

Since JSON output format uses Key-value pairs, the tables in the following sections list each key and the meaning of their values.

#### Common to all logs



The following key-value pairs are common for all logs.

| Key      | Type    | Verbosity Level | Description                                                                                                                                                                                                       |
|----------|---------|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| level    | string  | all             | The log level of the message. This is either 'error' for error messages, or 'info' for all other messages.                                                                                                        |
| ts       | string  | all             | The timestamp when the log message was generated. It uses RFC 3339 format with nanosecond precision and 00:00 offset from UTC, meaning Zulu time.                                                                 |
| logger   | string  | all             | Used to identify what produced the log entry. For Store, the name always starts with <code>MetadataStore</code> . For log entries that display the raw SQL queries, the name is <code>MetadataStore.gorm</code> . |
| msg      | string  | all             | A short description of the logged event.                                                                                                                                                                          |
| hostname | string  | all             | The Kubernetes hostname of the pod handling the request. This helps identify the specific instance of the Store when you deploy multiple instances on a cluster.                                                  |
| error    | string  | all             | The error message which is only available in error log entries.                                                                                                                                                   |
| endpoint | string  | default         | The API endpoint the Metadata Store attempts to handle the request. This also includes any query and path parameters passed in.                                                                                   |
| method   | string  | default         | The HTTP verb to access the endpoint. For example, 'GET' or 'POST'.                                                                                                                                               |
| code     | integer | default         | The HTTP response code.                                                                                                                                                                                           |
| response | string  | default         | The HTTP response in human-readable format. For example, 'OK', 'Bad Request', or 'Internal Server Error'.                                                                                                         |
| function | string  | debug           | The function name that handles the request.                                                                                                                                                                       |

#### Logging query and path parameter values

Those endpoints that use query or path parameters are logged on the `Request parameters` logline as key-value pairs. Afterward, they are appended to all other log lines of the same request as key-value pairs.

The key names are the query or path parameter's name, while the value is set to the value of those parameters in string format.

For example, the following log line contains the `digest` and `id` key, which represents the respective `digest` and `id` query parameters, as well as their values:

```
{ "level": "info", "ts": "2022-05-27T15:41:36.052063176Z", "logger": "MetadataStore", "msg": "Request parameters", "hostname": "metadata-store-app-c7c8648f7-8dmd1", "method": "GET", "endpoint": "/api/images?digest=sha256%3A20521f76ff3d27f436e03dc666cc97a511bbe71e8e8495f851d0f4bf57b0bab6", "id": 0, "digest": "sha256:20521f76ff3d27f436e03dc666cc97a511bbe71e8e8495f851d0f4bf57b0bab6", "name": "" }
```

These key/value pairs show up in all subsequent log lines of the same call. For example:

```
{ "level": "info", "ts": "2022-05-27T15:41:36.057393519Z", "logger": "MetadataStore", "msg": "Request response", "hostname": "metadata-store-app-c7c8648f7-8dmd1", "method": "GET", "endpoint": "/api/images?digest=sha256%3A20521f76ff3d27f436e03dc666cc97a511bbe71e8e8495f851d0f4bf57b0bab6", "id": 0, "digest": "sha256:20521f76ff3d27f436e03dc666cc97a511bbe71e8e8495f851d0f4bf57b0bab6", "name": "", "code": 200, "response": "OK" }
```

This is done to ensure:

- The application interprets the values of the query or path parameters correctly.
- Help figure out which log lines are associated with a particular API request. Since there can be several simultaneous endpoint calls, this is a first attempt at grouping logs by specific calls.

#### API payload log output

As mentioned at the start of this section, by setting the verbosity level to `debug`, the Store logs the body payload data for both the request and response of an API call.

The `debug` verbosity level, instead of the `default`, is used to display this information instead of `default` because:

- Body payloads can be huge, containing full CycloneDX and SBOM information. Moving the payload information at this level helps keep the production log output to a reasonable size.
- Some information in these payloads may be sensitive, and the user may not want them exposed in production environment logs.

## SQL Query log output

Some Store logs display the executed SQL query commands when you set the verbosity level to `trace` or a failed SQL call occurs.



#### Note

Some information in these SQL Query trace logs might be sensitive, and the user might not want them exposed in production environment logs.

## Format

When the Store display SQL query logs, it uses the following format:

```
{ "level": "info", "ts": "2022-05-27T15:37:26.186960324Z", "logger": "MetadataStore.gorm", "msg": "sql call", "hostname": "metadata-store-app-c7c8648f7-8dmd1", "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. |

## Connect to the PostgreSQL database

You can use a PostgreSQL database with Supply Chain Security Tools - Store. To connect to the PostgreSQL database, you need the following values:

- database name
- user name

- password
- database host
- database port
- database CA certificate

Connect to the PostgreSQL database:

1. Obtain the database name, user name, password, and CA certificate. Run:

```
db_name=$(kubectl get secret postgres-db-secret -n metadata-store -o json | jq
-r '.data.POSTGRES_DB' | base64 -d)
db_username=$(kubectl get secret postgres-db-secret -n metadata-store -o json |
jq -r '.data.POSTGRES_USER' | base64 -d)
db_password=$(kubectl get secret postgres-db-secret -n metadata-store -o json |
jq -r '.data.POSTGRES_PASSWORD' | base64 -d)

db_ca_dir=$(mktemp -d -t ca-cert-XXXX)
db_ca_path="$db_ca_dir/ca.crt"
kubectl get secrets postgres-db-tls-cert -n metadata-store -o json | jq -r '.da
ta."ca.crt"' | base64 -d > $db_ca_path
```

If the password was auto-generated, the `password` command returns an empty string. Run:

```
db_password=$(kubectl get secret postgres-db-password -n metadata-store -o json
| jq -r '.data.DB_PASSWORD' | base64 -d)
```

2. In a separate terminal, run:

```
kubectl port-forward service/metadata-store-db 5432:5432 -n metadata-store
```

3. Set the database host and port values on the first terminal:

```
db_host="localhost"
db_port=5432
```

4. To port forward to a different local port number, use the following command template:

```
kubectl port-forward service/metadata-store-db <LOCAL_PORT>:5432 -n metadata-st
ore
```

Where `LOCAL-PORT` is the port number for the database you want to use.

You can now connect to the database and make queries. For example:

```
psql "host=$db_host port=$db_port user=$db_username dbname=$db_name sslmode=verify-ca
sslrootcert=$db_ca_path" -c "SELECT * FROM images"
```

You can use GUI clients such as [Postico](#) or [DBeaver](#) to interact with the database.

## Troubleshooting Supply Chain Security Tools - Store

This topic contains ways you can troubleshoot known issues for Supply Chain Security Tools (SCST) - Store.

### Querying by `insight source` returns zero CVEs even though there are CVEs in the source scan

#### Symptom

When attempting to look up CVE and affected packages, querying `insight source get` (or other `insight source` commands) might return zero results due to supply chain configuration and repository URL.

## Solution

You might have to include different combinations of `--repo`, `--org`, `--commit` due to how the scan-controller populates the software bill of materials (SBOM). For more information see [Query vulnerabilities, images, and packages](#).

## 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 these steps to erase the data on the volume:

1. Deploy metadata-store app by using `kapp`.
2. Verify that the `metadata-store-db-*` pod fails.
3. Run:

```
kubectl exec -it metadata-store-db-<some-id> -n metadata-store /bin/bash
```

Where `<some-id>` is the ID generated by Kubernetes and appended to the pod name.

4. Run `rm -rf /var/lib/postgresql/data/*` to delete all database data.

Where `/var/lib/postgresql/data/*` is the path found in `postgres-db-deployment.yaml`.

5. Delete the `metadata-store` app by using `kapp`.
6. Deploy the `metadata-store` app by using `kapp`.

## Missing persistent volume

### Symptom

After SCST - Store is deployed, `metadata-store-db` pod might fail for missing volume while `postgres-db-pv-claim` pvc is in `PENDING` state.

This is because the cluster where SCST - Store is deployed does not have `storageclass` defined. `storageclass`'s provisioner is responsible for creating the persistent volume after `metadata-store-db` attaches `postgres-db-pv-claim`.

## Solution

1. Verify that your cluster has `storageclass` by running `kubectl get storageclass`.
2. Create a `storageclass` in your cluster before deploying SCST - Store. For example:

```
# This is the storageclass that Kind uses
kubectl apply -f https://raw.githubusercontent.com/rancher/local-path-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"}}}'
```

## Builds fail due to volume errors on EKS running Kubernetes v1.23

### Symptom

When installing SCST - Store on or upgrading an existing EKS cluster to Kubernetes v1.23, the database pod shows:

```
running PreBind plugin "VolumeBinding": binding volumes: provisioning failed for PVC
"postgres-db-pv-claim"
```

### Explanation

This is due to the [CSIMigrationAWS in this Kubernetes version](#) which requires users to install the [Amazon Elastic Block Store \(EBS\) CSI Driver](#) to use EBS volumes.

SCST - Store uses the default storage class which uses EBS volumes by default on EKS.

### Solution

Follow the AWS documentation to install the [Amazon EBS CSI Driver](#) before installing SCST - Store or before upgrading to Kubernetes v1.23.

## Certificate Expiries

### Symptom

The Insight CLI or the Scan Controller fails to connect to SCST - Store.

The logs of the metadata-store-app pod show the following error:

```
$ kubectl logs deployment/metadata-store-app -c metadata-store-app -n metadata-store
...
2022/09/12 21:22:07 http: TLS handshake error from 127.0.0.1:35678: write tcp 127.0.0.1:9443->127.0.0.1:35678: write: broken pipe
...
```

or

The logs of metadata-store-db show the following error:

```
$ kubectl logs statefulset/metadata-store-db -n metadata-store
...
2022-07-20 20:02:51.206 UTC [1] LOG: database system is ready to accept connections
2022-09-19 18:05:26.576 UTC [13097] LOG: could not accept SSL connection: sslv3 alert
```

```
bad certificate
...
```

## Explanation

cert-manager rotates the certificates, but the metadata-store and the PostgreSQL db are unaware of the change, and are using the old certificates.

## Solution

If you see **TLS handshake error** in the metadata-store-app logs, delete the metadata-store-app pod and wait for it to come back up.

```
kubectl delete pod metadata-store-app-xxxx -n metadata-store
```

If you see **could not accept SSL connection** in the metadata-store-db logs, delete the metadata-store-db pod and wait for it to come back up.

```
kubectl delete pod metadata-store-db-0 -n metadata-store
```

## Troubleshooting errors from Tanzu Application Platform GUI related to SCST - Store

Different Tanzu Application Platform GUI plug-ins use SCST - Store to display information about vulnerabilities and packages. Some errors visible in Tanzu Application Platform GUI are related to this connection.

### An error occurred while loading data from the Metadata Store

#### Symptom

In the Supply Chain Choreographer plug-in, you see the error message **An error occurred while loading data from the Metadata Store**.

tanzu-java-web-app-scan2 Errors: 1

Supply Chain: source-test-scan-to-url

Supply Chain Cluster: tkg-build-airgap-fuji

Delivery Cluster: tkg-run-airgap-fuji-config

**Stage Detail: Image Scanner**  
2 hours ago

|                 |                                                                          |               |                                      |
|-----------------|--------------------------------------------------------------------------|---------------|--------------------------------------|
| <b>Overview</b> |                                                                          | <b>Policy</b> |                                      |
| Registry        | harbor-airgap.dapdaws.net                                                | Name          | scan-policy                          |
| Image           | harbor-airgap.dapdaws.net/tap/workloads/tanzu-java-web-app-scan2-my-apps | UID           | 07e2e602-3c2b-4ad5-a0b4-e7a13ebb2158 |
| Digest          | sha256:81c064e7bb23d30ff66840c0c6474a45de5c1ef58d219ee6d12f17a6ecc61ed   | Generation    | 1                                    |
| Scan Template   | tanzu-java-web-app-scan2                                                 | Details       | No Violations Found                  |
| UID             | 35ba48d6-1d12-419b-84e1-217be6bf296                                      |               |                                      |
| Generation      | 1                                                                        |               |                                      |

ⓘ An error occurred while loading data from the Metadata Store.

| CVE ID | Severity | Package | Version | Description |
|--------|----------|---------|---------|-------------|
|        |          |         |         |             |

#### Cause

There are multiple potential causes. The most common cause is `tap-values.yaml` missing the configuration that enables Tanzu Application Platform GUI to communicate with Supply Chain Security Tools - Store.

### Solution

See [Supply Chain Choreographer - Enable CVE scan results](#) for the necessary configuration to add to `tap-values.yaml`. After adding the configuration, update your Tanzu Application Platform deployment or Tanzu Application Platform GUI deployment with the new values.

## Troubleshoot upgrading Supply Chain Security Tools - Store

This topic describes how you can troubleshoot upgrading issues Supply Chain Security Tools (SCST) - Store.

### Database deployment does not exist

To prevent issues with the metadata store database, such as the ones described in this topic, the database deployment is `StatefulSet` in

- Tanzu Application Platform v1.2 and later
- Metadata Store v1.1 and later

If you have scripts searching for a `metadata-store-db` deployment, edit the scripts to instead search for `StatefulSet`.

### Invalid checkpoint record

When using Tanzu to upgrade to a new version of the store, there is occasionally data corruption. Here is an example of how this shows up in the log:

```
PostgreSQL Database directory appears to contain a database; Skipping initialization

2022-01-21 21:53:38.799 UTC [1] LOG:  starting PostgreSQL 13.5 (Ubuntu 13.5-1.pgdg18.04+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 16: Aborted
2022-01-21 21:53:39.496 UTC [1] LOG:  aborting startup due to startup process failure
2022-01-21 21:53:39.507 UTC [1] LOG:  database system is shut down
```

The log shows a database pod in a failure loop. For steps to fix the issue so that the upgrade can proceed, see the [SysOpsPro documentation](#).

### Upgraded pod hanging

Because the default access mode in the PVC is `ReadWriteOnce`, if you are deploying in an environment with multiple nodes then each pod might be on a different node. This causes the upgraded pod to spin up but then get stuck initializing because the original pod does not stop. To resolve this issue, find and delete the original pod so that the new pod can attach to the persistent volume:

1. Discover the name of the app pod that is not in a pending state by running:

```
kubectl get pods -n metadata-store
```

2. Delete the pod by running:

```
kubectl delete pod METADATA-STORE-APP-POD-NAME -n metadata-store
```

## Failover, redundancy, and backups for Supply Chain Security Tools - Store

This topic describes how you can configure and use failover, redundancy, and backups for Supply Chain Security Tools (SCST) - Store.

### API Server

By default the API server has 1 replica. If the pod fails, the single instance restarts by normal Kubernetes behavior, but there is downtime. If you upgrade, some downtime is possible.

You can configure the number of replicas by using the `app_replicas` text box in the `scst-store-values.yaml` file.

### Database

By default, the database has one replica, and restarts with some downtime if it fails.



#### Caution

Although you can configure `db_replicas` in `scst-store-values.yaml`, this is discouraged because `db_replicas` is still experimental. The default internal database is not for production use. For production deployments, use an external database.

- [Use external postgres database](#)
- [AWS RDS postgres configuration](#)

For the default PostgreSQL database deployment, with `deploy_internal_db` set to `true`, you can use [Velero](#) as the backup method. For information about using [Velero](#) as the backup method, see [Backups](#).

## Custom certificate configuration for Supply Chain Security Tools - Store

This topic describes how you can configure the following certificates for Supply Chain Security Tools (SCST) - Store:

1. Default configuration
2. Custom certificate



## Default configuration

By default, SCST - Store creates a self-signed certificate and TLS communication is automatically enabled.

If [Ingress support](#) is enabled, SCST - Store installation creates an HTTPProxy entry with host routing by using the qualified name `metadata-store.<ingress_domain>`. For example, `metadata-store.example.com`. The created route supports HTTPS communication using the self-signed certificate with the same subject [Alternative Name](#).

## (Optional) Setting up custom ingress TLS certificate

(Optional) Users can configure TLS to use a custom certificate. To do that:

1. Place the certificates in the secret.
2. Edit the `tap-values.yaml` to use this secret.

### Place the certificates in secret

1. Create the certificate secret before deploying SCST - Store.
2. Create a Kubernetes object with kind `Secret` and type `kubernetes.io/tls`.

### Update `tap-values.yaml`

1. In the `tap-values.yaml` file, you can configure the metadata store to use the `namespace` and `secretName` from the secret created in the last step.

```
metadata_store:
  tls:
    namespace: "namespace"
    secretName: "secretName"
```

Where:

- `namespace` is the targeted namespace for secret consumption by the HTTPProxy.
- `secretName` is the name of secret for consumption by the HTTPProxy.

## Additional resources

- [Ingress support](#)
- [TLS configuration](#)

## TLS configuration for Supply Chain Security Tools - Store

This topic describes how you can configure TLS for Supply Chain Security Tools (SCST) - Store.



### Important

SCST - Store only supports TLS v1.2.

## Setting up custom ingress TLS ciphers

In the `tap-values.yaml` file, `tls.server.rfcCiphers` are set as shown in the following YAML:

```

metadata_store:
  tls:
    server:
      rfcCiphers:
        - TLS_AES_128_GCM_SHA256
        - TLS_AES_256_GCM_SHA384
        - TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
        - TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
        - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
        - TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

```

Where `tls.server.rfcCiphers` is a list of cipher suites for the server. Values are from the [Go TLS package constants](#). If you omit values, the default Go cipher suites are used. These are the default values:

- `TLS_AES_128_GCM_SHA256`
- `TLS_AES_256_GCM_SHA384`
- `TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256`
- `TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384`
- `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`
- `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`

## Example custom TLS settings

The following is a complete example of TLS configuration:

```

metadata_store:
  tls:
    namespace: NAMESPACE
    secretName: SECRET-NAME
    server:
      rfcCiphers:
        - TLS_AES_128_GCM_SHA256
        - TLS_AES_256_GCM_SHA384
        - TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
        - TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
        - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
        - TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

```

Where:

- `NAMESPACE` is the name of the namespace you want to configure TLS with.
- `SECRET-NAME` is the name of the secret you want to configure TLS with.

## Additional resources

- [Custom certificate configuration](#)
- [Ingress support](#)

## Certificate rotation for Supply Chain Security Tools - Store

This topic describes how you can rotate TLS certificates for Supply Chain Security Tools (SCST) - Store.

## Certificates

By default, the `use_cert_manager` setting is set to `"true"`. When the setting `use_cert_manager` is `"true"` the Store uses `cert-manager` to generate a CA certificate, an API certificate, and a database Certificate.

To see these certificates:

```
$ kubectl get certificate -n metadata-store
NAME                READY   SECRET                AGE
app-tls-ca-cert     True    app-tls-ca-cert       38d
app-tls-cert        True    app-tls-cert          38d
postgres-db-tls-cert True    postgres-db-tls-cert  38d
```

The earlier certificates are automatically rotated by `cert-manager`.

The Store can run these certificates automatically once `cert-manager` rotates them.

If the environment is a [multi-cluster](#) setup, the operator must manually copy over the new CA certificate to the build cluster.

## Certificate duration setting

In the `tap-values.yaml` file, `api_cert_duration`, `api_cert_renew_before`, `ca_cert_duration`, and `ca_cert_renew_before` are configurable as shown in the following YAML:

```
metadata_store:
  ca_cert_duration: CA-DURATION
  ca_cert_renew_before: CA-RENEW
  api_cert_duration: API-DURATION
  api_cert_renew_before: API-RENEW
```

Where:

- `CA-DURATION` is the duration that the ca certificate is valid for. Must be given in `h`, `m`, or `s`. Default value is 8760h.
- `CA-RENEW` is how long before the expiry of the ca certificate is renewed. Must be given in `h`, `m`, or `s`. Default value is 1h.
- `API-DURATION` is the duration that the API certificate is valid for. Must be given in `h`, `m`, or `s`. Default value is 2160h.
- `API-RENEW` is how long before the expiry of the API certificate is renewed. Must be given in `h`, `m`, or `s`. Default value is 24h.



### Important

The `*_cert_duration` and the corresponding `*_renew_before` settings must not be close. For more information, see the [cert-manager documentation](#). This can lead to a renewal loop. The `*_cert_duration` must be greater than the corresponding `*_renew_before`. The earlier settings only take effect when `use_cert_manager` is `"true"`. If the `use_cert_manager` is not set, it defaults to `"true"`.

## Ingress support for Supply Chain Security Tools - Store

This topic describes how to configure ingress for Supply Chain Security Tools (SCST) - Store.

## Ingress configuration

Supply Chain Security Tools (SCST) - Store has ingress support by using Contour's HTTPProxy resources. To enable ingress support, a Contour installation must be available in the cluster.

To change ingress configuration, edit your `tap-values.yaml` when you install a Tanzu Application Platform profile. When you configure the `shared.ingress_domain` property, SCST - Store automatically uses that setting.

Alternatively, you can customize SCST - Store's configuration under the `metadata_store` property. Under `metadata_store`, there are two values to configure the proxy:

- `ingress_enabled`
- `ingress_domain`

This is an example snippet in a `tap-values.yaml`:

```
...
metadata_store:
  ingress_enabled: "true"
  ingress_domain: "example.com"
  app_service_type: "ClusterIP" # Defaults to `LoadBalancer`. If ingress is enabled then this must be set to `ClusterIP`.
...
```

SCST - Store installation creates an HTTPProxy entry with host routing by using the qualified name `METADATA-STORE.INGRESS-DOMAIN`. For example, `metadata-store.example.com`. The route supports HTTPS communication using a certificate. By default, a self-signed certificate is used with the same subject `alternative name`. For more information, see [Custom certificate configuration](#).

Contour and DNS setup are not part of SCST - Store installation. Access to SCST - Store using Contour depends on the correct configuration of these two components.

Make the proper DNS record available to clients to resolve `metadata-store` and set `ingress_domain` to Envoy service's external IP address.

DNS setup example:

```
$ kubectl describe svc envoy -n tanzu-system-ingress
> ...
Type:                LoadBalancer
...
LoadBalancer Ingress: 100.2.3.4
...
Port:                https 443/TCP
...

$ nslookup metadata-store.example.com
> Server: 8.8.8.8
Address: 8.8.8.8#53

Non-authoritative answer:
Name: metadata-store.example.com
Address: 100.2.3.4

$ curl https://metadata-store.example.com/api/health -k -v
> ...
< HTTP/2 200
...
```



#### Note

The preceding `curl` example uses the not secure `-k` flag to skip TLS verification because the Store installs a self-signed certificate. The following section shows how

to access the CA certificate to enable TLS verification for HTTP clients.

## Get the TLS CA certificate

To get SCST - Store's TLS CA certificate, use `kubectl get secret`. In this example, you save the certificate for the environment variable to a file.

```
kubectl get secret CERT-NAME -n metadata-store -o json | jq -r '.data."ca.crt"' | base64 -d > OUTPUT-FILE
```

Where:

- `CERT-NAME` is the name of the certificate. This must be `ingress-cert` if no custom certificate is used.
- `OUTPUT-FILE` is the file you want to create to store the certificate.

For example:

```
$ kubectl get secret ingress-cert -n metadata-store -o json | jq -r '.data."ca.crt"' | base64 -d > insight-ca.crt
$ cat insight-ca.crt
```

## Additional Resources

- [Custom certificate configuration](#)
- [TLS configuration](#)
- [Certificate rotation](#)
- [Configure target endpoint and certificate](#)

## Use your LoadBalancer with Supply Chain Security Tools - Store

This topic describes how to use your LoadBalancer with Supply Chain Security Tools (SCST) - Store.

### Configure LoadBalancer



#### Note

`LoadBalancer` is not the recommended service type. Consider the recommended configuration of enabling [Ingress](#).

To configure a `LoadBalancer`:

1. Edit `/etc/hosts/` to use the external IP address of the `metadata-store-app` service.

```
METADATA_STORE_IP=$(kubectl get service/metadata-store-app --namespace metadata-store -o jsonpath="{.status.loadBalancer.ingress[0].ip}")
METADATA_STORE_PORT=$(kubectl get service/metadata-store-app --namespace metadata-store -o jsonpath="{.spec.ports[0].port}")
METADATA_STORE_DOMAIN="metadata-store-app.metadata-store.svc.cluster.local"

# Delete any previously added entry
sudo sed -i '' "/$METADATA_STORE_DOMAIN/d" /etc/hosts
```

```
echo "$METADATA_STORE_IP $METADATA_STORE_DOMAIN" | sudo tee -a /etc/hosts > /dev/null
```



### Note

On EKS, you must get the IP address for the LoadBalancer. Find the IP address by running something similar to the following: `dig RANDOM-SHA.us-east-2.elb.amazonaws.com`. Where `RANDOM-SHA` is the EXTERNAL-IP received for the LoadBalancer.

2. Select one of the IP addresses returned from the `dig` command and write it to the `/etc/hosts` file.

## Port forwarding

If you want to use port forwarding instead of the external IP address from the `LoadBalancer`, follow these steps:

Configure port forwarding for the service so the insight plug-in can access SCST - Store. Run:

```
kubectl port-forward service/metadata-store-app 8443:8443 -n metadata-store
```

**Note:** You must run the port forwarding command in a separate terminal window, or run the command in the background: `kubectl port-forward service/metadata-store-app 8443:8443 -n metadata-store &`

## Edit your `/etc/hosts` file for Port Forwarding

Use the following script to add a new local entry to `/etc/hosts`:

```
METADATA_STORE_PORT=$(kubectl get service/metadata-store-app --namespace metadata-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 's/"$METADATA_STORE_DOMAIN/d" /etc/hosts'

echo "127.0.0.1 $METADATA_STORE_DOMAIN" | sudo tee -a /etc/hosts > /dev/null
```

## Configure the Insight plug-in

Because you deployed Supply Chain Security Tools (SCST) - Store without using Ingress, you must use the Certificate resource `app-tls-cert` for HTTPS communication.

To get the CA Certificate:

```
kubectl get secret app-tls-cert -n metadata-store -o json | jq -r '.data."ca.crt"' | base64 -d > insight-ca.crt
```

Set the target by running:

```
tanzu insight config set-target https://$METADATA_STORE_DOMAIN:$METADATA_STORE_PORT --ca-cert insight-ca.crt
```



### Important

The `tanzu insight config set-target` does not initiate a test connection. Use `tanzu insight health` to test connecting using the configured endpoint and CA certificate. Neither commands test whether the access token is correct. For that you must use the plug-in to [add data](#) and [query data](#).

## Use your NodePort with Supply Chain Security Tools - Store

This topic describes how you can use your NodePort with Supply Chain Security Tools (SCST) - Store.

### Overview



#### Note

The recommended service type is [Ingress](#). NodePort is only recommended when the cluster does not support [Ingress](#) or the cluster does not support the [LoadBalancer](#) service type. [NodePort](#) is not supported for a multicluster setup, as certificates cannot be modified.

You must use port forwarding when using the [NodePort](#) configuration.

Configure port forwarding for the service so the insight plug-in can access SCST - Store. Run:

```
kubectl port-forward service/metadata-store-app 8443:8443 -n metadata-store
```

**Note:** You must run the port forwarding command in a separate terminal window, or run the command in the background: `kubectl port-forward service/metadata-store-app 8443:8443 -n metadata-store &`

### Edit your `/etc/hosts` file for Port Forwarding

Use the following script to add a new local entry to `/etc/hosts`:

```
METADATA_STORE_PORT=$(kubectl get service/metadata-store-app --namespace metadata-store -o jsonpath="{.spec.ports[0].port}")
METADATA_STORE_DOMAIN="metadata-store-app.metadata-store.svc.cluster.local"

# delete any previously added entry
sudo sed -i '' "$METADATA_STORE_DOMAIN/d" /etc/hosts

echo "127.0.0.1 $METADATA_STORE_DOMAIN" | sudo tee -a /etc/hosts > /dev/null
```

### Configure the Insight plug-in

Because you deployed Supply Chain Security Tools (SCST) - Store without using Ingress, you must use the Certificate resource `app-tls-cert` for HTTPS communication.

To get the CA Certificate:

```
kubectl get secret app-tls-cert -n metadata-store -o json | jq -r '.data."ca.crt"' | base64 -d > insight-ca.crt
```

Set the target by running:

```
tanzu insight config set-target https://$METADATA_STORE_DOMAIN:$METADATA_STORE_PORT --
ca-cert insight-ca.crt
```



### Important

The `tanzu insight config set-target` does not initiate a test connection. Use `tanzu insight health` to test connecting using the configured endpoint and CA certificate. Neither commands test whether the access token is correct. For that you must use the plug-in to [add data](#) and [query data](#).

## Multicluster setup for Supply Chain Security Tools - Store

This topic describes how you can deploy Supply Chain Security Tools (SCST) - Store in a multicluster setup, including installing multiple profiles such as, View, Build, Run, and Iterate.

### Overview

SCST - Store is deployed with the View profile. After installing the View profile, but before installing the Build profile, you must add configuration for SCST - Store to the Kubernetes cluster where you intend to install the Build profile. This topic explains how to add configuration which allows components in the Build cluster to communicate with SCST - Store in the View cluster.



### Note

If you already deployed the Build profile, you can follow this procedure. However, in the [Install Build profile](#) step, instead of deploying the Build profile again, update your deployment using `tanzu package installed update`.

## Prerequisites

You must install the View profile. See [Install View profile](#).

## Procedure summary

1. Copy SCST - Store CA certificate from the View cluster.
2. Copy SCST - Store authentication token from the View cluster.
3. Apply the CA certificate and authentication token to the Kubernetes cluster where you intend to install the Build profile.
4. Install the Build profile.

## Copy SCST - Store CA certificate from View cluster

With your `kubectl` targeted at the View cluster, you can view SCST - Store's TLS CA certificate. Run these commands to copy the CA certificate into a file `store_ca.yaml`.

```
CA_CERT=$(kubectl get secret -n metadata-store CERT-NAME -o json | jq -r ".data.\"ca.c
rt\"")
cat <<EOF > store_ca.yaml
---
apiVersion: v1
kind: Secret
```



```

type: Opaque
metadata:
  name: store-ca-cert
  namespace: metadata-store-secrets
data:
  ca.crt: $CA_CERT
EOF

```

Where `CERT-NAME` is the name of the certificate you want to reference in `store_ca.yaml`.

For example:

```

$ CA_CERT=$(kubectl get secret -n metadata-store ingress-cert -o json | jq -r ".data.a.\"ca.crt\"")
$ cat <<EOF > store_ca.yaml
---
apiVersion: v1
kind: Secret
type: Opaque
metadata:
  name: store-ca-cert
  namespace: metadata-store-secrets
data:
  ca.crt: $CA_CERT
EOF

```

## Copy SCST - Store authentication token from the View cluster

Copy the SCST - Store authentication token into an environment variable. You use this environment variable in the next step.

```

AUTH_TOKEN=$(kubectl get secrets metadata-store-read-write-client -n metadata-store -o jsonpath="{.data.token}" | base64 -d)

```

## Apply the CA certificate and authentication token to a new Kubernetes cluster

Before you deploy the Build profile, you must apply the CA certificate and authentication token from the earlier steps. Then the Build profile deployment has access to these values.

To apply the CA certificate and authentication token:

1. With your `kubectl` targeted at the Build cluster, create a namespace for the CA certificate and authentication token.

```

kubectl create ns metadata-store-secrets

```

2. Apply the CA certificate `store_ca.yaml` secret YAML you generated earlier.

```

kubectl apply -f store_ca.yaml

```

3. Create a secret to store the access token. This uses the `AUTH_TOKEN` environment variable.

```

kubectl create secret generic store-auth-token \
  --from-literal=auth_token=$AUTH_TOKEN -n metadata-store-secrets

```

The cluster now has a CA certificate named `store-ca-cert` and authentication token named `store-auth-token` in the namespace `metadata-store-secrets`.

## Install Build profile

If you came to this topic from the [Install multicluster Tanzu Application Platform profiles](#) topic after installing the View profile, return to that topic to [install the Build profile](#).

The Build profile `values.yaml` contains configuration that references the secrets in the `metadata-store-secrets` namespace you created in this guide. The names of these secrets are hard coded in the example `values.yaml`.

### More information about how Build profile uses the configuration

The secrets you created are used in the Build profile `values.yaml` to configure the Grype scanner which talks to SCST - Store. After performing a vulnerabilities scan, the Grype scanner sends the results to SCST - Store. Here's a snippet of what the configuration might look like.

```
...
grype:
  namespace: "MY-DEV-NAMESPACE" # (Optional) Defaults to default namespace.
  targetImagePullSecret: "TARGET-REGISTRY-CREDENTIALS-SECRET"
  metadataStore:
    url: METADATA-STORE-URL-ON-VIEW-CLUSTER
    caSecret:
      name: store-ca-cert
      importFromNamespace: metadata-store-secrets
    authSecret:
      name: store-auth-token
      importFromNamespace: metadata-store-secrets
...
```

Where:

- `METADATA-STORE-URL-ON-VIEW-CLUSTER` is the ingress URL of SCST - Store deployed to the View cluster. For example, <https://metadata-store.example.com>. See [Ingress support](#).
- `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the secret that contains the credentials to pull an image from the registry for scanning.
- `MY-DEV-NAMESPACE` is the name of the developer namespace. SCST - Scan deploys the ScanTemplates there. This allows the scanning feature to run in this namespace.

## Configure developer namespaces

After you finish the entire Tanzu Application Platform installation process, you are ready to configure developer namespaces. To prepare developer namespaces, you must export the secrets you created earlier to those namespaces.

### Exporting SCST - Store secrets to a developer namespace in a Tanzu Application Platform multicluster deployment

Export secrets to a developer namespace by creating `SecretExport` resources on the developer namespace. Run the following command to create the `SecretExport` resources. You must have created and populated the `metadata-store-secrets` namespace.

```
cat <<EOF | kubectl apply -f -
---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretExport
metadata:
  name: store-ca-cert
  namespace: metadata-store-secrets
spec:
```

```

  toNamespaces: [DEV-NAMESPACES]
  ---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretExport
metadata:
  name: store-auth-token
  namespace: metadata-store-secrets
spec:
  toNamespaces: [DEV-NAMESPACES]
EOF

```

Where `[DEV-NAMESPACES]` is an array of developer namespaces where the secrets are exported.

## Additional resources

- [Ingress support](#)
- [Custom certificate configuration](#)

## Developer namespace setup for Supply Chain Security Tools - Store

This topic describes how you can set up your developer namespace for Supply Chain Security Tools (SCST) - Store.

### Overview

After you finish the entire Tanzu Application Platform installation process, you are ready to configure the developer namespace. When you configure a developer namespace, you must export the Supply Chain Security Tools (SCST) - Store CA certificate and authentication token to the namespace. This enables SCST - Scan to find the credentials to send scan results to SCST - Store.

There are two ways to deploy Tanzu Application Platform:

- Single cluster, which entails using the Tanzu Application Platform values file
- Multicluster, which entails using `SecretExport`

### Single cluster - Using the Tanzu Application Platform values file

When deploy the Tanzu Application Platform Full or Build profile, edit the `tap-values.yaml` file you used to deploy Tanzu Application Platform.

```

metadata_store:
  ns_for_export_app_cert: "DEV-NAMESPACE"

```

Where `DEV-NAMESPACE` is the name of the developer namespace.

The `ns_for_export_app_cert` supports one namespace at a time. If you have multiple namespaces you can replace this value with a `"*"`, but this exports the CA to all namespaces. Consider whether this increased visibility presents a risk.

```

metadata_store:
  ns_for_export_app_cert: "*"

```

Update Tanzu Application Platform to apply the changes by running:

```
$ tanzu package installed update tap -f tap-values.yaml -n tap-install
```

## Multicluster - Using `SecretExport`

In a multicluster deployment, follow the steps in [Multicluster setup](#). It describes how to create secrets and export secrets to developer namespaces.

## Next steps

If you arrived in this topic from [Setting up the Out of the Box Supply Chain with testing and scanning](#), return to that topic and continue with the instructions.

## Retrieve access tokens for Supply Chain Security Tools - Store

This topic describes how you can retrieve access tokens for Supply Chain Security Tools (SCST) - Store.

## Overview

When you install Tanzu Application Platform, the Supply Chain Security Tools (SCST) - Store deployment automatically includes a read-write service account. This service account is bound to the `metadata-store-read-write` role.

There are two types of SCST - Store service accounts:

1. Read-write service account - full access to the `POST` and `GET` API requests
2. Read-only service account - can only use `GET` API requests

This topic shows how to retrieve the access token for these service accounts.

## Retrieving the read-write access token

To retrieve the read-write access token, run:

```
kubectl get secrets metadata-store-read-write-client -n metadata-store -o jsonpath="{.data.token}" | base64 -d
```

## Retrieving the read-only access token

In order to retrieve the read-only access token, you must first have a read-only service account. See [Create read-only service account](#).

To retrieve the read-only access token, run:

```
kubectl get secrets metadata-store-read-client -n metadata-store -o jsonpath="{.data.token}" | base64 -d
```

## Using an access token

The access token is a Bearer token used in the http request header `Authorization`. For example, `Authorization: Bearer eyJhbGciOiJSUzI1NiIsImtpZCI6IjhmV0...`

## Additional Resources

- [Create service accounts](#)
- [Create a service account with a custom cluster role](#)

## Retrieve and create service accounts for Supply Chain Security Tools - Store

This topic explains how you can create service accounts for Supply Chain Security Tools (SCST) - Store.

### Overview

When you install Tanzu Application Platform, the Supply Chain Security Tools (SCST) - Store deployment automatically includes a read-write service account. This service account is bound to the `metadata-store-read-write` role.

There are two types of SCST - Store service accounts:

1. Read-write service account - full access to the `POST` and `GET` API requests
2. Read-only service account - can only use `GET` API requests

### Create read-write service account

When you install Tanzu Application Platform, the SCST - Store deployment automatically includes a read-write service account. This service account is already bound to the `metadata-store-read-write` role.

To create an additional read-write service account, run the following command. The command creates a service account called `metadata-store-read-write-client`, depending on the Kubernetes version:

```
kubectl apply -f - -o yaml << EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: metadata-store-read-write
  namespace: metadata-store
rules:
- resources: ["all"]
  verbs: ["get", "create", "update"]
  apiGroups: [ "metadata-store/v1" ]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: metadata-store-read-write
  namespace: metadata-store
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: metadata-store-read-write
subjects:
- kind: ServiceAccount
  name: metadata-store-read-write-client
  namespace: metadata-store
---
apiVersion: v1
kind: ServiceAccount
```

```

metadata:
  name: metadata-store-read-write-client
  namespace: metadata-store
  annotations:
    kapp.k14s.io/change-group: "metadata-store.apps.tanzu.vmware.com/service-account"
automountServiceAccountToken: false
---
apiVersion: v1
kind: Secret
type: kubernetes.io/service-account-token
metadata:
  name: metadata-store-read-write-client
  namespace: metadata-store
  annotations:
    kapp.k14s.io/change-rule: "upsert after upserting metadata-store.apps.tanzu.vmw
    e.com/service-account"
    kubernetes.io/service-account.name: "metadata-store-read-write-client"
EOF

```



### Note

For Kubernetes v1.24 and later, services account secrets are no longer automatically created. This is why the example adds a `Secret` resource in the earlier YAML.

## Create a read-only service account

You can create a read-only service account with a default cluster role or with a custom cluster role.

### With a default cluster role

During Store installation, the `metadata-store-read-only` cluster role is created by default. This cluster role allows the bound user to have `get` access to all resources. To bind to this cluster role, run the following command depending on the Kubernetes version:

```

kubect1 apply -f - -o yaml << EOF
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: metadata-store-read-only
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: metadata-store-read-only
subjects:
- kind: ServiceAccount
  name: metadata-store-read-client
  namespace: metadata-store
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: metadata-store-read-client
  namespace: metadata-store
  annotations:
    kapp.k14s.io/change-group: "metadata-store.apps.tanzu.vmware.com/service-account"
automountServiceAccountToken: false
---
apiVersion: v1
kind: Secret

```

```

type: kubernetes.io/service-account-token
metadata:
  name: metadata-store-read-client
  namespace: metadata-store
  annotations:
    kapp.k14s.io/change-rule: "upsert after upserting metadata-store.apps.tanzu.vmware.com/service-account"
    kubernetes.io/service-account.name: "metadata-store-read-client"
EOF

```



### Note

For Kubernetes v1.24 and later, services account secrets are no longer automatically created. This is why the example adds a `Secret` resource in the earlier YAML.

## With a custom cluster role

If using the default role is not sufficient, see [Create a service account with a custom cluster role](#).

## Additional Resources

- [Retrieve access tokens](#)
- [Create a service account with a custom cluster role](#)

## Create a service account with a custom cluster role for Supply Chain Security Tools - Store

This topic describes how you can create a service account with a custom cluster role for Supply Chain Security Tools (SCST)- Store.

### Example service account

If you do not want to bind to the default cluster role, create a read-only role in the `metadata-store` namespace with a service account. The following example creates a service account named `metadata-store-read-client`, depending on the Kubernetes version:

```

kubectl apply -f - -o yaml << EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: metadata-store-ro
  namespace: metadata-store
rules:
- resources: ["all"]
  verbs: ["get"]
  apiGroups: [ "metadata-store/v1" ]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: metadata-store-ro
  namespace: metadata-store
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: metadata-store-ro

```

```

subjects:
- kind: ServiceAccount
  name: metadata-store-read-client
  namespace: metadata-store
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: metadata-store-read-client
  namespace: metadata-store
  annotations:
    kapp.k14s.io/change-group: "metadata-store.apps.tanzu.vmware.com/service-account"
automountServiceAccountToken: false
---
apiVersion: v1
kind: Secret
type: kubernetes.io/service-account-token
metadata:
  name: metadata-store-read-client
  namespace: metadata-store
  annotations:
    kapp.k14s.io/change-rule: "upsert after upserting metadata-store.apps.tanzu.vmware.com/service-account"
    kubernetes.io/service-account.name: "metadata-store-read-client"
EOF

```



#### Note

For Kubernetes v1.24 and later, service account secrets are no longer automatically created. This is why the example adds a `Secret` resource in the earlier YAML.

## Additional Resources

- [Retrieve access tokens](#)
- [Create service accounts](#)

## Install Supply Chain Security Tools - Store independent from Tanzu Application Platform profiles

This topic describes how you can install Supply Chain Security Tools (SCST) - Store from the Tanzu Application Platform package repository.



#### Note

Follow the steps in this topic if you do not want to use a profile to install Supply Chain Security Tools - Store. For more information about profiles, see [Components and installation profiles](#).

## Prerequisites

Before installing SCST - Store:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see [Prerequisites](#).
- Install cert-manager on the cluster. See [Install cert-manager](#).



- See [Deployment Details and Configuration](#) to review what resources are deployed. For more information, see the [overview](#).

## Install

To install SCST - Store:

1. To use this deployment, the user must have set up the Kubernetes cluster to provision persistent volumes on demand. Ensure that a default storage class is available in your cluster. Verify whether default storage class is set in your cluster using `kubectl get storageClass`.

```
kubectl get storageClass
```

For example:

```
$ kubectl get storageClass
NAME                                PROVISIONER                RECLAIMPOLICY   VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANSION  AGE
standard (default)    rancher.io/local-path     Delete          WaitForFirstConsumer
er    false                7s
```

2. List version information for the package using `tanzu package available list`.

```
tanzu package available list metadata-store.apps.tanzu.vmware.com --namespace 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 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:

```
$ 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
kube_rbac_proxy_limit_memory    512Mi
string  Memory limit for kube-rbac-proxy container in the metadata-store-app deployment
pg_req_cpu                        1Gi
string  CPU request for postgres container in metadata-store-db deployment
app_req_cpu                        100m
string  CPU request for metadata-store-app container
db_port                        5432
string  The database port to use. This is the port to use when connecting to the database pod.
db_sslmode                       verify-full
string  Determines the security connection between API server and Postgres dat
```

```

abase. This can be set to 'verify-ca' or 'verify-full'
kube_rbac_proxy_limit_cpu      250m
string      CPU limit for kube-rbac-proxy container in the metadata-store-app deployment
ns_for_export_app_cert        scan-link-system
string      The namespace where the "Supply Chain Security Tools for VMware Tanzu - Scan" component is installed in. Certain certificates will be exported to that namespace so that scan reports can be posted to the Metadata Store.
pg_limit_memory                4Gi
string      Memory limit for postgres container in metadata-store-db deployment
tls.namespace
string      The targeted namespace for secret consumption by the HTTPProxy.
api_port                       9443
integer     The internal port for the metadata app api endpoint. This will be used by the kube-rbac-proxy sidecar.
app_limit_cpu                  250m
string      CPU limit for metadata-store-app container
db_conn_max_lifetime          60
integer     Sets the maximum amount of time a database connection may be reused in seconds.
tls.server.minTLSVersion      VersionTLS12
string      Minimum TLS version supported. Value must match version names from https://golang.org/pkg/crypto/tls/#pkg-constants. (default "VersionTLS12")
db_ca_certificate
string      This should only be set in the case when 'deploy_internal_db' is 'false'. Set this to the trusted CA Certificate that signed the Postgres DB TLS Certificate
db_password
string      The database user password. If no value is provided, a 32 character value will be generated.
log_level                      default
string      Sets the log level. This can be set to "minimum", "less", "default", "more", "debug" or "trace". "minimum" currently does not output logs. "less" outputs log configuration options only. "default" and "more" outputs API endpoint access information. "debug" and "trade" outputs extended API endpoint access information (such as body payload) and other debug information.
kubernetes_distribution
string      Kubernetes platform distribution where the metadata-store is being installed on. Accepted values: ["", "openshift"]
pg_limit_cpu                   2Gi
string      CPU limit for postgres container in metadata-store-db deployment
tls.secretName
string      The name of secret for consumption by the HTTPProxy.
tls.server.rfcCiphers          [TLS_AES_128_GCM_SHA256 TLS_AES_256_GCM_SHA384 TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384]
array      List of cipher suites for the server. Values are from tls package constants (https://golang.org/pkg/crypto/tls/#pkg-constants). If omitted, the default Go cipher suites will be used
app_limit_memory               512Mi
string      Memory limit for metadata-store-app container
app_service_type               LoadBalancer
string      The type of service to use for the metadata app service. This can be set to 'Nodeport', 'ClusterIP' or 'LoadBalancer'.
db_replicas                    1
integer     The number of replicas for the metadata-store-db
kube_rbac_proxy_req_cpu        100m
string      CPU request for kube-rbac-proxy container in the metadata-store-app deployment
pg_req_memory                  1Gi
string      Memory request for postgres container in metadata-store-db deployment
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_max_idle_conns          100
integer   Sets the maximum number of database connections from the Metadata Store in the idle connection pool.
db_name                    metadata-store
string    The name of the database to use.
kube_rbac_proxy_req_memory 128Mi
string    Memory request for kube-rbac-proxy container in the metadata-store-app deployment
db_max_open_conns         10
integer   Sets the maximum number of open database connections from the Metadata Store to the database.
deploy_internal_db        true
string    If set to 'true', a postgres deployment will be created. If set to 'false', db_host and db_port should point to an accessible postgres instance. Postgres connections require TLS, so the corresponding db_ca_certification must be provided
ingress_domain
string    Domain to be used by the HTTPProxy ingress object. The "metadata-store" subdomain will be prepended to the value provided. For example: "example.com" would become "metadata-store.example.com". Required if ingress_enabled is true.
use_cert_manager           true
string    Cert manager is required to be installed to use this flag. When true, this creates certificates object to be signed by cert manager for the API server and Postgres database. If false, the certificate object have to be provided by the user.
add_default_rw_service_account true
string    Adds a read-write service account which can be used to obtain access token to use metadata-store CLI
api_host                   localhost
string    The internal hostname for the metadata api endpoint. This will be used by the kube-rbac-proxy sidecar.
auth_proxy_port            8443
integer   The external port address of the of the kube-rbac-proxy sidecar
database_request_storage   10Gi
string    The storage requested of the persistent volume used by Postgres database for storing data.
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.
ingress_enabled            false
string    Contour is required to be installed to use this flag. When true, this creates an HTTPProxy object for the metadata-store. If false, then no ingress is configured.
kubernetes_version
string    Kubernetes platform distribution version where the metadata-store is being installed on. This is required when openshift version is higher than 4.10
priority_class_name
string    If specified, this value is the name of the desired PriorityClass for the metadata-store-db deployment
app_replicas               1
integer   The number of replicas for the metadata-store-app
app_req_memory              128Mi
string    Memory request for metadata-store-app container
auth_proxy_host            0.0.0.0
string    The binding ip address of the kube-rbac-proxy sidecar
storage_class_name
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.

```

4. (Optional) Edit one of the deployment configurations by creating a configuration YAML with the custom configuration values you want. For example, if your environment does not

support `LoadBalancer`, and you want to use `ClusterIP`, then create a `metadata-store-values.yaml` and configure the `app_service_type` property.

```
---
app_service_type: "ClusterIP"
```

See [Deployment details and configuration](#) for more information about configuration options.

For information about ingress and custom domain name support, see [Ingress support](#).

5. Install the package using `tanzu package install`.

```
tanzu package install metadata-store \
  --package-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:

```
$ 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'
```

## Overview of Tanzu Developer Tools for Visual Studio

VMware Tanzu Developer Tools for Visual Studio is the official VMware Tanzu IDE extension for Visual Studio 2022. The extension helps you develop with Tanzu Application Platform and enables you to rapidly iterate on your workloads on supported Kubernetes clusters that have Tanzu Application Platform installed.

This plug-in extends Microsoft Visual Studio 2022 only. It is incompatible with Visual Studio Code and Visual Studio for Mac.



### Note

This extension is in the beta stage of development.

## Extension Features

The extension has the following features:

- **See code updates running on-cluster in seconds:**

With the use of Live Update facilitated by Tilt, deploy your workload once, save changes to the code and then, seconds later, see those changes reflected in the workload running on the cluster. All Live Update output is filtered to its own output pane window within Visual Studio.

- **Debug workloads directly on the cluster:**

Debug your application in a production-like environment by debugging on a Kubernetes cluster of yours that has Tanzu Application Platform. The similarity of an environment to production relies on keeping dependencies updated, among other variables.

- **Deploy a workload to a Kubernetes cluster:**

Deploy your workload straight to your Kubernetes cluster and, after you're finished using it, you can delete it. All the output for deleting a workload is filtered to its own output pane window within Visual Studio.

- **See workloads running on the cluster:**

From the Tanzu Panel, you can see any workload found within the cluster and namespace specified in the current kubectl context.

## Overview of Tanzu Developer Tools for Visual Studio

VMware Tanzu Developer Tools for Visual Studio is the official VMware Tanzu IDE extension for Visual Studio 2022. The extension helps you develop with Tanzu Application Platform and enables you to rapidly iterate on your workloads on supported Kubernetes clusters that have Tanzu Application Platform installed.

This plug-in extends Microsoft Visual Studio 2022 only. It is incompatible with Visual Studio Code and Visual Studio for Mac.



### Note

This extension is in the beta stage of development.

## Extension Features

The extension has the following features:

- **See code updates running on-cluster in seconds:**

With the use of Live Update facilitated by Tilt, deploy your workload once, save changes to the code and then, seconds later, see those changes reflected in the workload running on the cluster. All Live Update output is filtered to its own output pane window within Visual Studio.

- **Debug workloads directly on the cluster:**

Debug your application in a production-like environment by debugging on a Kubernetes cluster of yours that has Tanzu Application Platform. The similarity of an environment to production relies on keeping dependencies updated, among other variables.

- **Deploy a workload to a Kubernetes cluster:**

Deploy your workload straight to your Kubernetes cluster and, after you're finished using it, you can delete it. All the output for deleting a workload is filtered to its own output pane window within Visual Studio.

- **See workloads running on the cluster:**

From the Tanzu Panel, you can see any workload found within the cluster and namespace specified in the current kubectl context.

## Install Tanzu Developer Tools for Visual Studio

This topic tells you how to install VMware Tanzu Developer Tools for Visual Studio.

### Prerequisites

Ensure that you have the following installed on your workstation before installing Tanzu Developer Tools for Visual Studio:

- [Visual Studio 2022 v17.7](#) or later
- [kubectl](#)
- [Tilt v0.30.12](#) or later
- [Tanzu CLI and plug-ins](#)
- [A cluster with the Tanzu Application Platform Full profile or Iterate profile](#)



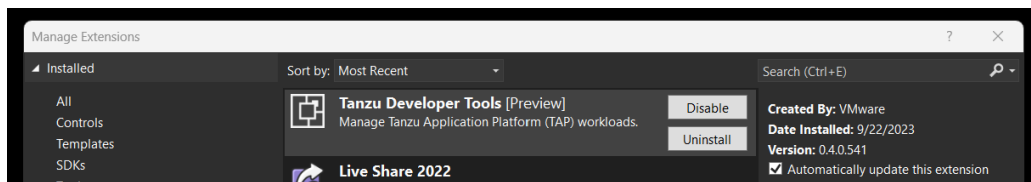
#### Note

If you are an app developer, someone else in your organization might have already set up the Tanzu Application Platform environment.

### Install

To install the extension:

1. Sign in to VMware Tanzu Network and download [Tanzu Developer Tools for Visual Studio](#).
2. Double-click the `.vsix` install file and click through the prompts.
3. Open Visual Studio and verify that the extension is installed and enabled.



### Uninstall

To uninstall:

1. From the top menu, click the **Extensions** tab and then click **Manage Extensions**.
2. Select the **Installed** section and then click the **Uninstall** button for this extension.

### Next steps

Use [Tanzu Developer Tools for Visual Studio](#).

# Use Tanzu Developer Tools for Visual Studio

This topic tells you how to use VMware Tanzu Developer Tools for Visual Studio.



## Note

This extension is in the beta stage of development.

## Configure settings

To configure settings, right-click anywhere in the Solution Explorer and click **Tanzu > Settings...**

- Tanzu CLI is installed in a location in your `PATH` environment variable.
- A valid `workload.yaml` file is in the project. For more information, see the specification for [Tanzu apps workload apply](#).
- You have a functional Tanzu Application Platform environment.
- Your kubeconfig file is modified for Tanzu Application Platform workload deployments.
- You have an image repository to which source code in the local file system can be uploaded before Build Service builds it.

## Workload Actions

The extension enables you to apply, debug, and Live Update your application on a Kubernetes cluster that has Tanzu Application Platform. The developer sandbox experience enables you to Live Update your code, and simultaneously debug the updated code, without deactivating Live Update.

### Apply a workload

To apply a workload, right-click anywhere in the Solution Explorer and click **Tanzu > Apply Workload**. Alternatively, right-click an associated workload in the Tanzu Panel and click **Apply Workload**.

### Delete a workload

To delete a workload, right-click anywhere in the Solution Explorer and click **Tanzu > Delete Workload**. Alternatively, right-click an associated workload in the Tanzu Panel and click **Delete Workload**.

### Start debugging on the cluster

To remote debug a workload, right-click anywhere in the Solution Explorer and click **Tanzu > Debug Workload**. Alternatively, right-click an associated workload in the Tanzu Panel and click **Debug Workload**.



## Caution

Do not use the red square Stop button to end your debugging session. Using the red square Stop button might cause the Tanzu Application Platform workload to fail. Instead, in the top menu click **Debug > Detach All**.

If the name of your running app process (the app DLL process), does not match the name of your .NET project as shown in the Visual Studio Solution Explorer, the remote debugging agent might fail to attach.

## Live Update

See the following sections for how to use Live Update.

### Start Live Update

Ensure that the following Tanzu Settings parameters are set:

- **Local Path**, which is the path on the local file system to a directory of source code to build.
- **Namespace**, which is the namespace that workloads are deployed into. Optional.
- **Source Image**, which is the registry location for publishing local source code. For example, `registry.io/yourapp-source`. It must include both a registry and a project name. The source image parameter is not needed if you configured Local Source Proxy.

To start Live Update, right-click anywhere in the Solution Explorer and click **Tanzu > Start Live Update**. Alternatively, right-click an associated workload in the Tanzu Panel and click **Start Live Update**.

After starting Live Update, local builds changes are synchronized with the container.

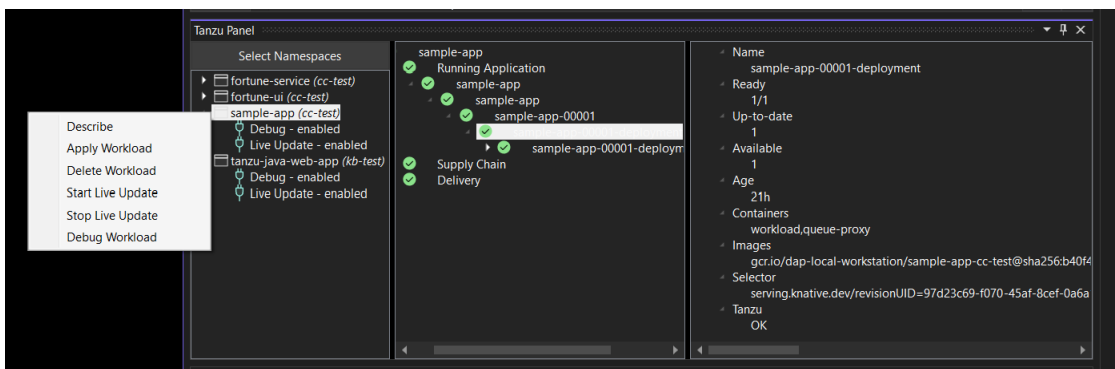
### Stop Live Update

To stop Live Update, right-click anywhere in the Solution Explorer and click **Tanzu > Stop Live Update**. Alternatively, right-click an associated workload in the Tanzu Panel and click **Stop Live Update**.

## Tanzu Workloads panel

### Stop Remote Debug

To view the Tanzu Workloads panel, right-click anywhere in the Solution Explorer and click **Tanzu > View Workloads**.



## Extension logs

The extension creates log entries in two files named `tanzu-dev-tools-{GUID}.log` and `tanzu-language-server-{GUID}.log`. These files are in the directory where Visual Studio Installer installed the extension.

To find the log files from PowerShell, run:



```
dir $Env:LOCALAPPDATA\Microsoft\VisualStudio\*\Extensions\*\Logs\tanzu-*.log
```

To find the log files from CMD, run:

```
dir %LOCALAPPDATA%\Microsoft\VisualStudio\*\Extensions\*\Logs\tanzu-*.log
```

This extension records logs in a `.log` file whose name starts with `tanzu-dev-tools` and ends with a string of numbers representing the date, such as `tanzu-dev-tools20221202.log`. A new log file is created for each day and retained for a maximum of 31 days. These log files are in the installation directory of the `.vsix` file. By default, this is

```
C:\Users\NAME\AppData\Local\Microsoft\VisualStudio\VERSION\Extensions\VMware\Tanzu Developer Tools\VSIX-VERSION
```

Where `NAME`, `VERSION`, and `VSIX-VERSION` are placeholders.

## Troubleshoot Tanzu Developer Tools for Visual Studio

This topic tells you how to troubleshoot issues you encounter with VMware Tanzu Developer Tools for Visual Studio.

### Erroneous `WorkloadNotRunningState` error message

#### Symptom

In v0.1.0 and earlier, the `Tanzu: Delete Workload` command might give the following error message even when a workload is running:

```
Invalid transition DeleteWorkload from state WorkloadNotRunningState
```

#### Solution

Re-apply your workload by running `Tanzu: Workload Apply` Or `Tanzu: Start Live Update`. This realigns the extension's internal state with the proper workload state. The delete operation is enabled again after the extension detects that the workload is running.

### Live Update fails to update remote app

#### Symptom

In v0.1.0 and earlier, the `Tanzu: Start Live Update` command does not update the remote app.

#### Cause

The Tiltfile might be specifying an incorrect local path.

#### Solution

In your Tiltfile, change the lines

```
live_update=[
  sync('./bin', '/workspace')
]
```

to

```
live_update=[
  sync('./bin/Debug/net6.0', '/workspace')
]
```

This copies the correct portion of the local workspace to the remote app. The actual path `bin/Debug/net6.0` might be different depending on your Visual Studio configuration and target.

## Delete workload command fails to delete workload

### Symptom

In v0.1.0 and earlier, the `Tanzu: Delete Workload` command appears to run but does not delete the workload.

### Cause

The workload is running in a namespace other than `default`.

### Solution

Only deploy workloads to the `default` namespace. Alternatively, set the default Kubernetes namespace to the one where your workload is running. To do so, run:

```
kubectl config set-context --current --namespace=NAMESPACE
```

## Live Update does not work with the Jammy ClusterBuilder

### Symptom

Live Update does not work when using the Jammy `ClusterBuilder`.

### Solution

A fix is planned for Tanzu Application Platform v1.5.1.

## Frequent application restarts

### Symptom

When an application is applied from Visual Studio it restarts frequently.

### Cause

An application or environment behavior is triggering the application to restart.

Observed trigger behaviors include:

- The application itself writing logs to the file system in the application directory that Live Update is watching
- Autosave being set to a very high frequency in the IDE configuration

### Solution

Prevent the trigger behavior. Example solutions include:

- Prevent 12-factor applications from writing to the file system.
- Reduce the autosave frequency to once every few minutes.

## Overview of Tanzu Developer Tools for VS Code

VMware Tanzu Developer Tools for Visual Studio Code (VS Code) is the official VMware Tanzu IDE extension for VS Code. The extension helps you develop with Tanzu Application Platform and enables you to rapidly iterate on your workloads on supported Kubernetes clusters that have Tanzu Application Platform installed.

Tanzu Developer Tools for VS Code currently supports VS Code on macOS and Windows OS for Java applications.

### Extension features

The extension has the following features:

- **Deploy applications directly from VS Code:**

Rapidly iterate on your applications on Tanzu Application Platform by deploying them as workloads directly from within VS Code.

- **See code updates running on-cluster in seconds:**

With Live Update (facilitated by Tilt), you can deploy your workload once, save changes to the code and then see those changes reflected within seconds in the workload running on the cluster.

- **Debug workloads directly on the cluster:**

Debug your application in a production-like environment by debugging on your Kubernetes cluster that has Tanzu Application Platform. An environment's similarity to production relies on keeping dependencies and other variables updated.

- **See workloads running on the cluster:**

From the Tanzu Workloads panel you can see any workload found within the cluster and namespace specified in the current kubectl context.

## Overview of Tanzu Developer Tools for VS Code

VMware Tanzu Developer Tools for Visual Studio Code (VS Code) is the official VMware Tanzu IDE extension for VS Code. The extension helps you develop with Tanzu Application Platform and enables you to rapidly iterate on your workloads on supported Kubernetes clusters that have Tanzu Application Platform installed.

Tanzu Developer Tools for VS Code currently supports VS Code on macOS and Windows OS for Java applications.

### Extension features

The extension has the following features:

- **Deploy applications directly from VS Code:**

Rapidly iterate on your applications on Tanzu Application Platform by deploying them as workloads directly from within VS Code.

- **See code updates running on-cluster in seconds:**

With Live Update (facilitated by Tilt), you can deploy your workload once, save changes to the code and then see those changes reflected within seconds in the workload running on the cluster.

- **Debug workloads directly on the cluster:**

Debug your application in a production-like environment by debugging on your Kubernetes cluster that has Tanzu Application Platform. An environment's similarity to production relies on keeping dependencies and other variables updated.

- **See workloads running on the cluster:**

From the Tanzu Workloads panel you can see any workload found within the cluster and namespace specified in the current kubectl context.

## Install Tanzu Developer Tools for your VS Code

This topic tells you how to install VMware Tanzu Developer Tools for Visual Studio Code (VS Code).

### Prerequisites

Before installing the extension, you must have:

- [VS Code](#)
- [kubectl](#)
- [Tilt v0.30.12 or later](#)
- [Tanzu CLI and plug-ins](#)
- [A cluster with the Tanzu Application Platform Full profile or Iterate profile](#)

If you are an app developer, someone else in your organization might have already set up the Tanzu Application Platform environment.

Docker Desktop and local Kubernetes are not prerequisites for using Tanzu Developer Tools for VS Code.

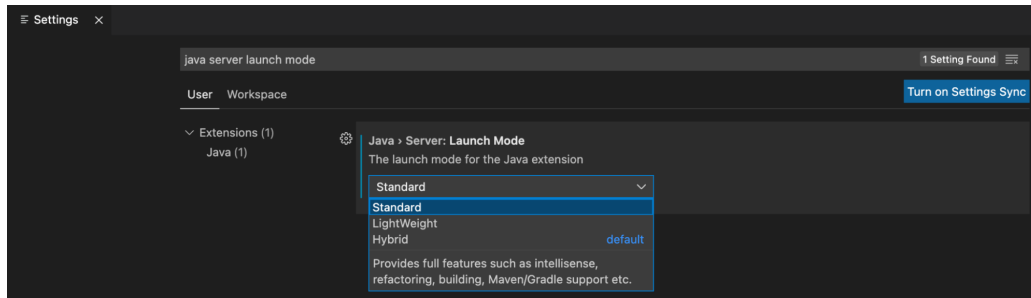
### Install

To install 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:
  - o **Confirm Delete:** This controls whether the extension asks for confirmation when deleting a workload.
  - o **Enable Live Hover:** For more information, see [Integrating Live Hover by using Spring Boot Tools](#). Reload VS Code for this change to take effect.
  - o **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.
  - o **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.
  - o **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](#).

## Get started with Tanzu Developer Tools for VS Code

This topic guides you through getting started with VMware Tanzu Developer Tools for Visual Studio Code (VS Code).

## Prerequisite

[Install VMware Tanzu Developer Tools for Visual Studio Code](#).

## Set up Tanzu Developer Tools

The extension makes use of the following files within your project:

- `workload.yaml`
- `catalog-info.yaml`
- `Tiltfile`
- `.tanzuignore`

You can create these files by using the instructions in this topic, or use the files in the [View an example project](#) section.

There are two ways to create these files:

- Using the code snippets that Tanzu Developer Tools provide, which create templates in empty files that you then fill in with the required information.
- Writing the files manually.

### Create the `workload.yaml` file

`workload.yaml` provides instructions to the Supply Chain Choreographer about how to build and manage a workload.

The extension requires only one `workload.yaml` file per project. `workload.yaml` must be a single-document YAML file, not a multidocument YAML file.

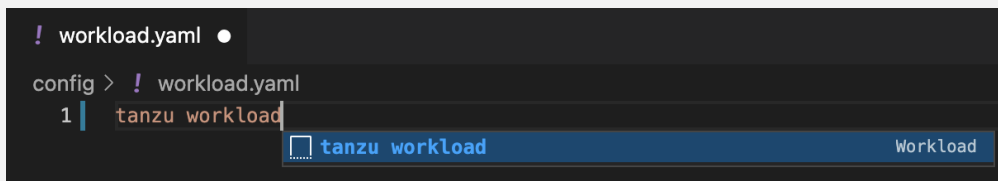
Before beginning to write your `workload.yaml` file, ensure that you know:

- The name of your application. For example, `my app`.
- The workload type of your application. For example, `web`.
- The GitHub source code URL. For example, `github.com/mycompany/myapp`.
- The Git branch of the source code that you intend to use. For example, `main`.

#### Code snippets

To create a `workload.yaml` file by using code snippets:

1. (Optional) Create a directory named `config` in the root directory of your project. For example, `my project/config`.
2. Create a file named `workload.yaml` in the new config directory. For example, `my project/config/workload.yaml`.
3. Open the new `workload.yaml` file in VS Code, enter `tanzu workload` in the file to trigger the code snippets, and either press Enter or left-click the `tanzu workload` text in the drop-down menu.



```
! workload.yaml ●
config > ! workload.yaml
1 | tanzu workload
   | tanzu workload Workload
```

4. Fill in the template by pressing the Tab key.

#### Manual

To create your `workload.yaml` file manually, follow this example:

```
apiVersion: carto.run/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

`catalog-info.yaml` enables the workloads of this project to appear in [Tanzu Application Platform GUI](#).

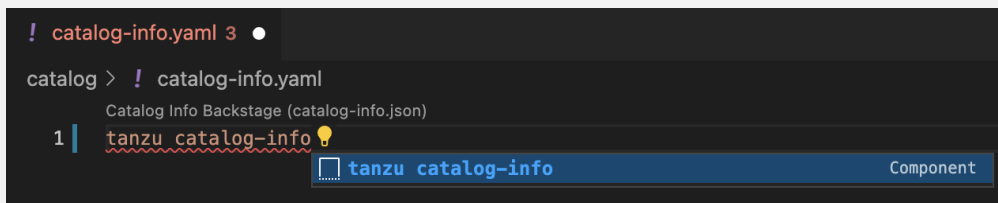
Before beginning to write your `catalog-info.yaml` file, ensure that you:

- 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.

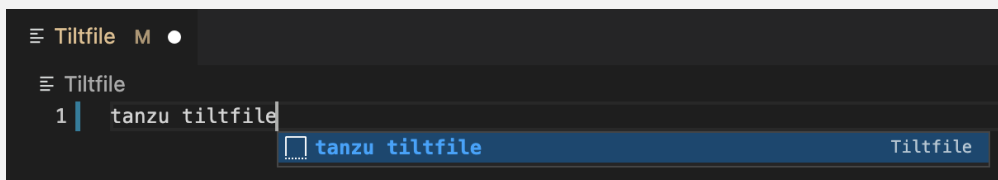
Before beginning to write your Tiltfile file, ensure that you know:

- 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](#).

## Create a `.tanziignore` file

The `.tanziignore` file specifies the file paths to exclude from the source code image. When working with local source code, you can exclude files from the source code to be uploaded within the image. Directories must not end with the system path separator (`/` or `\`). See this [example](#) in GitHub.

## View an example project

Before you begin, you need a container registry for the sample application.

You can view a sample application that demonstrates the necessary configuration files. There are two ways to obtain the sample application:

### Application Accelerator

If your company has configured [Application Accelerator](#), you can obtain the sample application there if it was not removed. To do so:

1. Open Application Accelerator.
2. Search for [Tanzu Java Web App](#) in Application Accelerator.

3. Add the required configuration information and generate the application.
4. Unzip the file and open the project in a VS Code workspace.

### Clone from GitHub

To clone the sample application from GitHub:

1. Run `git clone` to clone the `tanzu-java-web-app` repository from GitHub.
2. Change into the `tanzu-java-web-app` directory.
3. Open the Tiltfile and replace `your-registry.io/project` with your container registry.

## Next steps

[Use Tanzu Developer Tools for VS Code.](#)

## Use Tanzu Developer Tools for VS Code

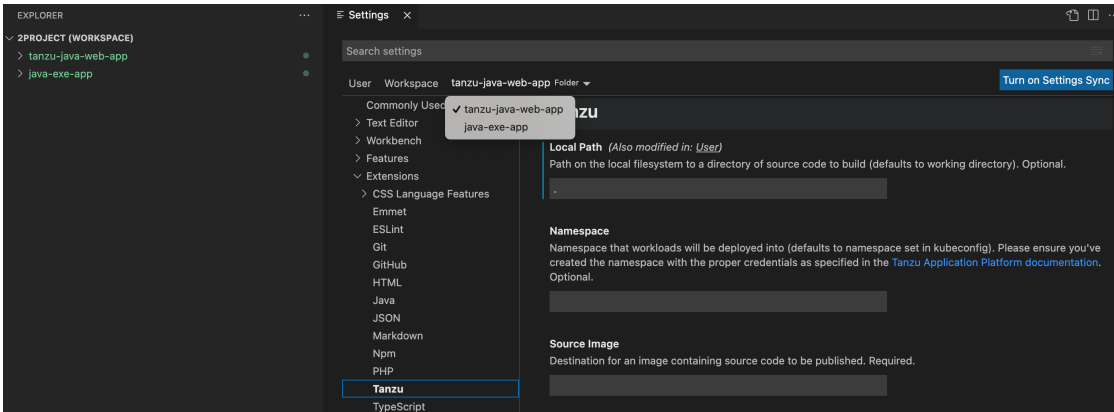
This topic tells you how to use VMware Tanzu Developer Tools for Visual Studio Code (VS Code).

Ensure that the project you want to use the extension with has the required files specified in [Get started with Tanzu Developer Tools for VS Code](#).

The extension requires only one Tiltfile and one `workload.yaml` per project. The `workload.yaml` must be a single-document YAML file, not a multidocument YAML file.

## Configure for multiple projects in the workspace

When working with multiple projects in a single workspace, you can configure the extension settings on a per-project basis by using the drop-down menu in **Settings**.

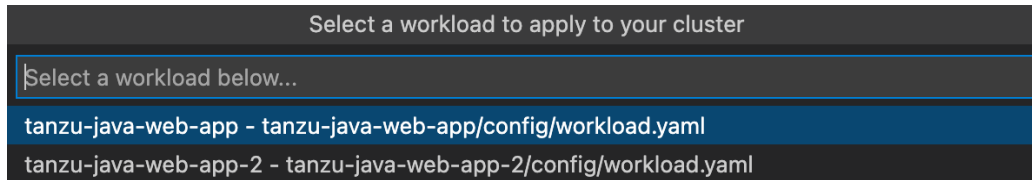


## Apply a workload

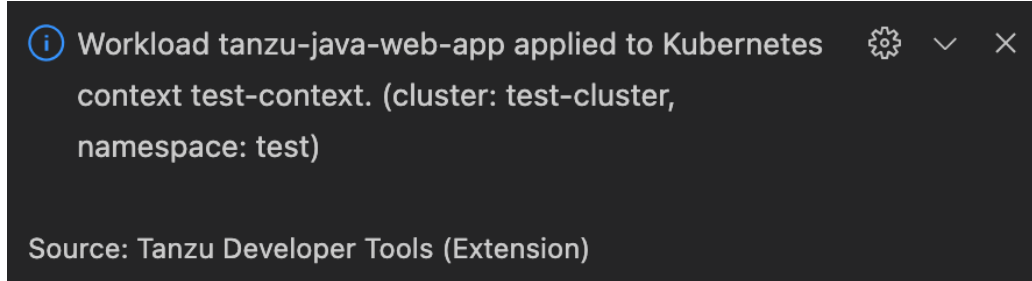
The extension enables you to apply workloads on your Kubernetes cluster that has Tanzu Application Platform.

To apply a workload:

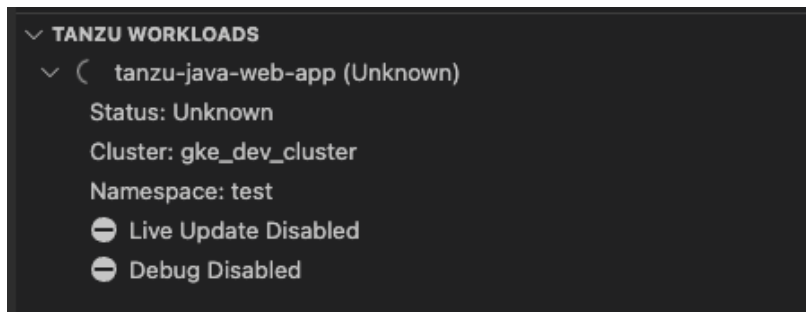
1. Right-click anywhere in the VS Code project explorer or open the Command Palette by pressing `⌘ P` (Ctrl+Shift+P on Windows).
2. Run `Tanzu: Apply Workload`.
3. If there are multiple projects with workloads, select the workload to apply.



A notification appears showing that the workload was applied.



A new workload appears on the Tanzu Workloads panel.



The Workloads panel shows the workloads running in the namespace that is defined in the current Kubernetes context.

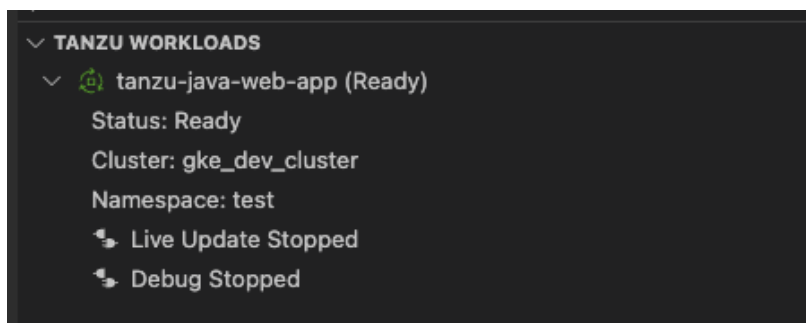
- (Optional) See the context and namespace currently configured by running:

```
kubectl config get-contexts
```

- (Optional) Set a namespace for the current context by running:

```
kubectl config set-context --current --namespace=YOUR-NAMESPACE
```

After the workload is deployed, the status on the Tanzu Workloads panel changes to **Ready**.



## Debugging on the cluster

The extension enables you to debug your application on your Kubernetes cluster that has Tanzu Application Platform.

Debugging requires a `workload.yaml` file in your project. For information about creating a `workload.yaml` file, see [Get Started with Tanzu Developer Tools for VS Code](#).

The developer sandbox experience enables developers to Live Update their code, and simultaneously debug the updated code, without having to deactivate Live Update when debugging.

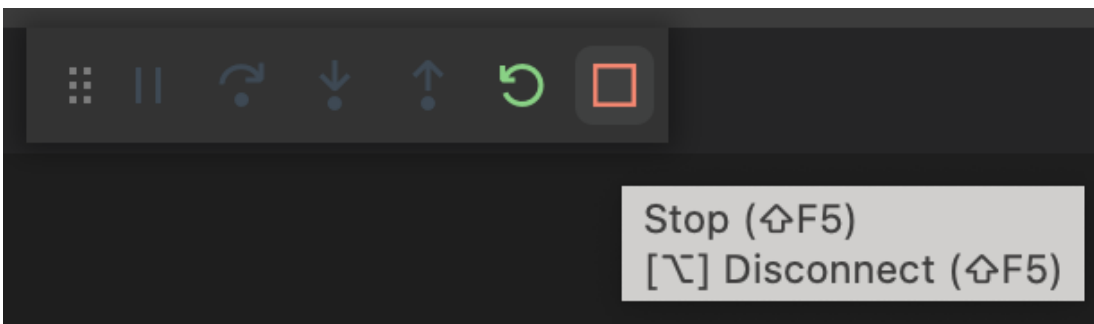
## Start debugging on the cluster

To start debugging on the cluster:

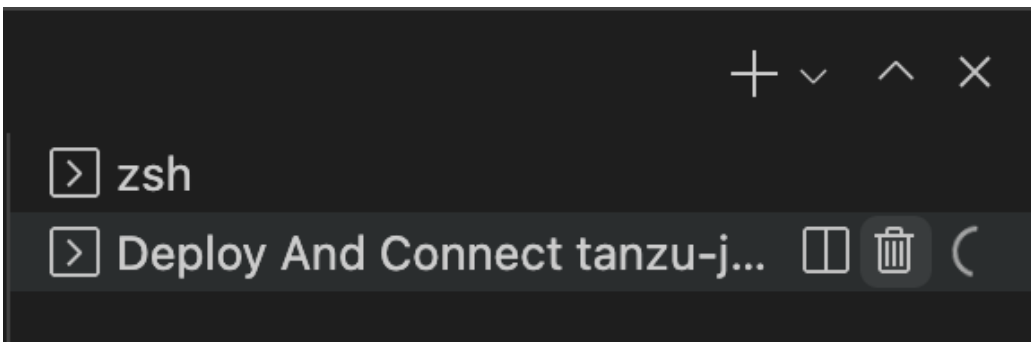
1. Add a [breakpoint](#) in your code.
2. Right-click anywhere in the VS Code project explorer or open the Command Palette by pressing `⇧ ⌘ P` (Ctrl+Shift+P on Windows).
3. Click **Tanzu: Java Debug Workload** from either menu.

## Stop Debugging on the cluster

To stop debugging on the cluster, you can click the stop button in the Debug overlay.



Alternatively, you can press `⇧ J` (Ctrl+J on Windows) to open the panel and then click the trash can button for the debug task running in the panel.



## Debug apps in a microservice repository

To debug multiple apps in a microservice repository:

1. Add each app folder as a workspace folder. For instructions, see the [Visual Studio Code documentation](#).
2. Update the `tanzu.debugPort` setting so that it does not conflict with other debugging sessions. For how to update individual workspace folder settings, see the [Visual Studio Code documentation](#).

## Live Update

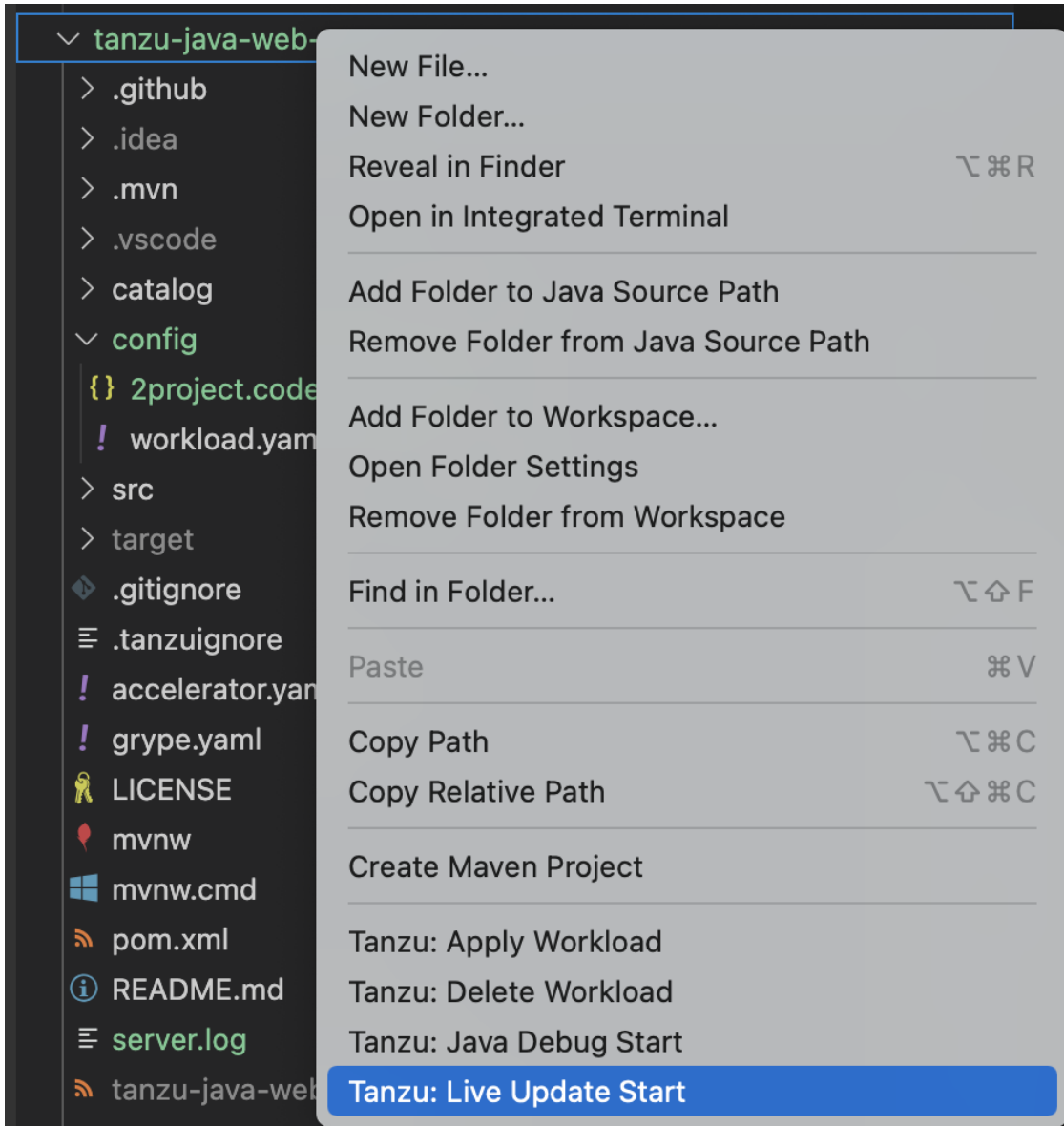
With the use of Live Update facilitated by [Tilt](#), the extension enables you to deploy your workload once, save changes to the code, and see those changes reflected in the workload running on the cluster within seconds.

Live Update requires a `workload.yaml` file and a Tiltfile in your project. For information about how to create a `workload.yaml` and a Tiltfile, see [Get Started with Tanzu Developer Tools for VS Code](#).

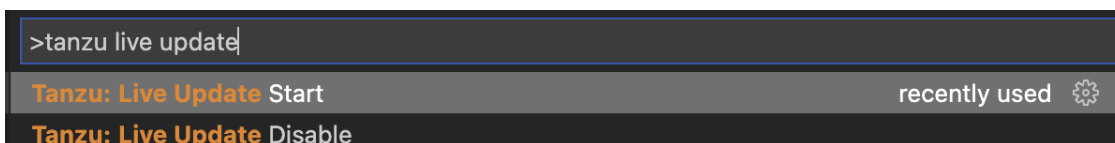
The developer sandbox experience enables developers to Live Update their code, and simultaneously debug the updated code, without having to deactivate Live Update when debugging.

## Start Live Update

You can start Live Update by right-clicking anywhere in the VS Code project explorer and then clicking **Tanzu: Live Update Start** in the pop-up menu.



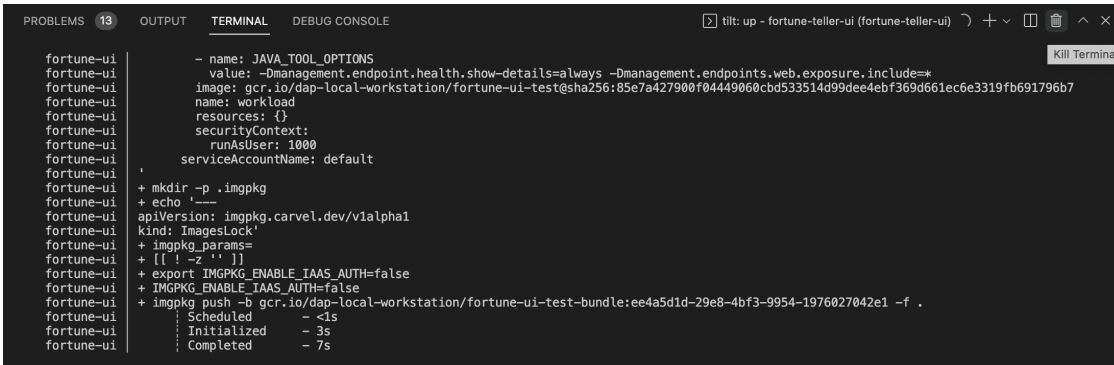
Alternatively, you can press `⇧⌘P` to open the Command Palette and then run the `Tanzu: Live Update Start` command.



## Stop Live Update

When Live Update stops, your application continues to run on the cluster, but the changes you made and saved in your editor are not present in your running application unless you redeploy your application to the cluster.

To stop Live Update, click the trash can button in the terminal pane to stop the Live Update process.



```

fortune-ui - name: JAVA_TOOL_OPTIONS
fortune-ui value: -Dmanagement.endpoint.health.show-details=always -Dmanagement.endpoints.web.exposure.include=*
fortune-ui image: gcr.io/dap-local-workstation/fortune-ui-test@sha256:85e7a427900f04449060cbd533514d99dee4ebf369d661ec6e3319fb691796b7
fortune-ui name: workload
fortune-ui resources: {}
fortune-ui securityContext:
fortune-ui runAsUser: 1000
fortune-ui serviceAccountName: default
fortune-ui
fortune-ui + mkdir -p .imgpkg
fortune-ui + echo '___
fortune-ui apiVersion: imgpkg.carvel.dev/v1alpha1
fortune-ui kind: ImagesLock'
fortune-ui + imgpkg_params=
fortune-ui + [[ ! -z '' ]]
fortune-ui + export IMGPKG_ENABLE_IAAS_AUTH=false
fortune-ui + IMGPKG_ENABLE_IAAS_AUTH=false
fortune-ui + imgpkg push -D gcr.io/dap-local-workstation/fortune-ui-test-bundle:ee4a5d1d-29e8-4bf3-9954-1976627042e1 -f .
fortune-ui | Scheduled - <1s
fortune-ui | Initialized - 3s
fortune-ui | Completed - 7s

```

## Deactivate Live Update

You can remove the Live Update capability from your application entirely. You might find this option 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  $\uparrow$   $\text{⌘}$ P (Ctrl+Shift+P on Windows) 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

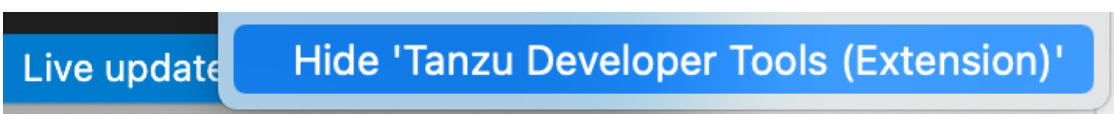
The current status of Live Update is visible on the right side of the status bar at the bottom of the VS Code window.



The Live Update status bar entry shows the following states:

- Live Update Stopped
- Live Update Starting...
- Live Update Running

To hide the Live Update status bar entry, right-click it and then click **Hide 'Tanzu Developer Tools (Extension)'**.



## Live Update apps in a microservices repository

To Live Update multiple apps in a microservice repository:

1. Add each app folder as a workspace folder. For instructions, see the [Visual Studio Code documentation](#).
2. Ensure that a port is available to port-forward the Knative service. For example, you might have this in your Tiltfile:

```
k8s_resource('tanzu-java-web-app', port_forwards=["NUMBER:8080"],
            extra_pod_selectors=[{'carto.run/workload-name': 'tanzu-java-web-app',
            'app.kubernetes.io/component': 'run'}])
```

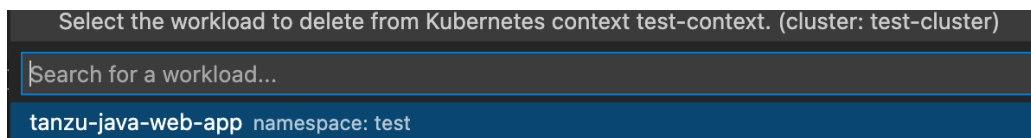
Where `NUMBER` is the port you choose. For example, `port_forwards=["9999:8080"]`.

## Delete a workload

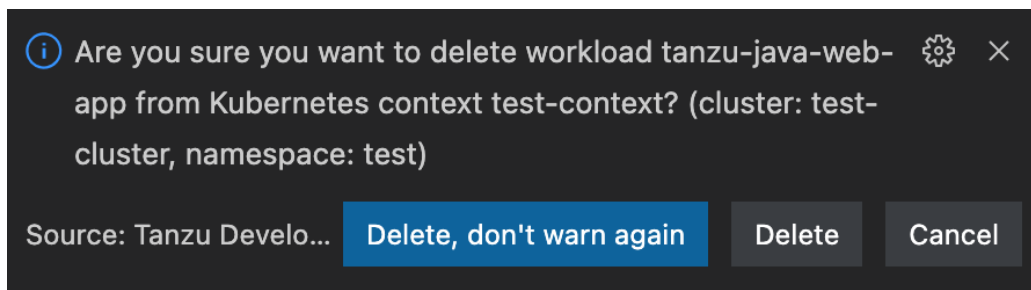
The extension enables you to delete workloads on your Kubernetes cluster that has Tanzu Application Platform.

To delete a workload:

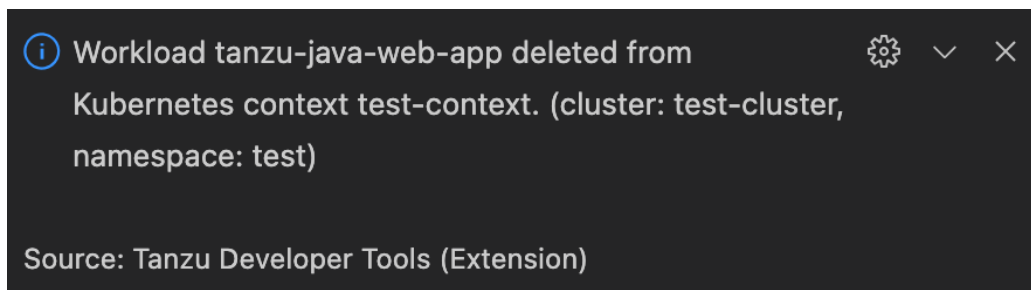
1. Right-click anywhere in the VS Code project explorer or open the Command Palette by pressing `⇧⌘P` (Ctrl+Shift+P on Windows).
2. Run `Tanzu: Delete Workload`.
3. Select the workload to delete.



If the **Tanzu: Confirm Delete** setting is enabled, a message appears that prompts you to delete the workload and not warn again, delete the workload, or cancel.



A notification appears showing that the workload was deleted.

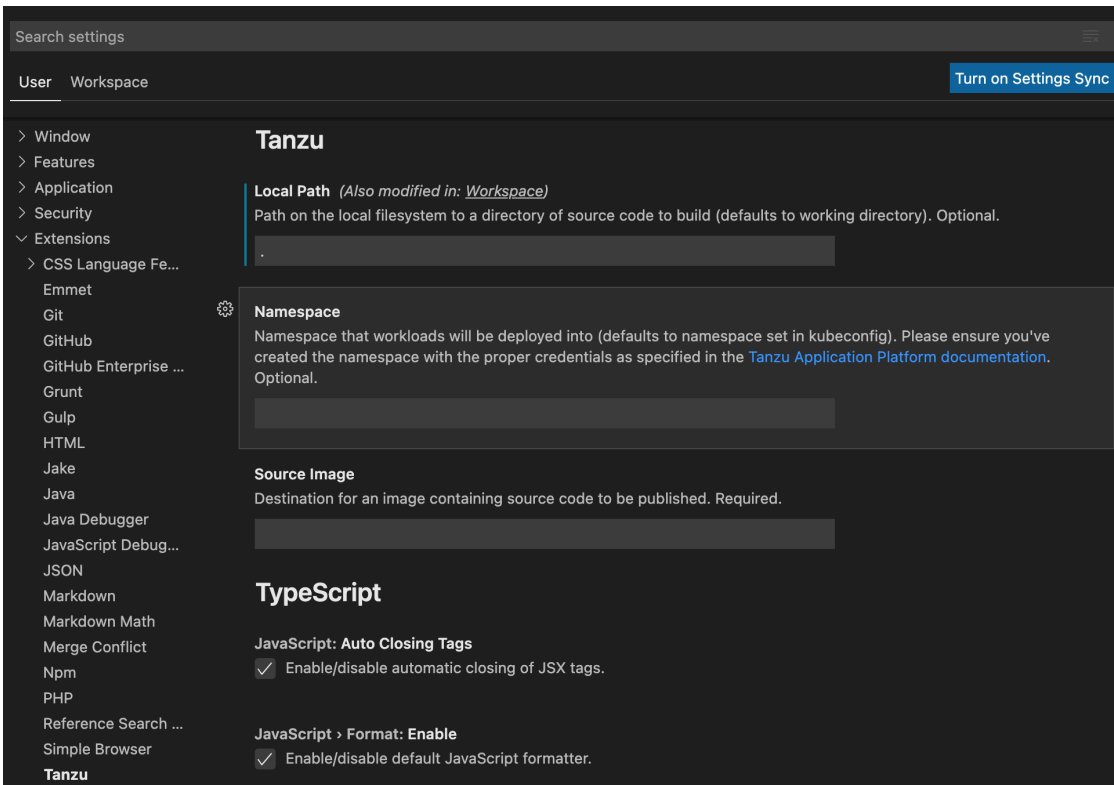


## Switch namespaces

To switch the namespace where you created the workload:

1. Go to **Code > Preferences > Settings**.
2. Expand the **Extensions** section of the settings and click **Tanzu**.

- In the **Namespace** option, add the namespace you want to deploy to. This is the **default** namespace by default.



## Tanzu Workloads panel

The current state of the workloads is visible on the Tanzu Workloads panel in the bottom left corner of the VS Code window. The panel shows the current status of each workload, namespace, and cluster. It also shows whether Live Update and Debug is running, stopped, or deactivated.

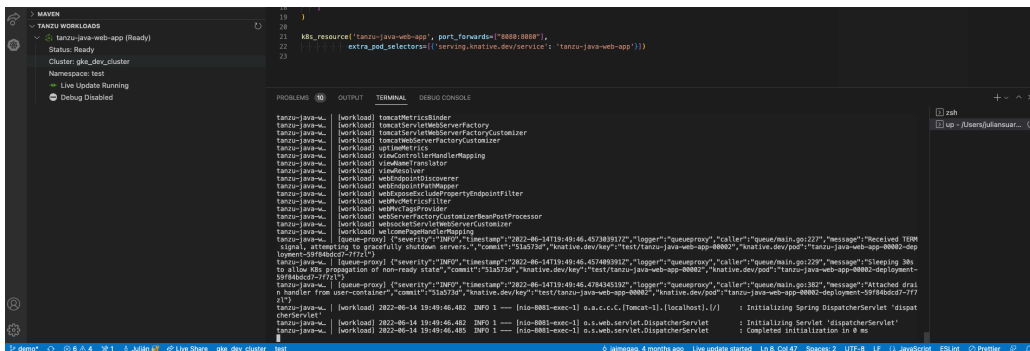
The Tanzu Workloads panel uses the cluster and namespace specified in the current kubectl context.

- View the current context and namespace by running:

```
kubectl config get-contexts
```

- Set a namespace for the current context by running:

```
kubectl config set-context --current --namespace=YOUR-NAMESPACE
```



## Working with Microservices in a Monorepo



A monorepo is single Git repository that contains multiple workloads. Each individual workload is placed in a subfolder of the main repository.

You can find an example of this in [Application Accelerator](#). The relevant accelerator is called Spring SMTP Gateway, and you can obtain its source code as an accelerator or directly from the [application-accelerator-samples](#) GitHub repository.

This project exemplifies a typical layout:

- MONO-REPO-ROOT/
  - pom.xml (parent pom)
  - microservice-app-1/
    - pom.xml
    - mvnw (and other mvn-related files for building the workload)
    - Tiltfile (supports Live Update)
    - config
      - workload.yaml (supports deploying and debugging from IntelliJ)
    - src/ (contains source code for this microservice)
  - microservice-app-2/
  - ...similar layout

## Recommended structure: Microservices that can be built independently

In this example, each of the microservices can be built independently of one another. Each subfolder contains everything needed to build that workload.

This is reflected in the `source` section of `workload.yaml` by using the `subPath` attribute:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: microservice-app-1
  ...
spec:
  source:
    git:
      ref:
        branch: main
      url: https://github.com/kdvolder/sample-mono-repo.git
      subPath: microservice-app-1 # build only this
    ...
```

For setting up your own repositories, it's best practice to set up a monorepo so that each microservice can be built completely independently.

To work with these monorepos:

- Import the monorepo as a project into VSCode.
- Interact with each of the subfolders in the same way you would a project containing a single workload.

## Alternative structure: Services with build-time interdependencies

Some monorepos do not have submodules that can be independently built. Instead the `pom.xml` files of the submodules are set up to have some build-time interdependencies. For example:

- A submodule `pom.xml` can reference the parent `pom.xml` as a common place for centralized dependency management.
- A microservice submodule can reference another, as a maven dependency.
- Several microservice submodules can reference one or more shared library modules.

For these projects, make these adjustments:

1. Make `workload.yaml` point to the repository root, not a subfolder. Because submodules have dependencies on code outside of their own subfolder, all source code from the repository must be supplied to the workload builder.
2. Make `workload.yaml` specify additional buildpack arguments through environment variables. They differentiate the submodule that the build is targeting.

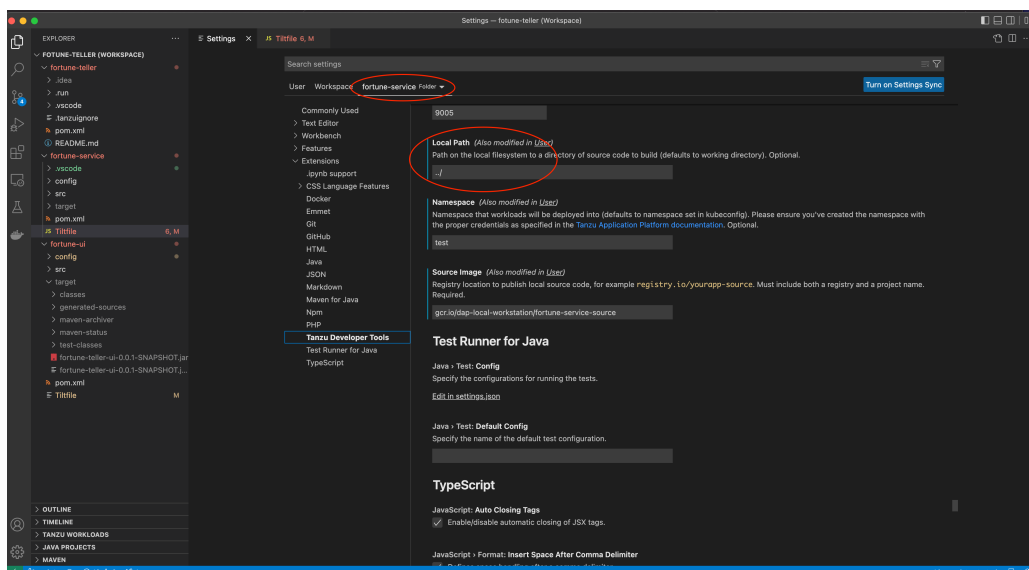
Both of these `workload.yaml` changes are in the following example:

```

apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: fortune-ui
labels:
  apps.tanzu.vmware.com/workload-type: web
  app.kubernetes.io/part-of: fortune-ui
spec:
  build:
    env:
      - name: BP_MAVEN_BUILD_ARGUMENTS
        value: package -pl fortune-teller-ui -am # indicate which module to build.
      - name: BP_MAVEN_BUILT_MODULE
        value: fortune-teller-ui # indicate where to find the built artefact to de
  source:
    git:
      url: https://github.com/my-user/fortune-teller # repo root
      ref:
        branch: main
    
```

For more information about these and other `BP_XXX` buildpack parameters, see the [Buildpack Documentation](#).

3. Make the local path preference for each subfolder point to the path of the repository root. Because submodules have dependencies on code outside of their own subfolder, all source code from the repository must be supplied to the workload builder.



## Pinniped compatibility

This topic tells you the compatibility details of [Pinniped](#) in GitHub.

## OAuth

OAuth login is compatible only when both `--skip-browser` and `--skip-listen` flags are not set.

## LDAP

LDAP authentication is not compatible with VMware Tanzu Developer Tools for Visual Studio Code.

## Integrate Live Hover by using Spring Boot Tools

For more information about this feature, see the [Live application information hovers](#) section of the [Spring Boot Tools Marketplace page](#).

## Prerequisites

To integrate Live Hover by using Spring Boot Tools you need:

- A Tanzu Spring Boot application, such as [tanzu-java-web-app](#)
- Spring Boot Extension Pack (includes Spring Boot Dashboard) [extension](#)

## Activate the Live Hover feature

Activate the Live Hover feature by enabling it in **Code > Preferences > Settings > Extensions > Tanzu Developer Tools**.

## Deploy a Workload to the Cluster

Follow these steps to deploy the workload for an app to a cluster, making live hovers appear. The examples in some steps reference the sample [tanzu-java-web-app](#).

1. Clone the repository by running:

```
git clone REPOSITORY-ADDRESS
```

Where `REPOSITORY-ADDRESS` is your repository address. For example, <https://github.com/vmware-tanzu/application-accelerator-samples>.

2. Open the project in VS Code, with the Live Hover feature enabled, by running:

```
TAP_LIVE_HOVER=true code ./PROJECT-DIRECTORY
```

Where `PROJECT-DIRECTORY` is your project directory. For example, `./application-accelerator-samples/tanzu-java-web-app`.

3. Verify that you are targeting the cluster on which you want to run the workload by running:

```
kubectl cluster-info
```

For example:

```
$ kubectl cluster-info
Kubernetes control plane is running at https://...
CoreDNS is running at https://...
```

```
To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

Tanzu Developer Tools for VS Code periodically connects to your cluster to search for pods from which live data can be extracted and shown. Tanzu Developer Tools for VS Code uses your current context from `~/.kube/config` to choose which cluster to connect with.

- If you don't have the workload running yet, run **Tanzu: Apply Workload** from the Command Palette. Tanzu Developer Tools for VS Code periodically searches for pods in your cluster that correspond to the workload configurations it finds in your workspace.
- The workload takes time to build and then start a running pod. To see if a pod has started running, run:

```
kubectl get pods
```

For example:

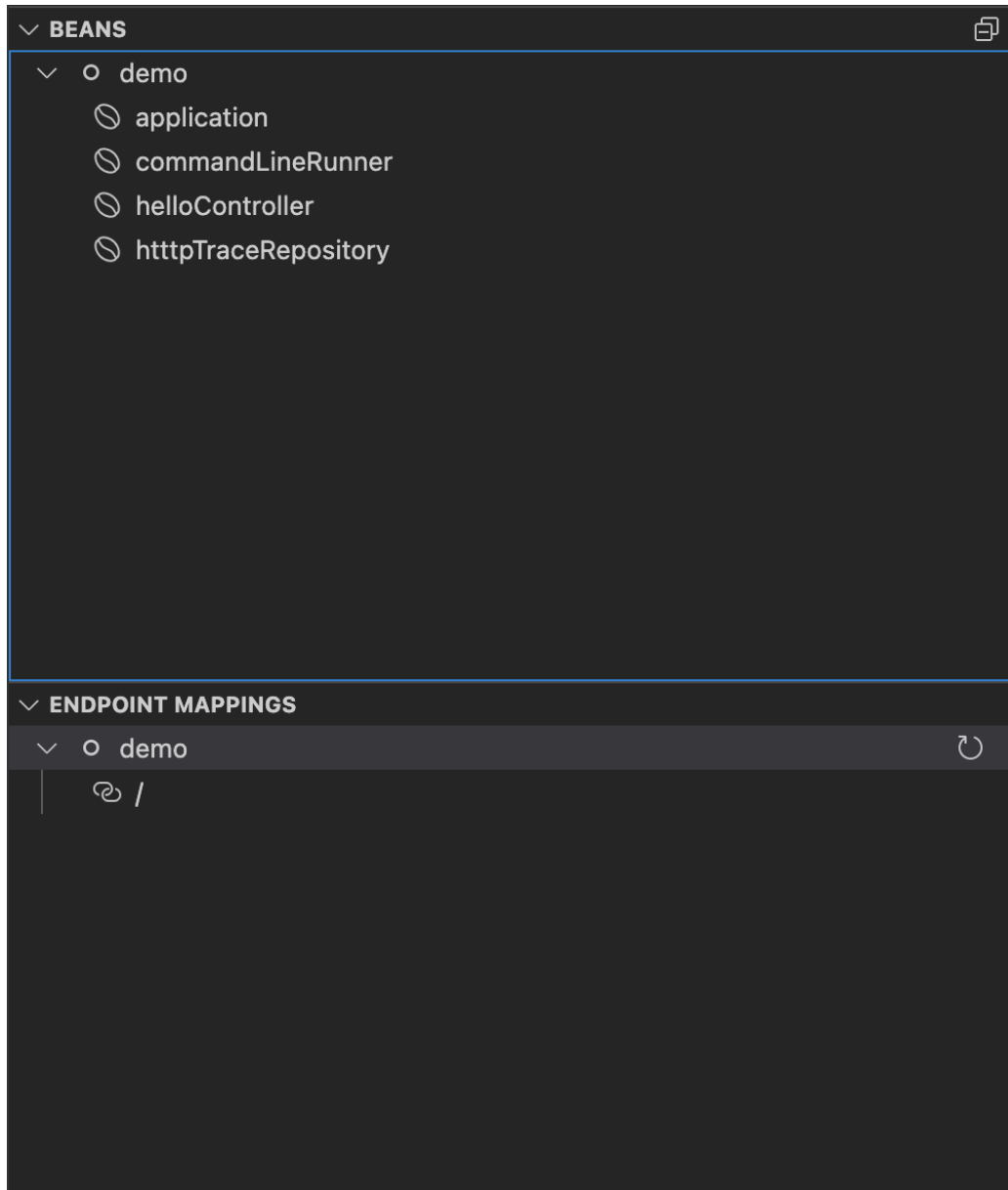
```
$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
tanzu-java-web-app-00001-deployment-8596bfd9b4-5vgx2  2/2     Running   0          20s
tanzu-java-web-app-build-1-build-pod                0/1     Completed 0          2m26s
tanzu-java-web-app-config-writer-fpnzb-pod          0/1     Completed 0          67s
```

In this example, live data can be extracted from the `...-0001-deployment-...` pod.

- Open a Java file, such as `HelloController.java`. After a delay of up to 30 seconds, because of a 30-second polling loop, green highlights appear in your code.

```
6 @RestController
7 public class HelloController {
8
9     http://tanzu-java-web-app.my-apps.vipins2appsso.cloudfocused.in/ (Count=1 Total=0.06s Max=0.06s)
10    @RequestMapping("/")
11    public String index() {
12
13        return "Greetings from Spring Boot + Tanzu + TAP!";
14
15    }
16 }
```

- Hover over any of the bubbles to see live information about the corresponding element.
- The **Live Beans** and **Live Endpoint Mapping** information are displayed in Spring Boot Dashboard. To view the Spring Boot Dashboard, run **View: Show Spring Boot Dashboard** from the Command Palette.



## Use Memory View in Spring Boot Dashboard

This topic tells you how to use Spring Boot Dashboard to view memory use.

For more information about Spring Boot Dashboard, see [Spring Boot Dashboard](#).

### Prerequisites

To see the Memory View in Spring Boot Dashboard you need:

- A Tanzu Spring Boot application, such as [tanzu-java-web-app](#)
- The [Spring Boot Extension Pack](#), which includes Spring Boot Dashboard

### Deploy a workload

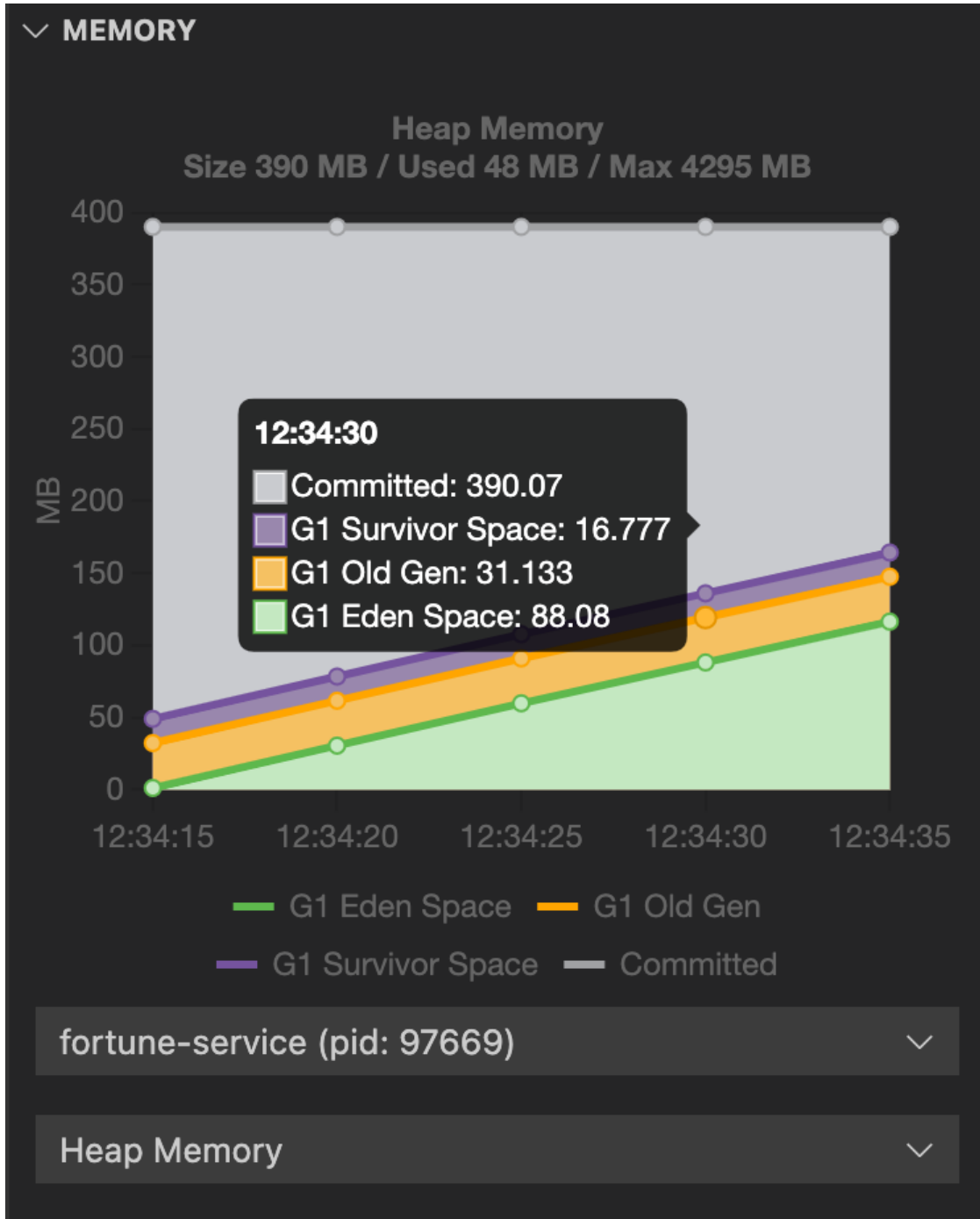
Deploy the workload for an app to a cluster by following the steps in [Deploy a Workload to the Cluster](#).

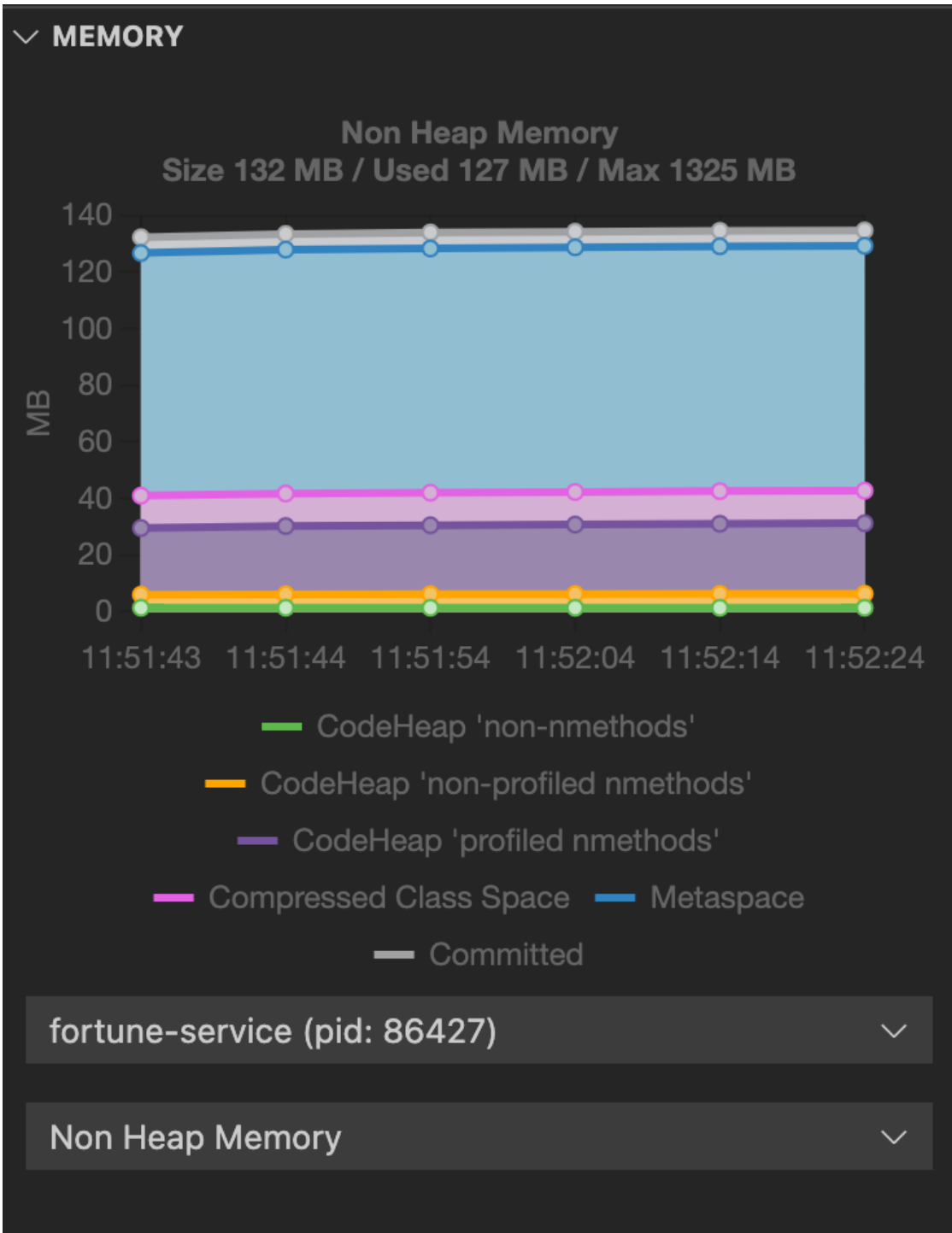
## View memory use in Spring Boot Dashboard

To view the Spring Boot Dashboard, run `View: Show Spring Boot Dashboard` from the Command Palette.

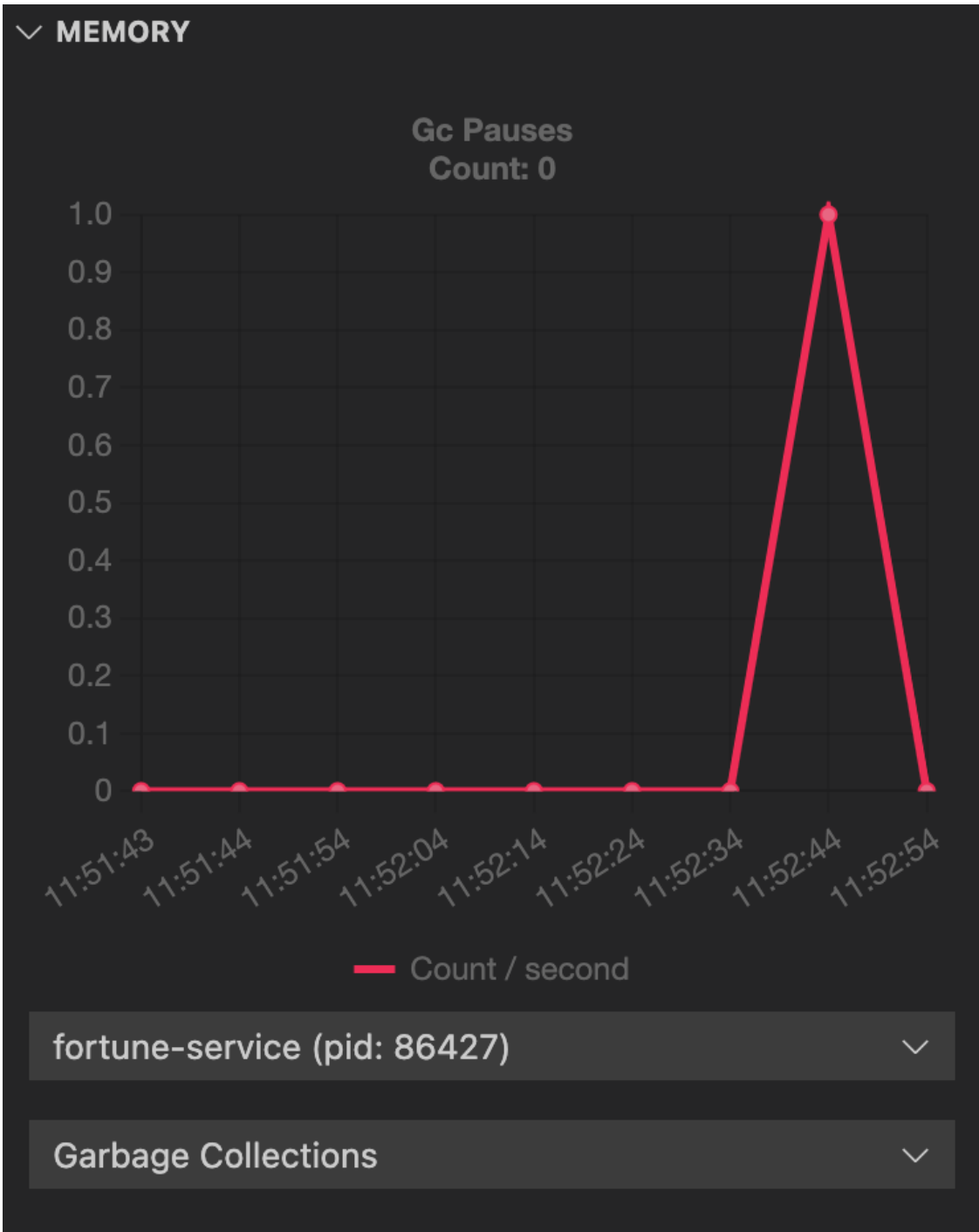
When the app is running, the Memory View section is displayed in Spring Boot Dashboard. The graphical representation in the memory view highlights the memory use inside the Java virtual machine (JVM). The drop-down menus beneath the graph enable you to switch between different running processes and graphical views.

The heap and non-heap memory regions provide memory insights into the application. The real-time graphs display a stacked overview of the different spaces in memory relative to the total memory used and total memory size.

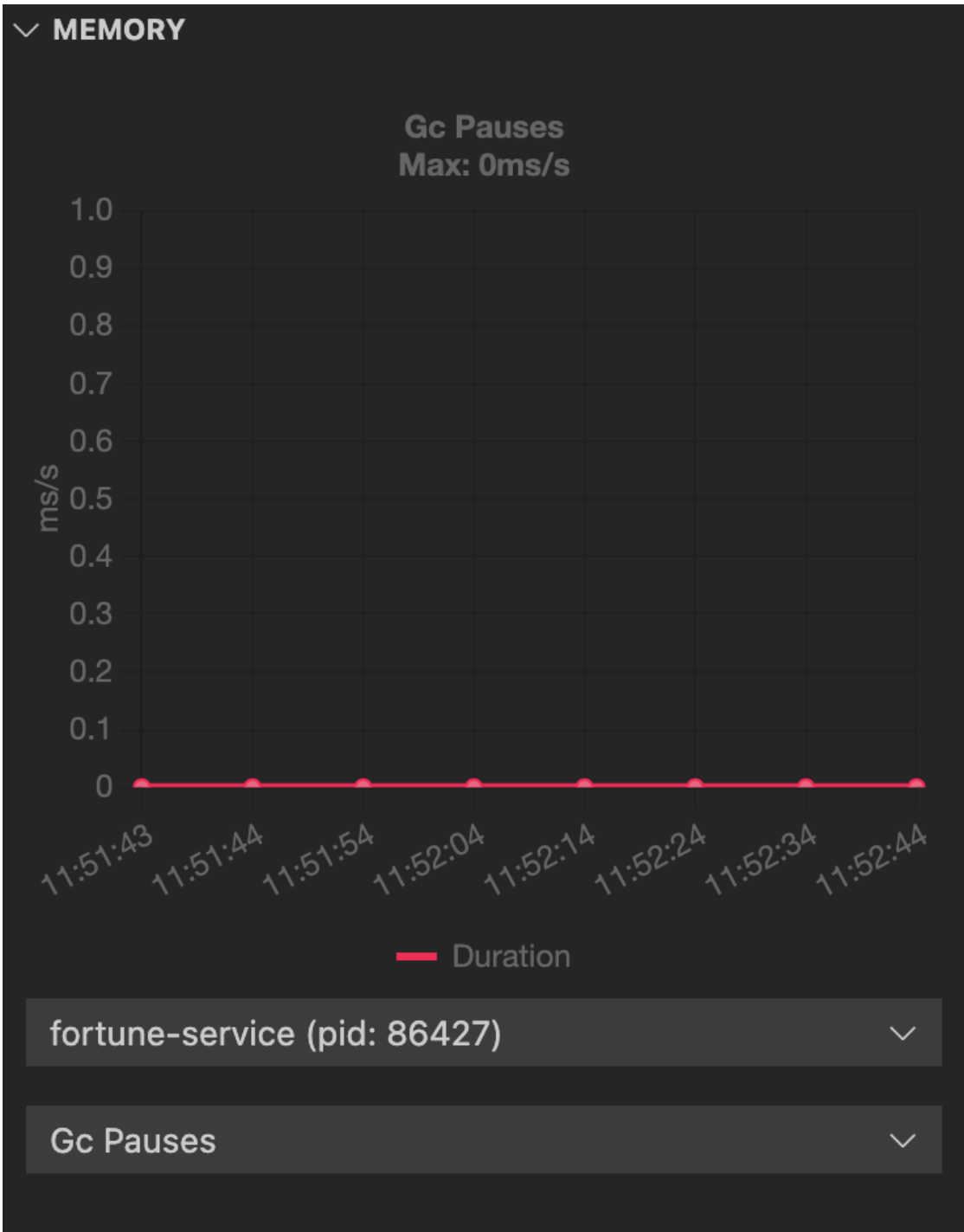




The memory view also contains graphs to display the garbage-collection pauses and garbage-collection events. Long and frequent garbage-collection pauses indicate that the app is having a memory problem that requires further investigation.







The graphs show only real-time data. You can configure the number of data points to view and the interval by changing the settings. To access the settings on macOS, go to **Code > Preferences > Settings > Extensions > Spring Boot Dashboard > Memory View Settings**. The navigation path might differ on other operating systems, such as Windows and Linux.

#### Memory View Settings

Spring > Dashboard > Memory-view > Display-data: **Max-datapoints**  
Spring boot dashboard memory view - Display max data points in the graph

5

Spring > Dashboard > Memory-view > Fetch-data: **Delay-in-milliseconds**  
Spring boot dashboard memory view - Delay between fetching new data in milliseconds

5000

## Troubleshoot Tanzu Developer Tools for VS Code

This topic tells you what to do when you encounter issues with VMware Tanzu Developer Tools for Visual Studio Code (VS Code).

### Unable to view workloads on the panel when connected to GKE cluster

#### Symptom

When connecting to Google's GKE clusters, an error appears with the text `WARNING: the gcp auth plugin is deprecated in v1.22+, unavailable in v1.25+; use gcloud instead.`

#### Cause

GKE authentication was extracted into a separate plug-in and is no longer inside the Kubernetes client or libraries.

#### Solution

Download and configure the GKE authentication plug-in. For instructions, see the [Google documentation](#).

### Live Update fails with `UnsupportedClassVersionError`

#### Symptom

After live-update has synchronized changes you made locally to the running workload, the workload pods start failing with an error message similar to the following:

```
Caused by: org.springframework.beans.factory.CannotLoadBeanClassException: Error loading
class
[com.example.springboot.HelloController] for bean with name 'helloController' defined
in file
[/workspace/BOOT-INF/classes/com/example/springboot/HelloController.class]: problem wi
th class file
or dependent class; nested exception is
java.lang.UnsupportedClassVersionError: com/example/springboot/HelloController has bee
n compiled by
a more recent version of the Java Runtime (class file version 61.0), this version of t
he
Java Runtime only recognizes class file versions up to 55.0
```

#### Cause

The classes produced locally on your machine are compiled to target a later Java virtual machine (JVM). The error message mentions `class file version 61.0`, which corresponds to Java 17. The buildpack, however, is set up to run the application with an earlier JVM. The error message mentions `class file versions up to 55.0`, which corresponds to Java 11.

The root cause of this is a misconfiguration of the Java compiler that VS Code uses. The cause might be a suspected issue with the VS Code Java tooling, which sometimes fails to properly configure the compiler source and target compatibility-level from information in the Maven POM.

For example, in the `tanzu-java-web-app` sample application the POM contains the following:

```
<properties>
  <java.version>11</java.version>
```

```
...
</properties>
```

This correctly specifies that the app must be compiled for Java 11 compatibility. However, the VS Code Java tooling sometimes fails to take this information into account.

## Solution

Force the VS Code Java tooling to re-read and synchronize information from the POM:

1. Right-click the `pom.xml` file.
2. Click **Reload Projects**.

This causes the internal compiler level to be set correctly based on the information from `pom.xml`. For example, Java 11 in `tanzu-java-web-app`.

## Timeout error when Live Updating

### Symptom

When you attempt to Live Update your workload, the following error message appears in the log:

```
ERROR: Build Failed: apply command timed out after 30s - see https://docs.tilt.dev/v/api.html#api.update\_settings for how to increase
```

### Cause

Kubernetes times out on upserts over 30 seconds.

### Solution

Add `update_settings (k8s_upsert_timeout_secs = 300)` to the Tiltfile. For more information, see the [Tiltfile documentation](#).

## Task-related error when running a Tanzu Debug launch configuration

### Symptom

When you attempt to run a Tanzu Debug launch configuration, you see a task-related error message similar to the following:

```
Could not find the task 'tanzuManagement: Kill Port Forward my-app'
```

### Cause

The task you're trying to run is no longer supported.

### Solution

Delete the launch configuration from your `launch.json` file in your `.vscode` directory.

## Frequent application restarts

### Symptom

When an application is applied from VS Code it restarts frequently.

## Cause

An application or environment behavior is triggering the application to restart.

Observed trigger behaviors include:

- The application itself writing logs to the file system in the application directory that Live Update is watching
- Autosave being set to a very high frequency in the IDE configuration

## Solution

Prevent the trigger behavior. Example solutions include:

- Prevent 12-factor applications from writing to the file system.
- Reduce the autosave frequency to once every few minutes.

## Overview of Tanzu Developer Tools for IntelliJ

Tanzu Developer Tools for IntelliJ is the official VMware Tanzu IDE extension for IntelliJ IDEA. The extension helps you develop with Tanzu Application Platform and enables you to rapidly iterate on your workloads on supported Kubernetes clusters that have Tanzu Application Platform installed.

Tanzu Developer Tools for IntelliJ currently supports Java applications on macOS and Windows.

## Extension features

This extension gives the following features.

- **Deploy applications directly from IntelliJ:**  
Rapidly iterate on your applications on Tanzu Application Platform and deploy them as workloads directly from within IntelliJ.
- **See code updates running on-cluster in seconds:**  
With the use of Live Update facilitated by Tilt, deploy your workload once, save changes to the code and then, seconds later, see those changes reflected in the workload running on the cluster.
- **Debug workloads directly on the cluster:**  
Debug your application in a production-like environment by debugging on your Kubernetes cluster that has Tanzu Application Platform. An environment's similarity to production relies on keeping dependencies updated, among other variables.
- **See workloads running on the cluster:**  
From the Workloads panel you can see any workload found within the cluster and namespace specified in the current kubectl context.
- **Work with microservices in a Java monorepo:**  
Tanzu Developer Tools for IntelliJ v1.3 and later supports working with a monorepo containing multiple modules that represent different microservices. This makes it possible to deploy, debug, and live update multiple workloads simultaneously from the same IntelliJ multimodule project. For more information about projects with multiple modules, see the [IntelliJ documentation](#). For more information about a typical monorepo setup, see [Working with microservices in a monorepo](#).

## Next steps

[Follow the steps to install the extension.](#)

## Overview of Tanzu Developer Tools for IntelliJ

Tanzu Developer Tools for IntelliJ is the official VMware Tanzu IDE extension for IntelliJ IDEA. The extension helps you develop with Tanzu Application Platform and enables you to rapidly iterate on your workloads on supported Kubernetes clusters that have Tanzu Application Platform installed.

Tanzu Developer Tools for IntelliJ currently supports Java applications on macOS and Windows.

## Extension features

This extension gives the following features.

- **Deploy applications directly from IntelliJ:**

Rapidly iterate on your applications on Tanzu Application Platform and deploy them as workloads directly from within IntelliJ.

- **See code updates running on-cluster in seconds:**

With the use of Live Update facilitated by Tilt, deploy your workload once, save changes to the code and then, seconds later, see those changes reflected in the workload running on the cluster.

- **Debug workloads directly on the cluster:**

Debug your application in a production-like environment by debugging on your Kubernetes cluster that has Tanzu Application Platform. An environment's similarity to production relies on keeping dependencies updated, among other variables.

- **See workloads running on the cluster:**

From the Workloads panel you can see any workload found within the cluster and namespace specified in the current kubectl context.

- **Work with microservices in a Java monorepo:**

Tanzu Developer Tools for IntelliJ v1.3 and later supports working with a monorepo containing multiple modules that represent different microservices. This makes it possible to deploy, debug, and live update multiple workloads simultaneously from the same IntelliJ multimodule project. For more information about projects with multiple modules, see the [IntelliJ documentation](#). For more information about a typical monorepo setup, see [Working with microservices in a monorepo](#).

## Next steps

[Follow the steps to install the extension.](#)

## Install Tanzu Developer Tools for IntelliJ

This topic explains how to install the VMware Tanzu Developer Tools for IntelliJ IDE extension. The extension currently only supports Java applications on macOS and Windows. The extension currently supports IntelliJ IDEA v2022.2 to v2022.3.

## Prerequisites

Before installing the extension, you must have:

- [IntelliJ](#)

- [kubectl](#)
- [Tilt v0.30.12](#) or later
- [Tanzu CLI and plug-ins](#)
- [A cluster with the Tanzu Application Platform Full profile or Iterate profile](#)



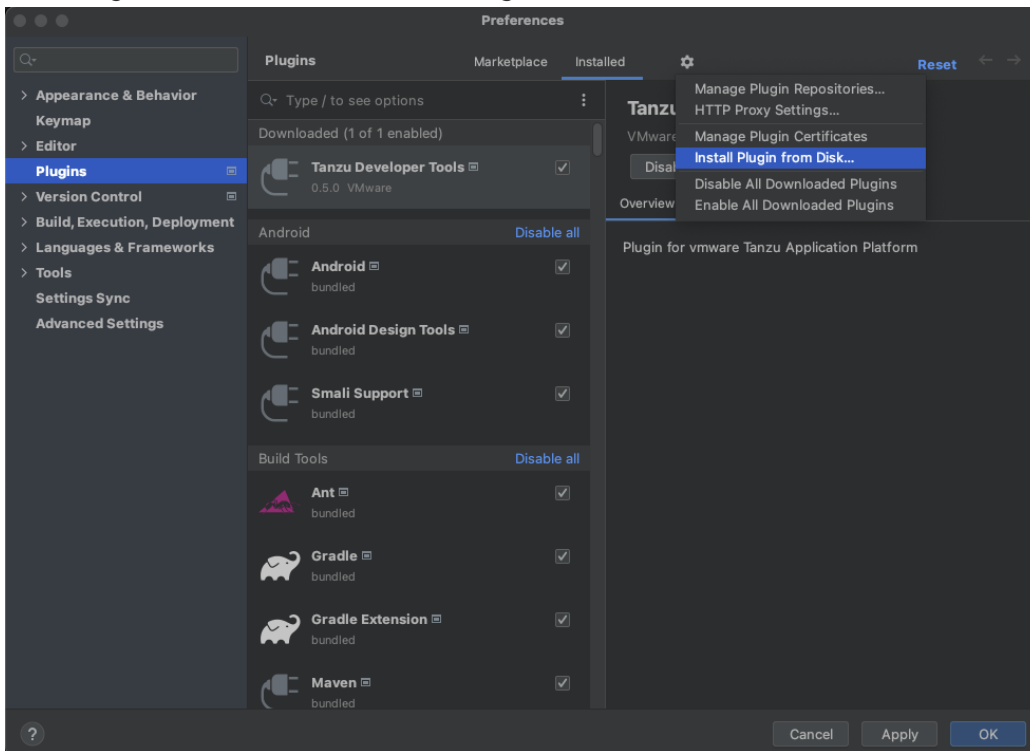
**Note**

If you are an app developer, someone else in your organization might have already set up the Tanzu Application Platform environment.

## Install

To install VMware Tanzu Developer Tools for IntelliJ:

1. Download VMware Tanzu Developer Tools for IntelliJ from the [VMware Tanzu Network](#).
2. Open IntelliJ.
3. Open the **Preferences** pane and then go to **Plugins**.
4. Click the gear icon and then click **Install Plugin from disk...**



5. Use the file picker to select the ZIP file downloaded from the VMware Tanzu Network.

## Update

To update to a later version, repeat the steps in the [Install](#) section. You do not need to uninstall the current version.

## Uninstall

To uninstall the VMware Tanzu Developer Tools for IntelliJ:

1. Open the **Preferences** pane and then go to **Plugins**.
2. Select the extension, click the gear icon, and then click **Uninstall**.
3. Restart IntelliJ.

## Next steps

Proceed to [Getting started](#).

## Get Started with Tanzu Developer Tools for IntelliJ

This topic guides you through getting started with Tanzu Developer Tools for IntelliJ.

## Prerequisite

[Install Tanzu Developer Tools for IntelliJ](#).

## Run Tanzu Developer Tools for IntelliJ

Run IntelliJ from a CLI, instead of through your operating system GUI, to avoid restricting the set of environment variables the app receives. This is especially relevant for macOS.

Limited environment variables can cause problems with cluster authentication for Tanzu Developer Tools for IntelliJ. For example, a common situation is that a sanitized `PATH` does not provide access to the `gke-cloud-auth-plugin` installed on your system. This makes Tanzu Developer Tools for IntelliJ unable to authenticate and access your GKE cluster.

This situation is complex and different things can go wrong depending on:

- Precisely how you installed various cloud-related CLI tools
- How you set environment variables
- Your OS version
- Which cloud provider and authentication method you are using

All of these problems are most easily avoided by running IntelliJ from a CLI. Run IntelliJ from a CLI in macOS by running:

```
open /Applications/IntelliJ\ IDEA.app
```

## Set up Tanzu Developer Tools

The extension makes use of the following files within your project:

- `workload.yaml`
- `catalog-info.yaml`
- `Tiltfile`
- `.tanzuignore`

You can create these files by using the instructions in this topic, or use the files in the [View an example project](#) section.

There are two ways to create these files:

- Using the code snippets that Tanzu Developer Tools provide, which create templates in empty files that you then fill in with the required information.

- Writing the files manually.

## Create the `workload.yaml` file

You must include a file named `workload.yaml` in your project. For example, `my-project/config/workload.yaml`.

`workload.yaml` provides instructions to Supply Chain Choreographer about how to build and manage a workload. For more information, see [Supply Chain Choreographer for Tanzu](#).

The Tanzu Developer Tools for IntelliJ extension requires only one `workload.yaml` file per project. `workload.yaml` must be a single-document YAML file, not a multi-document YAML file.

To create a `workload.yaml` file by using code snippets:

1. Right-click on the IntelliJ project explorer and then click **New**.
2. Select the Tanzu workload.
3. Add the filename `workload`.
4. Fill in the template.

See the following `workload.yaml` example:

```
apiVersion: carto.run/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. For example, `my app`.
- `WORKLOAD-TYPE` is the type of workload for your app. For example, `web`. For more information, see [Workload types](#).
- `GIT-SOURCE-URL` is the Git source code URL for your app. For example, `github.com/mycompany/myapp`.
- `GIT-BRANCH-NAME` is the branch of the Git source code you want to use. For example, `main`.

Alternatively you can use the Tanzu CLI to create a `workload.yaml` file. For more information about the relevant Tanzu CLI command, see [Tanzu apps workload apply](#).

## Create the `catalog-info.yaml` file

You must include a file named `catalog-info.yaml` in your project. For example, `my-project/catalog/catalog-info.yaml`.

`catalog-info.yaml` enables the workloads created with Tanzu Developer Tools for IntelliJ to be visible in Tanzu Application Platform GUI. For more information, see [Overview of Tanzu Application Platform GUI](#).

To create a `catalog-info.yaml` file by using the code snippets:



1. Right-click on the IntelliJ project explorer and then click **New**.
2. Select the Tanzu Catalog.
3. Add the filename `catalog-info`.
4. Fill in the template.

See the following `workload.yaml` example:

```
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: APP-NAME
  description: APP-DESCRIPTION
  tags:
    - tanzu
  annotations:
    'backstage.io/kubernetes-label-selector': 'app.kubernetes.io/part-of=APP-NAME'
spec:
  type: service
  lifecycle: experimental
  owner: default-team
```

Where:

- `APP-NAME` is the name of your application.
- `APP-DESCRIPTION` is a description of your application.

## Create the Tiltfile file

In your project you must include a file named `Tiltfile` with no extension (no filetype), such as `my-project/Tiltfile`.

The `Tiltfile` provides the configuration for Tilt to enable your project to [Live Update](#) on the Tanzu Application Platform-enabled Kubernetes cluster. For more information, see the [Tilt documentation](#).

The Tanzu Developer Tools for IntelliJ extension requires only one Tiltfile per project.

The following is an example `Tiltfile`:

```
SOURCE_IMAGE = os.getenv("SOURCE_IMAGE", default='SOURCE-IMAGE-VALUE')
LOCAL_PATH = os.getenv("LOCAL_PATH", default='.')
NAMESPACE = os.getenv("NAMESPACE", default='default')

k8s_custom_deploy(
  'APP-NAME',
  apply_cmd="tanzu apps workload apply -f PATH-TO-WORKLOAD-YAML --live-update" +
    " --local-path " + LOCAL_PATH +
    " --source-image " + SOURCE_IMAGE +
    " --namespace " + NAMESPACE +
    " --yes >/dev/null" +
    " && kubectl get workload APP-NAME --namespace " + NAMESPACE + " -o yaml",
  delete_cmd="tanzu apps workload delete -f PATH-TO-WORKLOAD-YAML --namespace " + 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/com
```

```
ponent': 'run'}})
allow_k8s_contexts('CONTEXT-NAME')
```

Where:

- `SOURCE-IMAGE-VALUE` is your [source image](#).
- `APP-NAME` is the name of your application.
- `PATH-TO-WORKLOAD-YAML` is the local file system path to your `workload.yaml` file. For example, `config/workload.yaml`.
- `CONTEXT-NAME` is the name of your current [Kubernetes context](#). If your Tanzu Application Platform-enabled Kubernetes cluster is running on your local machine, you can remove the entire `allow_k8s_contexts` line. For more information about this line, see the [Tilt documentation](#).

If you want to compile the source image from a local directory other than the project directory, change the value of `local path`. For more information, see [local path](#) in the glossary.

## Create the `.tanziignore` file

In your project, you can include a file named `.tanziignore` with no file extension. For example, `my-project/.tanziignore`.

When working with local source code, `.tanziignore` excludes files from the source code that are uploaded within the image. It has syntax similar to the `.gitignore` file.

For an example, see the `.tanziignore` file in GitHub that is used for the sample Tanzu Java web app. You can use the file as it is or edit it for your needs.

## View an example project

Before you begin, you need a container image registry to use the sample application. There are two ways to view a sample application that demonstrates the necessary configuration files.

### Use Application Accelerator

If your company has configured [Application Accelerator](#), you can obtain the sample application there if it was not removed. To view the example using Application Accelerator:

1. Open Application Accelerator. The Application Accelerator location varies based on where your company placed it. Contact the appropriate team to learn its location.
2. Search for `Tanzu Java Web App` in the Application Accelerator.
3. Add the required configuration information and generate the application.
4. Unzip the application and open the directory in IntelliJ.

### Clone from GitHub

To clone the example from GitHub:

1. Use `git clone` to clone the `application-accelerator-samples` repository from GitHub.
2. Go to the `tanzu-java-web-app` directory.
3. Open the `Tiltfile` and replace `your-registry.io/project` with your registry.

## Next steps

Use [Tanzu Developer Tools for IntelliJ](#).

## Use Tanzu Developer Tools for IntelliJ

Ensure that the project you want to use the Tanzu Developer Tools for IntelliJ extension with has the required files specified in [Getting started](#).

The extension requires only one Tiltfile and one `workload.yaml` file per project. `workload.yaml` must be a single-document YAML file, not a multi-document YAML file.

## Apply a workload

The extension enables you to apply workloads on your Kubernetes cluster that has Tanzu Application Platform.

To apply a workload:

1. Right-click anywhere in the IntelliJ project explorer.
2. Click **Tanzu > Apply Workload**.
3. Click **Tanzu > Modify Apply Configuration**.

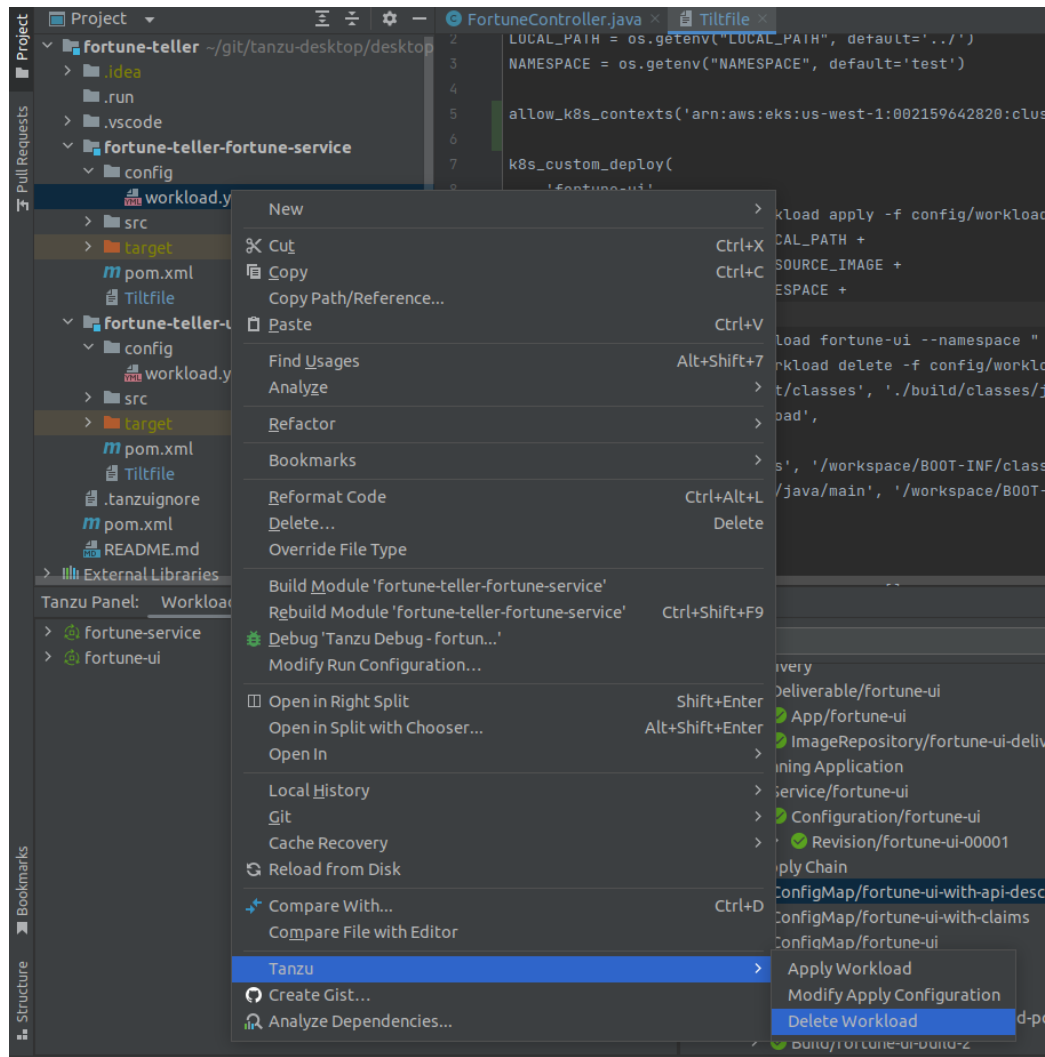
The `Tanzu workload apply` command is triggered in the terminal and the workload is applied. A new workload appears on the Tanzu panel.

## Delete a workload

The extension enables you to delete workloads on your Kubernetes cluster that has Tanzu Application Platform.

To delete a workload:

1. Right-click anywhere in the IntelliJ project explorer.
2. Click **Tanzu > Delete Workload**.



A message appears that prompts you to delete the workload and not warn again, delete the workload, or cancel. A notification appears showing that the workload was deleted.

## Debugging on the cluster

The extension enables you to debug your application on a Kubernetes cluster that has Tanzu Application Platform.

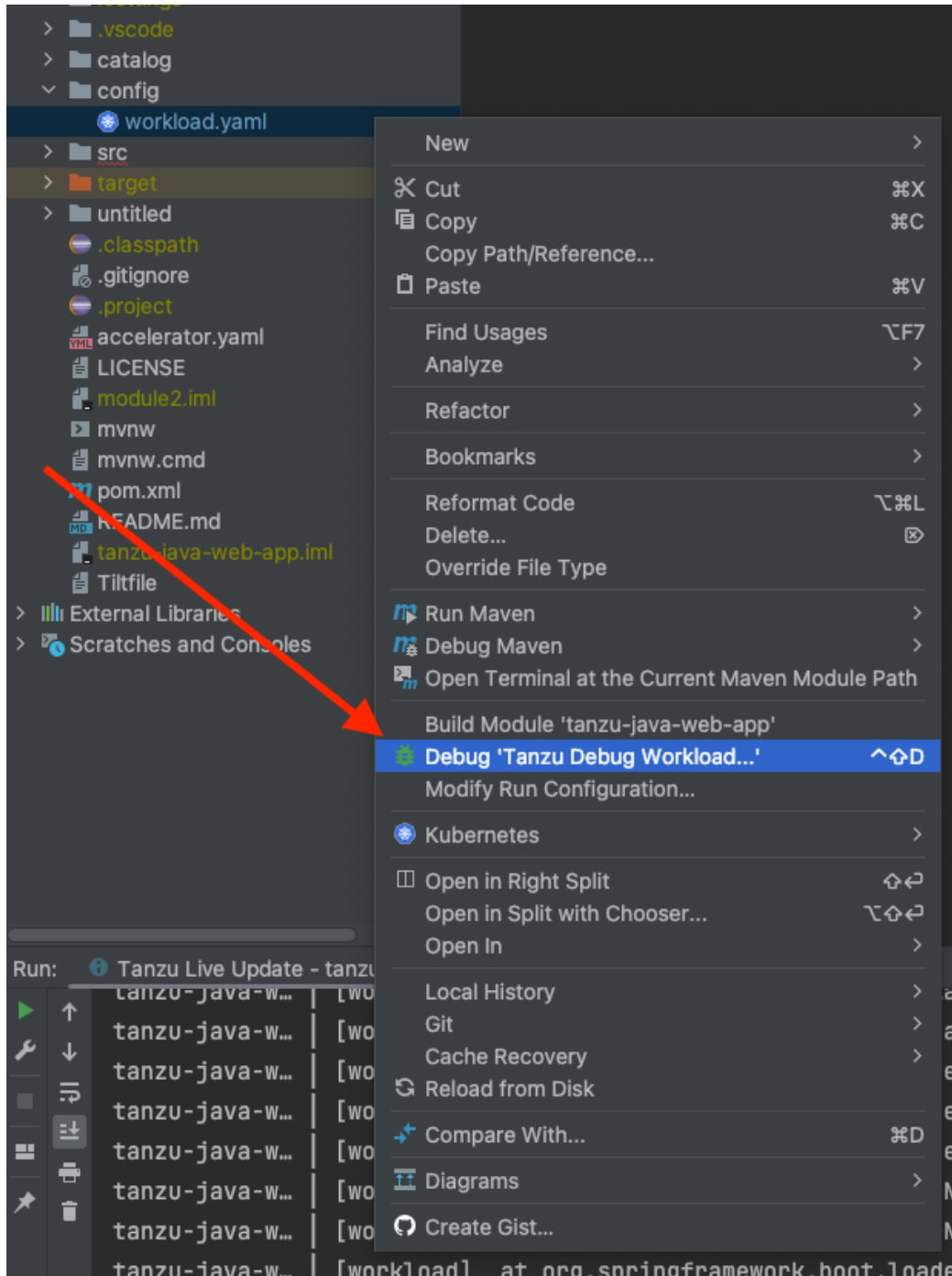
Debugging requires a single-document `workload.yaml` file in your project. For how to create `workload.yaml`, see [Set up Tanzu Developer Tools](#).

The developer sandbox experience enables developers to Live Update their code, and simultaneously debug the updated code, without having to deactivate Live Update when debugging.

## Start debugging on the cluster

To start debugging on the cluster:

1. Add a `breakpoint` in your code.
2. Right-click the `workload.yaml` file in your project.
3. Click **Debug 'Tanzu Debug Workload...'** in the pop-up menu.

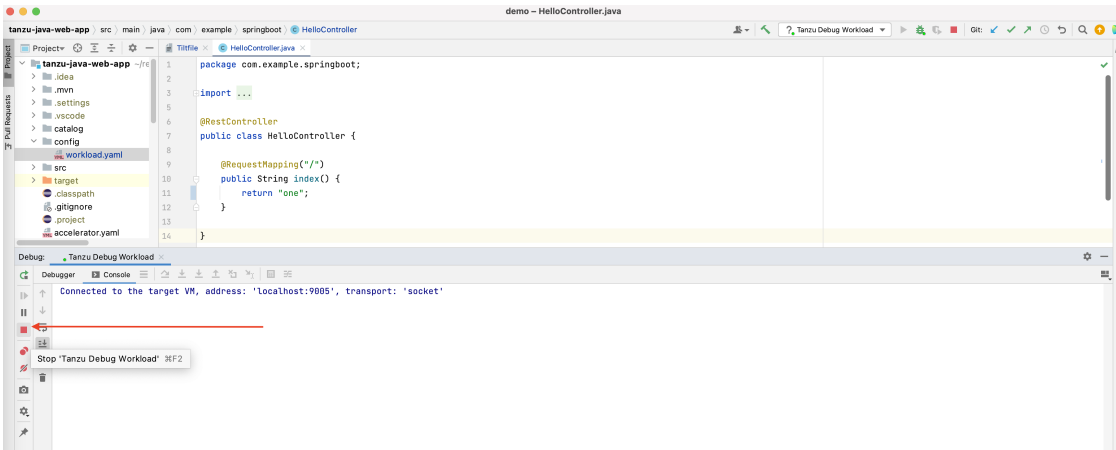


4. Ensure that the configuration parameters are set:
  - **Source Image:** This is the registry location for publishing local source code. For example, `registry.io/yourapp-source`. It must include both a registry and a project name.
  - **Local Path:** This is the path on the local file system to a directory of source code to build.
  - **Namespace:** This is the namespace that workloads are deployed into.

You can also manually create Tanzu Debug configurations by using the **Edit Configurations** IntelliJ UI.

## Stop Debugging on the Cluster

Click the stop button in the **Debug** overlay to stop debugging on the cluster.



## Live Update

See the following sections for how to use Live Update.

### Start Live Update

To start Live Update:

1. Right-click your project's Tiltfile and then click **Run 'Tanzu Live Update - ...'**.
2. Ensure that the configuration parameters are set:
  - o **Source Image:** This is the registry location for publishing local source code. For example, `registry.io/yourapp-source`. It must include both a registry and a project name.
  - o **Local Path:** This is the path on the local file system to a directory of source code to build.
  - o **Namespace:** This is the namespace that workloads are deployed into.

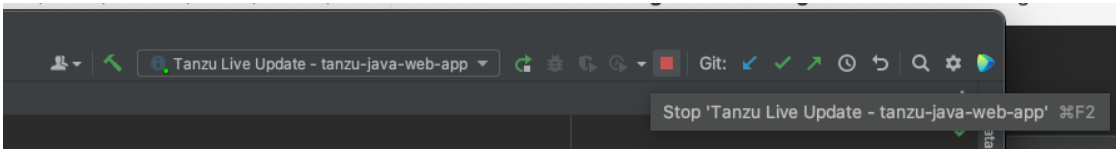


#### Note

You must compile your code before the changes are synchronized to the container. For example, `Build Project: ⌘+F9`.

### Stop Live Update

To stop Live Update, use the native controls to stop the Tanzu Live Update Run Configuration that is running.

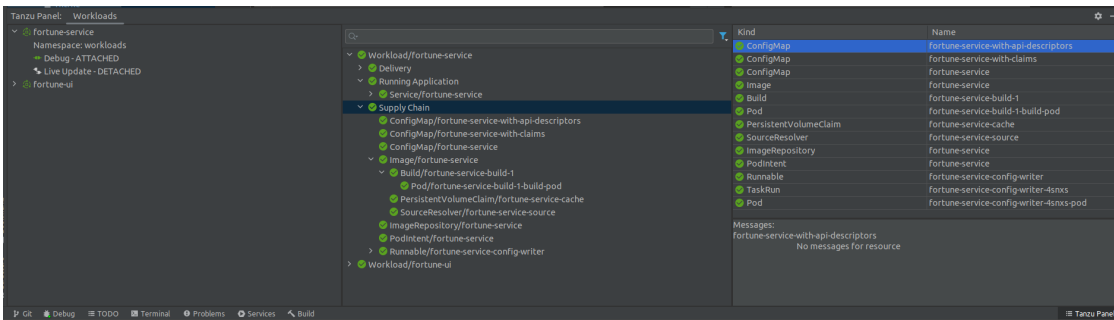


## Tanzu Workloads panel

The current state of the workloads is visible on the Tanzu Workloads panel in the bottom of the IDE window. The panel shows the current status of each workload, namespace, and cluster. It also shows whether Live Update and Debug is running, stopped, or deactivated.

Because each workload is deployed on the cluster, the activity section on the right in the Tanzu Workloads panel enables developers to visualize the supply chain, delivery, and running application

Pods. The panel displays detailed error messages on each resource and enables a developer to view and describe logs on these resources from within their IDE.



The Tanzu Workloads panel uses the cluster and namespace specified in the current kubectl context.

1. View the current context and namespace by running:

```
kubectl config get-contexts
```

2. Set a namespace for the current context by running:

```
kubectl config set-context --current --namespace=YOUR-NAMESPACE
```

## Working with microservices in a monorepo

A monorepo is single Git repository that contains multiple workloads. Each individual workload is placed in a subfolder of the main repository.

You can find an example of this in [Application Accelerator](#).

The relevant accelerator is called Spring SMTP Gateway, and you can obtain its source code as an accelerator or directly from the [application-accelerator-samples](#) GitHub repository.

This project is an example of a typical layout:

- `MONO-REPO-ROOT/`
  - `pom.xml` (parent pom)
  - `microservice-app-1/`
    - `pom.xml`
    - `mvnw` (and other mvn-related files for building the workload)
    - `Tiltfile` (supports Live Update)
    - `config`
      - `workload.yaml` (supports deploying and debugging from IntelliJ)
    - `src/` (contains source code for this microservice)
  - `microservice-app-2/`
  - ...similar layout

## Recommended structure: Microservices that can be built independently

In this example, each of the microservices can be built independently of one another. Each subfolder contains everything needed to build that workload.

This is reflected in the `source` section of `workload.yaml` by using the `subPath` attribute:

```

apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: microservice-app-1
  ...
spec:
  source:
    git:
      ref:
        branch: main
      url: https://github.com/kdvolder/sample-mono-repo.git
      subPath: microservice-app-1 # build only this
    ...

```

For setting up your own repositories, it's best practice to set up a monorepo so that each microservice can be built completely independently.

To work with these monorepos:

1. Import the monorepo as a project into IntelliJ.
2. Interact with each of the subfolders as you would interact with a project containing a single workload.

## Alternative structure: Services with build-time interdependencies

Some monorepos do not have submodules that can be independently built. Instead the `pom.xml` files of the submodules are set up to have some build-time interdependencies. For example:

- A submodule `pom.xml` can reference the parent `pom.xml` as a common place for centralized dependency management.
- A microservice submodule can reference another, as a maven dependency.
- Several microservice submodules can reference one or more shared library modules.

For these projects, make these adjustments:

1. Make `workload.yaml` point to the repository root, not a subfolder. Because submodules have dependencies on code outside of their own subfolder, all source code from the repository must be supplied to the workload builder.
2. Make `workload.yaml` specify additional buildpack arguments through environment variables. They differentiate the submodule that the build is targeting.

Both of these `workload.yaml` changes are in the following example:

```

apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: fortune-ui
  labels:
    apps.tanzu.vmware.com/workload-type: web
    app.kubernetes.io/part-of: fortune-ui
spec:
  build:
    env:
      - name: BP_MAVEN_BUILD_ARGUMENTS
        value: package -pl fortune-teller-ui -am # indicate which module to build.
      - name: BP_MAVEN_BUILT_MODULE
        value: fortune-teller-ui # indicate where to find the built artefact to de
  ploy.
  source:
    git:
      url: https://github.com/my-user/fortune-teller # repository root

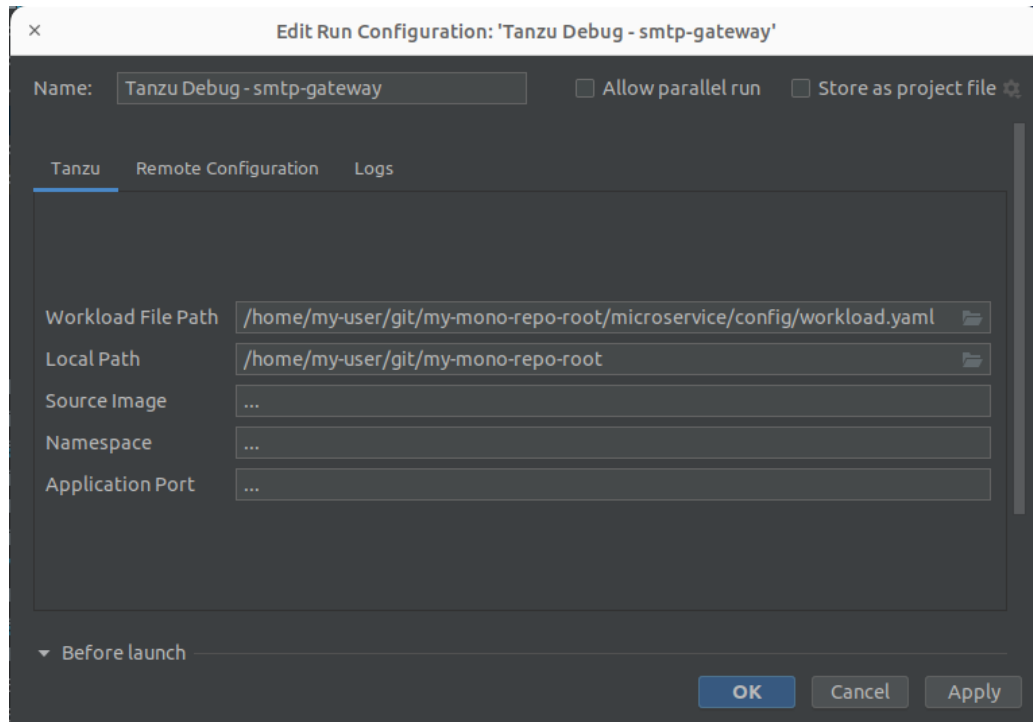
```



```
ref:
branch: main
```

For more information about these and other `BP_XXX` buildpack parameters, see the [Buildpack documentation](#).

3. Make the local path attribute in the launch configuration for each workload point to the path of the repository root. Because submodules have dependencies on code outside of their own subfolder, all source code from the repository must be supplied to the workload builder.



## Glossary of terms

This topic gives you explanations of common terms used throughout the Tanzu Developer Tools for IntelliJ documentation, and within the extension itself. Some of these terms are unique to Tanzu Application Platform, while others might have a different meaning outside of Tanzu Application Platform and are included here for clarification.

## Live Update

Live Update, facilitated by [Tilt](#), enables you to deploy your workload once, save changes to the code, and see those changes reflected in the workload running on the cluster within seconds.

## Tiltfile

The Tiltfile is a file with no extension that is required for Tilt to enable the Live Update feature. For more information about the Tiltfile, see the [Tilt documentation](#).

## Debugging on the cluster

The Tanzu Developer Tools on IntelliJ extension enables you to debug your application in an environment similar to production by debugging on your Tanzu Application Platform enabled Kubernetes cluster.

**Note**

An environment's similarity to production relies on keeping dependencies updated, among other variables.

## YAML file format

YAML is a human-readable data-serialization language. It is commonly used for configuration files. For more information, see the [YAML Wikipedia entry](#).

## workload.yaml file

The workload YAML file is a required configuration file used by the Tanzu Application Platform to specify the details of an application including its name, type, and source code URL.

## catalog-info.yaml file

The catalog-info YAML file enables the workloads created with the Tanzu Developer Tools for IntelliJ extension to be visible in the [Tanzu Application Platform GUI](#).

## Code snippet

[Code snippets](#) enable you to quickly add project files that are necessary to develop using Tanzu Application Platform by creating a template in an empty file that you fill out with the required information.

## Source image

The source image is the registry location to publish local source code, for example, [registry.io/yourapp-source](#). This must include both a registry and a project name.

## Local path

The local path value tells the Tanzu Developer Tools for IntelliJ extension which directory on your local file system to bring into the [source image](#) container image. The default local path value is the current directory where you saved the files for your open IntelliJ project.

## Kubernetes context

A Kubernetes context is a set of access parameters that contains a Kubernetes cluster, a user, and a namespace. A Kubernetes context acts like a set of coordinates that describe the target of the Kubernetes commands that you run. For more information, see the [Kubernetes documentation](#).

## Kubernetes namespace

As defined by the [Kubernetes documentation](#), in Kubernetes, namespaces provide a mechanism for isolating groups of resources within a single cluster. Names of resources need to be unique within a namespace, but not across namespaces.

## Troubleshoot Tanzu Developer Tools for IntelliJ

This topic helps you troubleshoot issues with Tanzu Developer Tools for IntelliJ.

## Tanzu Debug re-applies the workload when namespace field is empty

### Symptoms

If the `namespace` field of the debug launch configuration is empty, the workload is re-applied even if it exists on the cluster.

### Cause

Internally, workloads are gathered in the cluster in the current namespace and compared with the information that you specify. If the `namespace` field is empty, it is considered `null` and the internal checks fail.

### Solution

Do not leave the `namespace` field blank.

## Workload is wrongly re-applied because of debug configuration selected from the launch configuration drop-down menu

### Symptoms

If your debug configuration is created from the launch configuration drop-down menu, it re-applies the workload even if the workload already exists on the cluster.

### Cause

There is internal logic that is not run when debug configuration is created from the drop-down menu. However, the logic is run when debug configuration is selected from the right-click pop-up menu.

### Solution

Select debug configuration from the right-click pop-up menu.

## Unable to view workloads on the panel when connected to GKE cluster

### Symptom

When connecting to Google's GKE clusters, an error appears with the text `WARNING: the gcp auth plugin is deprecated in v1.22+, unavailable in v1.25+; use gcloud instead.`

### Cause

GKE authentication was extracted into a separate plug-in and is no longer inside the Kubernetes client or libraries.

### Solution

Download and configure the GKE authentication plug-in. For instructions, see the [Google documentation](#).

## Deactivated launch controls after running a launch configuration

### Symptom

When you run or debug a launch configuration, IntelliJ deactivates the launch controls.

### Cause

IntelliJ deactivates the launch controls to prevent other launch configurations from being launched at the same time. These controls are reactivated when the launch configuration is started. As such, starting multiple Tanzu debug and live update sessions is a synchronous activity.

## Starting a Tanzu Debug session fails with `Unable to open debugger port`

### Symptom

You try to start a Tanzu Debug session and it immediately fails with an error message similar to:

```
Error running 'Tanzu Debug - fortune-teller-fortune-service': Unable to open debugger port (localhost:5005): java.net.ConnectException "Connection refused"
```

### Cause

Old Tanzu Debug launch configurations sometimes appear to be corrupted after installing a later version of the plug-in. You can see whether this is the problem you are experiencing by opening the launch configuration:

1. Right-click `workload.yaml`.
2. Click **Modify Run Configuration...** in the menu.
3. Scroll down and expand the **Before Launch** section of the dialog.
4. Verify that it contains the two Unknown Task entries `com.vmware.tanzu.tanzuBeforeRunPortForward` and `com.vmware.tanzu.tanzuBeforeRunWorkloadApply`.

Because these two tasks are unknown causes, these steps of the debug launch are not run. This in turn means that the target application is not deployed and accessible on the expected port, which causes an error when the debugger tries to connect to it.

It might be that although the launch configuration appears corrupt when seen in the launch config editor, in fact there is no corruption. It's suspected that this problem only occurs when you install a new version of the plug-in and start using it before first restarting IntelliJ.

There is possibly an issue in the IntelliJ platform that prevents completely or correctly initializing the plug-in when the plug-in is hot-swapped into an active session instead of loaded on startup.

### Solution

Closing and restarting IntelliJ typically fixes this problem. If that doesn't work for you, delete the old corrupted launch configuration and recreate it.

## Timeout error when Live Updating

### Symptom

When you attempt to Live Update your workload, the following error message appears in the log:

```
ERROR: Build Failed: apply command timed out after 30s - see https://docs.tilt.dev/v/api.html#api.update\_settings for how to increase
```

## Cause

Kubernetes times out on upserts over 30 seconds.

## Solution

Add `update_settings (k8s_upsert_timeout_secs = 300)` to the Tiltfile. For more information, see the [Tiltfile documentation](#).

# Tanzu Panel empty when using a GKE cluster on macOS

## Symptom

On macOS, the Tanzu Panel doesn't display workloads or any other resources when using a GKE cluster. Other tools, such as the [Tanzu CLI Apps plug-in](#), display resources correctly.

## Cause

`gke-cloud-auth-plugin` is required to properly authenticate to a GKE cluster. However, when starting IntelliJ from Dock or Spotlight, environment variables set by using `.profile`, `.bash_profile`, or `.zshrc` are not available. For more information, see this [YouTrack issue](#).

This might cause `gke-cloud-auth-plugin` to be missing from `PATH` when launching IntelliJ and prevent the Tanzu Panel from reaching the cluster.

## Solution

Open IntelliJ from the CLI. Example command:

```
open /Applications/IntelliJ\ IDEA.app
```

# The Describe action in the Activity panel fails when used on PodIntent resources

## Symptom

The pop-up menu **Describe** action in the Activity panel fails when used on PodIntent resources. The error message is similar to the following:

```
Warning: conventions.apps.tanzu.vmware.com/v1alpha1 PodIntent is deprecated; \
use conventions.carto.run/v1alpha1 PodIntent instead
Error from server (NotFound): podintents.conventions.apps.tanzu.vmware.com "my-app" not found

Process finished with exit code 1
```

## Cause

When there are multiple resource types with the same kind, attempting to describe a resource of that kind without fully qualifying the API version causes this error.

## Solution

There is no workaround for this issue at present. A fix is planned for this issue in the next version.

## Tanzu panel shows workloads but doesn't show Kubernetes resources

### Symptom

The Tanzu panel shows workloads but doesn't show Kubernetes resources in the center panel of the activity pane.

### Cause

When switching the Kubernetes context, the activity pane doesn't automatically update the namespace, but the workload pane detects the new namespace. Therefore, the Tanzu panel shows workloads but doesn't show Kubernetes resources in the center panel of the activity pane.

### Solution

Restart IntelliJ to properly detect the context change.

## Live Update does not work with the Jammy `ClusterBuilder`

### Symptom

Live Update does not work when using the Jammy `ClusterBuilder`.

### Solution

A fix is planned for Tanzu Application Platform v1.5.1.

## Frequent application restarts

### Symptom

When an application is applied from IntelliJ it restarts frequently.

### Cause

An application or environment behavior is triggering the application to restart.

Observed trigger behaviors include:

- The application itself writing logs to the file system in the application directory that Live Update is watching
- Autosave being set to a very high frequency in the IDE configuration

### Solution

Prevent the trigger behavior. Example solutions include:

- Prevent 12-factor applications from writing to the file system.
- Reduce the autosave frequency to once every few minutes.

## Overview of Tanzu Application Platform GUI

Tanzu Application Platform GUI (commonly called TAP GUI) is a tool for your developers to view your applications and services running for your organization. This portal provides a central location in which you can view dependencies, relationships, technical documentation, and the service status.

Tanzu Application Platform GUI is built from the [Cloud Native Computing Foundation's project Backstage](#).

Tanzu Application Platform GUI consists of the following components:

- **Your organization catalog:**

The catalog serves as the primary visual representation of your running services (components) and applications (systems).

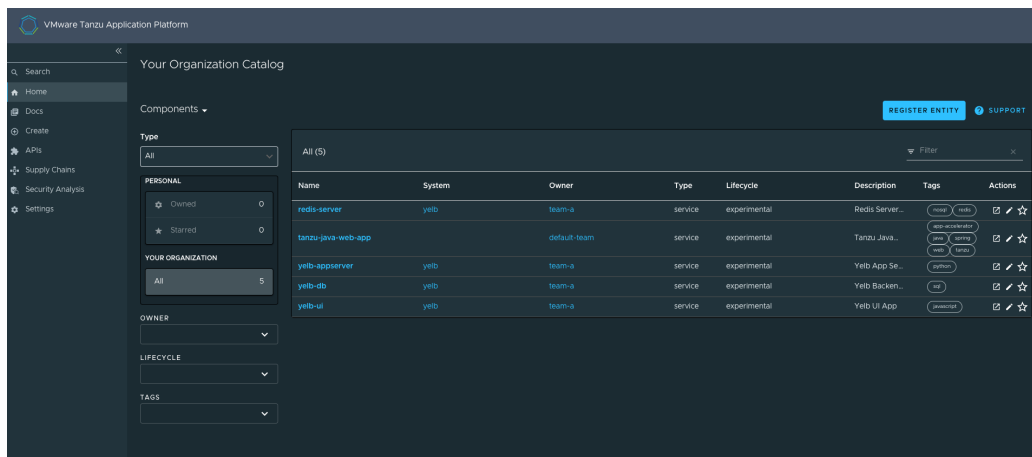
- **Tanzu Application Platform GUI plug-ins:**

These plug-ins expose capabilities regarding specific Tanzu Application Platform tools. Initially the included plug-ins are:

- Runtime Resources Visibility
- Application Live View
- Application Accelerator
- API Documentation
- Supply Chain Choreographer

- **TechDocs:**

This plug-in enables you to store your technical documentation in Markdown format in a source-code repository and display it alongside the relevant catalog entries.



- **Search:**

This plug-in enables you to search your organization's catalog, including domains, systems, components, APIs, accelerators, and TechDocs.

- **A Git repository:**

Tanzu Application Platform GUI stores the following in a Git repository:

- The structure for your application catalog.
- Your technical documentation about the catalog items, if you enable Tanzu Application Platform GUI TechDocs capabilities.

You can host the structure for your application catalog and your technical documentation in the same repository as your source code.

## Overview of Tanzu Application Platform GUI

Tanzu Application Platform GUI (commonly called TAP GUI) is a tool for your developers to view your applications and services running for your organization. This portal provides a central location in which you can view dependencies, relationships, technical documentation, and the service status.

Tanzu Application Platform GUI is built from the [Cloud Native Computing Foundation's project Backstage](#).

Tanzu Application Platform GUI consists of the following components:

- **Your organization catalog:**

The catalog serves as the primary visual representation of your running services (components) and applications (systems).

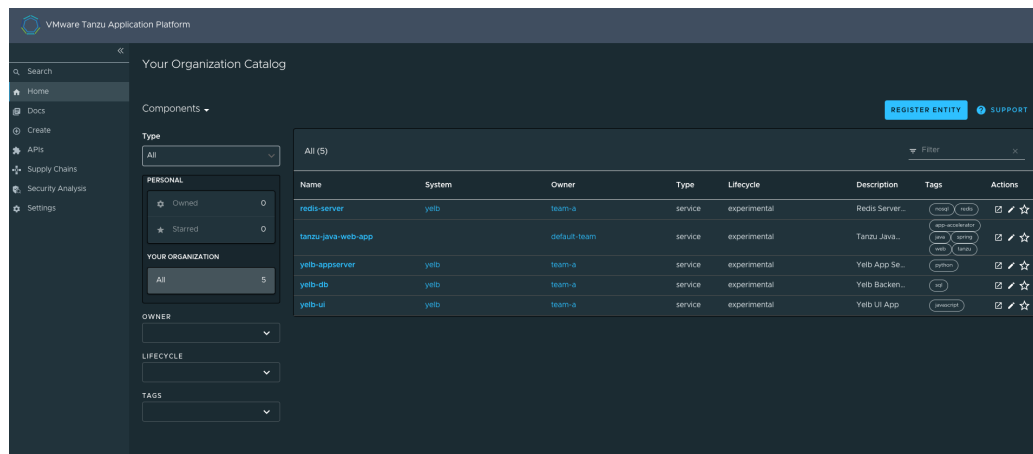
- **Tanzu Application Platform GUI plug-ins:**

These plug-ins expose capabilities regarding specific Tanzu Application Platform tools. Initially the included plug-ins are:

- Runtime Resources Visibility
- Application Live View
- Application Accelerator
- API Documentation
- Supply Chain Choreographer

- **TechDocs:**

This plug-in enables you to store your technical documentation in Markdown format in a source-code repository and display it alongside the relevant catalog entries.



- **Search:**

This plug-in enables you to search your organization's catalog, including domains, systems, components, APIs, accelerators, and TechDocs.

- **A Git repository:**

Tanzu Application Platform GUI stores the following in a Git repository:

- The structure for your application catalog.
- Your technical documentation about the catalog items, if you enable Tanzu Application Platform GUI TechDocs capabilities.



You can host the structure for your application catalog and your technical documentation in the same repository as your source code.

## Install Tanzu Application Platform GUI

This topic tells you how to install Tanzu Application Platform GUI (commonly called TAP GUI) from the Tanzu Application Platform package repository.



### Note

Follow the steps in this topic if you do not want to use a profile to install Tanzu Application Platform GUI. For more information about profiles, see [Components and installation profiles](#).

## Prerequisites

Before installing Tanzu Application Platform GUI:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see the Tanzu Application Platform [Prerequisites](#).
- Create a Git repository for Tanzu Application Platform GUI software catalogs, with a token allowing read access. Supported Git infrastructure includes:
  - GitHub
  - GitLab
  - Azure DevOps
- Install Tanzu Application Platform GUI Blank Catalog
  1. Go to the [Tanzu Application Platform](#) section of [VMware Tanzu Network](#).
  2. Under the list of available files to download, open the **tap-gui-catalogs-latest** folder.
  3. Extract Tanzu Application Platform GUI Blank Catalog to your Git repository. This serves as the configuration location for your organization's Catalog inside Tanzu Application Platform GUI.

## Procedure

To install Tanzu Application Platform GUI on a compliant Kubernetes cluster:

1. List version information for the package by running:

```
tanzu package available list tap-gui.tanzu.vmware.com --namespace tap-install
```

For example:

```
$ tanzu package available list tap-gui.tanzu.vmware.com --namespace tap-install
- Retrieving package versions for tap-gui.tanzu.vmware.com...
NAME                                VERSION    RELEASED-AT
tap-gui.tanzu.vmware.com            1.0.1     2022-01-10T13:14:23Z
```

2. (Optional) Make changes to the default installation settings by running:

```
tanzu package available get tap-gui.tanzu.vmware.com/VERSION-NUMBER --values-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 YAML:

```
ingressEnabled: true
ingressDomain: "INGRESS-DOMAIN"
app_config:
  catalog:
    locations:
      - type: url
        target: https://GIT-CATALOG-URL/catalog-info.yaml
```

Where:

- `INGRESS-DOMAIN` is the subdomain for the host name that you point at the `tanzu-shared-ingress` service's External IP address.
- `GIT-CATALOG-URL` is the path to the `catalog-info.yaml` catalog definition file. It is from either the included Blank catalog (provided as an additional download named **Blank Tanzu Application Platform GUI Catalog**) or a Backstage-compliant catalog that you've already built and posted on the Git infrastructure specified in [Adding Tanzu Application Platform GUI integrations](#).

4. Install the package by running:

```
tanzu package install tap-gui \
--package-name tap-gui.tanzu.vmware.com \
--version VERSION -n tap-install \
-f tap-gui-values.yaml
```

Where `VERSION` is the version that you want. For example, `1.0.1`.

For example:

```
$ tanzu package install tap-gui -package-name tap-gui.tanzu.vmware.com --versio
n 1.0.1 -n \
tap-install -f tap-gui-values.yaml
- Installing package 'tap-gui.tanzu.vmware.com'
| Getting package metadata for 'tap-gui.tanzu.vmware.com'
| Creating service account 'tap-gui-default-sa'
| Creating cluster admin role 'tap-gui-default-cluster-role'
| Creating cluster role binding 'tap-gui-default-cluster-rolebinding'
| Creating secret 'tap-gui-default-values'
- Creating package resource
- Package install status: Reconciling

Added installed package 'tap-gui' in namespace 'tap-install'
```

5. Verify that the package installed by running:

```
tanzu package installed get tap-gui -n tap-install
```

For example:

```
$ tanzu package installed get tap-gui -n tap-install
| Retrieving installation details for cc...
NAME:                tap-gui
PACKAGE-NAME:        tap-gui.tanzu.vmware.com
```

```

PACKAGE-VERSION:      1.0.1
STATUS:               Reconcile succeeded
CONDITIONS:           [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:

```

Verify that `STATUS` is `Reconcile succeeded`.

- To access Tanzu Application Platform GUI, use the service you exposed in the `service_type` field in the values file.

## Runtime configuration options for Tanzu Application Platform GUI

You can provide a series of options to the Tanzu Application Platform GUI (commonly called TAP GUI) package to configure it and do some basic [runtime customization](#).

## Identify the Tanzu Application Platform GUI version you have available

From the Tanzu CLI, discover the Tanzu Application Platform GUI package version that is available to configure by running:

```
tanzu package available get tap-gui.tanzu.vmware.com -n INSTALL-NAMESPACE
```

Where `INSTALL-NAMESPACE` is the namespace in which you configured the Tanzu Application Platform installation. In most cases the namespace is `tap-install`.

For example:

```

$ tanzu package available get tap-gui.tanzu.vmware.com -n tap-install

NAME:                tap-gui.tanzu.vmware.com
DISPLAY-NAME:        Tanzu Application Platform GUI
CATEGORIES:
SHORT-DESCRIPTION:   web app graphical user interface for Tanzu Application Platform
LONG-DESCRIPTION:   web app graphical user interface for Tanzu Application Platform
PROVIDER:            VMware
MAINTAINERS:         - name: VMware
SUPPORT-DESCRIPTION: https://tanzu.vmware.com/support

VERSION  RELEASED-AT
1.7.6    2023-10-17 00:25:21 +0000 UTC

```

## Display the possible values options for Tanzu Application Platform GUI

From the Tanzu CLI, identify possible values options for Tanzu Application Platform GUI by running:

```
tanzu package available get tap-gui.tanzu.vmware.com/VERSION --values-schema -n INSTALL-NAMESPACE
```

Where:

- `VERSION` is the Tanzu Application Platform GUI package version you learned earlier
- `INSTALL-NAMESPACE` is the namespace in which you configured the Tanzu Application Platform installation. In most cases the namespace is `tap-install`.

For example:

```
$ tanzu package available get tap-gui.tanzu.vmware.com/1.7.6 --values-schema -n tap-in
stall

KEY  DEFAULT  TYPE
DESCRIPTION
#Details of all the possible configuration values
...
```

## Customize the Tanzu Application Platform GUI portal

This section describes how to customize the Tanzu Application Platform GUI portal.

### Customize branding

To customize the branding in your portal, you can choose the name of the portal and the logo for it. To make these customizations:

1. Provide additional configuration parameters to the `app_config` section of `tap-values.yaml`:

```
tap_gui:
  app_config:
    customize:
      custom_logo: 'BASE-64-IMAGE'
      custom_name: 'PORTAL-NAME'
```

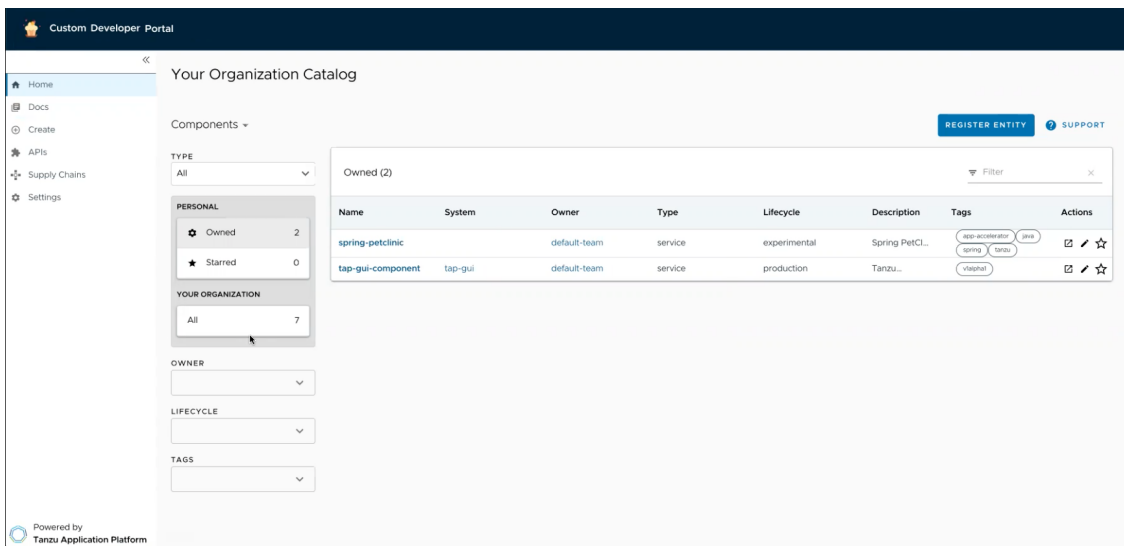
Where:

- o `BASE-64-IMAGE` is the image encoded in base64. A 512-pixel by 512-pixel PNG image with a transparent background is optimal.
- o `PORTAL-NAME` is the name of your portal, such as `Our Custom Developer Experience Portal`.

2. Reinstall your Tanzu Application Platform GUI package by following steps in [Upgrading Tanzu Application Platform](#).

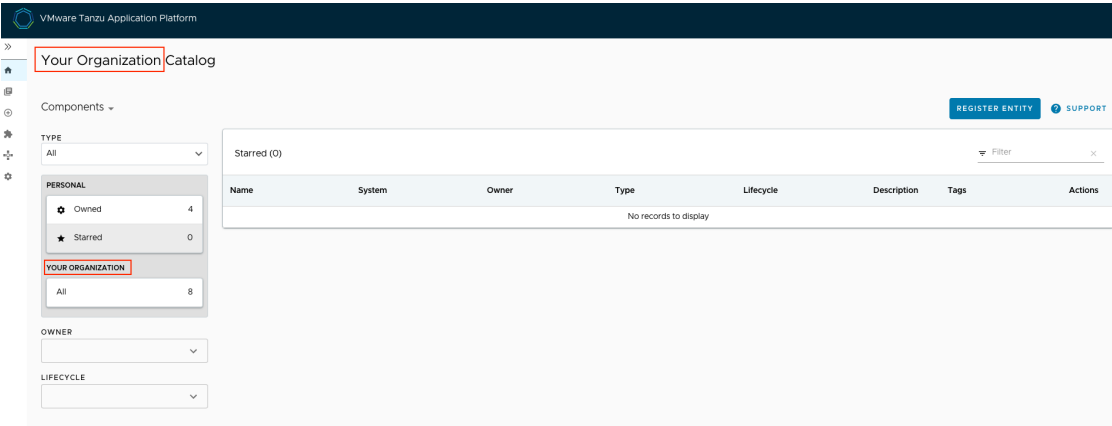
After the updated values configuration file is applied in Tanzu Application Platform GUI, you see the customized version of your portal.

If there is an error in any of the supplied images encoded in base64 or in your choice of portal name, Tanzu Application Platform GUI reverts to the original branding template.



## Customize the Software Catalog page

You can customize the name of your organization on the Software Catalog page of Tanzu Application Platform GUI portal. By default, the portal displays **Your Organization** next to **Catalog** and in the selection box.



## Customize the name of the organization

To customize the name of the organization for the software catalog in your portal:

1. Provide additional configuration parameters to the `app_config` section of your `tap-values.yaml` file:

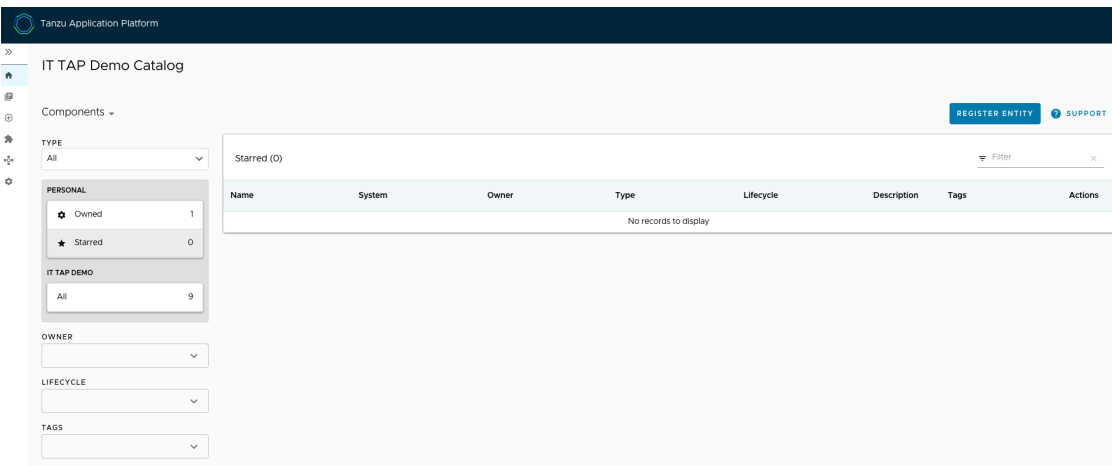
```
tap_gui:
  app_config:
    organization:
      name: 'ORG-NAME'
```

Where `ORG-NAME` is the name of your organization for the software catalog, such as `Our Organization Name`. You don't need to add `Catalog` to the `ORG-NAME`.

2. Reinstall your Tanzu Application Platform GUI package by following the steps in [Upgrading Tanzu Application Platform](#).

After the updated values configuration file is applied in Tanzu Application Platform GUI, you see the customized version of your portal.

If there is an error in the provided configuration parameters, Tanzu Application Platform GUI reverts to the original organization name.



## Prevent changes to the software catalog

You can deactivate the **Register Entity** button to prevent a user from making changes to the software catalog, including registering and deregistering locations. To do so, add `readonly: true` to the `catalog` section in `tap-values.yaml`, as in this example:

```
tap_gui:
  app_config:
    catalog:
      readonly: true
```

## Customize the Authentication page

To customize the portal name on the **Authentication** page and the name of the browser tab for Tanzu Application Platform GUI:

1. Provide additional configuration parameters to the `app_config` section of your `tap-values.yaml` file:

```
tap_gui:
  app_config:
    app:
      title: 'CUSTOM-TAB-NAME'
```

Where `CUSTOM-TAB-NAME` is the name on the Authentication page and the browser tab of your portal, such as `Our Organization Full Name`.

2. Reinstall your Tanzu Application Platform GUI package by following the steps in [Upgrading Tanzu Application Platform](#).

After the updated values configuration file is applied in Tanzu Application Platform GUI, you see the customized version of your portal.

## Customize the default view

You can set your default route when the user is accessing your portal. Without this customization, when the user accesses the Tanzu Application Platform GUI URL, it displays the list of owned components of the software catalog.

To change the default view:

1. Provide additional configuration parameters to the `app_config` section of your `tap-values.yaml` file:

```
tap_gui:
  app_config:
    customize:
      default_route: 'YOUR-PREFERRED-ROUTE'
```

Where `YOUR-PREFERRED-ROUTE` is the path to the route that the portal uses by default. For example, you can type `/catalog?filters%5Bkind%5D=component&filters%5Buser%5D=all` to show all components of the software catalog instead of defaulting to owned components. As another example, you can type `/create` to show Application Accelerator when the portal starts.



### Caution

Tanzu Application Platform GUI redirects you to `tap-gui.INGRESS-DOMAIN/YOUR-PREFERRED-ROUTE` even if there is an error in `YOUR-PREFERRED-ROUTE`.

2. Reinstall your Tanzu Application Platform GUI package by following the steps in [Upgrading Tanzu Application Platform](#).

After the updated values configuration file is applied in Tanzu Application Platform GUI, you see the customized version of your portal.

## Customize the Tanzu Application Platform GUI portal

This section describes how to customize the Tanzu Application Platform GUI portal.

### Customize branding

To customize the branding in your portal, you can choose the name of the portal and the logo for it. To make these customizations:

1. Provide additional configuration parameters to the `app_config` section of `tap-values.yaml`:

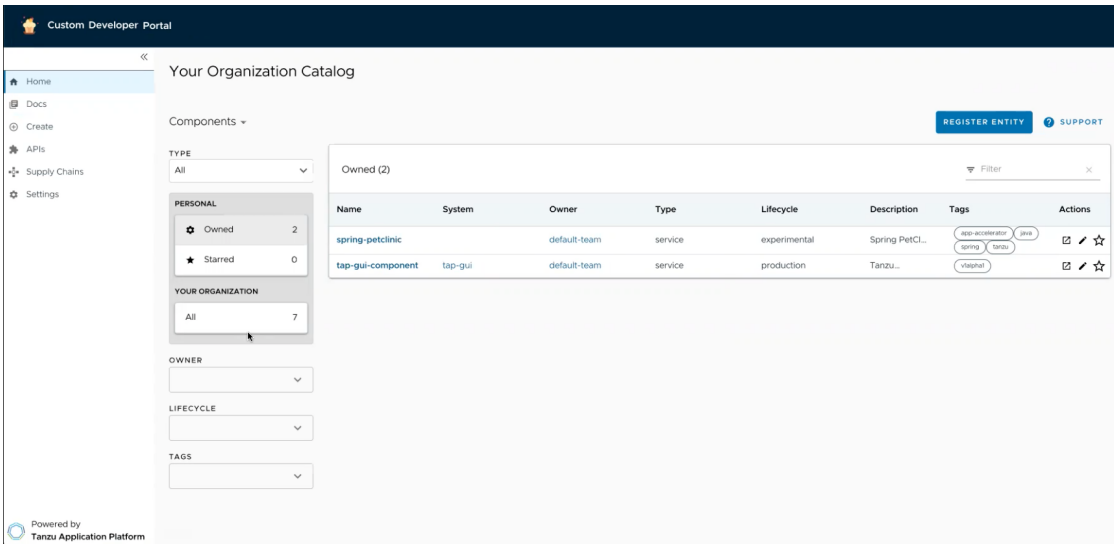
```
tap_gui:
  app_config:
    customize:
      custom_logo: 'BASE-64-IMAGE'
      custom_name: 'PORTAL-NAME'
```

Where:

- `BASE-64-IMAGE` is the image encoded in base64. A 512-pixel by 512-pixel PNG image with a transparent background is optimal.
  - `PORTAL-NAME` is the name of your portal, such as `Our Custom Developer Experience Portal`.
2. Reinstall your Tanzu Application Platform GUI package by following steps in [Upgrading Tanzu Application Platform](#).

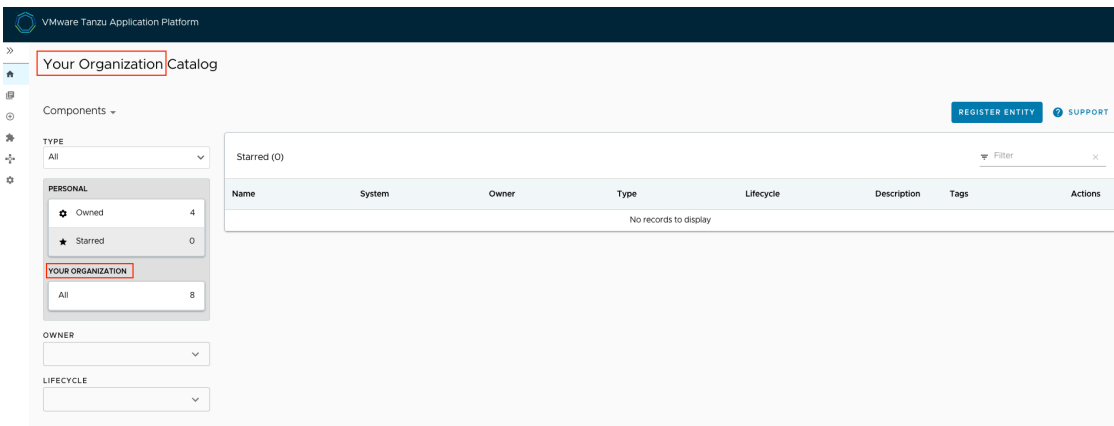
After the updated values configuration file is applied in Tanzu Application Platform GUI, you see the customized version of your portal.

If there is an error in any of the supplied images encoded in base64 or in your choice of portal name, Tanzu Application Platform GUI reverts to the original branding template.



## Customize the Software Catalog page

You can customize the name of your organization on the Software Catalog page of Tanzu Application Platform GUI portal. By default, the portal displays **Your Organization** next to **Catalog** and in the selection box.



## Customize the name of the organization

To customize the name of the organization for the software catalog in your portal:

1. Provide additional configuration parameters to the `app_config` section of your `tap-values.yaml` file:

```
tap_gui:
  app_config:
    organization:
      name: 'ORG-NAME'
```

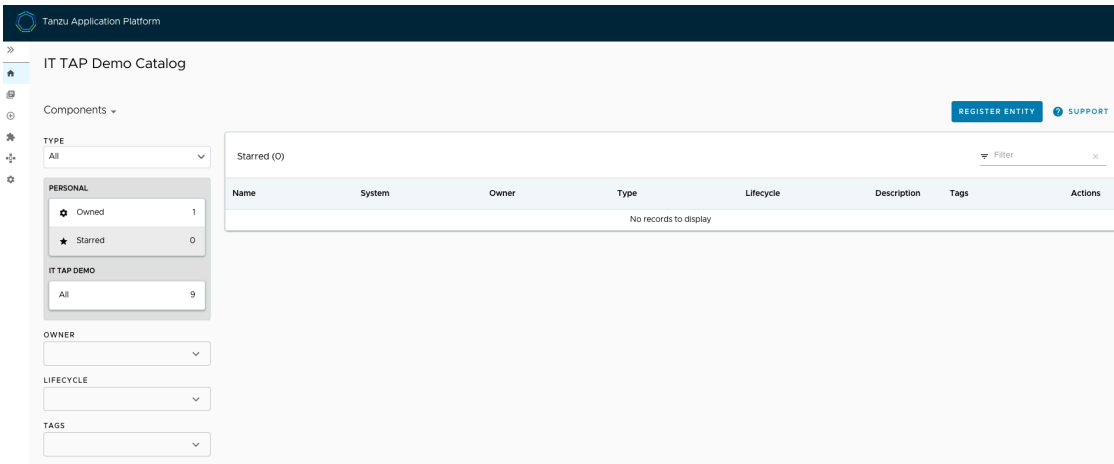
Where `ORG-NAME` is the name of your organization for the software catalog, such as `Our Organization Name`. You don't need to add `Catalog` to the `ORG-NAME`.

2. Reinstall your Tanzu Application Platform GUI package by following the steps in [Upgrading Tanzu Application Platform](#).

After the updated values configuration file is applied in Tanzu Application Platform GUI, you see the customized version of your portal.



If there is an error in the provided configuration parameters, Tanzu Application Platform GUI reverts to the original organization name.



## Prevent changes to the software catalog

You can deactivate the **Register Entity** button to prevent a user from making changes to the software catalog, including registering and deregistering locations. To do so, add `readonly: true` to the `catalog` section in `tap-values.yaml`, as in this example:

```
tap_gui:
  app_config:
    catalog:
      readonly: true
```

## Customize the Authentication page

To customize the portal name on the **Authentication** page and the name of the browser tab for Tanzu Application Platform GUI:

1. Provide additional configuration parameters to the `app_config` section of your `tap-values.yaml` file:

```
tap_gui:
  app_config:
    app:
      title: 'CUSTOM-TAB-NAME'
```

Where `CUSTOM-TAB-NAME` is the name on the Authentication page and the browser tab of your portal, such as `Our Organization Full Name`.

2. Reinstall your Tanzu Application Platform GUI package by following the steps in [Upgrading Tanzu Application Platform](#).

After the updated values configuration file is applied in Tanzu Application Platform GUI, you see the customized version of your portal.

## Customize the default view

You can set your default route when the user is accessing your portal. Without this customization, when the user accesses the Tanzu Application Platform GUI URL, it displays the list of owned components of the software catalog.

To change the default view:

1. Provide additional configuration parameters to the `app_config` section of your `tap-values.yaml` file:

```
tap_gui:
  app_config:
    customize:
      default_route: 'YOUR-PREFERRED-ROUTE'
```

Where `YOUR-PREFERRED-ROUTE` is the path to the route that the portal uses by default. For example, you can type `/catalog?filters%5Bkind%5D=component&filters%5Buser%5D=all` to show all components of the software catalog instead of defaulting to owned components. As another example, you can type `/create` to show Application Accelerator when the portal starts.



#### Caution

Tanzu Application Platform GUI redirects you to `tap-gui.INGRESS-DOMAIN/YOUR-PREFERRED-ROUTE` even if there is an error in `YOUR-PREFERRED-ROUTE`.

2. Reinstall your Tanzu Application Platform GUI package by following the steps in [Upgrading Tanzu Application Platform](#).

After the updated values configuration file is applied in Tanzu Application Platform GUI, you see the customized version of your portal.

## Customize the Support menu

This topic describes how to customize the support menu.

### Overview

Many important pages of Tanzu Application Platform GUI have a **Support** button that displays a pop-out menu. This menu contains a one-line description of the page the user is looking at, and a list of support item groupings.

## All your software catalog entities



Contact Support  
[Tanzu Support Page](#)



Documentation  
[Tanzu Application Platform Documentation](#)

[CLOSE](#)

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
```

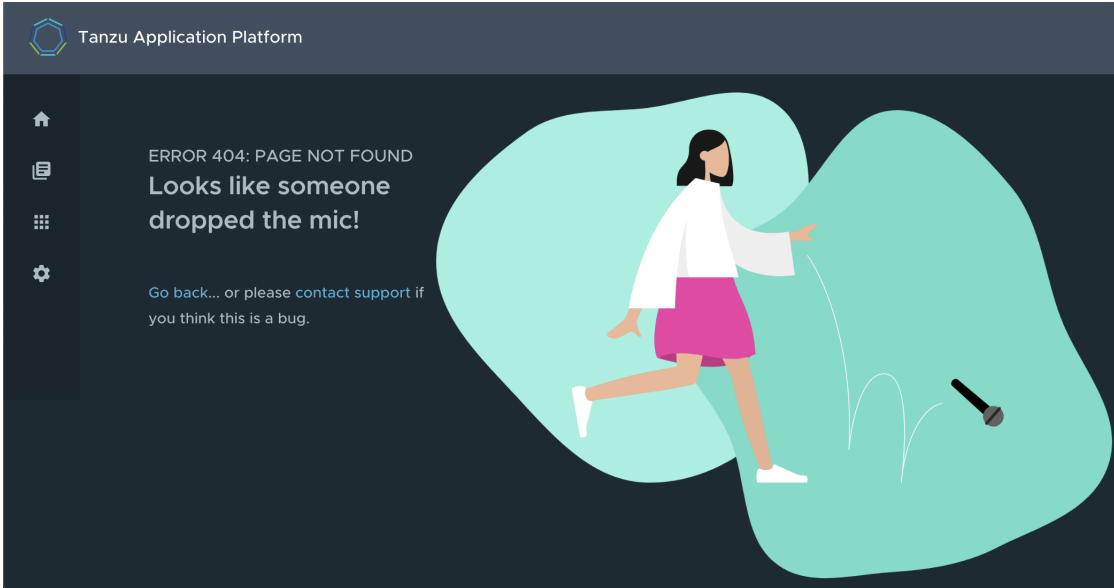
## Structure of the support configuration

### 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.



## 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.

## Access Tanzu Application Platform GUI

This topic tells you how to access Tanzu Application Platform GUI (commonly called TAP GUI) by using one of the following methods:

- Access with the LoadBalancer method (default)
- Access with the shared Ingress method

### Access with the LoadBalancer method (default)

1. Verify that you specified the `service_type` for Tanzu Application Platform GUI in `tap-values.yaml`, as in this example:

```
tap_gui:
  service_type: LoadBalancer
```

2. Obtain the external IP address of your LoadBalancer by running:

```
kubectl get svc -n tap-gui
```

3. Access Tanzu Application Platform GUI by using the external IP address with the default port of 7000. It has the following form:

```
http://EXTERNAL-IP:7000
```

Where `EXTERNAL-IP` is the external IP address of your LoadBalancer.

### Access with the shared Ingress method

The Ingress method of access for Tanzu Application Platform GUI uses the shared `tanzu-system-ingress` instance of Contour that is installed as part of the Profile installation.

1. The Ingress method of access requires that you have a DNS host name that you can point at the External IP address of the `envoy` service that the shared `tanzu-system-ingress` uses. Retrieve this IP address by running:

```
kubectl get service envoy -n tanzu-system-ingress
```

This returns a value similar to this example:

```
$ kubectl get service envoy -n tanzu-system-ingress
NAME      TYPE           CLUSTER-IP      EXTERNAL-IP      PORT(S)
AGE
envoy     LoadBalancer   10.0.242.171    40.118.168.232  80:31389/TCP,443:31780/TCP
CP       27h
```

The IP address in the `EXTERNAL-IP` field is the one that you point a DNS host record to. Tanzu Application Platform GUI prepends `tap-gui` to your provided subdomain. This makes the final host name `tap-gui.YOUR-SUBDOMAIN`. You use this host name in the appropriate fields in the `tap-values.yaml` file mentioned later.

- Specify parameters in `tap-values.yaml` related to Ingress. For example:

```
shared:
  ingress_domain: "example.com"
```

- Update your other host names in the `tap_gui` section of your `tap-values.yaml` with the new host name. For example:

```
shared:
  ingress_domain: "example.com"

tap_gui:
  # Existing tap-values.yaml above
  app_config:
    app:
      baseUrl: http://tap-gui.example.com # No port needed with Ingress
    integrations:
      github: # Other are integrations available
        - host: github.com
          token: GITHUB-TOKEN
    catalog:
      locations:
        - type: url
          target: https://GIT-CATALOG-URL/catalog-info.yaml
    backend:
      baseUrl: http://tap-gui.example.com # No port needed with Ingress
    cors:
      origin: http://tap-gui.example.com # No port needed with Ingress
```

- 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, `1.4.13`.

- 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](#).

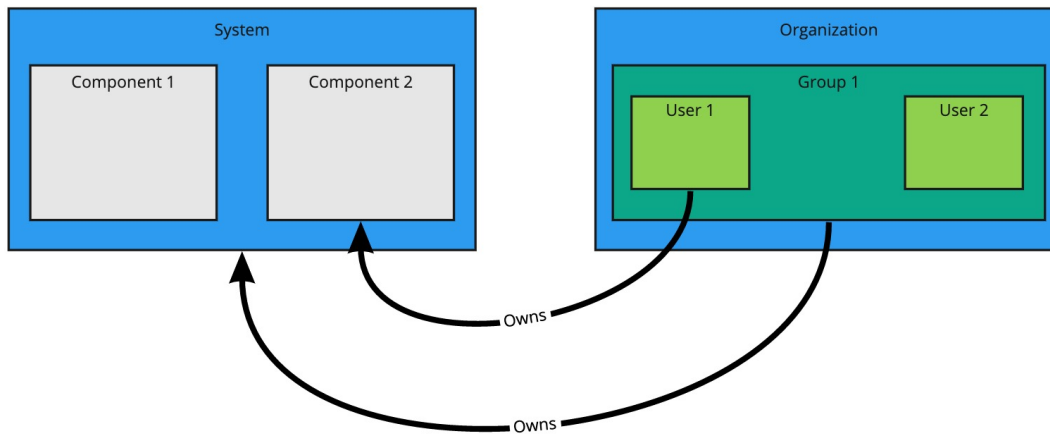
## Adding catalog entities

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.

The organization contains Group 1, and Group 1 contains Users 1 and 2. System contains Components 1 and 2. User 1 owns Component 2. Group 1 owns System.



## Users and groups

A user entity describes a specific person and is used for identity purposes. Users are members of one or more groups. A group entity describes an organizational team or unit.

Users and groups have different descriptor requirements in their descriptor files:

- User descriptor files require `apiVersion`, `kind`, `metadata.name`, and `spec.memberOf`.
- Group descriptor files require `apiVersion`, `kind`, and `metadata.name`. They also require `spec.type` and `spec.children` where `spec.children` is another group.

To link a logged-in user to a user entity, include the optional `spec.profile.email` field.

Sample user entity:

```
apiVersion: backstage.io/v1alpha1
kind: User
metadata:
  name: default-user
spec:
  profile:
    displayName: Default User
    email: guest@example.com
    picture: https://avatars.dicebear.com/api/avataaars/guest@example.com.svg?background=%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: []

```

For more information about user entities and group entities, see the [Backstage documentation](#).

## Systems

A system entity is a collection of resources and components.

System descriptor files require values for `apiVersion`, `kind`, `metadata.name`, and also `spec.owner` where `spec.owner` is a user or group.

A system has components when components specify the system name in the field `spec.system`.

Sample system entity:

```

apiVersion: backstage.io/v1alpha1
kind: System
metadata:
  name: backstage
  description: Tanzu Application Platform GUI System
spec:
  owner: default-team

```

For more information about system entities, see the [Backstage documentation](#).

## Components

A component describes a software component, or what might be described as a unit of software.

Component descriptor files require values for `apiVersion`, `kind`, `metadata.name`, `spec.type`, `spec.lifecycle`, and `spec.owner`.

Some useful optional fields are `spec.system` and `spec.subcomponentOf`, both of which link a component to an entity that it is part of.

```

apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: backstage-component
  description: Tanzu 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

```

For more information about component entities, see the [Backstage documentation](#).

## Update software catalogs



The following procedures describe how to update software catalogs.

## Register components

To update your software catalog with new entities without re-deploying the entire `tap-gui` package:

1. Go to your **Software Catalog** page.
2. Click **Register Entity** at the top-right of the page.
3. Enter the full path to link to an existing entity file and start tracking your entity.
4. Import the entities and view them in your **Software Catalog** page.

## Deregister components

To deregister an entity:

1. Go to your **Software Catalog** page.
2. Select the entity to deregister, such as component, group, or user.
3. Click the three dots at the top-right of the page and then click **Unregister...**

## Add or change organization catalog locations

To add or change organization catalog locations, you can use static configuration or you can use `GitLabDiscoveryProcessor` to discover and register catalog entities that match the configured path.

### Use static configuration

To use static configuration to add or change catalog locations:

1. Update components by changing the catalog location in either the `app_config` section of `tap-gui-values.yaml` or the custom values file you used when installing. For example:

```
tap_gui:
  app_config:
    catalog:
      locations:
        - type: url
          target: UPDATED-CATALOG-LOCATION
```

2. Register components by adding the new catalog location in either the `app_config` section of `tap-gui-values.yaml` or the custom values file you used when installing. For example:

```
tap_gui:
  app_config:
    catalog:
      locations:
        - type: url
          target: EXISTING-CATALOG-LOCATION
        - type: url
          target: EXTRA-CATALOG-LOCATION
```

When targeting GitHub, don't write the raw URL. Instead, use the URL that 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](#).

### Use `GitLabDiscoveryProcessor`

To use `GitLabDiscoveryProcessor` to discover and register catalog entities:

1. Use `type: gitlab-discovery` to make `GitLabDiscoveryProcessor` crawl the GitLab instance to discover and register catalog entities that match the configured path. For more information, see the [Backstage documentation](#).
2. Update the package to include the catalog:
  - If you installed Tanzu Application Platform GUI by using a profile, run:

```
tanzu package installed update tap \
--package-name tap.tanzu.vmware.com \
--version PACKAGE-VERSION \
--values-file tap-values.yaml \
--namespace tap-install
```

- If you installed Tanzu Application Platform GUI as an individual package, run:

```
tanzu package installed update tap-gui \
--package-name tap-gui.tanzu.vmware.com \
--version PACKAGE-VERSION \
--values-file tap-gui-values.yaml \
--namespace tap-install
```

3. Verify the status of this update by running:

```
tanzu package installed list -n tap-install
```

## Install demo apps and their catalogs

To set up one of the demos, you can choose a blank catalog or a sample catalog.

### Yelb system

The [Yelb](#) demo catalog in GitHub includes all the components that make up the Yelb system and the default Backstage components.

#### Install Yelb

1. Download the necessary file for running the Yelb application itself from [GitHub](#).
2. Install the application on the Kubernetes cluster that you used for Tanzu Application Platform. Preserve the metadata labels on the Yelb application objects.

## Install the Yelb catalog

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`.

## View resources on multiple clusters in Tanzu Application Platform GUI

You can configure Tanzu Application Platform GUI (commonly called TAP GUI) to retrieve Kubernetes object details from multiple clusters and then surface those details in the various Tanzu Application Platform GUI plug-ins.



### Important

In this topic the terms `Build`, `Run`, and `View` describe the cluster's roles and distinguish which steps to apply to which cluster.

`Build` clusters are where the code is built and packaged, ready to be run.

`Run` clusters are where the Tanzu Application Platform workloads themselves run.

`View` clusters are where the Tanzu Application Platform GUI is run from.

In multicluster configurations, these can be separate clusters. However, in many configurations these can also be the same cluster.

## Set up a Service Account to view resources on a cluster

To view resources on the `Build` or `Run` clusters, create a service account on the `View` cluster that can `get`, `watch`, and `list` resources on those clusters.

You first create a `ClusterRole` with these rules and a `ServiceAccount` in its own `Namespace`, and then bind the `ClusterRole` to the `ServiceAccount`. Depending on your topology, not every cluster has all of the following objects. For example, the `Build` cluster doesn't have any of the `-serving.knative.dev` objects, by design, because it doesn't run the workloads themselves. You can edit the following object lists to reflect your topology.

To set up a Service Account to view resources on a cluster:

1. Copy this YAML content into a file called `tap-gui-viewer-service-account-rbac.yaml`.

```
apiVersion: v1
kind: Namespace
metadata:
  name: tap-gui
---
apiVersion: v1
kind: ServiceAccount
metadata:
  namespace: tap-gui
  name: tap-gui-viewer
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: tap-gui-read-k8s
subjects:
- kind: ServiceAccount
```

```

namespace: tap-gui
name: tap-gui-viewer
roleRef:
  kind: ClusterRole
  name: k8s-reader
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: k8s-reader
rules:
- apiGroups: ['']
  resources: ['pods', 'pods/log', 'services', 'configmaps', 'limitranges']
  verbs: ['get', 'watch', 'list']
- apiGroups: ['metrics.k8s.io']
  resources: ['pods']
  verbs: ['get', 'watch', 'list']
- apiGroups: ['apps']
  resources: ['deployments', 'replicasets', 'statefulsets', 'daemonsets']
  verbs: ['get', 'watch', 'list']
- apiGroups: ['autoscaling']
  resources: ['horizontalpodautoscalers']
  verbs: ['get', 'watch', 'list']
- apiGroups: ['networking.k8s.io']
  resources: ['ingresses']
  verbs: ['get', 'watch', 'list']
- apiGroups: ['networking.internal.knative.dev']
  resources: ['serverlessservices']
  verbs: ['get', 'watch', 'list']
- apiGroups: ['autoscaling.internal.knative.dev']
  resources: ['podautoscalers']
  verbs: ['get', 'watch', 'list']
- apiGroups: ['serving.knative.dev']
  resources:
  - configurations
  - revisions
  - routes
  - services
  verbs: ['get', 'watch', 'list']
- apiGroups: ['carto.run']
  resources:
  - clusterconfigtemplates
  - clusterdeliveries
  - clusterdeploymenttemplates
  - clusterimagetemplates
  - clusterruntemplates
  - clustersourcetemplates
  - clustersupplychains
  - clustertemplates
  - deliverables
  - runnables
  - workloads
  verbs: ['get', 'watch', 'list']
- apiGroups: ['source.toolkit.fluxcd.io']
  resources:
  - gitrepositories
  verbs: ['get', 'watch', 'list']
- apiGroups: ['source.apps.tanzu.vmware.com']
  resources:
  - imagerepositories
  - mavenartifacts
  verbs: ['get', 'watch', 'list']
- apiGroups: ['conventions.apps.tanzu.vmware.com']
  resources:
  - podintents

```

```

  verbs: ['get', 'watch', 'list']
- apiGroups: ['kpack.io']
  resources:
  - images
  - builds
  verbs: ['get', 'watch', 'list']
- apiGroups: ['scanning.apps.tanzu.vmware.com']
  resources:
  - sourcescans
  - imagescans
  - scanpolicies
  - scantemplates
  verbs: ['get', 'watch', 'list']
- apiGroups: ['tekton.dev']
  resources:
  - taskruns
  - pipelineruns
  verbs: ['get', 'watch', 'list']
- apiGroups: ['kappctrl.k14s.io']
  resources:
  - apps
  verbs: ['get', 'watch', 'list']
- apiGroups: [ 'batch' ]
  resources: [ 'jobs', 'cronjobs' ]
  verbs: [ 'get', 'watch', 'list' ]
- apiGroups: ['conventions.carto.run']
  resources:
  - podintents
  verbs: ['get', 'watch', 'list']

```

This YAML content creates [Namespace](#), [ServiceAccount](#), [ClusterRole](#), and [ClusterRoleBinding](#).

2. On the [Build](#) and [Run](#) clusters, create [Namespace](#), [ServiceAccount](#), [ClusterRole](#), and [ClusterRoleBinding](#) by running:

```
kubectl create -f tap-gui-viewer-service-account-rbac.yaml
```

3. Again, on the [Build](#) and [Run](#) clusters, discover the [CLUSTER\\_URL](#) and [CLUSTER\\_TOKEN](#) values.

#### v1.23 or earlier Kubernetes cluster

If you're watching a v1.23 or earlier Kubernetes cluster, run:

```

CLUSTER_URL=$(kubectl config view --minify -o jsonpath='{.clusters[0].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

```

#### v1.24 or later Kubernetes cluster

If you're watching a v1.24 or later Kubernetes cluster, run:

```

CLUSTER_URL=$(kubectl config view --minify -o jsonpath='{.clusters[0].cluster.server}')

kubectl apply -f - <<EOF
apiVersion: v1

```

```

kind: Secret
metadata:
  name: tap-gui-viewer
  namespace: tap-gui
  annotations:
    kubernetes.io/service-account.name: tap-gui-viewer
type: kubernetes.io/service-account-token
EOF

CLUSTER_TOKEN=$(kubectl -n tap-gui get secret tap-gui-viewer -o=json \
| jq -r '.data["token"]' \
| base64 --decode)

echo CLUSTER_URL: $CLUSTER_URL
echo CLUSTER_TOKEN: $CLUSTER_TOKEN

```



### Note

You can create a short-lived token with the `kubectl create token` command if that is the preferred method. This method requires frequent token rotation.

- (Optional) Configure the Kubernetes client to verify the TLS certificates presented by a cluster's API server. To do this, discover `CLUSTER_CA_CERTIFICATES` by running:

```

CLUSTER_CA_CERTIFICATES=$(kubectl config view --raw -o jsonpath='{.clusters[?(@.name=="CLUSTER-NAME")].cluster.certificate-authority-data}')

echo CLUSTER_CA_CERTIFICATES: $CLUSTER_CA_CERTIFICATES

```

Where `CLUSTER-NAME` is your cluster name.

- Record the `Build` and `Run` clusters' `CLUSTER_URL` and `CLUSTER_TOKEN` values for when you [Update Tanzu Application Platform GUI to view resources on multiple clusters](#) later.

## Update Tanzu Application Platform GUI to view resources on multiple clusters

The clusters must be identified to Tanzu Application Platform GUI with the `ServiceAccount` token and the cluster Kubernetes control plane URL.

You must add a `kubernetes` section to the `app_config` section in the `tap-values.yaml` file that Tanzu Application Platform used when you installed it. This section must have an entry for each `Build` and `Run` cluster that has resources to view.

To do so:

- Copy this YAML content into `tap-values.yaml`:

```

tap_gui:
  ## Previous configuration above
  app_config:
    kubernetes:
      serviceLocatorMethod:
        type: 'multiTenant'
      clusterLocatorMethods:
        - type: 'config'
          clusters:
            ## Cluster 1
            - url: CLUSTER-URL

```

```

name: CLUSTER-NAME
authProvider: serviceAccount
serviceAccountToken: "CLUSTER-TOKEN"
skipTLSVerify: true
skipMetricsLookup: true
## Cluster 2+
- url: CLUSTER-URL
  name: CLUSTER-NAME
  authProvider: serviceAccount
  serviceAccountToken: "CLUSTER-TOKEN"
  skipTLSVerify: true
  skipMetricsLookup: true

```

Where:

- `CLUSTER-URL` is the value you discovered earlier.
- `CLUSTER-TOKEN` is the value you discovered earlier.
- `CLUSTER-NAME` is a unique name of your choice.

If there are resources to view on the `View` cluster that hosts Tanzu Application Platform GUI, add an entry to `clusters` for it as well.

If you would like the Kubernetes client to verify the TLS certificates presented by a cluster's API server, set the following properties for the cluster:

```

skipTLSVerify: false
caData: CLUSTER-CA-CERTIFICATES

```

Where `CLUSTER-CA-CERTIFICATES` is the value you discovered earlier.

2. Update the `tap` package by running this command:

```
tanzu package installed update tap -n tap-install --values-file tap-values.yaml
```

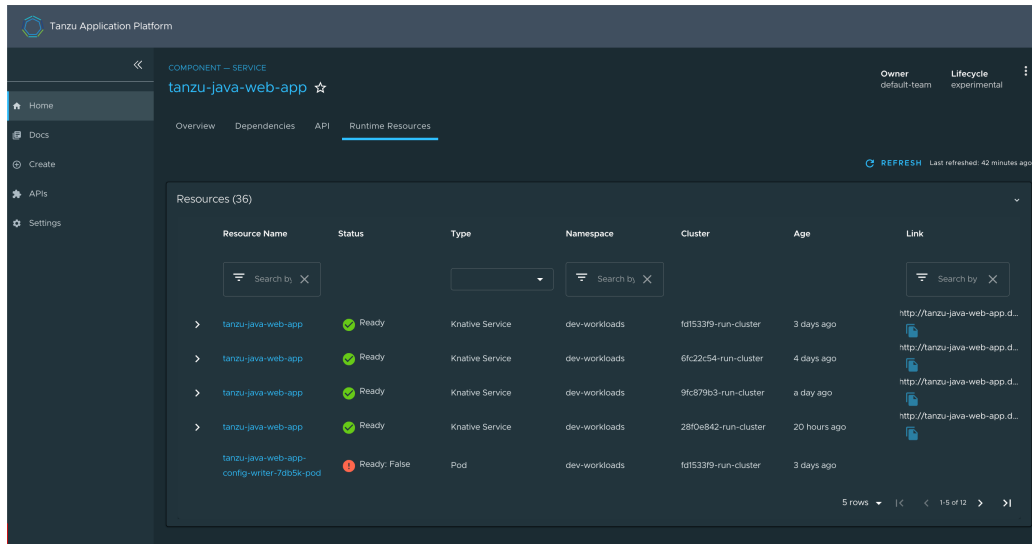
3. Wait a moment for the `tap` and `tap-gui` packages to update and then verify that `STATUS` is `Reconcile succeeded` by running:

```
tanzu package installed get all -n tap-install
```

## View resources on multiple clusters in the Runtime Resources Visibility plug-in

To view resources on multiple clusters in the Runtime Resources Visibility plug-in:

1. Go to the Runtime Resources Visibility plug-in for a component that is running on multiple clusters.
2. View the multiple resources and their statuses across the clusters.



## Set up authentication for Tanzu Application Platform GUI

Tanzu Application Platform GUI (commonly called TAP GUI) extends the current Backstage authentication plug-in so that you can see a login page based on the authentication providers configured at installation. This feature is a work in progress.

Tanzu Application Platform GUI currently supports the following authentication providers:

- [Auth0](#)
- [Azure](#)
- [Bitbucket](#)
- [GitHub](#)
- [GitLab](#)
- [Google](#)
- [Okta](#)
- [OneLogin](#)

You can also configure a custom OpenID Connect (OIDC) provider.

## View your Backstage Identity

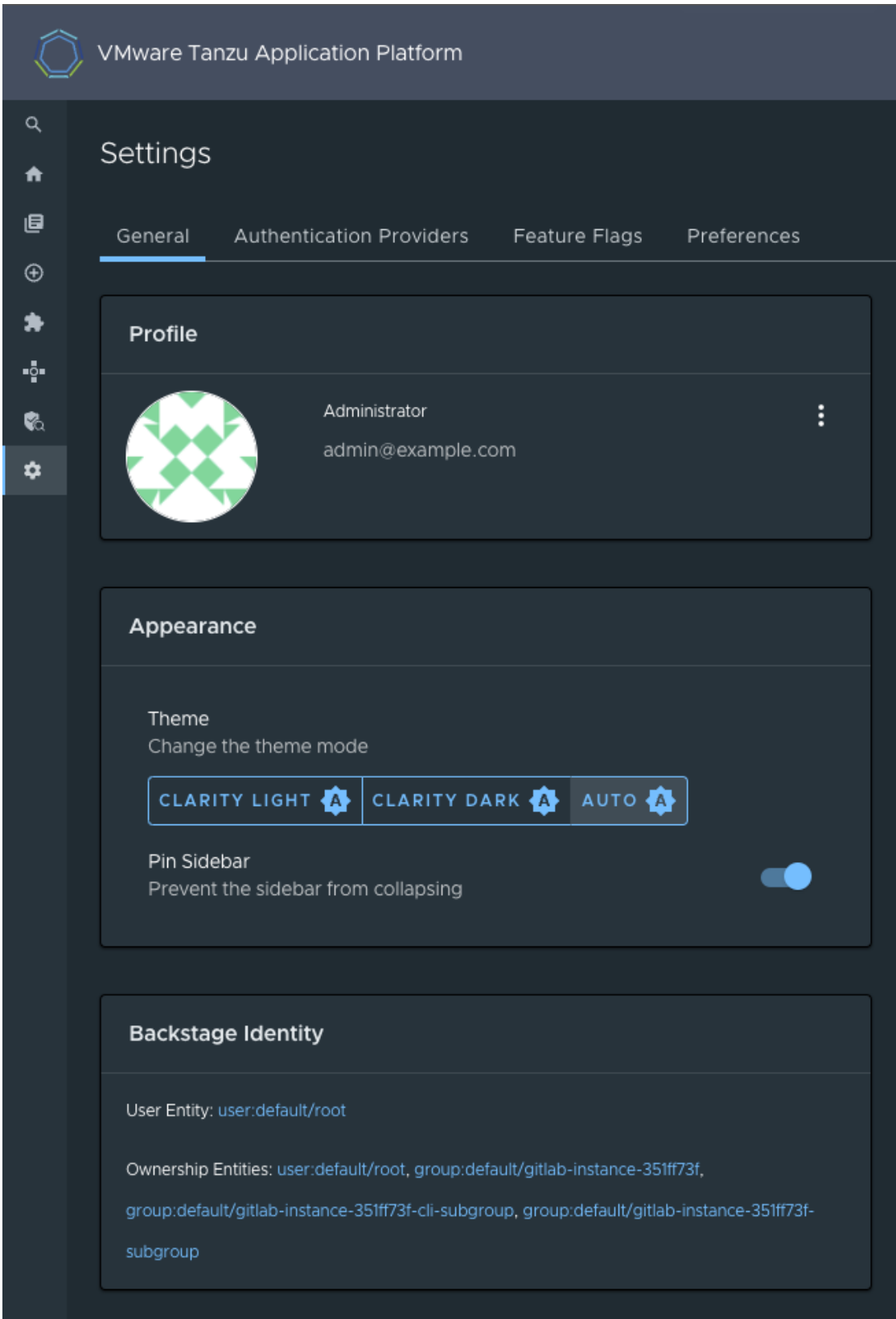
A Backstage identity is defined as a combination of:

- The user reference: each entity in the catalog is uniquely identified by the triplet of its `kind`
- A `namespace`
- A `name`

For example, the user Jane can be assigned to the user entity `user:default/jane` and an ownership reference, which is used to determine what that user owns. Jane (`user:default/jane`) might have the ownership references `user:default/jane`, `group:default/team-a`, and `group:default/admins`. This would mean that Jane belongs to those groups and, therefore, owns those references.

To view your current Backstage identity, in the **Settings** section of the left side navigation pane click the **General** tab.





## Configure an authentication provider

Configure a supported authentication provider or a custom OIDC provider:

- To configure a supported authentication provider, see the [Backstage authentication documentation](#).

- To configure a custom OIDC provider, edit your `tap-values.yaml` file or your custom configuration file to include an OIDC authentication provider. Configure the OIDC provider with your OAuth App values. For example:

```
shared:
  ingress_domain: "INGRESS-DOMAIN"

# ... any existing values

tap_gui:
  # ... any other TAP GUI values
  app_config:
    auth:
      environment: development
      session:
        secret: custom session secret
      providers:
        oidc:
          development:
            metadataUrl: AUTH-OIDC-METADATA-URL
            clientId: AUTH-OIDC-CLIENT-ID
            clientSecret: AUTH-OIDC-CLIENT-SECRET
            tokenSignedResponseAlg: AUTH-OIDC-TOKEN-SIGNED-RESPONSE-ALG # 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) Configure offline access scope for the OIDC provider by adding the `scope` parameter `offline_access` to either `tap-values.yaml` or your custom configuration file. For example:

```
auth:
  providers:
    oidc:
      development:
```

```
... # auth configs
scope: 'openid profile email offline_access'
```

By default, `scope` is not configured to provide persistence to user login sessions, such as in the case of a page refresh. Not all identity providers support the `offline_access` scope. For more information, see your identity provider documentation.

## (Optional) Allow guest access

Enable guest access with other providers by adding the following flag under your authentication configuration:

```
auth:
  allowGuestAccess: true
```

## (Optional) Customize the login page

Change the card's title or description for a specific provider with the following configuration:

```
auth:
  environment: development
  providers:
    ... # auth providers config
  loginPage:
    github:
      title: Github Login
      message: Enter with your GitHub account
```

For a provider to appear on the login page, ensure that it is properly configured under the `auth.providers` section of your values file.

## View resources on remote clusters

You can control the access to Kubernetes runtime resources on Tanzu Application Platform GUI (commonly called TAP GUI) based on user roles and permissions for each of the visible remote clusters.



### Caution

Setting up role-based access control (RBAC) might impact the user's ability to view workloads in the Security Analysis GUI and the Workloads table of the Supply Chain Choreographer plug-in GUI.

RBAC is currently supported for the following Kubernetes cluster providers:

- [EKS](#) (Elastic Kubernetes Service) on AWS
- [GKE](#) (Google Kubernetes Engine) on GCP

Support for other Kubernetes providers is planned for future releases of Tanzu Application Platform.

Tanzu Application Platform GUI is designed under the assumption that the roles and permissions for the Kubernetes clusters are already defined and that the users are already assigned to their roles. For information about assigning roles and permissions to users, see [Assigning roles and permissions on Kubernetes clusters](#).

Adding access-controlled visibility for a remote cluster is similar to [Setting up unrestricted remote cluster visibility](#).

The steps are:

1. Set up the OIDC provider
2. Configure the Kubernetes cluster with the OIDC provider
3. Configure the Tanzu Application Platform GUI to view the remote cluster
4. Upgrade the Tanzu Application Platform GUI package

After following these steps, you can view your runtime resources on a remote cluster in Tanzu Application Platform GUI. For more information, see [View runtime resources on remote clusters](#).

## View resources on remote clusters

You can control the access to Kubernetes runtime resources on Tanzu Application Platform GUI (commonly called TAP GUI) based on user roles and permissions for each of the visible remote clusters.



### Caution

Setting up role-based access control (RBAC) might impact the user's ability to view workloads in the Security Analysis GUI and the Workloads table of the Supply Chain Choreographer plug-in GUI.

RBAC is currently supported for the following Kubernetes cluster providers:

- [EKS](#) (Elastic Kubernetes Service) on AWS
- [GKE](#) (Google Kubernetes Engine) on GCP

Support for other Kubernetes providers is planned for future releases of Tanzu Application Platform.

Tanzu Application Platform GUI is designed under the assumption that the roles and permissions for the Kubernetes clusters are already defined and that the users are already assigned to their roles. For information about assigning roles and permissions to users, see [Assigning roles and permissions on Kubernetes clusters](#).

Adding access-controlled visibility for a remote cluster is similar to [Setting up unrestricted remote cluster visibility](#).

The steps are:

1. Set up the OIDC provider
2. Configure the Kubernetes cluster with the OIDC provider
3. Configure the Tanzu Application Platform GUI to view the remote cluster
4. Upgrade the Tanzu Application Platform GUI package

After following these steps, you can view your runtime resources on a remote cluster in Tanzu Application Platform GUI. For more information, see [View runtime resources on remote clusters](#).

## View resources on remote EKS clusters

This topic tells you how to view your runtime resources on a remote EKS cluster in Tanzu Application Platform GUI (commonly called TAP GUI). For more information, see [View runtime resources on remote clusters](#).

## Set up the OIDC provider

You must set up the OIDC provider to enable RBAC visibility of remote EKS clusters. You can see the list of supported OIDC providers in [Setting up a Tanzu Application Platform GUI authentication provider](#).

Tanzu Application Platform GUI supports multiple OIDC providers. Auth0 is used here as an example.

1. Log in to the Auth0 dashboard.
2. Go to **Applications**.
3. Create an application of the type **Single Page Web Application** named **TAP-GUI** or a name of your choice.
4. Click the **Settings** tab.
5. Under **Application URIs > Allowed Callback URLs**, add

```
https://tap-gui.INGRESS-DOMAIN/api/auth/auth0/handler/frame
```

Where **INGRESS-DOMAIN** is the domain you chose for your Tanzu Application Platform GUI in [Installing the Tanzu Application Platform package and profiles](#).

6. Click **Save Changes**.

After creating an application with your OIDC provider, you receive the following credentials for setting up RBAC for your remote cluster:

- **Domain**, which is used as **ISSUER-URL** in the following sections (**AUTH0\_DOMAIN** for Auth0)
- **Client ID**, which is used as **CLIENT-ID** in the following sections
- **Client Secret**, which is used as **CLIENT-SECRET** in the following sections

For more information, see [Auth0 Setup Walkthrough](#) in the Backstage documentation. To configure other OIDC providers, see [Authentication in Backstage](#) in the Backstage documentation.

## Configure the Kubernetes cluster with the OIDC provider

To configure the cluster with the OIDC provider's credentials:

1. Create a file with the following content and name it `rbac-setup.yaml`. This content applies to EKS clusters.

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: "CLUSTER-NAME"
  region: "AWS-REGION"
identityProviders:
  - name: auth0
    type: oidc
    issuerUrl: "ISSUER-URL"
    clientId: "CLIENT-ID"
    usernameClaim: email
```

Where:

- **CLUSTER-NAME** is the cluster name for your EKS cluster as an AWS identifier
- **AWS-REGION** is the AWS region of the EKS cluster
- **CLIENT-ID** is the Client ID you obtained while setting up the OIDC provider
- **ISSUER-URL** is the Issuer URL you obtained while setting up the OIDC provider. For Auth0, this is `https://${AUTH0_DOMAIN}/`.

- Using `eksctl`, run:

```
eksctl associate identityprovider -f rbac-setup.yaml
```

- Verify that the association of the OIDC provider with the EKS cluster was successful by running:

```
eksctl get identityprovider --cluster CLUSTER-NAME
```

Where `CLUSTER-NAME` is the cluster name for your EKS cluster as an AWS identifier

Verify that the output shows `ACTIVE` in the `STATUS` column.

## Configure the Tanzu Application Platform GUI

Configure visibility of the remote cluster in Tanzu Application Platform GUI:

- Obtain your cluster's URL by running:

```
CLUSTER_URL=$(kubectl config view --minify -o jsonpath='{.clusters[0].cluster.server}')
echo CLUSTER-URL: $CLUSTER_URL
```

This command returns the URL of the first configured cluster in your `kubeconfig` file. To view other clusters one by one, edit the number in `.clusters[0].cluster.server` or edit the command to view all the configured clusters.

- Ensure you have an `auth` section in the `app_config` section that Tanzu Application Platform GUI uses. In the example for Auth0, copy this YAML content into `tap-values.yaml`:

```
auth:
  environment: development
  providers:
    auth0:
      development:
        clientId: "CLIENT-ID"
        clientSecret: "CLIENT-SECRET"
        domain: "ISSUER-URL"
```

Where:

- `CLIENT-ID` is the Client ID you obtained while setting up the OIDC provider.
  - `CLIENT-SECRET` is the Client Secret you obtained while setting up the OIDC provider.
  - `ISSUER-URL` is the Issuer URL you obtained while setting up the OIDC provider. For Auth0, it is only `AUTH0_DOMAIN`.
- Add a `kubernetes` section to the `app_config` section that Tanzu Application Platform GUI uses. This section must have an entry for each cluster that has resources to view. To do so, copy this YAML content into `tap-values.yaml`:

```
kubernetes:
  serviceLocatorMethod:
    type: 'multiTenant'
  clusterLocatorMethods:
    - type: 'config'
      clusters:
        - name: "CLUSTER-NAME-UNCONSTRAINED"
          url: "CLUSTER-URL"
          authProvider: oidc
```

```
oidcTokenProvider: auth0
skipTLSVerify: true
skipMetricsLookup: true
```

Where:

- `CLUSTER-NAME-UNCONSTRAINED` is the cluster name of your choice for your EKS cluster
- `CLUSTER-URL` is the URL for the remote cluster you are connecting to Tanzu Application Platform GUI. You obtained this earlier in the procedure.

If there are any other clusters that you want to make visible in Tanzu Application Platform GUI, add their entries to `clusters` as well.

## Upgrade the Tanzu Application Platform GUI package

After the new configuration file is ready, update the `tap` package:

1. Run:

```
tanzu package installed update tap --values-file tap-values.yaml
```

2. Wait a moment for the `tap-gui` package to update and then verify that `STATUS` is `Reconcile succeeded` by running:

```
tanzu package installed get tap-gui -n tap-install
```

## View resources on remote GKE clusters

This topic tells you about two supported options to add access-controlled visibility for a remote GKE cluster:

- [Leverage an external OIDC provider](#)
- [Leveraging Google's OIDC provider](#)

After the authorization is enabled, you can view your runtime resources on a remote cluster in Tanzu Application Platform GUI. For more information, see [View runtime resources on remote clusters](#).

## Leverage an external OIDC provider

To leverage an external OIDC provider, such as Auth0:

1. Set up the OIDC provider
2. Configure the GKE cluster with the OIDC provider
3. Configure Tanzu Application Platform GUI to view the remote GKE cluster
4. Upgrade `tap-gui` package

### Set up the OIDC provider

You must set up the OIDC provider to enable RBAC visibility of remote clusters. You can see the list of supported OIDC providers in [Setting up a Tanzu Application Platform GUI authentication provider](#).

Tanzu Application Platform GUI supports multiple OIDC providers. Auth0 is used here as an example.

1. Log in to the Auth0 dashboard.

2. Go to **Applications**.
3. Create an application of the type `Single Page Web Application` named `TAP-GUI` or a name of your choice.
4. Click the **Settings** tab.
5. Under **Application URIs > Allowed Callback URLs**, add

```
https://tap-gui.INGRESS-DOMAIN/api/auth/auth0/handler/frame
```

Where `INGRESS-DOMAIN` is the domain you chose for your Tanzu Application Platform GUI in [Installing the Tanzu Application Platform package and profiles](#).

6. Click **Save Changes**.

After creating an application with your OIDC provider, you receive the following credentials for setting up RBAC for your remote cluster:

- **Domain**, which is used as `issuerURL` in the following sections
- **Client ID**, which is used as `CLIENT-ID` in the following sections
- **Client Secret**, which is used as `CLIENT-SECRET` in the following sections

For more information, see [Auth0 Setup Walkthrough](#) in the Backstage documentation. To configure other OIDC providers, see [Authentication in Backstage](#) in the Backstage documentation.

## Configure the GKE cluster with the OIDC provider

Add redirect configuration on the OIDC side by following the [Google Cloud documentation](#).

## Configure visibility of the remote cluster

Configure visibility of the remote cluster in Tanzu Application Platform GUI:

1. Obtain your cluster's URL by running:

```
CLUSTER_URL=$(kubectl config view --minify -o jsonpath='{.clusters[0].cluster.server}')
echo CLUSTER-URL: $CLUSTER_URL
```

This command returns the URL of the first configured cluster in your `kubeconfig` file. To view other clusters one by one, edit the number in `.clusters[0].cluster.server` or edit the command to view all the configured clusters.

2. Ensure you have an `auth` section in the `app_config` section that Tanzu Application Platform GUI uses. In the example for Auth0, copy this YAML content into `tap-values.yaml`:

```
auth:
  environment: development
  providers:
    auth0:
      development:
        clientId: "CLIENT-ID"
        clientSecret: "CLIENT-SECRET"
        domain: "ISSUER-URL"
```

Where:

- `CLIENT-ID` is the Client ID you obtained while setting up the OIDC provider
- `CLIENT-SECRET` is the Client Secret you obtained while setting up the OIDC provider



- `ISSUER-URL` is the Issuer URL you obtained while setting up the OIDC provider
3. Add a `kubernetes` section to the `app_config` section that Tanzu Application Platform GUI uses. This section must have an entry for each cluster that has resources to view. To do so, copy this YAML content into `tap-values.yaml`:

```
kubernetes:
  serviceLocatorMethod:
    type: 'multiTenant'
  clusterLocatorMethods:
    - type: 'config'
    clusters:
      - name: "CLUSTER-NAME-UNCONSTRAINED"
        url: "CLUSTER-URL"
        authProvider: oidc
        oidcTokenProvider: auth0
        skipTLSVerify: true
        skipMetricsLookup: true
```

Where:

- `CLUSTER-NAME-UNCONSTRAINED` is the cluster name of your choice for your GKE cluster
- `CLUSTER-URL` is the URL for the remote cluster you are connecting to Tanzu Application Platform GUI. You obtained this earlier in the procedure.

If there are any other clusters that you want to make visible in Tanzu Application Platform GUI, add their entries to `clusters` as well.

## Update the `tap-gui` package to finish leveraging the external OIDC provider

After the new configuration file is ready, update the `tap-gui` package:

1. Run:

```
tanzu package installed update tap --values-file tap-values.yaml
```

2. Wait a moment for the `tap-gui` package to update and then verify that `STATUS` is `Reconcile succeeded` by running:

```
tanzu package installed get tap-gui -n tap-install
```

## Leverage Google's OIDC provider

When leveraging Google's OIDC provider, fewer steps are needed to enable authorization:

1. Add redirect configuration on the OIDC side.
2. Configure the Tanzu Application Platform GUI to view the remote GKE cluster
3. Upgrade the Tanzu Application Platform GUI package

### Add redirect configuration on the OIDC side

Add redirect configuration on the OIDC side by following the [Google Cloud documentation](#).

### Configure visibility of the remote GKE cluster

Configure visibility of the remote GKE cluster in Tanzu Application Platform GUI:

1. Obtain your cluster's URL by running:

```
CLUSTER_URL=$(kubectl config view --minify -o jsonpath='{.clusters[0].cluster.server}')

echo CLUSTER-URL: $CLUSTER_URL
```

This command returns the URL of the first configured cluster in your `kubeconfig` file. To view other clusters one by one, edit the number in `.clusters[0].cluster.server` or edit the command to view all the configured clusters.

2. Ensure you have an `auth` section in the `app_config` section that Tanzu Application Platform GUI uses. In the example for Auth0, copy this YAML content into `tap-values.yaml`:

```
auth:
  environment: development
  providers:
    google:
      development:
        clientId: "CLIENT-ID"
        clientSecret: "CLIENT-SECRET"
```

Where:

- `CLIENT-ID` is the Client ID you obtained while setting up the OIDC provider
  - `CLIENT-SECRET` is the Client Secret you obtained while setting up the OIDC provider
3. Add a `kubernetes` section to the `app_config` section that Tanzu Application Platform GUI uses. This section must have an entry for each cluster that has resources to view. To do so, copy this YAML content into `tap-values.yaml`:

```
kubernetes:
  clusterLocatorMethods:
    - type: 'config'
  clusters:
    - name: "CLUSTER-NAME-UNCONSTRAINED"
      url: "CLUSTER-URL"
      authProvider: google
      caData: "CA-DATA"
```

Where:

- `CLUSTER-NAME-UNCONSTRAINED` is the cluster name of your choice for your GKE cluster.
- `CLUSTER-URL` is the URL for the remote cluster you are connecting to Tanzu Application Platform GUI. You obtained this earlier in the procedure.
- `CA-DATA` is the CA certificate data.

If there are any other clusters that you want to make visible in Tanzu Application Platform GUI, add their entries to `clusters` as well.

## Update the `tap-gui` package to finish leveraging the Google OIDC provider

After the new configuration file is ready, update the `tap-gui` package:

1. Run:

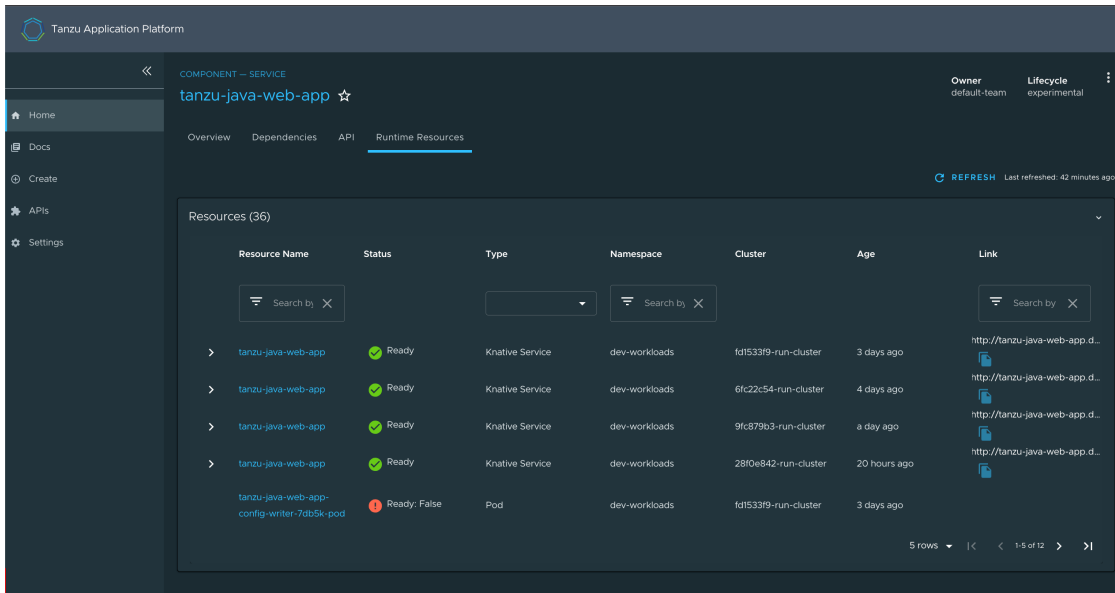
```
tanzu package installed update tap --values-file tap-values.yaml
```

2. Wait a moment for the `tap-gui` package to update and then verify that `STATUS` is `Reconcile succeeded` by running:

```
tanzu package installed get tap-gui -n tap-install
```

## View runtime resources on authorization-enabled clusters

To visualize runtime resources on authorization-enabled clusters in Tanzu Application Platform GUI (commonly called TAP GUI), proceed to the software catalog component of choice and click the **Runtime Resources** tab on top of the ribbon.



After you click **Runtime Resources**, Tanzu Application Platform GUI uses your credentials to query the clusters for the respective API runtime resources. The system verifies that you are authenticated with the OIDC providers configured for the remote clusters. If you are not authenticated, the system prompts you for your OIDC credentials.

Remote clusters that are not restricted by authorization are visible by using the general Service Account of Tanzu Application Platform GUI. It is not restricted for users. For more information about how to set up unrestricted remote cluster Service visibility, see [Viewing resources on multiple clusters in Tanzu Application Platform GUI](#).

The type of query to the remote cluster depends on the definition of the software catalog component. In Tanzu Application Platform GUI, there are globally-scoped components and namespace-scoped components.

This property of the component affects runtime resource visibility, depending on your permissions on a specific cluster.

If your permissions on the authorization-enabled cluster are limited to specific namespaces, you do not have visibility into runtime resources of globally-scoped components.

You need cluster-scoped access to have visibility into runtime resources of globally-scoped components.

## Globally-scoped components

For globally-scoped components, when you access **Runtime Resources** Tanzu Application Platform GUI queries all Kubernetes namespaces for runtime resources that have a matching `kubernetes-label-selector`, usually with a `part-Of` prefix.

For example, `demo-component-a` does not have a `backstage.io/kubernetes-namespace` in the `metadata.annotations` section. This makes it a globally-scoped component. See the following

example YAML.

```
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: demo-component-a
  description: Demo Component A
  tags:
    - java
  annotations:
    'backstage.io/kubernetes-label-selector': 'app.kubernetes.io/part-of=component-a'
spec:
  type: service
  lifecycle: experimental
  owner: team-a
```

## Namespace-scoped components

If a component is namespace-scoped, when you access **Runtime Resources** Tanzu Application Platform GUI queries only the associated Kubernetes namespace for each remote cluster that is visible to Tanzu Application Platform GUI.

To make a component namespace-scoped, pass the following annotation to the definition YAML file of the component:

```
annotations:
  'backstage.io/kubernetes-namespace': NAMESPACE-NAME
```

Where **NAMESPACE-NAME** is the Kubernetes namespace you want to associate your component with.

For example, `demo-component-b` has a `kubernetes-namespace` in the `metadata.annotations` section, which associates it with the `component-b` namespaces on each of the visible clusters. This makes it a namespace-scoped component. See the following example YAML.

```
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: demo-component-b
  description: Demo Component B
  tags:
    - java
  annotations:
    'backstage.io/kubernetes-label-selector': 'app.kubernetes.io/part-of=component-b'
    'backstage.io/kubernetes-namespace': component-b
spec:
  type: service
  lifecycle: experimental
  owner: team-b
```

When the `kubernetes-namespace` annotation is absent, the component is considered globally-scoped by default. For more information, see [Adding Namespace Annotation](#) in the Backstage documentation.

## Assign roles and permissions on Kubernetes clusters

This topic gives you an overview of creating roles and permissions on Kubernetes clusters and assigning these roles to users. For more information, see [Using RBAC Authorization](#) in the Kubernetes documentation.

The steps to define and assign roles are:

1. [Create roles](#)
2. [Create users](#)
3. [Assign users to their roles](#)

## Create roles

To control the access to Kubernetes runtime resources on Tanzu Application Platform GUI based on users' roles and permissions for each of visible remote clusters, VMware recommends two role types:

- [Cluster-scoped roles](#)
- [Namespace-scoped roles](#)

### Cluster-scoped roles

Cluster-scoped roles provide cluster-wide privileges. They enable visibility into runtime resources across all of a cluster's namespaces.

In this example YAML snippet, the `pod-viewer` role enables pod visibility on the cluster:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: pod-viewer
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

### Namespace-scoped roles

Namespace-scoped roles provide privileges that are limited to a certain namespace. They enable visibility into runtime resources inside namespaces.

In this example YAML snippet, the `pod-viewer-app1` role enables pod visibility in the `app1` namespace:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: app1
  name: pod-viewer-app1
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list"]
```

## Create users

You can create users by running the `kubectl create` command. In this example YAML snippet, the user `john` is defined:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: User
metadata:
  namespace: default
  name: john
```

## Assign users to their roles

After the users and role are created, the next step is to bind them together.

To bind a Tanzu Application Platform default role, see [Bind a user or group to a default role](#).

In this example YAML snippet, the user `john` is bound with the `pod-viewer` cluster role:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: john-pod-viewer
  namespace: default
subjects:
- kind: User
  name: john
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: pod-viewer
  apiGroup: rbac.authorization.k8s.io
```

In this example YAML snippet, the user `john` is bound with the `pod-viewer-app1` namespace-specific role:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: john-pod-viewer-app1
  namespace: app1
subjects:
- kind: User
  name: john
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-viewer-app1
  apiGroup: rbac.authorization.k8s.io
```

To verify the user's permissions, run the `can-i` commands to get a `yes` or `no` answer. To verify that you can list pods in your cluster-wide role, run:

```
kubectl auth can-i get pods --all-namespaces
```

To verify that you can list pods in namespace `app1` in your namespace-specific role, run:

```
kubectl auth can-i get pods --namespace app1
```

## Add Tanzu Application Platform GUI integrations

You can integrate Tanzu Application Platform GUI (commonly called TAP GUI) with several Git providers. To use an integration, you must enable it and provide the necessary token or credentials in `tap-values.yaml`.

### Add a GitHub provider integration

To add a GitHub provider integration, edit `tap-values.yaml` as in this example:

```
app_config:
  app:
```

```

baseUrl: http://EXTERNAL-IP:7000
# Existing tap-values.yaml above
integrations:
  github: # Other integrations available see NOTE below
    - host: github.com
      token: GITHUB-TOKEN

```

Where:

- `EXTERNAL-IP` is the external IP address.
- `GITHUB-TOKEN` is a valid token generated from your Git infrastructure of choice. Ensure that `GITHUB-TOKEN` has the necessary read permissions for the catalog definition files you extracted from the blank software catalog introduced in the [Tanzu Application Platform GUI prerequisites](#).

## Add a Git-based provider integration that isn't GitHub

To enable Tanzu Application Platform GUI to read Git-based non-GitHub repositories containing component information:

1. Add the following YAML to `tap-values.yaml`:

```

app_config:
  # Existing tap-values.yaml above
  backend:
    reading:
      allow:
        - host: "GIT-CATALOG-URL-1"
        - host: "GIT-CATALOG-URL-2" # Including more than one URL is optional

```

Where `GIT-CATALOG-URL-1` and `GIT-CATALOG-URL-2` are URLs in a list of URLs that Tanzu Application Platform GUI can read when registering new components. For example, `git.example.com`. For more information about registering new components, see [Adding catalog entities](#).

2. Adding the YAML from the previous step currently causes the **Accelerators** page to break and not show any accelerators. Provide a value for Application Accelerator as a workaround, as in this example:

```

app_config:
  # Existing tap-values.yaml above
  backend:
    reading:
      allow:
        - host: acc-server.accelerator-system.svc.cluster.local

```

## Add a non-Git provider integration

To add an integration for a provider that isn't associated with GitHub, see the [Backstage documentation](#).

## Update the package profile

After changing `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, `1.4.13`.

For example:

```
$ tanzu package installed update tap --package-name tap.tanzu.vmware.com --version \
1.4.13 --values-file tap-values.yaml -n tap-install
| Updating package 'tap'
| Getting package install for 'tap'
| Getting package metadata for 'tap.tanzu.vmware.com'
| Updating secret 'tap-tap-install-values'
| Updating package install for 'tap'
/ Waiting for 'PackageInstall' reconciliation for 'tap'

Updated package install 'tap' in namespace 'tap-install'
```

## Configure the Tanzu Application Platform GUI database

The Tanzu Application Platform GUI (commonly called TAP GUI) catalog gives you two approaches for storing catalog information:

- **In-memory database:**

The default option uses an in-memory database and is suitable for test and development scenarios only. The in-memory database reads the catalog data from Git URLs that you write in `tap-values.yaml`.

This data is temporary. Any operations that cause the `server` pod in the `tap-gui` namespace to be re-created also cause this data to be rebuilt from the Git location.

This can cause issues when you manually register entities by using the UI because they only exist in the database and are lost when that in-memory database is rebuilt. If you choose this method, you lose all user preferences and any manually registered entities when the Tanzu Application Platform GUI server pod is re-created.

- **PostgreSQL database:**

For production use-cases, use a PostgreSQL database that exists outside the Tanzu Application Platform packaging. The PostgreSQL database stores all the catalog data persistently both from the Git locations and the UI manual entity registrations.

For production or general-purpose use-cases, a PostgreSQL database is recommended.

## Configure a PostgreSQL database

See the following sections for configuring Tanzu Application Platform GUI to use a PostgreSQL database.

### Edit `tap-values.yaml`

Apply the following values in `tap-values.yaml`:

```
# ... existing tap-values.yaml above
tap_gui:
# ... existing tap_gui values
app_config:
  backend:
    database:
      client: pg
      connection:
        host: PG-SQL-HOSTNAME
        port: 5432
```



```

user: PG-SQL-USERNAME
password: PG-SQL-PASSWORD
ssl: {rejectUnauthorized: false} # Set to true if using SSL

```

Where:

- `PG-SQL-HOSTNAME` is the host name of your PostgreSQL database
- `PG-SQL-USERNAME` is the user name of your PostgreSQL database
- `PG-SQL-PASSWORD` is the password of your PostgreSQL database

### (Optional) Configure extra parameters

Beyond the minimum configuration options needed to make Tanzu Application Platform GUI work with the `pg` driver, there are many more configuration options for other purposes. For example, you can restrict Tanzu Application Platform GUI to a single database. For more information about this restriction, see the [Backstage documentation](#).

By default, Tanzu Application Platform GUI creates a database for each plug-in, but you can configure it to divide plug-ins based on different PostgreSQL schemas and use a single specified database.

See the following example of extra configuration parameters:

```

# ... existing tap-values.yaml above
tap_gui:
  # ... existing tap_gui values
  app_config:
    backend:
      # ... other backend details
      database:
        client: pg

        # This parameter tells Tanzu Application Platform GUI to put plug-ins in their
        # own schema instead
        # of their own database.
        # default: database
        pluginDivisionMode: schema

    connection:
      # ... other connection details
      database: PG-SQL-DATABASE

```

Where `PG-SQL-DATABASE` is the database name for Tanzu Application Platform GUI to use

For the complete list of these configuration options, see the [node-postgres documentation](#).

## Update the package profile

You can apply your new configuration by updating Tanzu Application Platform with your modified values. Doing so updates Tanzu Application Platform GUI because it belongs to Tanzu Application Platform.

To apply your new configuration, run:

```

tanzu package installed update tap --package-name tap.tanzu.vmware.com --version VERS
ION-NUMBER --values-file tap-values.yaml -n tap-install

```

Where `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.4.13`.

For example:

```

$ tanzu package installed update tap --package-name tap.tanzu.vmware.com --version 1.4.13 --values-file tap-values.yaml -n tap-install
| Updating package 'tap'
| Getting package install for 'tap'
| Getting package metadata for 'tap.tanzu.vmware.com'
| Updating secret 'tap-tap-install-values'
| Updating package install for 'tap'
/ Waiting for 'PackageInstall' reconciliation for 'tap'

Updated package install 'tap' in namespace 'tap-install'

```

## Generate and publish TechDocs

This topic tells you how to generate and publish TechDocs for catalogs as part of Tanzu Application Platform GUI (commonly called TAP GUI). For more information about TechDocs, see the [Backstage.io documentation](#).

## Create an Amazon S3 bucket

To create an Amazon S3 bucket:

1. Go to [Amazon S3](#).
2. Click **Create bucket**.
3. Give the bucket a name.
4. Select the AWS region.
5. Keep **Block all public access** checked.
6. Click **Create bucket**.

## Configure Amazon S3 access

The TechDocs are published to the S3 bucket that was recently created. You need an AWS user's access key to read from the bucket when viewing TechDocs.

## Create an AWS IAM user group

To create an [AWS IAM User Group](#):

1. Click **Create Group**.
2. Give the group a name.
3. Click **Create Group**.
4. Click the new group and navigate to **Permissions**.
5. Click **Add permissions** and click **Create Inline Policy**.
6. Click the **JSON** tab and replace contents with this JSON replacing `BUCKET-NAME` with the bucket name.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ReadTechDocs",
      "Effect": "Allow",
      "Action": [

```

```

        "s3:ListBucket",
        "s3:GetObject"
    ],
    "Resource": [
        "arn:aws:s3:::BUCKET-NAME",
        "arn:aws:s3:::BUCKET-NAME/*"
    ]
}
]
}

```

7. Click **Review policy**.
8. Give the policy a name and click **Create policy**.

## Create an AWS IAM user

To create an [AWS IAM User](#) to add to this group:

1. Click **Add users**.
2. Give the user a name.
3. Verify **Access key - Programmatic access** and click **Next: Permissions**.
4. Verify the IAM Group to add the user to and click **Next: Tags**.
5. Click **Next: Review** then click **Create user**.
6. Record the **Access key ID** (`AWS_READONLY_ACCESS_KEY_ID`) and the **Secret access key** (`AWS_READONLY_SECRET_ACCESS_KEY`) and click **Close**.

## Find the catalog locations and their entities' namespace, kind, and name

TechDocs are generated for catalogs that have Markdown source files for TechDocs. To find the catalog locations and their entities' namespace, kind, and name:

1. The catalogs appearing in Tanzu Application Platform GUI are listed in the config values under `app_config.catalog.locations`.
2. For a catalog, clone the catalog's repository to the local file system.
3. Find the `mkdocs.yml` that is at the root of the catalog. There is a YAML file describing the catalog at the same level called `catalog-info.yaml`.
4. Record the values for `namespace`, `kind`, and `metadata.name`, and the directory path containing the YAML file.
5. Record the `spec.targets` in that file.
6. Find the namespace, kind, or name for each of the targets:
  1. Go to the target's YAML file.
  2. The `namespace` value is the value of `namespace`. If it is not specified, it has the value `default`.
  3. The `kind` value is the value of `kind`.
  4. The `name` value is the value of `metadata.name`.
  5. Record the directory path containing the YAML file.

## Use the TechDocs CLI to generate and publish TechDocs

VMware uses `npx` to run the TechDocs CLI, which requires `Node.js` and `npm`. To generate and publish TechDocs by using the TechDocs CLI:

1. Download and install `Node.js` and `npm`.
2. Install `npx` by running:

```
npm install -g npx
```

3. Generate the TechDocs for the root of the catalog by running:

```
npx @techdocs/cli generate --source-dir DIRECTORY-CONTAINING-THE-ROOT-YAML-FILE
--output-dir ./site
```



#### Note

This creates a temporary `site` directory in your current working directory that contains the generated TechDocs files.

4. Review the contents of the `site` directory to verify the TechDocs were generated.
5. Set environment variables for authenticating with Amazon S3 with an account that has read/write access:

```
export AWS_ACCESS_KEY_ID=AWS-ACCESS-KEY-ID
export AWS_SECRET_ACCESS_KEY=AWS-SECRET-ACCESS-KEY
export AWS_REGION=AWS-REGION
```

6. Publish the TechDocs for the root of the catalog to the Amazon S3 bucket you created earlier by running:

```
npx @techdocs/cli publish --publisher-type awsS3 --storage-name BUCKET-NAME --e
ntity \
NAMESPACE/KIND/NAME --directory ./site
```

Where `NAMESPACE/KIND/NAME` are the values for `namespace`, `kind`, and `metadata.name` you recorded earlier. For example, `default/location/yelb-catalog-info`.

7. For each of the `spec.targets` found earlier, repeat the `generate` and `publish` commands.



#### Note

The `generate` command erases the contents of the `site` directory before creating new TechDocs files. Therefore, the `publish` command must follow the `generate` command for each target.

## Update the `techdocs` section in `app-config.yaml` to point to the Amazon S3 bucket

Update the config values you used during installation to point to the Amazon S3 bucket that has the published TechDocs files:

1. Add or edit the `techdocs` section under `app_config` in the config values with the following YAML, replacing placeholders with the appropriate values.

```

techdocs:
  builder: 'external'
  publisher:
    type: 'awsS3'
    awsS3:
      bucketName: BUCKET-NAME
      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.

### Tanzu Application Platform package installation

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 and `VALUES-FILE` is your values file

### Separate package installation

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 and `VALUES-FILE` is your values file

3. Verify the status of the update by running:

```

tanzu package installed list

```

4. Go to the **Docs** section of your catalog and view the TechDocs pages to verify the content is loaded from the S3 bucket.

## Overview of Tanzu Application Platform GUI plug-ins

Tanzu Application Platform GUI (commonly called TAP GUI) has many pre-integrated plug-ins. You need not configure the plug-ins. To use a plug-in, you must install the relevant Tanzu Application Platform component.

Tanzu Application Platform has the following GUI plug-ins:

- [Runtime Resources Visibility](#)
- [Application Live View](#)
- [Application Accelerator](#)
- [API Documentation](#)
- [Security Analysis](#)
- [Supply Chain Choreographer](#)

## Overview of Tanzu Application Platform GUI plug-ins

Tanzu Application Platform GUI (commonly called TAP GUI) has many pre-integrated plug-ins. You need not configure the plug-ins. To use a plug-in, you must install the relevant Tanzu Application Platform component.

Tanzu Application Platform has the following GUI plug-ins:

- [Runtime Resources Visibility](#)
- [Application Live View](#)
- [Application Accelerator](#)
- [API Documentation](#)
- [Security Analysis](#)
- [Supply Chain Choreographer](#)

## Runtime resources visibility in Tanzu Application Platform GUI

This topic tells you about runtime resources visibility.

The Runtime Resources Visibility plug-in enables users to visualize their Kubernetes resources associated with their workloads.

### Prerequisite

Do one of the following actions to access the Runtime Resources Visibility plug-in:

- [Install the Tanzu Application Platform Full or View profile](#)
- [Install Tanzu Application Platform without using a profile and then install Tanzu Application Platform GUI separately](#)
- [Review the section If you have a metrics server](#)

### If you have a metrics server

By default, the Kubernetes API does not attempt to use any metrics servers on your clusters. To access metrics information for a cluster, set `skipMetricsLookup` to `false` for that cluster in the `kubernetes` section of `app-config.yaml`. Example:

```
tap_gui:
  # ... existing configuration
  app_config:
    # ... existing configuration
    kubernetes:
      clusterLocatorMethods:
        - type: 'config'
          clusters:
            - url: https://KUBERNETES-SERVICE-HOST:KUBERNETES-SERVICE-PORT
              name: host
              authProvider: serviceAccount
              serviceAccountToken: KUBERNETES-SERVICE-ACCOUNT-TOKEN
              skipTLSVerify: true
              skipMetricsLookup: false
```

Where:

- `KUBERNETES-SERVICE-HOST` and `KUBERNETES-SERVICE-PORT` are the URL and ports of your Kubernetes cluster. You can gather these through `kubectl cluster-info`.
- `KUBERNETES-SERVICE-ACCOUNT-TOKEN` is the token from your `tap-gui-token-id`.

You can retrieve this secret's ID by running:

```
kubectl get secrets -n tap-gui
```

and then running

```
kubectl describe secret tap-gui-token-ID
```

Where `ID` is the secret name from the first step.

**Caution**

If you enable metrics for a cluster but do not have a metrics server running on it, Tanzu Application Platform web interface users see an error notifying them that there is a problem connecting to the back end.

## Visualize Workloads on Tanzu Application Platform GUI

In order to view your applications on Tanzu Application Platform GUI, use the following steps:

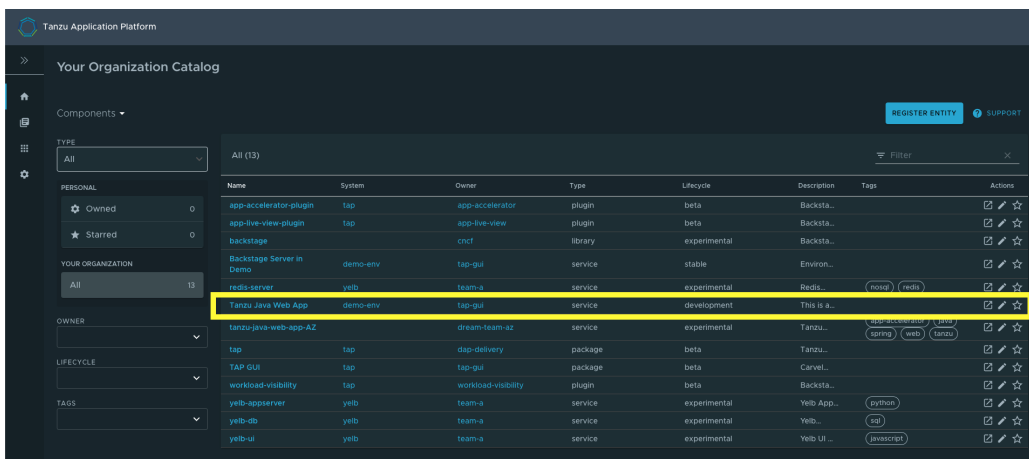
1. [Deploy your first application on the Tanzu Application Platform](#)
2. [Add your application to Tanzu Application Platform GUI Software Catalog](#)

## Navigate to the Runtime Resources Visibility screen

You can view the list of running resources and the details of their status, type, namespace, cluster, and public URL if applicable for the resource type.

To view the list of your running resources:

1. Select your component from the Catalog index page.



2. Select the **Runtime Resources** tab.

## Resources

Built-in Kubernetes resources in this view are:

- Services

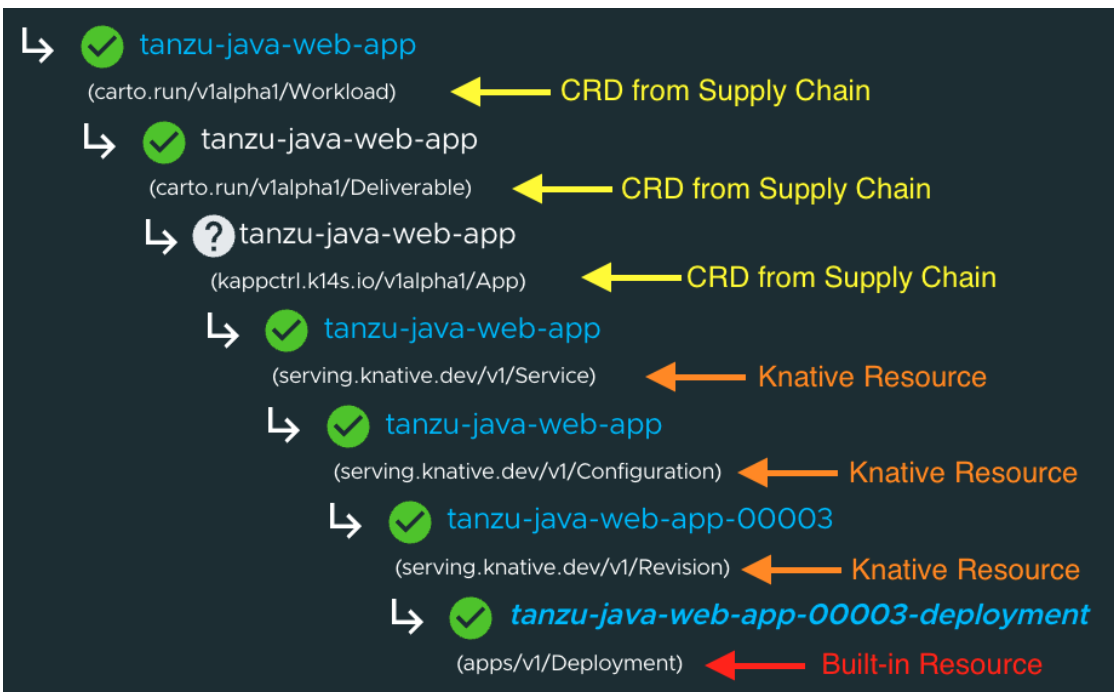
- Deployments
- ReplicaSets
- Pods
- Jobs
- Cronjobs
- DaemonSets
- ReplicaSets

The Runtime Resource Visibility plug-in also displays CRDs created with the Supply Chain, including:

- Cartographer Workloads
- Knative Services, Configurations, Revisions, and Routes

For more information, see [Supply Chain Choreographer in Tanzu Application Platform GUI](#).

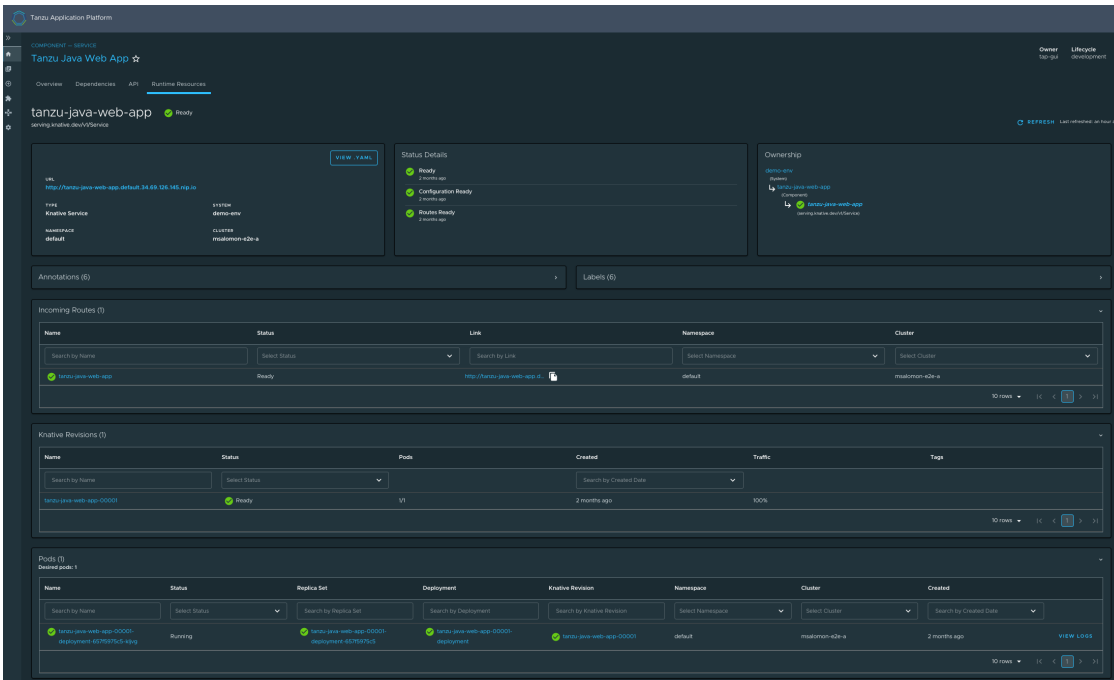
CRDs from Supply Chain are associated with Knative Resources, further down the chain, and built-in resources even further down the chain.



## Resources details page

To get more information about a particular workload, select it from the table on the main **Runtime Resources** page to visit a page that provides details about the workload. These details include the workload status, ownership, and resource-specific information.






## Overview card

All detail pages provide an overview card with information related to the selected resource. Most of the information feeds from the `metadata` attribute in each object. The following are some attributes that are displayed in the overview card:

- **View Pod Logs** button
- **View .YAML** button
- URL, which is for Knative and Kubernetes service detail pages
- Type
- System
- Namespace
- Cluster

VIEW POD LOGS
VIEW .YAML

<p><b>CREATED</b> a month ago</p>	<p><b>TYPE</b> Pod</p>
<p><b>NAMESPACE</b> default</p>	<p><b>CLUSTER</b> dev</p>
<p><b>TOTAL CPU (REQUESTED / LIMIT)</b> Unlimited/Unlimited</p>	<p><b>TOTAL MEMORY (REQUESTED / LIMIT)</b> Unlimited/Unlimited</p>
<p><a href="#" style="color: #3498db; text-decoration: none;">VIEW CPU AND MEMORY DETAILS →</a></p>	<p><a href="#" style="color: #3498db; text-decoration: none;">VIEW THREADS →</a></p>



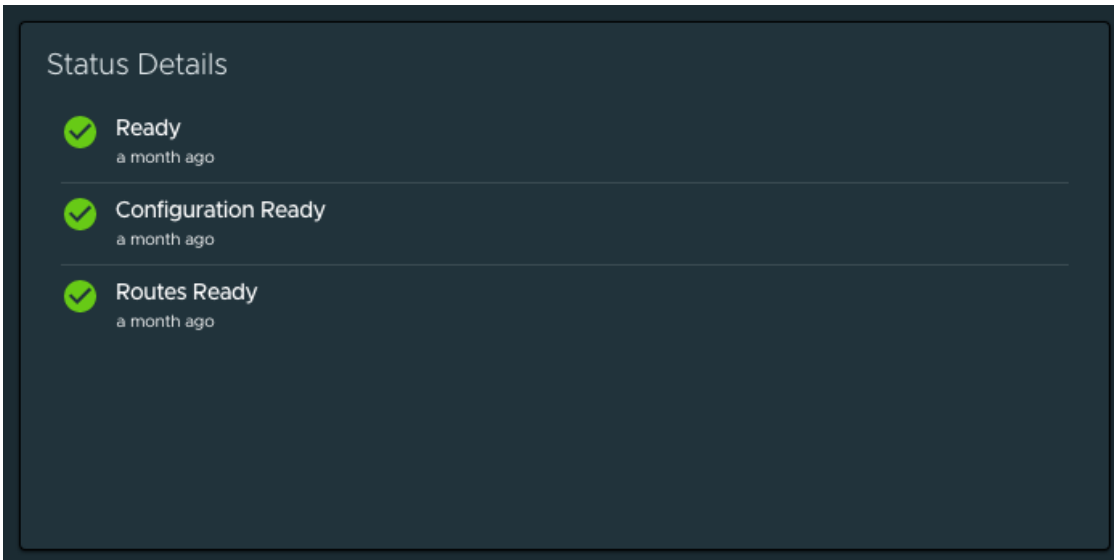
**Note**

The **VIEW CPU AND MEMORY DETAILS** and **VIEW THREADS** sections are only available for applications supporting Application Live View.

## Status card

The status section displays all of the conditions in the resource's attribute `status.conditions`. Not all resources have conditions, and they can vary from one resource to the other.

For more information about object `spec` and `status`, see the [Kubernetes documentation](#).



## Ownership card

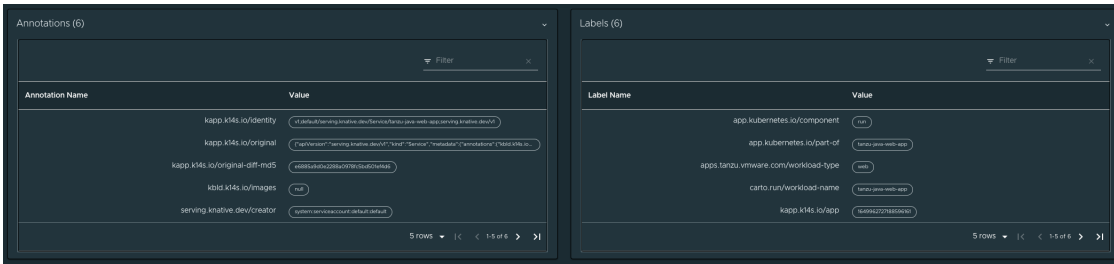
Depending on the resource that you are viewing, the ownership section displays all the resources specified in `metadata.ownerReferences`. You can use this section to navigate between resources.

For more information about owners and dependents, see the [Kubernetes documentation](#).



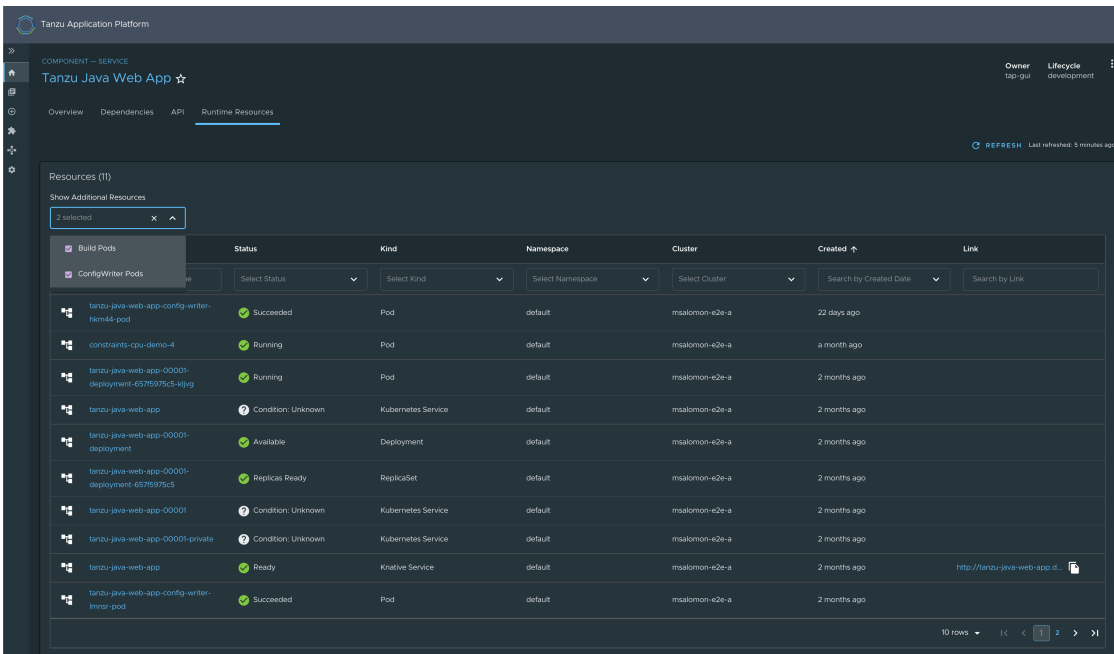
## Annotations and Labels

The Annotations and Labels card displays information about `metadata.annotations` and `metadata.labels`.



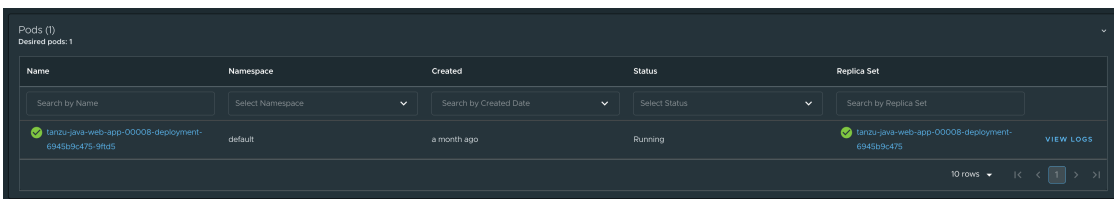
## Selecting completed supply chain pods

Completed supply chain pods (build pods and ConfigWriter pods) are hidden by default in the index table. Users can choose to display them from the **Show Additional Resources** drop-down menu above the Resources index table. This drop-down menu is only visible if the resources include Build or ConfigWriter pods.



## Navigating to the pod Details page

Users can see the pod table in each resource details page.



## Overview of pod metrics

If you have a metrics server running on your cluster, the overview card displays realtime metrics for pods.

If you do not have a metrics server, the overview card displays the user-configured resource limits on the pod, defined in accordance with the [Kubernetes documentation](#).

For applications built using Spring Boot, you can also monitor the actual real-time resource use using [Screenshot of Application Live View for Spring Boot Applications in Tanzu Application Platform GUI](#).

Metrics and limits are also displayed for each container on a pod details page. If a particular container’s current limit conflicts with a namespace-level LimitRange, a small warning indicator is displayed next to the container limit. Most conflicts are due to creating a container before applying a LimitRange.

Container	Status	CPU (Requested / Limit)	Memory (Requested / Limit)	Restarts	Image
queue-proxy	✓	25m / Unlimited	Unlimited / Unlimited	0	sha256:472cd366270790cc14b3737a89cdda6b5208fbf598
workload	✓	Unlimited / Unlimited	Unlimited / Unlimited	0	sha256:53c0e9d6c41ab3a72dce1875c7a09451360c30c04a93

Pods display the sum of the limits of all their containers. If a limit is not specified for a container, both the container and its pod are deemed to require unlimited resources.

Namespace-level resource limits, such as default memory limits and default CPU limits, are not considered as part of these calculations.

For more information about [default memory limits](#) and [default CPU limits](#) see the Kubernetes documentation.

These limits apply only for Memory and CPU that a pod or container can use. Kubernetes manages these resource units by using a binary base, which is explained in the [Kubernetes documentation](#).

## Navigating to Application Live View

To view additional information about your running applications, see the [Application Live View](#) section in the **Pod Details** page.

**Application Live View Details**

- Application Name:** java-web-app
- Namespace:** default
- Location:** https://192.168.1.10:8080
- Health Endpoint:** http://192.168.1.10:8080/health

**Containers Table:**

Container	Status	CPU (Requested / Limit)	Memory (Requested / Limit)	Restarts	Image
java-web-app	✓	25m / Unlimited	Unlimited / Unlimited	0	sha256:472cd366270790cc14b3737a89cdda6b5208fbf598
workload	✓	Unlimited / Unlimited	Unlimited / Unlimited	0	sha256:53c0e9d6c41ab3a72dce1875c7a09451360c30c04a93

## Viewing pod logs

To view logs for a pod, click **View Pod Logs** from the **Pod Details** page. By default, logs for the pod's first container are displayed, dating back to when the pod was created.

The screenshot shows the 'Pod Logs' interface for a pod named 'tanzu-java-web-app-00013-deployment-9586b7bb9-6sgb4'. The pod is in a 'Running' state. The logs are displayed in a scrollable area with a 'Container' dropdown menu set to the first container. The log entries include:

```

280 uptimeMetrics
281 viewControllerHandlerMapping
282 viewNameTranslator
283 viewResolver
284 webEndpointDiscoverer
285 webEndpointPathMapper
286 webExposeExcludePropertyEndpointFilter
287 webMvcMetricsFilter
288 webMvcTagsProvider
289 webServerFactoryCustomizerBeanPostProcessor
290 websocketServletWebServerCustomizer
291 welcomePageHandlerMapping
292 2022-06-23 03:03:40.629 INFO 1 --- [nio-8081-exec-2] o.a.c.c.C.[Tomcat-1].[localhost].[/] : Initializing Spring DispatcherServlet
293 2022-06-23 03:03:40.629 INFO 1 --- [nio-8081-exec-2] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
294 2022-06-23 03:03:40.630 INFO 1 --- [nio-8081-exec-2] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
295 2022-06-28 15:06:58.472 INFO 1 --- [nio-8080-exec-4] o.apache.tomcat.util.http.parser.Cookie : A cookie header was received [times
296 Note: further occurrences of this error will be logged at DEBUG level.
297 2022-06-28 15:06:58.475 INFO 1 --- [nio-8080-exec-4] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet
298 2022-06-28 15:06:58.475 INFO 1 --- [nio-8080-exec-4] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
299 2022-06-28 15:06:58.479 INFO 1 --- [nio-8080-exec-4] o.s.web.servlet.DispatcherServlet : Completed initialization in 4 ms
300 2022-07-14 15:41:11.120 WARN 1 --- [nio-8081-exec-4] .w.s.m.s.DefaultHandlerExceptionResolver : Resolved [org.springframework.web.H

```

## Pausing and resuming logs

Log entries are streamed in real time. New entries appear at the bottom of the log content area. Click or scroll the log content area to pause the log stream. Pausing the log stream enables you to focus on specific entries.

To resume the stream, click the **Follow Latest** button that appears after pausing.

## Filtering by container

To display logs for a different container, select the container that you want from the **Container** drop-down menu.

## Filtering by date and time

To see logs since a specific date and time, select or type the UTC timestamp in the **Since date** field. If no logs are displayed, adjust the timestamp to an earlier time. If you do not select a timestamp, all logs produced since the pod was created are displayed.

For optimal performance, the pod logs page limits the total log entries displayed to the last 10,000, at most.

## Changing log levels

If the pod is associated with an application that supports [Application Live View](#), you can change the application's log levels by clicking the **Change Log Levels** button. You then see a panel that enables you to select levels for each logger associated with your application.

The screenshot displays the 'Log Levels' configuration panel in the Tanzu Application Platform. The panel is titled 'Log Levels' and has a close button (X) in the top right corner. It features a search bar 'Search by Log' and a 'Changes Only' toggle. Below, a list of loggers is shown with buttons for OFF, ERR, WARN, INFO, DEBUG, and TRACE, and a 'RESET' button for each. The 'INFO' level is selected for most loggers. The background shows a pod log stream for 'tanzu-java-web-app-00006-cw1z5q'.

To change the levels for your application, select the desired level for each logger presented, and then click **X** in the upper-right corner of the panel, or press the Escape key, to close the panel.

Because adjusting log levels makes a real-time configuration change to your application, log-level adjustments are only reflected in log entries that your application produces after the change.

If no log entries for the expected levels appear, ensure that:

1. You adjusted the correct application loggers
2. You are viewing logs for the correct container and time frame
3. Your application is currently producing logs at the expected levels

## Line wrapping

By default, log entries are not wrapped. To activate or deactivate line wrapping, click the **Wrap lines** toggle.

## Downloading logs

To download current log content, click the **Download logs** button.

For optimal performance, the pod logs page limits the total log entries downloaded to the last 10,000, at most.

## Connection interruptions

If the log stream connection is interrupted for any reason, such as a network error, a notification appears after the most recent log entry, and the page attempts to reconnect to the log stream. If reconnection fails, an error message displays at the top of the page, and you can click the **Refresh** button at the upper-right of the page to attempt to reconnect.

If you notice frequent disconnections at regular intervals, contact your administrator. Your administrator might need to update the back-end configuration for your installation to allow long-lived HTTP connections to log endpoints (endpoints starting with `BACKEND-HOST/api/k8s-logging/`).

## Application Live View in Tanzu Application Platform GUI

This topic tells you about Application Live View in Tanzu Application Platform GUI (commonly called TAP GUI).

### Overview

The Application Live View features of Tanzu Application Platform include sophisticated components to give developers and operators a view into their running workloads on Kubernetes.

Application Live View shows an individual running process, for example, a Spring Boot application deployed as a workload resulting in a JVM process running inside of a pod. This is an important concept of Application Live View. Application Live View only recognizes running processes. If there is not a running process inside of a running pod, Application Live View does not show anything.

Under the hood, Application Live View uses the concept of actuators to gather data from those running processes. It visualizes them in a semantically meaningful way and allows users to interact with the inner workings of the running processes within limited boundaries.

The actuator data serves as the source of truth. Application Live View provides a live view of the data from inside of the running processes only. It does not store any of that data for further analysis or historical views.

This easy-to-use interface provides ways to troubleshoot, learn, and maintain an overview of certain aspects of the running processes. It gives a level of control to the users to change some parameters, such as environment properties, without a restart (where the Spring Boot application, for example, supports that).

### Entry point to Application Live View plug-in

The Application Live View UI plug-in is part of Tanzu Application Platform GUI. To use the Application Live View plug-in:

1. Select the relevant component under the **Organization Catalog** in Tanzu Application Platform GUI.
2. Select the desired service under the **Runtime Resources** tab.
3. Select the desired pod from the **Pods** section under the **Runtime Resources** tab.
4. You can now see all the details, do some lightweight troubleshooting, and interact with the application within certain boundaries under the **Live View** section.

## Application Live View in Tanzu Application Platform GUI

This topic tells you about Application Live View in Tanzu Application Platform GUI (commonly called TAP GUI).

### Overview

The Application Live View features of Tanzu Application Platform include sophisticated components to give developers and operators a view into their running workloads on Kubernetes.



Application Live View shows an individual running process, for example, a Spring Boot application deployed as a workload resulting in a JVM process running inside of a pod. This is an important concept of Application Live View. Application Live View only recognizes running processes. If there is not a running process inside of a running pod, Application Live View does not show anything.

Under the hood, Application Live View uses the concept of actuators to gather data from those running processes. It visualizes them in a semantically meaningful way and allows users to interact with the inner workings of the running processes within limited boundaries.

The actuator data serves as the source of truth. Application Live View provides a live view of the data from inside of the running processes only. It does not store any of that data for further analysis or historical views.

This easy-to-use interface provides ways to troubleshoot, learn, and maintain an overview of certain aspects of the running processes. It gives a level of control to the users to change some parameters, such as environment properties, without a restart (where the Spring Boot application, for example, supports that).

## Entry point to Application Live View plug-in

The Application Live View UI plug-in is part of Tanzu Application Platform GUI. To use the Application Live View plug-in:

1. Select the relevant component under the **Organization Catalog** in Tanzu Application Platform GUI.
2. Select the desired service under the **Runtime Resources** tab.
3. Select the desired pod from the **Pods** section under the **Runtime Resources** tab.
4. You can now see all the details, do some lightweight troubleshooting, and interact with the application within certain boundaries under the **Live View** section.

## Application Live View for Spring Boot applications in Tanzu Application Platform GUI

This topic tells you about the Application Live View pages for Spring Boot Applications in Tanzu Application Platform GUI (commonly called TAP GUI).

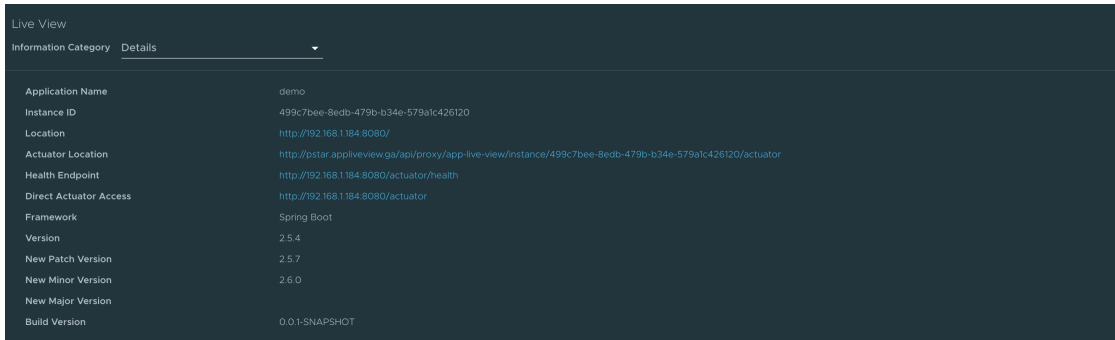
### Details page

This is the default page loaded in the **Live View** section. This page gives a tabular overview containing the following information:

- application name
- instance ID
- location
- actuator location
- health endpoint
- direct actuator access
- framework
- version
- new patch version
- new major version

- build version

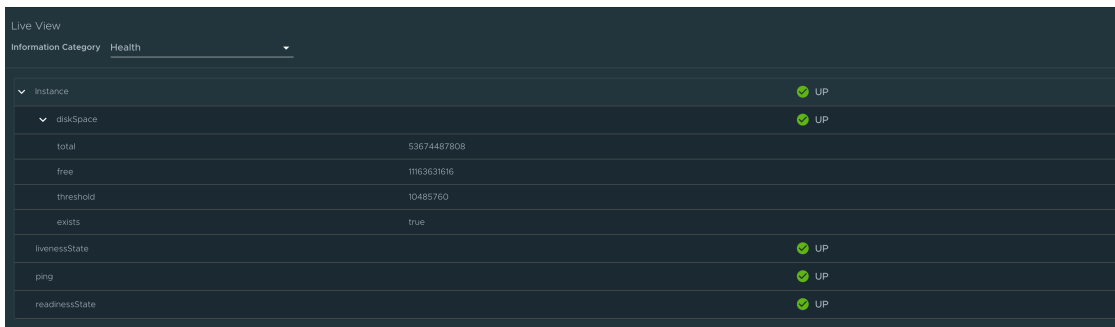
You can navigate between **Information Categories** by selecting from the drop-down menu on the top right corner of the page.



Property	Value
Application Name	demo
Instance ID	499c7bee-8edb-479b-b34e-579a1c426120
Location	http://192.168.1.184:8080/
Actuator Location	http://pstar-appliveview.gx/api/proxy/app-live-view/instance/499c7bee-8edb-479b-b34e-579a1c426120/actuator
Health Endpoint	http://192.168.1.184:8080/actuator/health
Direct Actuator Access	http://192.168.1.184:8080/actuator
Framework	Spring Boot
Version	2.5.4
New Patch Version	2.5.7
New Minor Version	2.6.0
New Major Version	
Build Version	0.0.1-SNAPSHOT

## Health page

To go to the health page, select the **Health** option from the **Information Category** drop-down menu. The health page provides detailed information about the health of the application. It lists all the components that make up the health of the application such as readiness, liveness, and disk space. It displays the status and details associated with each component.



Component	Status
Instance	UP
diskSpace	UP
total	53674487808
free	1163631616
threshold	10485760
exists	true
livenessState	UP
ping	UP
readinessState	UP

## Environment page

To go to the **Environment** page, select the **Environment** option from the **Information Category** drop-down menu. The Environment page contains details of the applications' environment. It contains properties including, but not limited to, system properties, environment variables, and configuration properties (such as application.properties) in a Spring Boot application.

The page includes the following capabilities for **viewing** configured environment properties:

- The UI has a search feature that enables you to search for a property or values.
- Each property has a search icon at the right corner which helps you quickly see all the occurrences of a specific property key without manually typing in the search box. Clicking the search button locates the property name.
- The **Refresh Scope** button on the top right corner of the page probes the application to refresh all the environment properties.

The page also includes the following capabilities for **editing** configured environment properties:

- The UI allows you to edit environment properties and see the live changes in the application. These edits are temporary and go away if the underlying pod is restarted.
- For each configured environment property, you can edit its value by clicking on the **Override** button in the same row. After the value is saved, you can view the message that the property was overridden from the initial value. The updated property is visible in the

**Applied Overrides** section at the top of the page. The **Reset** button in the same row resets the environment property to the initial state.

- You can edit or remove the overridden environment variables in the **Applied Overrides** section.
- The **Applied Overrides** section also enables you to add new environment properties to the application.

**Note**

`management.endpoint.env.post.enabled=true` must be set in the application config properties of the application and a corresponding, editable environment must be present in the application.

Live View
Information Category: Environment

Search by Property or Value...
✕

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	🔍

## Log Levels page

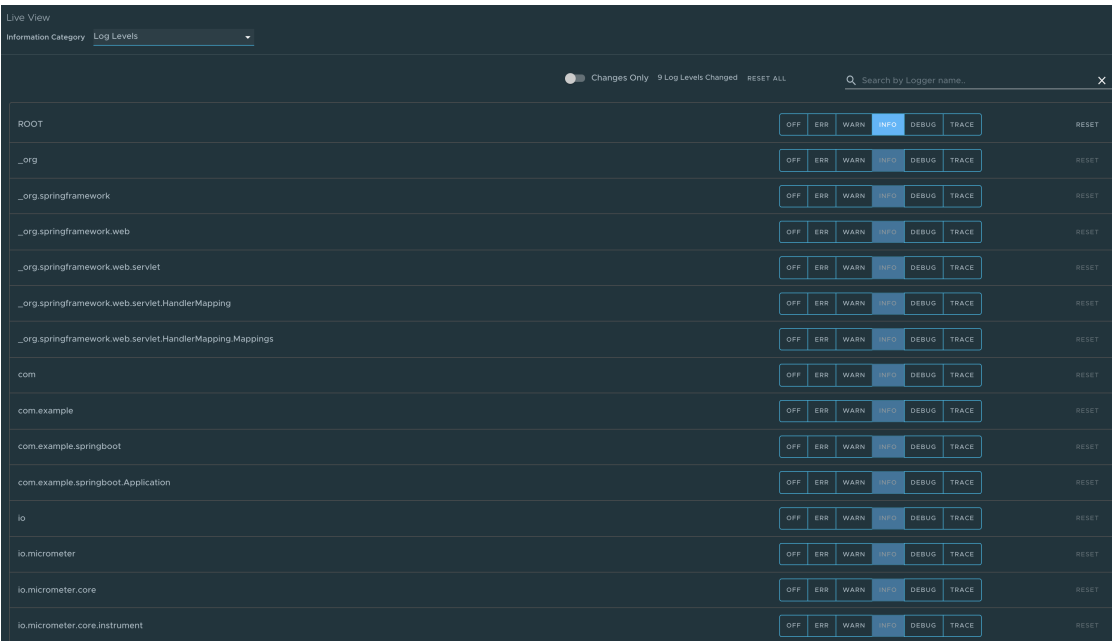
To go to the **Log Levels** page, select the **Log Levels** option from the **Information Category** drop-down menu. The log levels page provides access to the application’s loggers and the configuration of their levels.

You can configure the log levels such as INFO, DEBUG, and TRACE in real time from the UI. You can search for a package and edit its respective log level. You can configure the log levels at a specific class and package. They can deactivate all the log levels by modifying the log level of root logger to OFF.

The toggle **Changes Only** displays the changed log levels. Use the search feature to search by logger name. The **Reset** resets the log levels to the original state. The **Reset All** on top right corner of the page resets all the loggers to default state.

**Note**

Use the UI to change the log levels and see the live changes on the application. These changes are temporary and go away if the underlying pod is restarted.

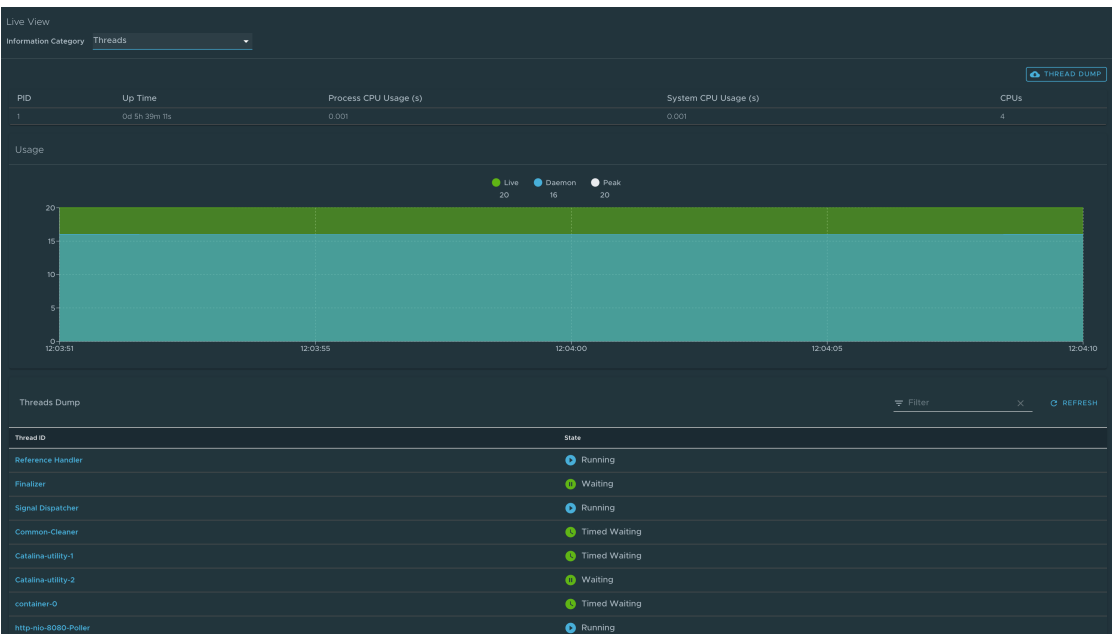


## Threads page

To go to the **Threads** page, select the **Threads** option from the **Information Category** drop-down menu.

This page displays all details related to Java Virtual Machine (JVM) threads and running processes of the application. This tracks live threads and daemon threads real-time. It is a snapshot of different thread states. Navigating to a thread state displays all the information about a particular thread and its stack trace.

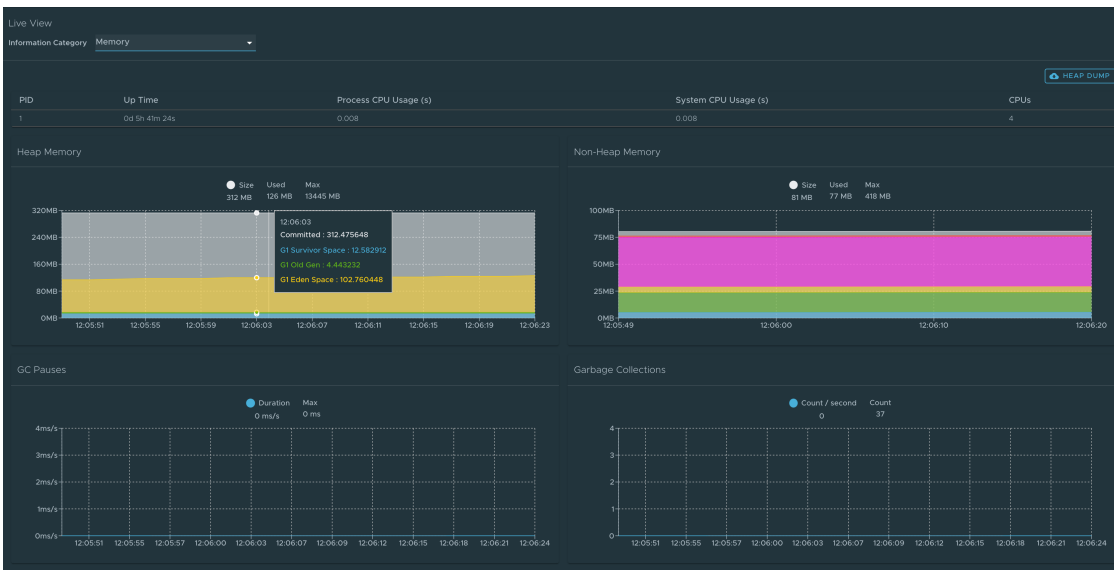
Use the search feature to search for threads by thread ID or state. The refresh icon refreshes to the latest state of the threads. You can view more thread details by clicking on the Thread ID. The page also has a feature to download thread dump data for analysis purposes.



## Memory page

To go to the **Memory** page, select the **Memory** option from the **Information Category** drop-down menu.

- The memory page highlights the memory use inside of the JVM. It displays a graphical representation of the different memory regions within heap and non-heap memory. This visualizes data from inside of the JVM (in case of Spring Boot apps running on a JVM) and therefore provides memory insights into the application in contrast to “outside” information about the Kubernetes pod level.
- The real-time graphs displays a stacked overview of the different spaces in memory with the total memory used and total memory size. The page contains graphs to display the GC pauses and GC events.
- The **Heap Dump** at the top-right corner enables you to download heap dump data.



**Note**

This graphical visualization happens in real time and shows real-time data only. As mentioned at the top, the Application Live View features do not store any information. That means the graphs visualize the data over time only for as long as you stay on that page.

## Request Mappings page

To go to the Request Mappings page, select the **Request Mappings** option from the **Information Category** drop-down menu.

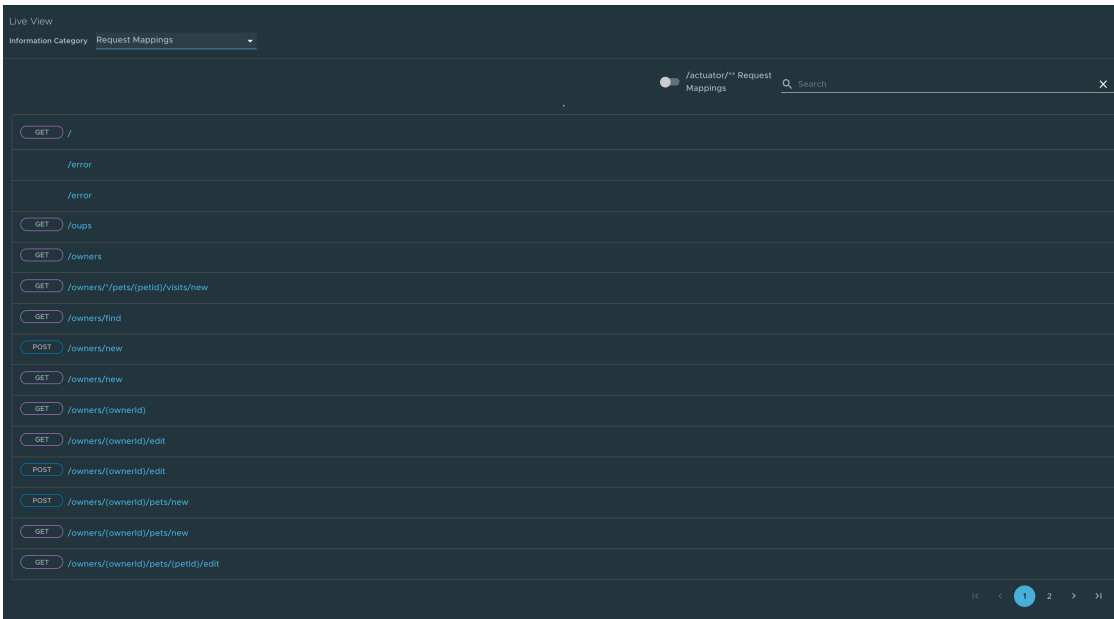
This page provides information about the application’s request mappings. For each mapping, it displays the request handler method. You can view more details of the request mapping, such as the header metadata of the application.

When you click on the request mapping, a side panel appears. This panel contains information about the mapping-media types **Produces** and **Consumes**. The panel also displays the **Handler** class for the request. Use the search feature to search for the request mapping or the method. The toggle **/actuator/\*\* Request Mappings** displays the actuator related mappings of the application.



**Note**

When application actuator endpoint is exposed on `management.server.port`, the application does not return any actuator request mappings data in the context. The application displays a message when the actuator toggle is enabled.



## HTTP Requests page

To go to the HTTP Requests page, click **HTTP Requests** from the **Information Category** drop-down menu. The HTTP Requests page provides information about HTTP request-response exchanges to the application.

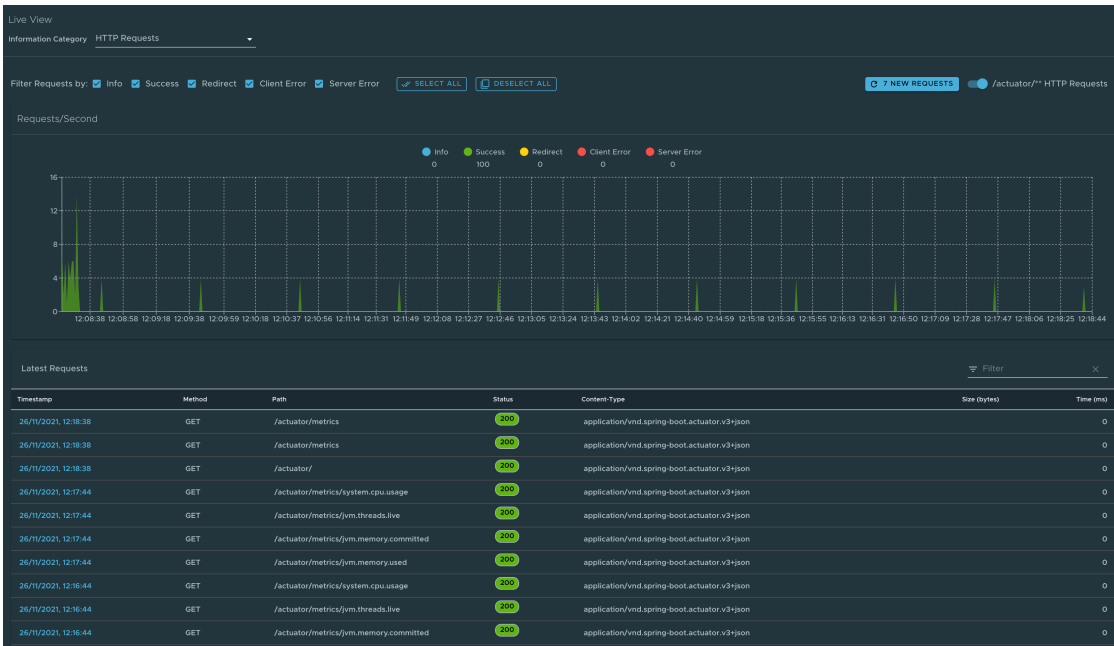
The graph visualizes the requests per second indicating the response status of all the requests. You can filter the response statuses, which include info, success, redirects, client-errors, and server-errors. The trace data is captured in detail in a tabular format with metrics such as timestamp, method, path, status, content-type, length, time.

The search feature on the table filters the traces based on the search field value. You can view more details of the request, such as method, headers, and response of the application by clicking on the timestamp. The refresh icon above the graph loads the latest traces of the application. The toggle `/actuator/**` on the top right corner of the page displays the actuator related traces of the application.



### Note

When application actuator endpoint is exposed on `management.server.port`, no actuator HTTP Traces data is returned for the application. In this case, a message is displayed when the actuator toggle is enabled.

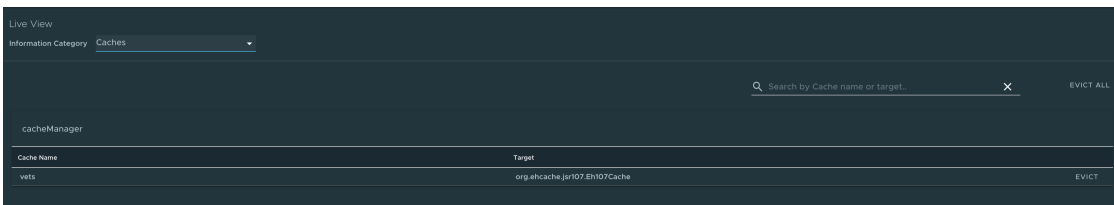


## Caches page

To go to the **Caches** page, select the **Caches** option from the **Information Category** drop-down menu.

The Caches page provides access to the application’s caches. It gives the details of the cache managers associated with the application including the fully qualified name of the native cache.

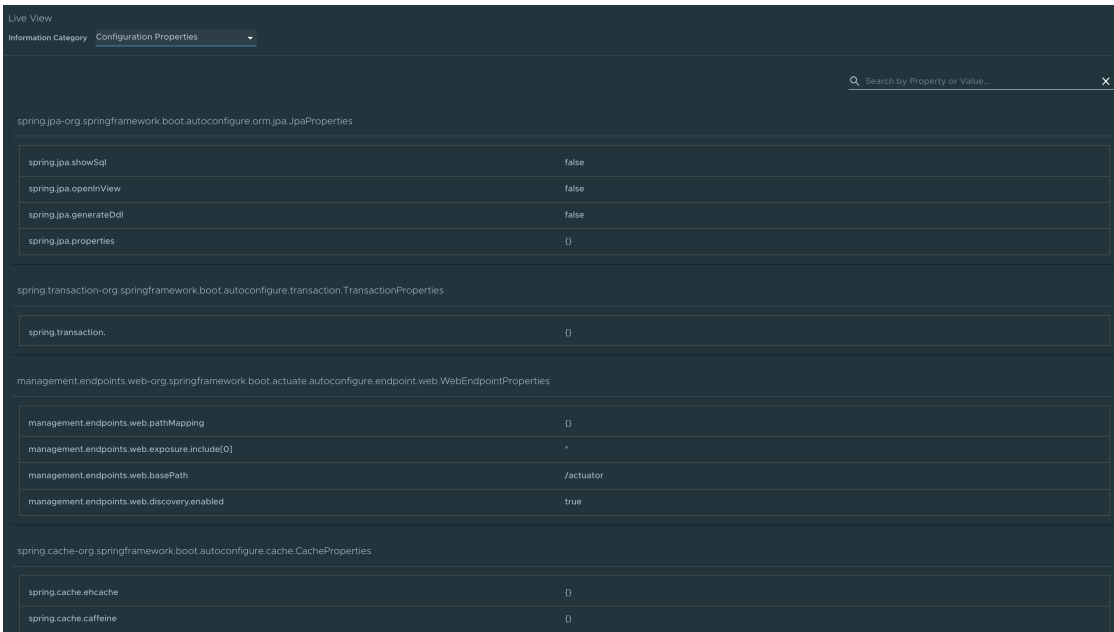
Use the search feature in the Caches Page to search for a specific cache/cache manager. You can clear individual caches by clicking **Evict**. You can clear all the caches completely by clicking **Evict All**. If there are no cache managers for the application, the message **No cache managers available for the application** is displayed.



## Configuration Properties page

To go to the **Configuration Properties** page, select the **Configuration Properties** option from the **Information Category** drop-down menu.

The configuration properties page provides information about the configuration properties of the application. In case of Spring Boot, it displays application’s `@ConfigurationProperties` beans. It gives a snapshot of all the beans and their associated configuration properties. Use the search feature to search for a property’s key/value or the bean name.

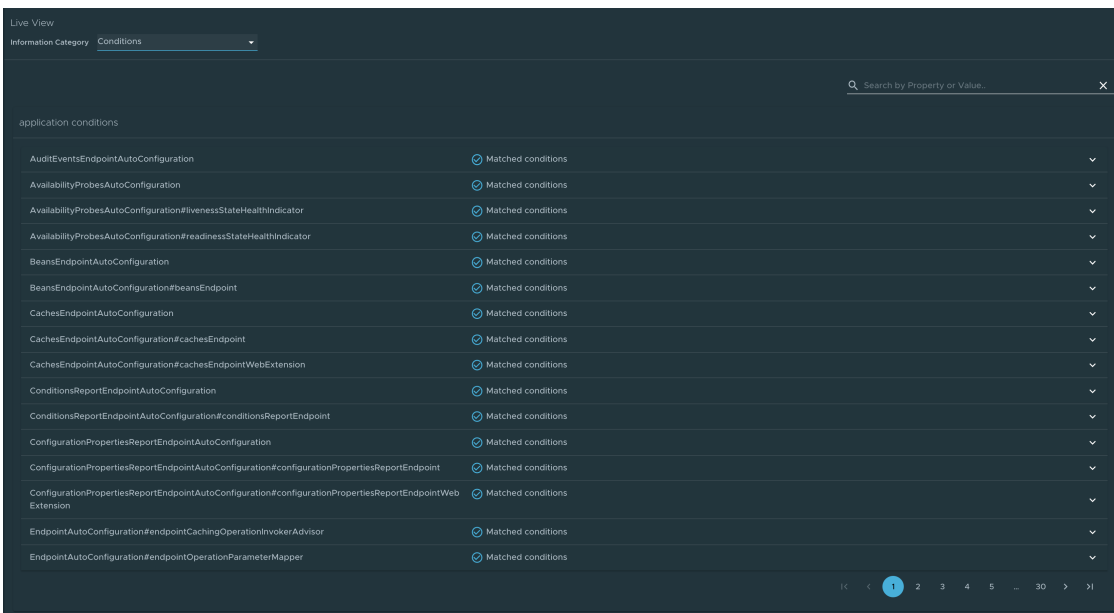


## Conditions page

To go to the **Conditions** page, select the **Conditions** option from the **Information Category** drop-down menu. The conditions evaluation report provides information about the evaluation of conditions on configuration and auto-configuration classes.

In the case of Spring Boot, this gives you a view of all the beans configured in the application. When you click on the bean name, the conditions and the reason for the conditional match is displayed.

In the case of non-configured beans, it shows both the matched and unmatched conditions of the bean if any. In addition to this, it also displays names of unconditional auto configuration classes if any. You can use the search feature to filter out the beans and the conditions.



## Scheduled Tasks page

To go to the **Scheduled Tasks** page, select the **Scheduled Tasks** option from the **Information Category** drop-down menu.



The scheduled tasks page provides information about the application’s scheduled tasks. It includes cron tasks, fixed delay tasks and fixed rate tasks, custom tasks and the properties associated with them.

You can search for a particular property or a task in the search bar to retrieve the task or property details.

Runnable	Expression	Initial Delay	Interval
<b>cron</b>			
com.github.kdvdolder.fortune.service.ScheduledTasks.scheduleTaskUsingCronExpression	0 15 10 15 * ?		
org.springframework.session.jdbc.config.annotation.web.http.JdbcHttpSessionConfiguration\$SessionCleanUpConfiguration\$\$Lambda\$1476/0x0000000000c2d040	0 * * * *		
<b>fixedDelay</b>			
com.github.kdvdolder.fortune.service.ScheduledTasks.lateTime		1000	50000
com.github.kdvdolder.fortune.service.ScheduledTasks.delayedTime		0	15000
<b>fixedRate</b>			
com.github.kdvdolder.fortune.service.FortuneServiceApplication.reportFortne		0	10000
com.github.kdvdolder.fortune.service.LogPing.ping		0	2000
com.github.kdvdolder.fortune.service.ScheduledTasks.reportCurrentTime		0	5000

## Beans page

To go to the **Beans** page, select the **Beans** option from the **Information Category** drop-down menu. The beans page provides information about a list of all application beans and its dependencies. It displays the information about the bean type, dependencies, and its resource. You can search by the bean name or its corresponding fields.

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

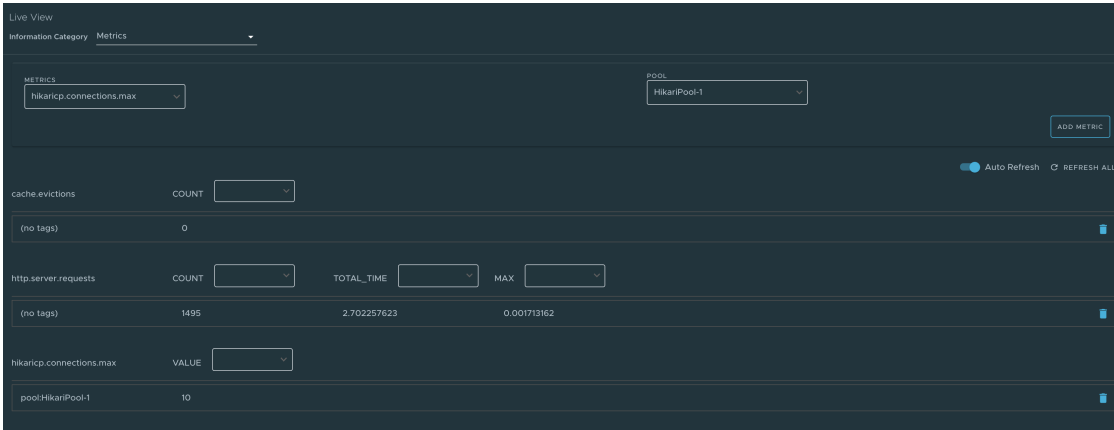
## Metrics page

To go to the **Metrics** page, select the **Metrics** option from the **Information Category** drop-down menu.

The metrics page provides access to application metrics information. You can choose from the list of various metrics available for the application, such as `jvm.memory.used`, `jvm.memory.max`, `http.server.request`, and so on.

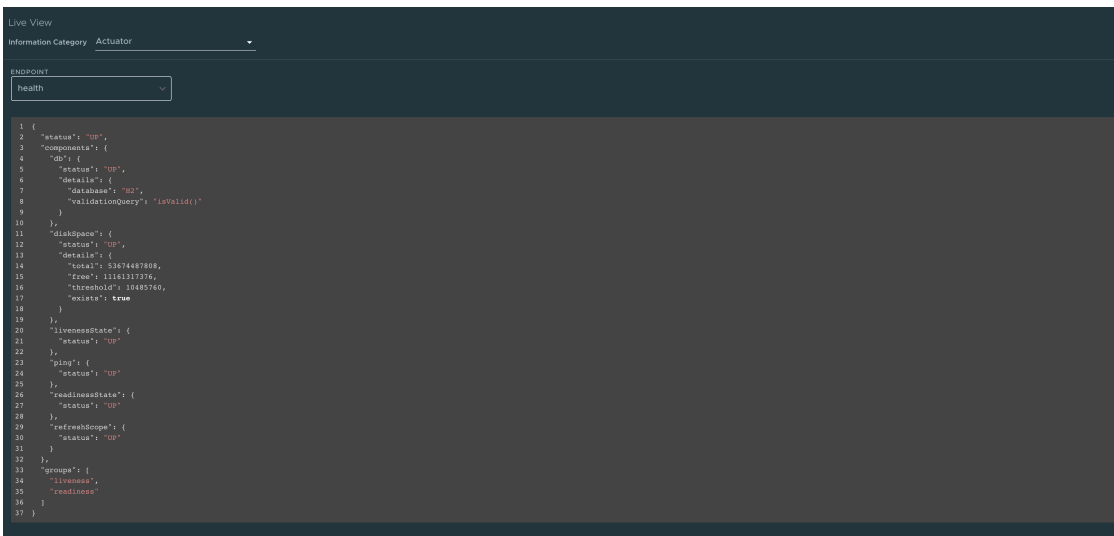
After the metric is chosen, you can view the associated tags. You can choose the value of each tag based on filtering criteria. Clicking **Add Metric** adds the metric to the page which is refreshed every 5 seconds by default.

You can pause the auto refresh feature by deactivating the **Auto Refresh** toggle. You can refresh the metrics manually by clicking **Refresh All**. The format of the metric value can be changed according to your needs. They can delete a particular metric by clicking the minus symbol in the same row.



## Actuator page

To go to the **Actuator** page, select the **Actuator** option from the **Information Category** drop-down menu. The actuator page provides a tree view of the actuator data. You can choose from a list of actuator endpoints and parse through the raw actuator data.



## Troubleshooting

You might run into cases where a workload running on your cluster does not appear in the Application Live View overview, the detail pages do not load any information while running, or similar issues. If you encounter issues, see [Troubleshooting](#) in the Application Live View documentation.

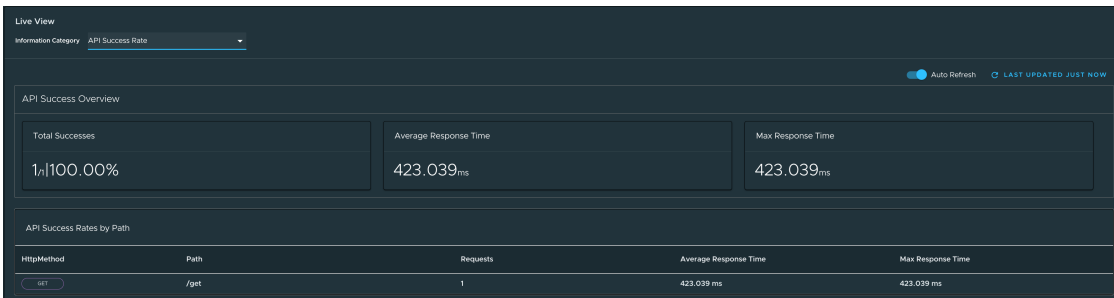
## Application Live View for Spring Cloud Gateway applications in Tanzu Application Platform GUI

This topic tells you about the Application Live View pages for Spring Cloud Gateway applications in Tanzu Application Platform GUI (commonly called TAP GUI).

## API Success Rate page

To access to the API Success Rate page, select the **API Success Rate** option from the **Information Category** drop-down menu.

The API success rate page displays the total successes, average response time, and maximum response time for the gateway routes. It also displays the details of each successful route path.



## API Overview page

To access the API Overview page, select the **API Overview** option from the **Information Category** drop-down menu.

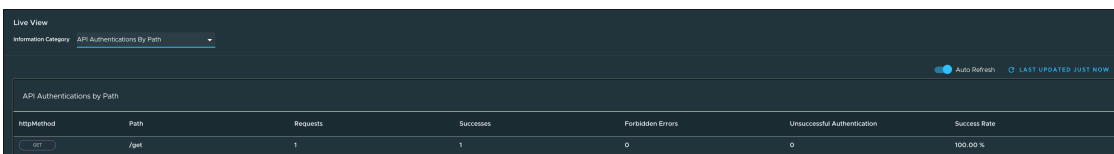
The API Overview page provides route count, number of successes, errors, and the rate-limited requests. It also provides an **auto refresh** feature to get the updated results. These metrics are depicted in a line graph.



## API Authentications By Path page

To access the API Authentications By Path page, select the **API Authentications By Path** option from the **Information Category** drop-down menu.

The API Authentications By Path page displays the total requests, number of successes, and forbidden and unsuccessful authentications grouped by the HTTP method and gateway route path. The page also displays the success rate for each route.



**Note**

In addition to the preceding three pages, the Spring Boot actuator pages are also displayed.

## Troubleshooting

You might run into cases where a workload running on your cluster does not appear in the Application Live View overview, or the detail pages do not load any information while running, or other similar issues. For more information about such issues, see [Troubleshooting](#) in the Application Live View documentation.

## Application Live View for Steeltoe applications in Tanzu Application Platform GUI

This topic tells you about the Application Live View pages for Steeltoe applications in Tanzu Application Platform GUI (commonly called TAP GUI).

### Details page

This is the default page loaded in the **Live View** section. This page gives a tabular overview containing the following information:

- Application name
- Instance ID
- Location
- Actuator location
- Health endpoint
- Direct actuator access
- Framework
- Version
- New patch version
- New major version
- Build version

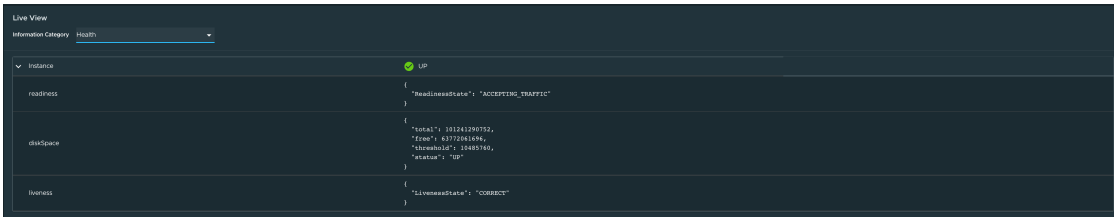
You can navigate between **Information Categories** by selecting from the drop-down menu on the top right corner of the page.

Information Category	Details
Application Name	sample-app
Instance ID	0167d11f-c568-4f5c-a43c-02a6031827
Location	http://70.20.2.10:8080/
Actuator Location	http://tap-gui-34.106.153.203.np.io/app/proxy/app-live-view/instance/0167d11f-c568-4f5c-02a6031827/actuator
Health Endpoint	http://70.20.2.10:8080/actuator/health
Direct Actuator Access	http://70.20.2.10:8080/actuator
Framework	Steeltoe
Version	UNKNOWN
New Patch Version	
New Minor Version	
New Major Version	
Build Version	1.0.0.0

### Health page

To access the health page, select the **Health** option from the **Information Category** drop-down menu.

The health page provides detailed information about the health of the application. It lists all the components that make up the health of the application, such as readiness, liveness, and disk space. It displays the status and details associated with each component.



## Environment page

To access the **Environment** page, select the **Environment** option from the **Information Category** drop-down menu.

The Environment page contains details of the applications' environment. It contains properties including, but not limited to, system properties, environment variables, and configuration properties (such as `appsettings.json`) in a Steeltoe application.

The page includes the following capabilities for **viewing** configured environment properties:

- The UI has a search feature that enables you to search for a property or values.
- Each property has a search icon at the right corner which helps you quickly see all the occurrences of a specific property key without manually typing in the search box. Clicking the search button locates the property name.
- The **Refresh Scope** button on the top right corner of the page probes the application to refresh all the environment properties.

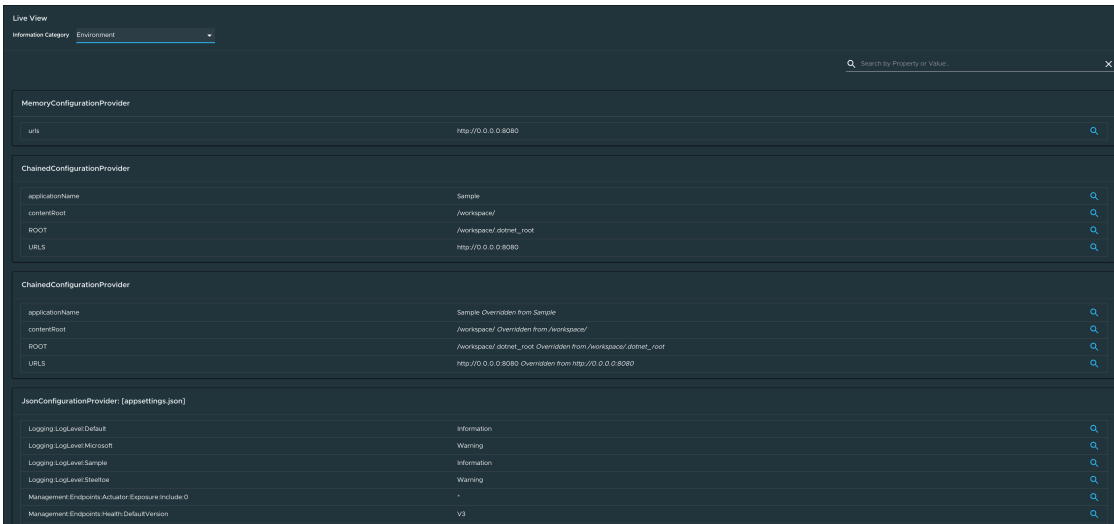
The page also includes the following capabilities for **editing** configured environment properties:

- The UI allows you to edit environment properties and see the live changes in the application. These edits are temporary and go away if the underlying pod is restarted.
- For each of the configured environment properties, you can edit its value by clicking on the **Override** button in the same row. After the value is saved, you can view the message that the property was overridden from the initial value. Also, the updated property is visible in the **Applied Overrides** section at the top of the page. The **Reset** button in the same row resets the environment property to the initial state.
- You can also edit or remove the overridden environment variables in the **Applied Overrides** section.
- The **Applied Overrides** section also enables you to add new environment properties to the application.



### Note

The `management.endpoint.env.post.enabled=true` must be set in the application config properties of the application, and a corresponding editable environment must be present in the application.



## Log Levels page

To go to the **Log Levels** page, select the **Log Levels** option from the **Information Category** drop-down menu. The **Log Levels** page provides access to the application’s loggers and the configuration of the levels.

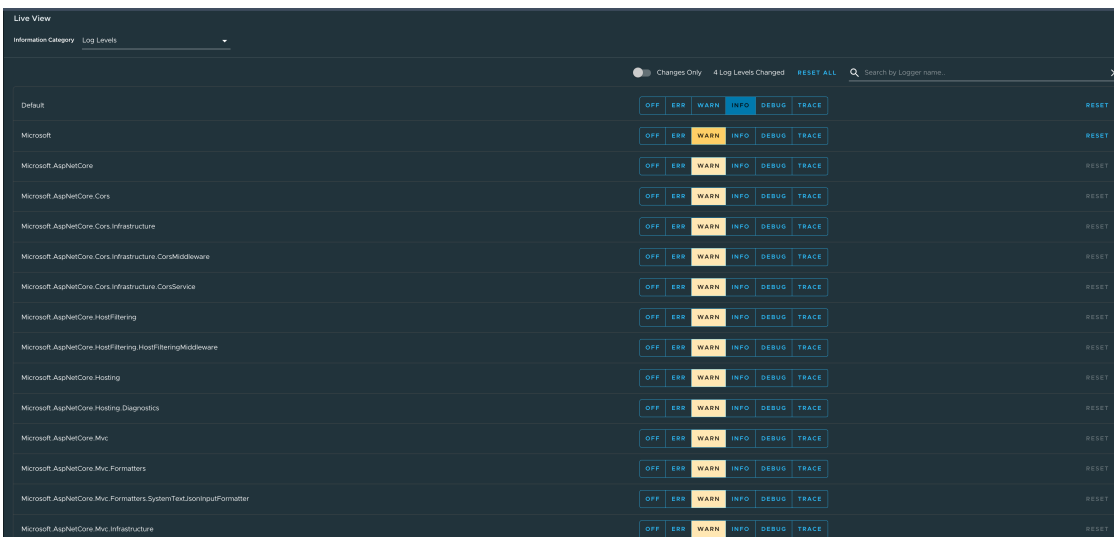
You can:

- Configure log levels, such as **INFO**, **DEBUG**, and **TRACE**, in real time from the UI
- Search for a package and edit its respective log level
- Configure the log levels at a specific class and package
- Deactivate all the log levels by changing the log level of root logger to **OFF**

Use the **Changes Only** toggle to display the changed log levels. Use the search feature to search by logger name. Click **Reset All** to reset all the loggers to the default state.

**Note**

The UI allows you to change the log levels and see the live changes on the application. These changes are temporary and go away if the underlying pod is restarted.

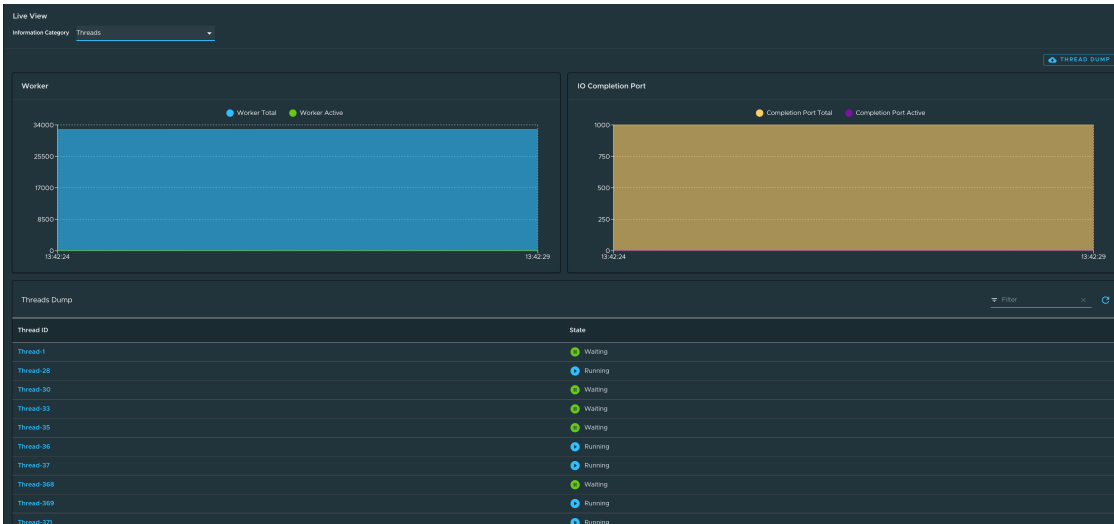


## Threads page

To access the **Threads** page, select the **Threads** option from the **Information Category** drop-down menu.

This page displays all details related to CLR threads and running processes of the application. This tracks worker threads and completion port threads real-time. Navigating to a thread state displays all the information about a particular thread and its stack trace.

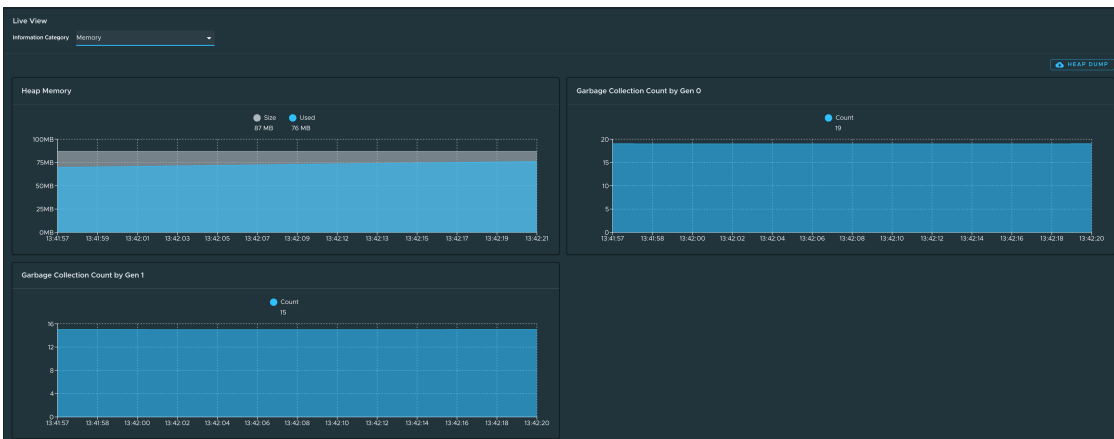
- The refresh icon refreshes to the latest state of the threads.
- To view more thread details, click the thread ID.
- The page also has a feature to download the thread dump for analysis.



## Memory page

To access the **Memory** page, select the **Memory** option from the **Information Category** drop-down menu.

This page displays all details related to used and committed memory of the application. This also displays the garbage collection count by generation (gen0/gen1). The page also has a feature to download the heap dump for analysis.

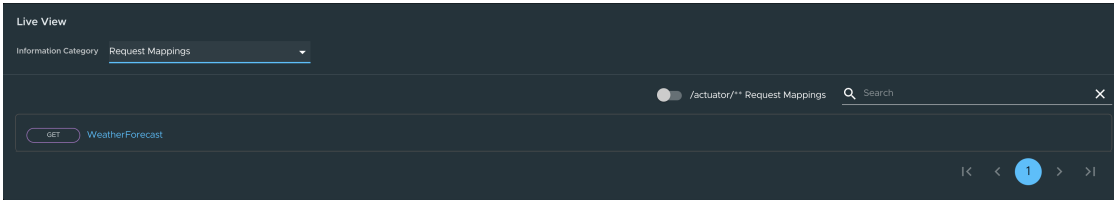


## Request Mappings page

To access the **Request Mappings** page, select the **Request Mappings** option from the **Information Category** drop-down menu.

This page provides information about the application’s request mappings. For each mapping, the page displays the request handler method. You can view more details of the request mapping, such as the header metadata of the application.

When you click on the request mapping, a side panel appears. This panel contains information about the mapping-media types **Produces** and **Consumes**. The panel also displays the **Handler** class for the request. The search feature enables you to search for the request mapping or the method. The toggle **/actuator/\*\* Request Mappings** displays the actuator-related mappings of the application.



## HTTP Requests page

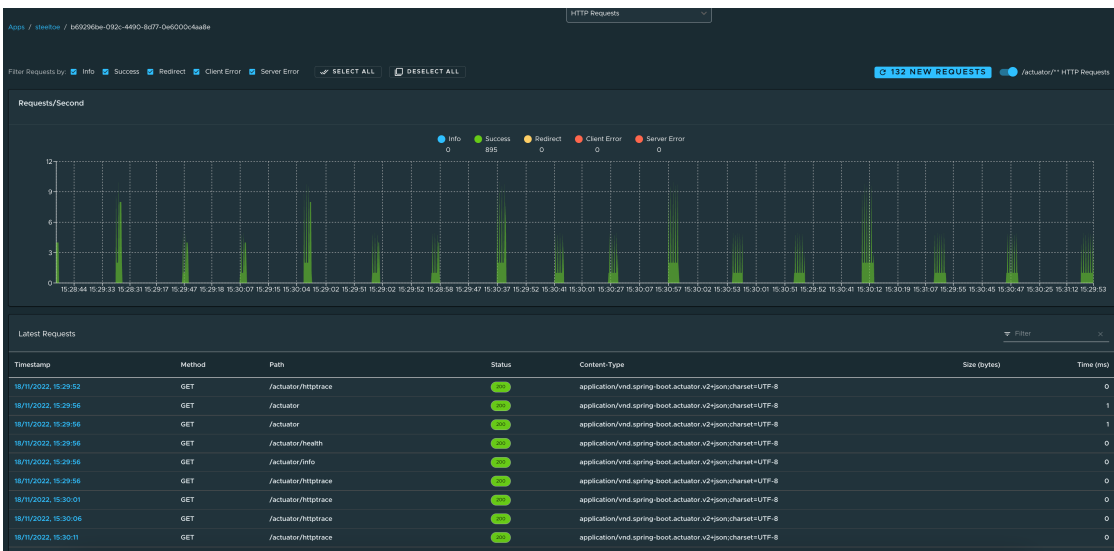
To access the **HTTP Requests** page, select the **HTTP Requests** option from the **Information Category** drop-down menu.

The **HTTP Requests** page provides information about HTTP request-response exchanges to the application.

The graph visualizes the requests per second, which indicates the response status of all the requests. You can filter by the response statuses, which include **info**, **success**, **redirects**, **client-errors**, and **server-errors**. The trace data is captured in detail in a tabular format with metrics, such as **timestamp**, **method**, **path**, **status**, **content-type**, **length**, and **time**.

The search feature on the table filters the traces based on the search text box value. By clicking on the timestamp, you can view more details of the request, such as method, headers, and the response of the application.

The refresh icon above the graph loads the latest traces of the application. The toggle **/actuator/\*\*** on the top-right corner of the page displays the actuator-related traces of the application.



## Metrics page

To access the **Metrics** page, select the **Metrics** option from the **Information Category** drop-down menu.

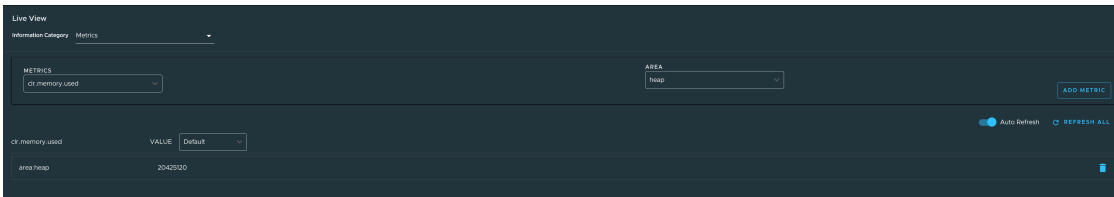


The metrics page provides access to application metrics information. You can choose from the list of various metrics available for the application, such as `clr.memory.used`, `System.Runtime.gc-committed`, `clr.threadpool.active`, and so on.

After you choose the metric, you can view the associated tags. You can choose the value of each of the tags based on filtering criteria. Click **Add Metric** to add the metric to the page. The page is refreshed every 5 seconds by default.

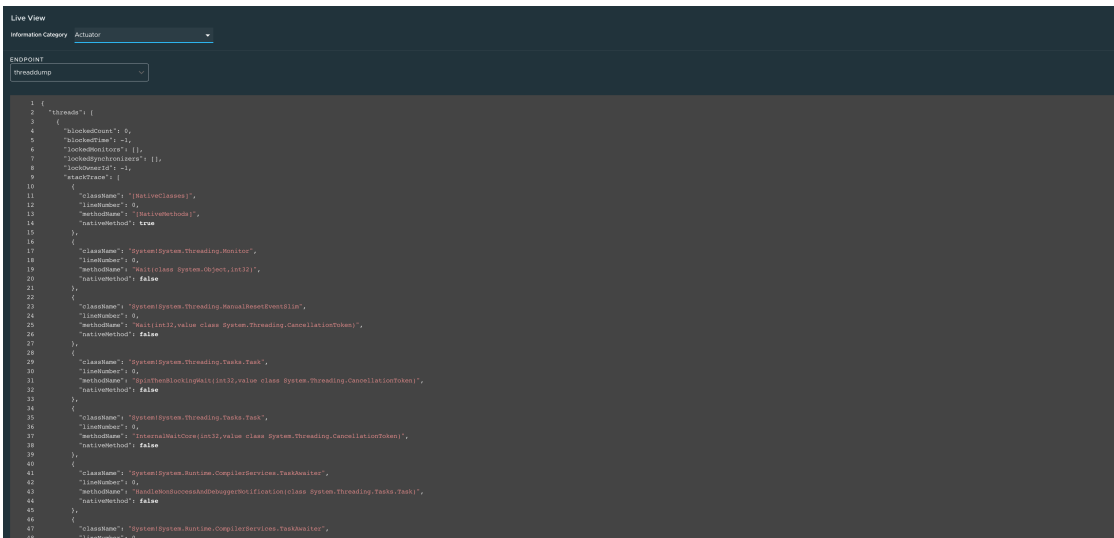
The UI on the **Metrics** page includes features that enable you to:

- Pause the auto refresh feature by deactivating the **Auto Refresh** toggle.
- Refresh the metrics manually by clicking **Refresh All**.
- Change the format of the metric value according to your needs.
- Delete a particular metric by clicking the minus-sign button in the relevant row.



## Actuator page

To access the **Actuator** page, select the **Actuator** option from the **Information Category** drop-down menu. The actuator page provides a tree view of the actuator data. You can choose from a list of actuator endpoints and parse through the raw actuator data.



## Troubleshooting

You might run into cases where a workload running on your cluster does not appear in the Application Live View overview, or the Details pages do not load any information while running, or other similar issues. For help with troubleshooting common issues, see [Troubleshooting](#).

## Application Accelerator in Tanzu Application Platform GUI

This topic tells you how to use Application Accelerator in Tanzu Application Platform GUI (commonly called TAP GUI).

## Overview

Application Accelerator for VMware Tanzu helps you bootstrap developing and deploying your applications in a discoverable and repeatable way.

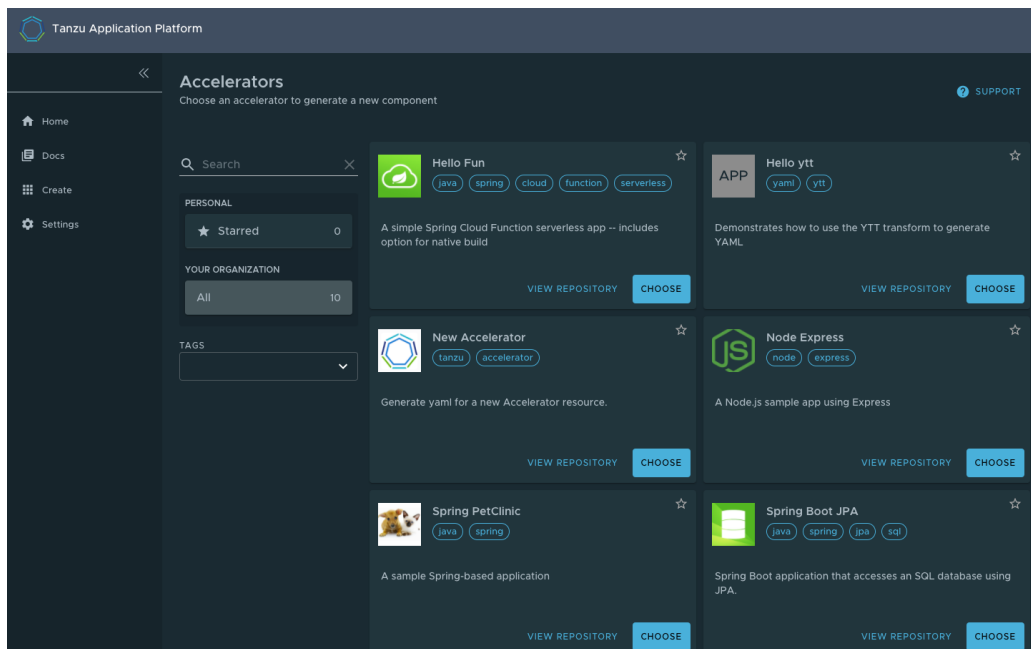
Enterprise architects author and publish accelerator projects that provide developers and operators with ready-made, enterprise-conforming code and configurations. You can then use Application Accelerator to create new projects based on those accelerator projects.

The Application Accelerator UI enables you to discover available accelerators, configure them, and generate new projects to download.

## Access Application Accelerator

To open the Application Accelerator UI plug-in and select an accelerator:

1. Within Tanzu Application Platform, click **Create** in the left navigation pane to open the **Accelerators** page.



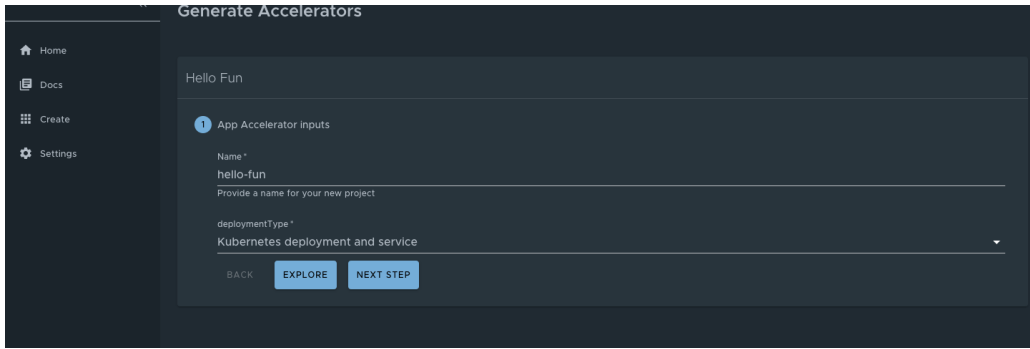
Here you can view accelerators already registered with the system. Developers can add new accelerators by registering them with Kubernetes.

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.

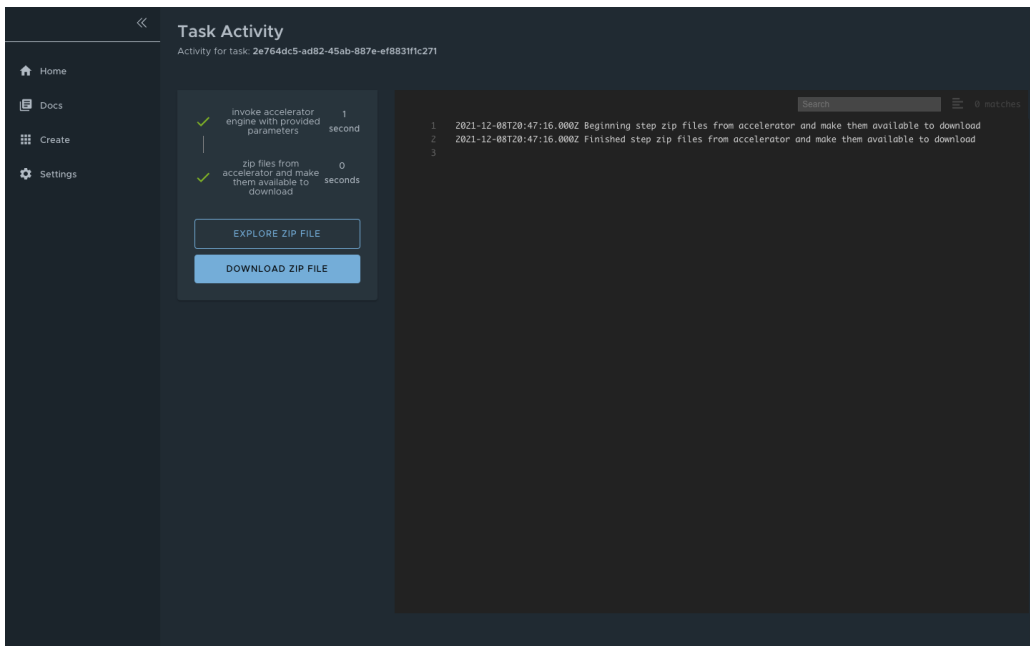


2. Click **EXPLORE** to open the **Explore Project** page and view the project before it is generated.
3. After configuring your project, click **NEXT STEP** to see the project summary page.
4. Review the values you specified for the configurable options.
5. Click **BACK** to make more changes, if necessary. 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).

```

HelloAppApplication.java - hello-fun
src > main > java > com > example > helloapp > HelloAppApplication.java > {} com.example.helloapp
1 package com.example.helloapp;
2
3 import java.util.function.Function;
4
5 import org.springframework.boot.SpringApplication;
6 import org.springframework.boot.autoconfigure.SpringBootApplication;
7 import org.springframework.context.annotation.Bean;
8
9 @SpringBootApplication
10 public class HelloAppApplication {
11
12     @Bean
13     public Function<String, String> hello() {
14         return (in) -> {
15             return "Hello " + in;
16         };
17     };
18
19     Run | Debug
20     public static void main(String[] args) {
21         SpringApplication.run(HelloAppApplication.class, args);
22     }
23 }
24
Ln 1, Col 1 Tab Size: 4 UTF-8 LF Java JavaSE-11

```

## Next steps

To learn more about Application Accelerator for VMware Tanzu, see the [Application Accelerator documentation](#).

## Application Accelerator in Tanzu Application Platform GUI

This topic tells you how to use Application Accelerator in Tanzu Application Platform GUI (commonly called TAP GUI).

### Overview

Application Accelerator for VMware Tanzu helps you bootstrap developing and deploying your applications in a discoverable and repeatable way.

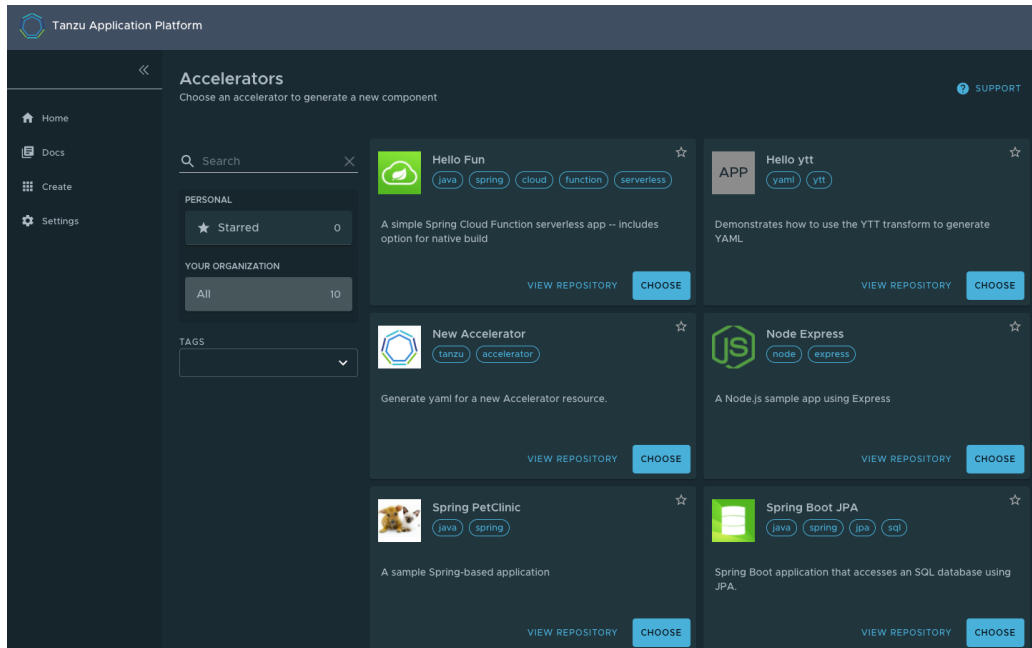
Enterprise architects author and publish accelerator projects that provide developers and operators with ready-made, enterprise-conforming code and configurations. You can then use Application Accelerator to create new projects based on those accelerator projects.

The Application Accelerator UI enables you to discover available accelerators, configure them, and generate new projects to download.

### Access Application Accelerator

To open the Application Accelerator UI plug-in and select an accelerator:

1. Within Tanzu Application Platform, click **Create** in the left navigation pane to open the **Accelerators** page.



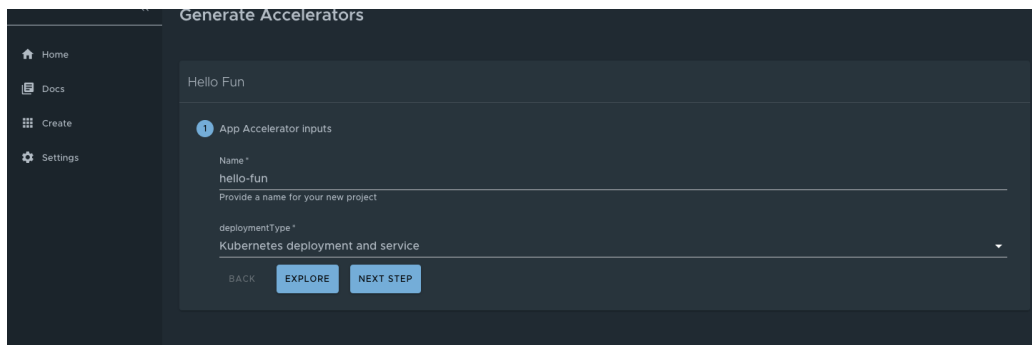
Here you can view accelerators already registered with the system. Developers can add new accelerators by registering them with Kubernetes.

2. Every accelerator has a title and short description. Click **VIEW REPOSITORY** to view an accelerator definition. This opens the accelerator's Git repository in a new browser tab.
3. Search and filter based on text and tags associated with the accelerators to find the accelerator representing the project you want to create.
4. Click **CHOOSE** for the accelerator you want. This opens the **Generate Accelerators** page.

## Configure project generation

To configure how projects are generated:

1. On the **Generate Accelerators** page, add any configuration values needed to generate the project. The application architect defined these values in `accelerator.yaml` in the accelerator definition. Filling some text boxes can cause other text boxes to appear. Fill them all in.

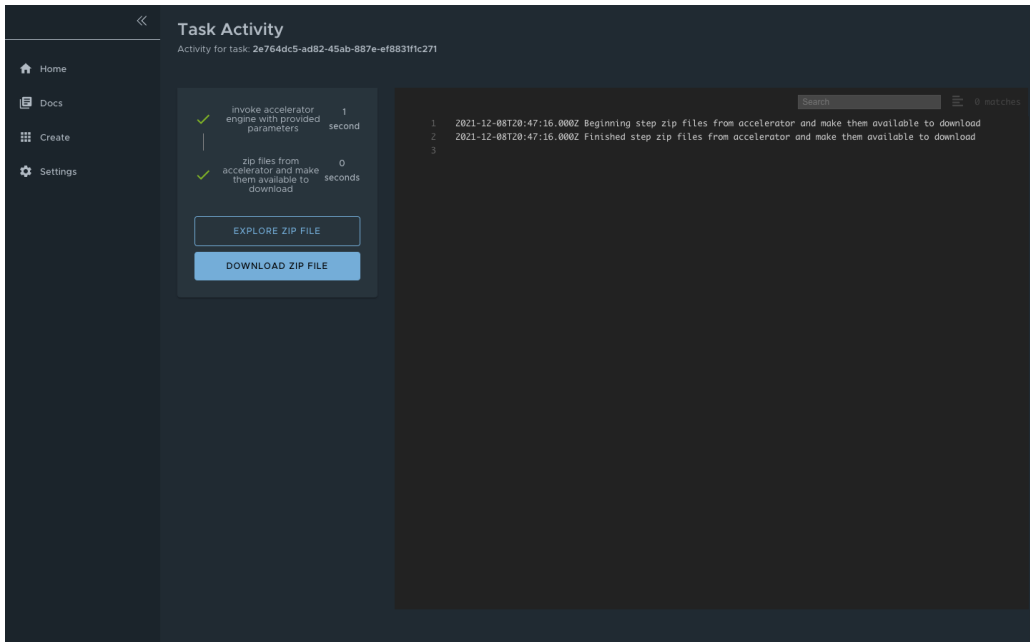


2. Click **EXPLORE** to open the **Explore Project** page and view the project before it is generated.
3. After configuring your project, click **NEXT STEP** to see the project summary page.
4. Review the values you specified for the configurable options.
5. Click **BACK** to make more changes, if necessary. Otherwise, proceed to [create the project](#).

## Create the project

To create the project:

1. Click **Create** to start generating your project. See the progress on the **Task Activity** page. A detailed log is displayed on the right.

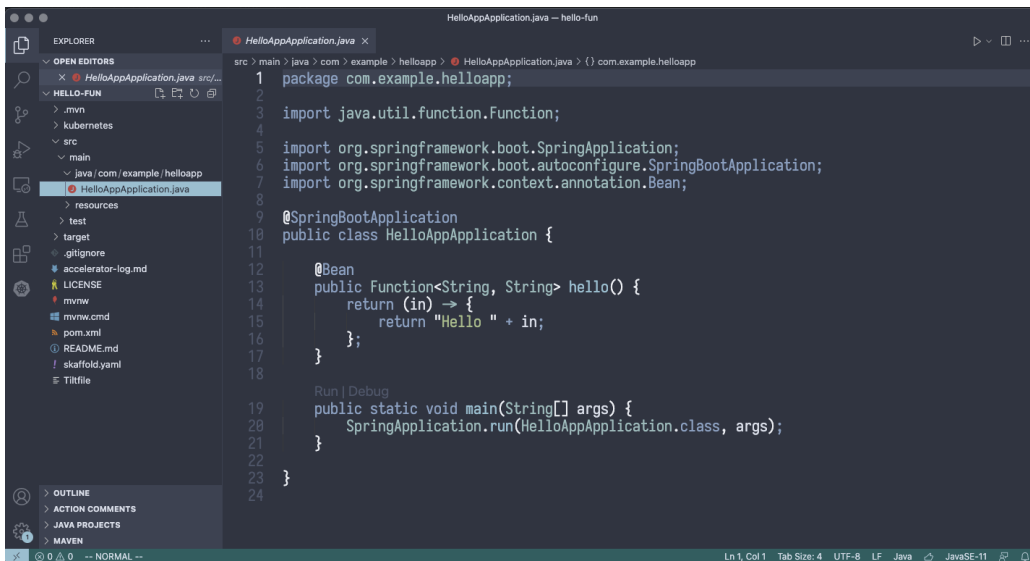


2. After the project is generated, click **EXPLORE ZIP FILE** to open the **Explore Project** page to verify configuration.
3. Click **DOWNLOAD ZIP FILE** to download the project in a ZIP file.

## Develop your code

To develop your code:

1. Expand the ZIP file.
2. Open the project in your integrated development environment (IDE).



## Next steps

To learn more about Application Accelerator for VMware Tanzu, see the [Application Accelerator documentation](#).

## Install Application Accelerator

This topic tells you how to install Application Accelerator from the Tanzu Application Platform (commonly known as TAP) package repository.



### Note

Follow the steps in this topic if you do not want to use a profile to install Application Accelerator. For more information about profiles, see [About Tanzu Application Platform components and profiles](#).

## Prerequisites

Before installing Application Accelerator:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see [Prerequisites](#).
- Install Flux SourceController on the cluster. See [Install Flux CD Source Controller](#).
- Install Source Controller on the cluster. See [Install Source Controller](#).

## Configure properties and resource use

When you install the Application Accelerator, you can configure the following optional properties from within your `.yaml` configuration file:

Property	Default	Description
<code>registry.secret_ref</code>	<code>registry.tanzu&lt;.vmware.com</code>	The secret used for accessing the registry where the App-Accelerator images are located
<code>server.service_type</code>	ClusterIP	The service type for the acc-ui-server service including LoadBalancer, NodePort, or ClusterIP
<code>server.watched_namespace</code>	<code>accelerator-system</code>	The namespace the server watches for accelerator resources
<code>server.engine_invocation_url</code>	<code>http://acc-engine.accelerator-system.svc.cluster.local/invocations</code>	The URL to use for invoking the accelerator engine
<code>engine.service_type</code>	ClusterIP	The service type for the acc-engine service including LoadBalancer, NodePort, or ClusterIP
<code>engine.max_direct_memory_size</code>	32M	The maximum size for the Java <code>-XX:MaxDirectMemorySize</code> setting
<code>samples.include</code>	True	Option to include the bundled sample Accelerators in the installation
<code>ingress.include</code>	False	Option to include the ingress configuration in the installation
<code>ingress.enable_tls</code>	False	Option to include TLS for the ingress configuration
<code>domain</code>	<code>tap.example.com</code>	Top-level domain to use for ingress configuration, default is <code>shared.ingress_domain</code>

Property	Default	Description
tls.secret_name	tls	The name of the secret
tls.namespace	tanzu-system-ingress	The namespace for the secret
telemetry.retain_invocation_events_for_no_days	30	The number of days to retain recorded invocation events resources
telemetry.record_invocation_events	true	The system records each engine invocation when generating files for an accelerator?
git_credentials.secret_name	git-credentials	The name to use for the secret storing Git credentials for accelerators
git_credentials.username	null	The user name to use in secret storing Git credentials for accelerators
git_credentials.password	null	The password to use in secret storing Git credentials for accelerators
git_credentials.ca_file	null	The CA certificate data to use in secret storing Git credentials for accelerators
managed_resources.enable	false	Whether to enable the App used to control managed accelerator resources
managed_resources.git_url	none	Required if managed_resources are enabled. Git repository URL containing manifests for managed accelerator resources
managed_resources.git_ref	origin/main	Required if managed_resources are enabled. Git ref to use for repository containing manifests for managed accelerator resources
managed_resources.git_sub_path	null	Git subPath to use for repository containing manifests for managed accelerator resources
managed_resources.git_secret_ref	git-credentials	Secret name to use for repository containing manifests for managed accelerator resources

VMware recommends that you do not override the default setting for `registry.secret_ref`, `server.engine_invocation_url`, or `engine.service_type`. These properties are only used to configure non-standard installations.

The following table is the resource use configurations for the components of Application Accelerator.

Component	Resource requests	Resource limits
acc-controller	CPU: 100m memory: 20Mi	CPU: 100m memory: 30Mi
acc-server	CPU: 100m memory: 20Mi	CPU: 100m memory: 30Mi
acc-engine	CPU: 500m memory: 1Gi	CPU: 500m memory: 2Gi

## Install

To install Application Accelerator:

1. List version information for the package by running:



```
tanzu package available list accelerator.apps.tanzu.vmware.com --namespace tap-
install
```

For example:

```
$ tanzu package available list accelerator.apps.tanzu.vmware.com --namespace ta
p-install
- Retrieving package versions for accelerator.apps.tanzu.vmware.com...
  NAME                                VERSION  RELEASED-AT
  accelerator.apps.tanzu.vmware.com  1.4.0   2022-12-08 12:00:00 -0500 EST
```

2. (Optional) To make changes to the default installation settings, run:

```
tanzu package available get "accelerator.apps.tanzu.vmware.com/${ACCELERATOR_VE
RSION_NUMBER}" --values-schema --namespace tap-install
```

Where `ACCELERATOR-VERSION-NUMBER` is the version of the Accelerator package that was listed earlier.

For example:

```
tanzu package available get accelerator.apps.tanzu.vmware.com/1.4.0 --values-sc
hema --namespace tap-install
```

For more information about values schema options, see the properties listed earlier.

3. Create an `app-accelerator-values.yaml` using the following example code:

```
server:
  service_type: "LoadBalancer"
  watched_namespace: "accelerator-system"
samples:
  include: true
```

Edit the values if needed or leave the default values.

4. (Optional) For clusters that do not support the `LoadBalancer` service type, override the default value for `server.service_type`. For example:

```
server:
  service_type: "ClusterIP"
  watched_namespace: "accelerator-system"
samples:
  include: true
```

5. Install the package by running:

```
tanzu package install app-accelerator -p accelerator.apps.tanzu.vmware.com -v
${ACCELERATOR_VERSION_NUMBER} -n tap-install -f app-accelerator-values.yaml
```

Where `ACCELERATOR-VERSION-NUMBER` is the version of the Application Accelerator package included with the Tanzu Application Platform installation.

For example:

```
$ tanzu package install app-accelerator -p accelerator.apps.tanzu.vmware.com -v
1.4.0 -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'
```

- Verify the package install by running:

```
tanzu package installed get app-accelerator -n tap-install
```

For example:

```
$ tanzu package installed get app-accelerator -n tap-install
| Retrieving installation details for cc...
NAME:                app-accelerator
PACKAGE-NAME:        accelerator.apps.tanzu.vmware.com
PACKAGE-VERSION:     1.4.0
STATUS:              Reconcile succeeded
CONDITIONS:          [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`.

- To see the IP address for the Application Accelerator API when the `server.service_type` is set to `LoadBalancer`, run:

```
kubectl get service -n accelerator-system
```

This lists an external IP address for use with the `--server-url` Tanzu CLI flag for the Accelerator plug-in `generate & generate-from-local` command.

## Troubleshooting

Depending on the error output, there are some actions that you can take.

### Verify installed packages

The package might be already installed. Verify this by running:

```
tanzu package installed list -n tap-install
```

Look for any package called `accelerator.apps.tanzu.vmware.com`.

### Look at resource events

The error might be within the custom resources such as accelerator, Git repository, fragment, and so on. These errors are checked by using Kubernetes command line interface tool (kubectl).

Here is an example using the custom resource `accelerator`:

```
kubectl get acc -n accelerator-system.
```

It displays the output:

NAME	READY	REASON	AGE
appsso-starter-java	True	Ready	5h2m
where-for-dinner	True	Ready	5h2m
java-function	True	Ready	5h2m
java-rest-service	True	Ready	5h2m
java-server-side-ui	True	Ready	5h2m
node-express	True	Ready	5h2m
node-function	False	Not-Ready	5h2m

python-function	True	Ready	5h2m
spring-cloud-serverless	True	Ready	5h2m
spring-smtp-gateway	True	Ready	5h2m
tanzu-java-web-app	True	Ready	5h2m
tap-initialize	True	Ready	5h2m
weatherforecast-csharp	True	Ready	5h2m
weatherforecast-steeltoe	True	Ready	5h2m

To verify the error event, run:

```
kubectl get acc node-function -n accelerator-system -o yaml
```

You can then look at the event section for more information about the error.

## Create an Application Accelerator Git repository during project creation

This topic tells you how to enable and use GitHub repository creation in the Application Accelerator plug-in of Tanzu Application Platform GUI (commonly called TAP GUI).

### Overview

The Application Accelerator plug-in uses the Backstage GitHub provider integration and the authentication mechanism to retrieve an access token. Then it can interact with the provider API to create GitHub repositories.

### Supported Providers

The supported Git providers are GitHub and GitLab.



#### Note

To create a repository in a self-hosted GitLab, you must add a custom GitLab integration in `tap-values.yaml` as described in the [Full Profile sample](#).

### Configure

The following steps describe an example configuration that uses GitHub:

1. Create an **OAuth App** in GitHub based on the configuration described in this [Backstage documentation](#). GitHub Apps are not supported. For more information about creating an OAuth App in GitHub, see the [GitHub documentation](#).

These values appear in your `app-config.yaml` or `app-config.local.yaml` for local development. For example:

```
auth:
  environment: development
  providers:
    github:
      development:
        clientId: GITHUB-CLIENT-ID
        clientSecret: GITHUB-CLIENT-SECRET
```

2. Add a GitHub integration in your `app-config.yaml` configuration. For example:

```
app_config:
  integrations:
```

```
github:
  - host: github.com
```

For more information, see the [Backstage documentation](#).

## (Optional) Deactivate Git repository creation in the Application Accelerator extension for VS Code

From Tanzu Application Platform v1.4, the Application Accelerator extension for VS Code uses the Tanzu Application Platform GUI URL to interact with the accelerator system. There is a new plug-in called `gitProviders` that you can configure to deactivate Git repository creation in the VS Code extension.

To deactivate Git repository creation, set `app_config.gitProviders.active` to `false` in `tap-values.yaml` as shown in the following example:

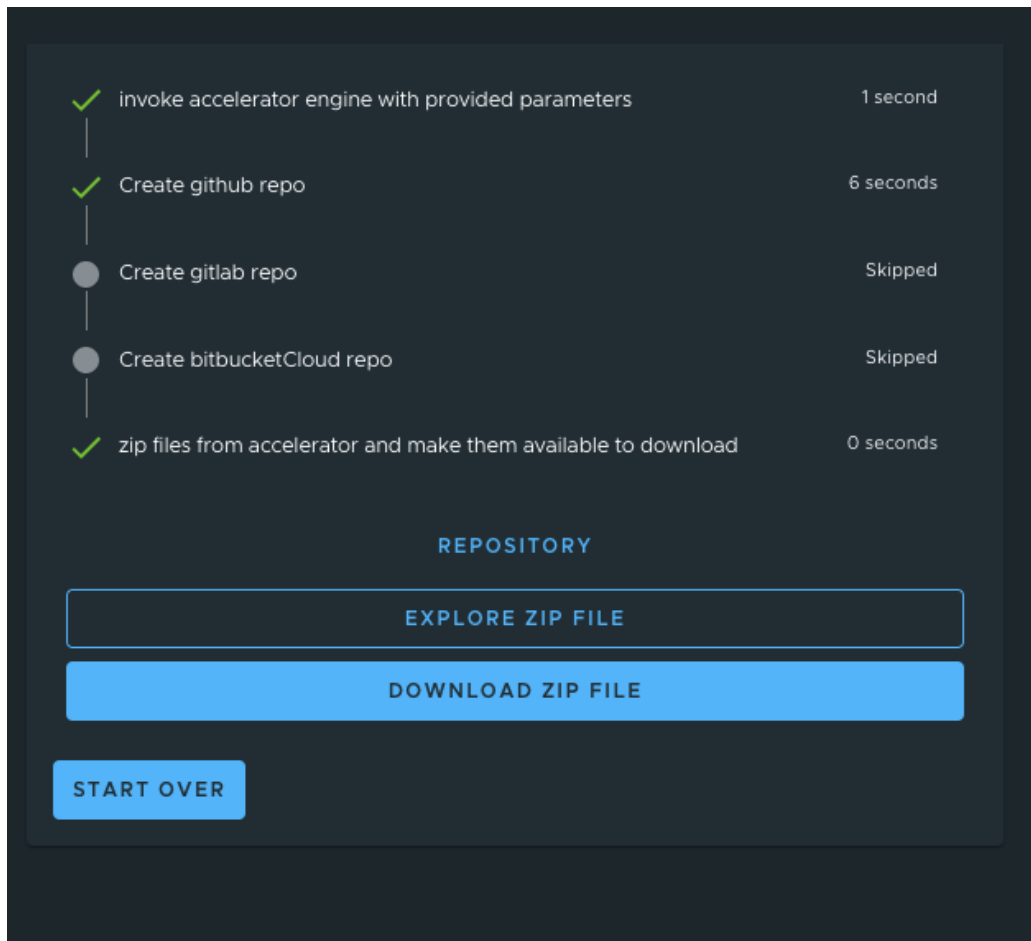
```
app_config:
  gitProviders:
    active: false
```

## Create a Project

To create a project:

1. Go to Tanzu Application Platform GUI, access the Accelerators section, and then select an accelerator. The accelerator form now has a second step named **Git repository**.
2. Fill in the accelerator options and click **Next**.
3. Select the **Create Git repo?** check box.
4. Fill in the **Owner**, **Repository**, and **Default Branch** text boxes.

5. After entering the repository name, a dialog box appears that requests GitHub credentials. Log in and then click **Next**.
6. Click **GENERATE ACCELERATOR**. A link to the repository location appears.



## API documentation plug-in in Tanzu Application Platform GUI

This topic gives you an overview of the API documentation plug-in of Tanzu Application Platform GUI (commonly called TAP GUI). For more information, see [Get started with the API documentation plug-in](#).

### Overview

The API documentation plug-in provides a standalone list of APIs that can be connected to components and systems of the Tanzu Application Platform GUI software catalog.

Each API entity can reflect the components that provide that API and the list of components that are consumers of that API. Also, an API entity can be associated with systems and appear on the system diagram. To show this dependency, make the `spec.providesApis:` and `spec.consumesApis:` sections of the component definition files reference the name of the API entity.

Here's a sample of how you can add `providesApis` and `consumesApis` to an existing component's catalog definition, linking them together.

```
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: example-component
  description: Example Component
spec:
  type: service
  lifecycle: experimental
```

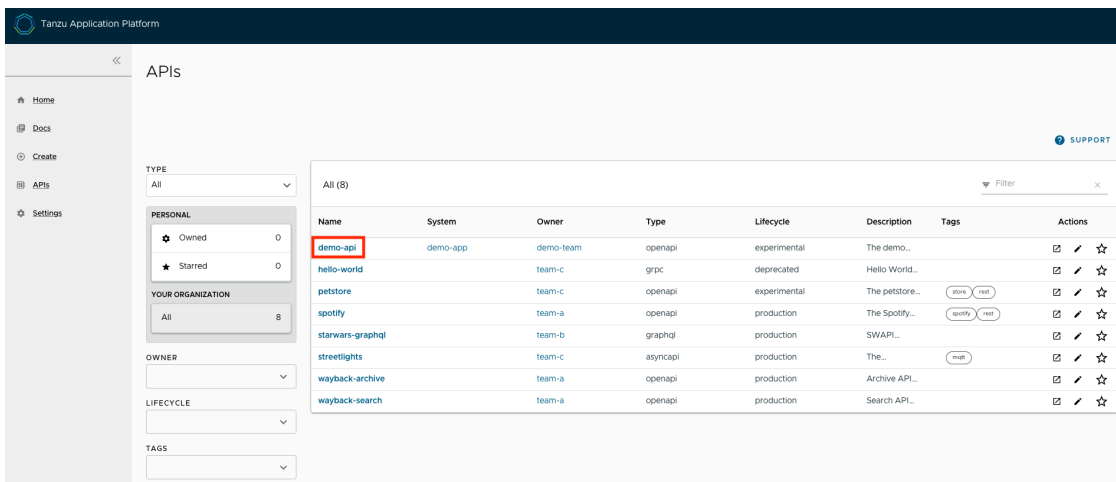
```
owner: team-a
system: example-system
providesApis: # list of APIs provided by the Component
  - example-api-1
consumesApis: # list of APIs consumed by the Component
  - example-api-2
```

For more information about the structure of the definition file for an API entity, see the [Backstage Kind: API documentation](#). For more information about the API documentation plug-in, see the [Backstage API documentation](#) in GitHub.

## Use the API documentation plug-in

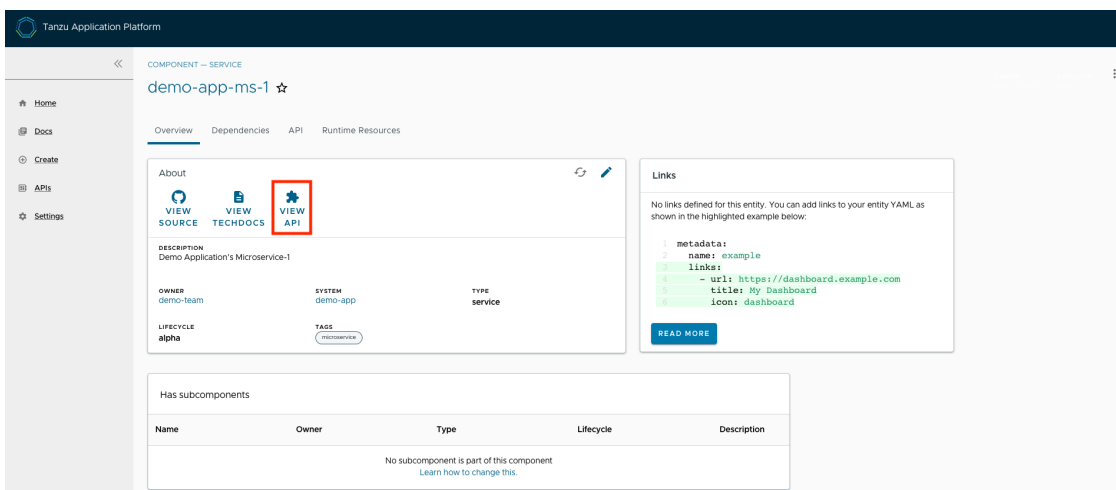
The API documentation plug-in is part of Tanzu Application Platform GUI.

The first way to use the API documentation plug-in is API-first. Click **APIs** in the left navigation pane of Tanzu Application Platform GUI. This opens the **API catalog page**.

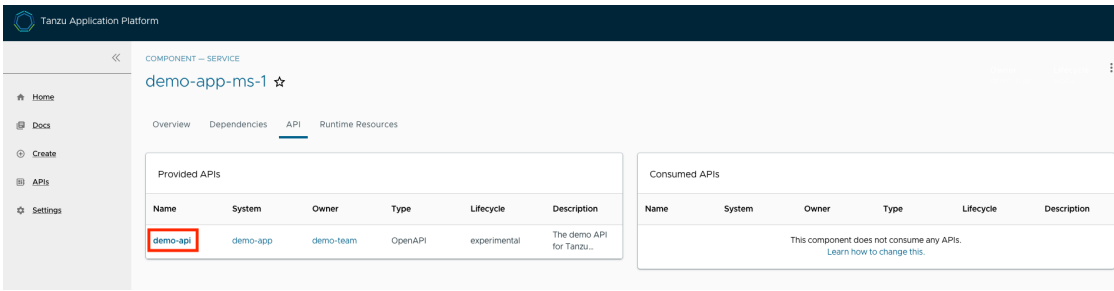


On that page, you can view all the APIs already registered in the catalog regardless of whether they are associated with components or systems.

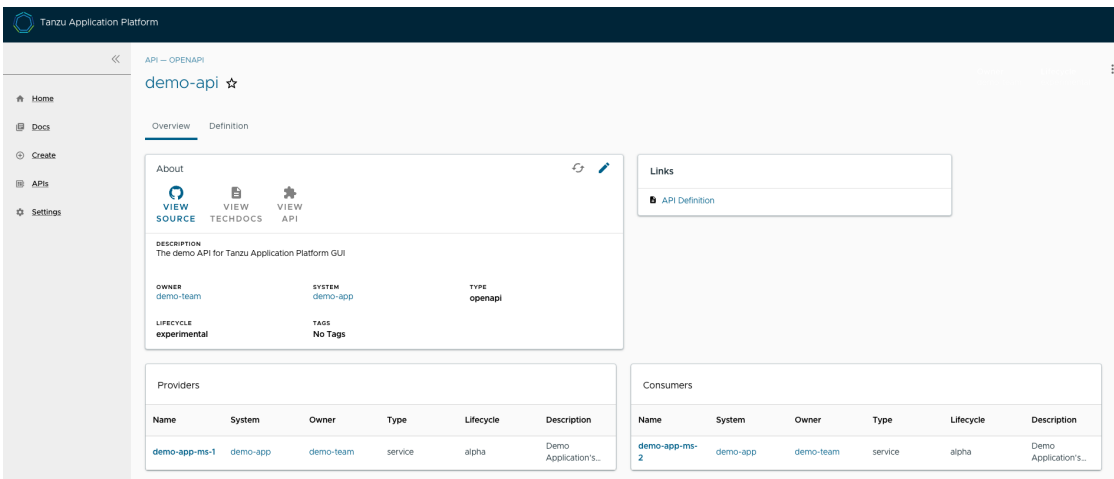
The second way to use the API documentation plug-in is to use components and systems of the software catalog, listed on the home page of Tanzu Application Platform GUI. If there is an API entity associated with the selected component or system, the **VIEW API** icon is active.



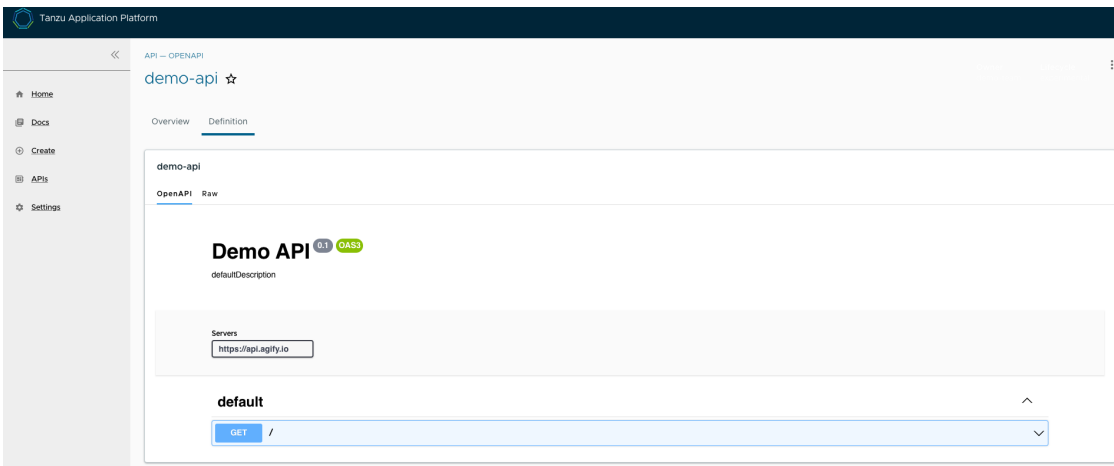
The **VIEW API** tab displays which APIs are being consumed by a component and which APIs are being provided by the component.



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.



Select the **Definition** tab on the top of the API page to see the definition of that API in human-readable and machine-readable format.



The API documentation plug-in supports the following API formats:

- OpenAPI 2 & 3
- AsyncAPI
- GraphQL
- Plain (to support any other format)

## Create a new API entry

You can create a new API entry manually or automatically.

## Manually create a new API entry

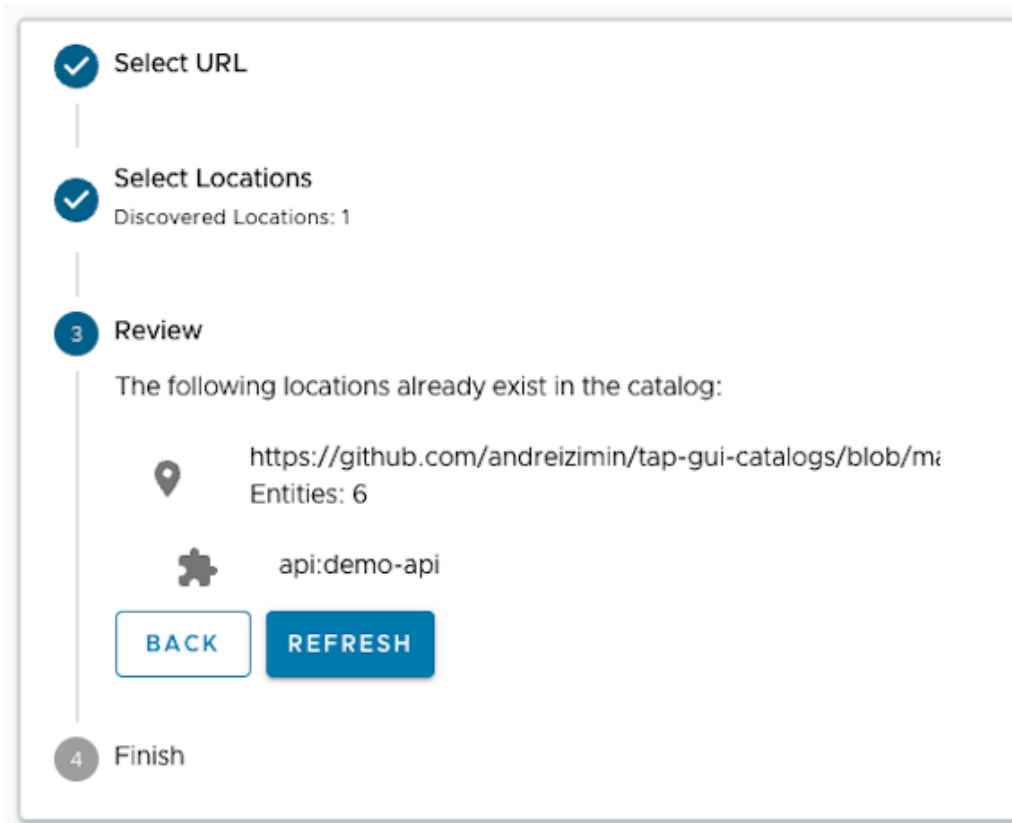
Manually creating a new API entity is similar to registering any other software catalog entity. To manually create a new API entity:

1. Click the **Home** button on the left navigation pane to access the home page of Tanzu 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 navigation pane to view entries on the **API** page.

## Automatically create a new API entry

Tanzu Application Platform v1.3 introduced a feature called **API Auto Registration** that can automatically register your APIs. For more information, see [API Auto Registration](#).

## API documentation plug-in in Tanzu Application Platform GUI

This topic gives you an overview of the API documentation plug-in of Tanzu Application Platform GUI (commonly called TAP GUI). For more information, see [Get started with the API documentation plug-in](#).

### Overview

The API documentation plug-in provides a standalone list of APIs that can be connected to components and systems of the Tanzu Application Platform GUI software catalog.

Each API entity can reflect the components that provide that API and the list of components that are consumers of that API. Also, an API entity can be associated with systems and appear on the system diagram. To show this dependency, make the `spec.providesApis:` and `spec.consumesApis:` sections of the component definition files reference the name of the API entity.

Here's a sample of how you can add `providesApis` and `consumesApis` to an existing component's catalog definition, linking them together.

```
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
```

```

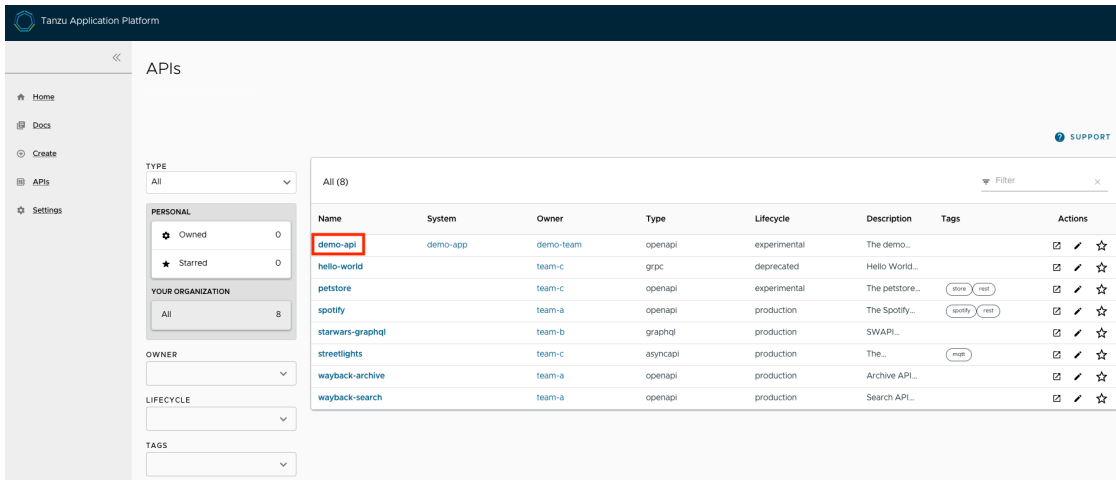
name: example-component
description: Example Component
spec:
  type: service
  lifecycle: experimental
  owner: team-a
  system: example-system
  providesApis: # list of APIs provided by the Component
  - example-api-1
  consumesApis: # list of APIs consumed by the Component
  - example-api-2
    
```

For more information about the structure of the definition file for an API entity, see the [Backstage Kind: API documentation](#). For more information about the API documentation plug-in, see the [Backstage API documentation](#) in GitHub.

## Use the API documentation plug-in

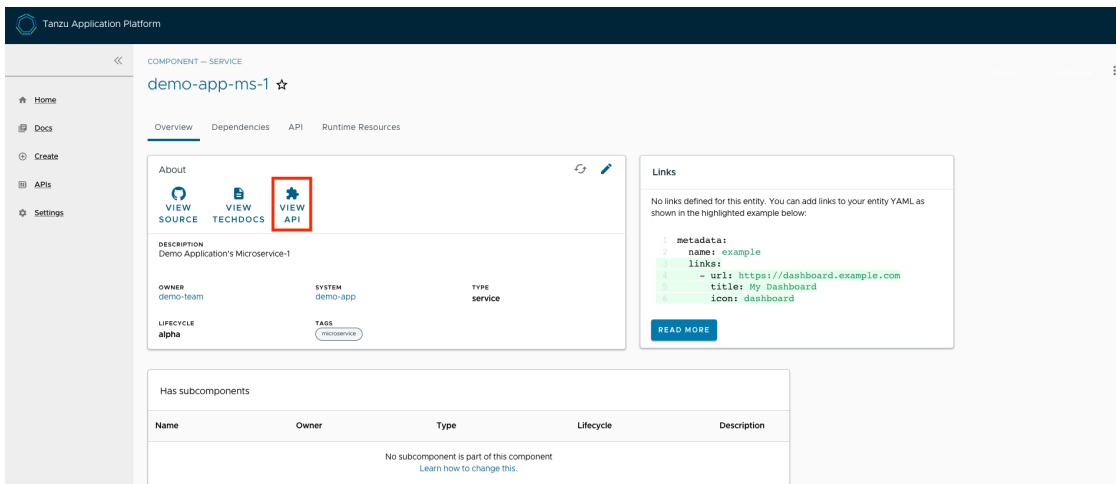
The API documentation plug-in is part of Tanzu Application Platform GUI.

The first way to use the API documentation plug-in is API-first. Click **APIs** in the left navigation pane of Tanzu Application Platform GUI. This opens the **API catalog page**.

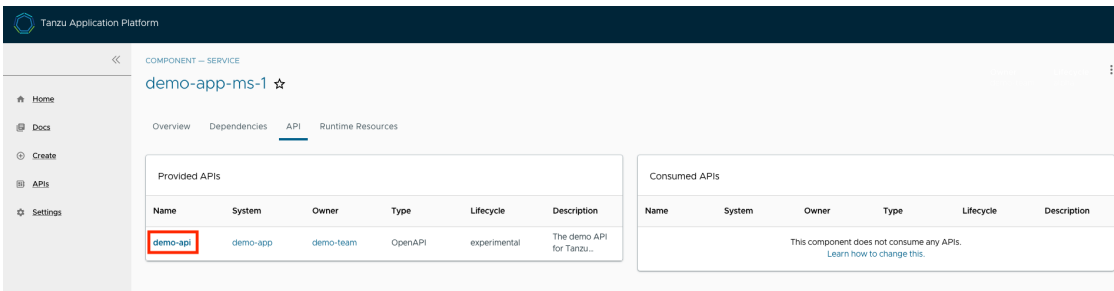


On that page, you can view all the APIs already registered in the catalog regardless of whether they are associated with components or systems.

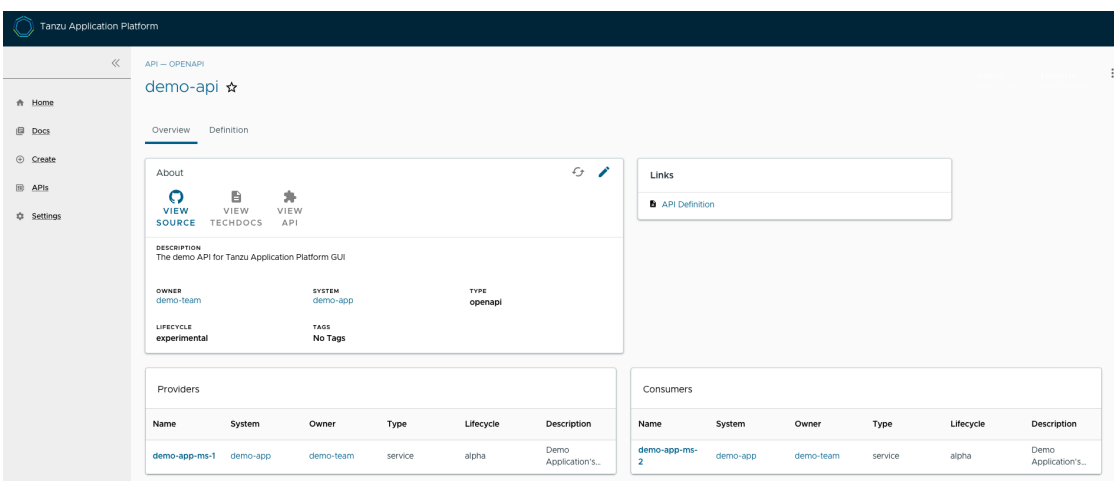
The second way to use the API documentation plug-in is to use components and systems of the software catalog, listed on the home page of Tanzu Application Platform GUI. If there is an API entity associated with the selected component or system, the **VIEW API** icon is active.



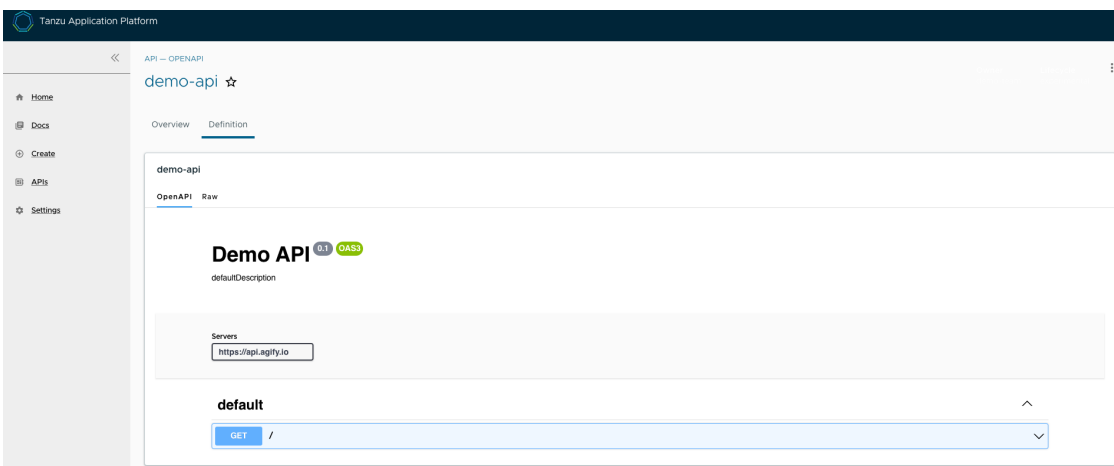
The **VIEW API** tab displays which APIs are being consumed by a component and which APIs are being provided by the component.



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.



Select the **Definition** tab on the top of the API page to see the definition of that API in human-readable and machine-readable format.



The API documentation plug-in supports the following API formats:

- OpenAPI 2 & 3
- AsyncAPI
- GraphQL
- Plain (to support any other format)

## Create a new API entry

You can create a new API entry manually or automatically.

### Manually create a new API entry

Manually creating a new API entry is similar to registering any other software catalog entity. To manually create a new API entry:

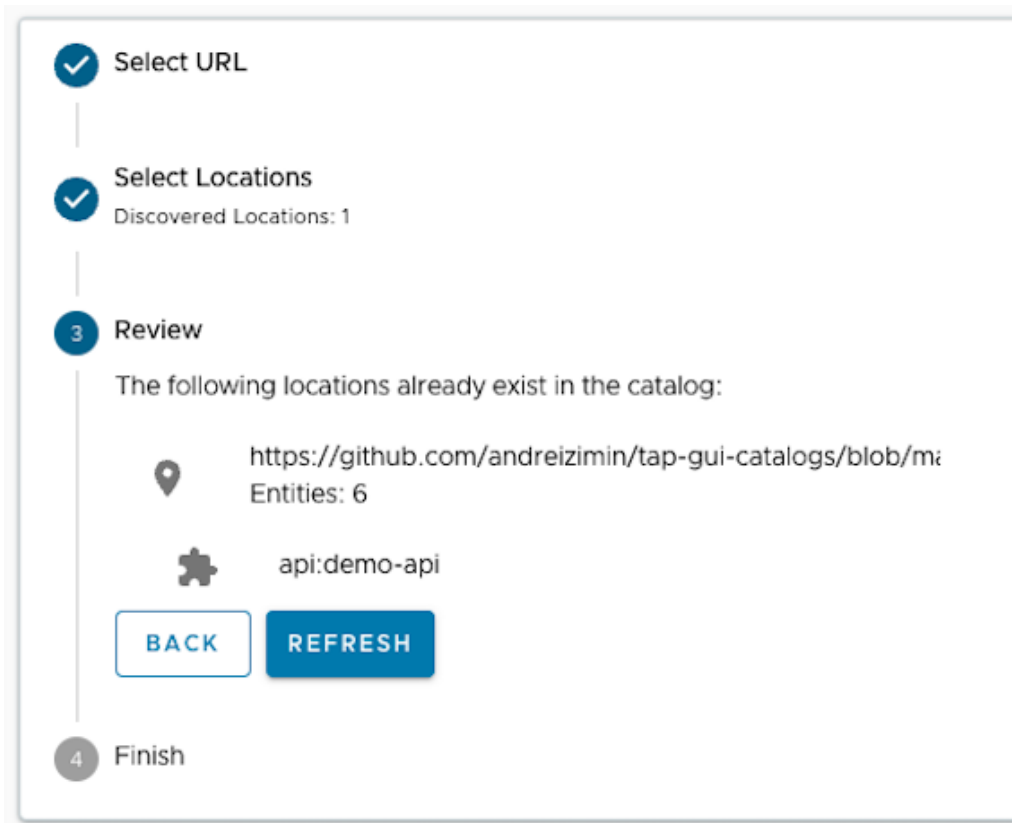
1. Click the **Home** button on the left navigation pane to access the home page of Tanzu Application Platform GUI.
2. Click **REGISTER ENTITY**.
3. **Register an existing component** prompts you to type a repository URL. Paste the link to the `catalog-info.yaml` file of your choice that contains the definition of your API entry. 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 navigation pane to view entries on the **API** page.

## Automatically create a new API entry

Tanzu Application Platform v1.3 introduced a feature called **API Auto Registration** that can automatically register your APIs. For more information, see [API Auto Registration](#).

## Get started with the API documentation plug-in

This topic tells you how to get started with the API documentation plug-in in Tanzu Application Platform GUI (commonly called TAP GUI).

## API entries

This section describes API entities, how to add them, and how to update them.

### About API entities

The list of API entities is visible on the left side navigation pane of Tanzu 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 raw machine-readable and human-readable formats. For more information, see the [API plug-in documentation](#).

### Add a demo API entity to the Tanzu Application Platform GUI software catalog

To add a demo API entity and its related Catalog objects, follow the steps used for registering any other software catalog entity:

1. Go to the Home page of Tanzu Application Platform GUI by clicking **Home** on the left-side navigation pane.
2. Click **REGISTER ENTITY**.
3. In the repository URL text box, type the link to the `catalog-info.yaml` file of your choice or use the following sample definition.
4. Save this code block as `catalog-info.yaml`.
5. Upload it to the Git repository of your choice and copy the link to `catalog-info.yaml`. This demo setup includes a domain named `demo-domain` with a single system named `demo-system`. This systems consists of two microservices (`demo-app-ms-1` and `demo-app-ms-1`) and one API named `demo-api` that `demo-app-ms-1` provides and that `demo-app-ms-2` consumes.

```

apiVersion: backstage.io/v1alpha1
kind: Domain
metadata:
  name: demo-domain
  description: Demo Domain for Tanzu Application Platform
  annotations:
    'backstage.io/techdocs-ref': dir:.
spec:
  owner: demo-team

---

apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: demo-app-ms-1
  description: Demo Application's Microservice-1
  tags:
    - microservice
  annotations:
    'backstage.io/kubernetes-label-selector': 'app.kubernetes.io/part-of=demo-a
pp-ms-1'
    'backstage.io/techdocs-ref': dir:.
spec:
  type: service
  providesApis:
    - demo-api
  lifecycle: alpha
  owner: demo-team
  system: demo-app

---

apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: demo-app-ms-2
  description: Demo Application's Microservice-2
  tags:
    - microservice
  annotations:
    'backstage.io/kubernetes-label-selector': 'app.kubernetes.io/part-of=demo-a
pp-ms-2'
    'backstage.io/techdocs-ref': dir:.
spec:
  type: service
  consumesApis:
    - demo-api
  lifecycle: alpha
  owner: demo-team

```

```

system: demo-app

---

apiVersion: backstage.io/v1alpha1
kind: System
metadata:
  name: demo-app
  description: Demo Application for Tanzu Application Platform
  annotations:
    'backstage.io/techdocs-ref': dir:.
spec:
  owner: demo-team
  domain: demo-domain

---

apiVersion: backstage.io/v1alpha1
kind: API
metadata:
  name: demo-api
  description: The demo API for Tanzu 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: {}

```

6. Paste the link to [catalog-info.yaml](#) and click **ANALYZE**.
7. Review the catalog entities and click **IMPORT**.

✓ Select URL

✓ Select Locations  
Discovered Locations: 1

3 Review

The following entities will be added to the catalog:

- 📍 <https://github.com/andreizimin/tap-gui-catalogs/blob/m>  
Entities: 6
- 🧩 api:demo-api
- ⚙️ component:demo-app-ms-1
- ⚙️ component:demo-app-ms-2
- 🏠 domain:demo-domain
- 📍 location:generated-f2d07ff2e480829c875974257e47c7
- 🏠 system:demo-app

BACK IMPORT

4 Finish

- Go to the **API** page by clicking **APIs** on the left side navigation pane. The catalog changes and entries are visible for further inspection. If you select the system **demo-app**, the diagram appears as follows:



## Update your demo API entry



To update your demo API entity, click on **demo-api** from the list of available APIs in your software catalog and click the **Edit** icon on the Overview page.

It opens the source `catalog-info.yaml` file that you can edit. For example, you can change the `spec.paths.parameters.example` from `type_any_name` to `Tanzu` and then save your changes.

```

![Screenshot of the overview of demo dash api. The edit button on the card labeled Abo
ut is framed in red.](../images/api-plugin-9.png)

It opens the source `catalog-info.yaml` file that you can edit. For example, you can c
hange the
`spec.paths.parameters.example` from `type_any_name` to `Tanzu` and then save your cha
nges.
    
```

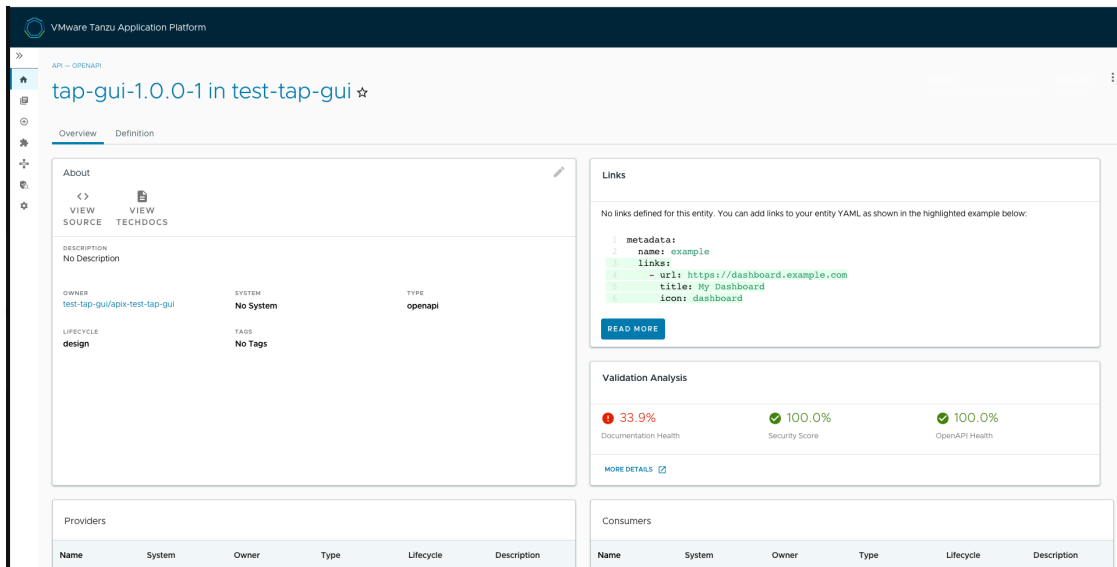
After making any edits, Tanzu Application Platform GUI re-renders the API entry with the next refresh cycle.

## Validation Analysis of API specifications

This section describes the Validation Analysis card, the data format needed to populate the card, and how to get automatic scores for your OpenAPI entities.

### About the Validation Analysis card

When viewing entities of the kind `API` on the Overview tab, you see the Validation Analysis card that displays the health of an API through various scoring parameters.



To display the health scores, an API entity must contain the following metadata structure:

```

apiVersion: backstage.io/v1alpha1
kind: API
metadata:
  name: NAME
  description: DESCRIPTION
apiscores:
  scores:
    - id: documentationReport
      label: "Documentation Health"
      value: 34.375
      valueType: percentage
      status: failed
    - id: securityReport
      label: "Security Score"
    
```

```

value: 70.0
valueType: percentage
status: warning
- id: openApiHealthReport
  label: "OpenAPI Health"
  value: 89.0625
  valueType: percentage
  status: passed
  scoreDetailsURL: VALIDATION-REPORT-URL-FOR-MORE-DETAILS
# Other API Entity parameters

```

If an API entity follows this schema, the Validation Analysis card displays helpful information about the API.

```

- id:          # Unique ID
  label:       # Descriptive label displayed as a title over the numerical value
  value:       # Any number value
  valueType:   # One of the types (percentage or other). Displays the % symbol or di
splays nothing.
  status:      # One of the statuses (passed, warning, or failed). Displays the numb
er in green, yellow, or red.

```

## Automatic OpenAPI specification validation

To receive automatic validation analysis for OpenAPI specifications by using API Validation Scoring:

1. [Install API Validation and Scoring](#).
2. Use [API Auto Registration](#) or API Validation Scoring Design GitOps to automatically generate the API entities in Tanzu Application Platform GUI.

The automatic scoring cannot score or replace API entities created through other methods, such as regular GitOps or manual registration. You might see the following message signaling that the OpenAPI specification was registered with regular GitOps methods or manual registration.

```

Validation analysis is currently unavailable for APIs registered via TAP GUI without b
eing attached \
to a workload.

```

## Security Analysis in Tanzu Application Platform GUI

This topic tells you about the Security Analysis plug-in in Tanzu Application Platform GUI (commonly called TAP GUI).

### Overview

The Security Analysis plug-in summarizes vulnerability data across all workloads running in Tanzu Application Platform, enabling faster identification and remediation of CVEs.

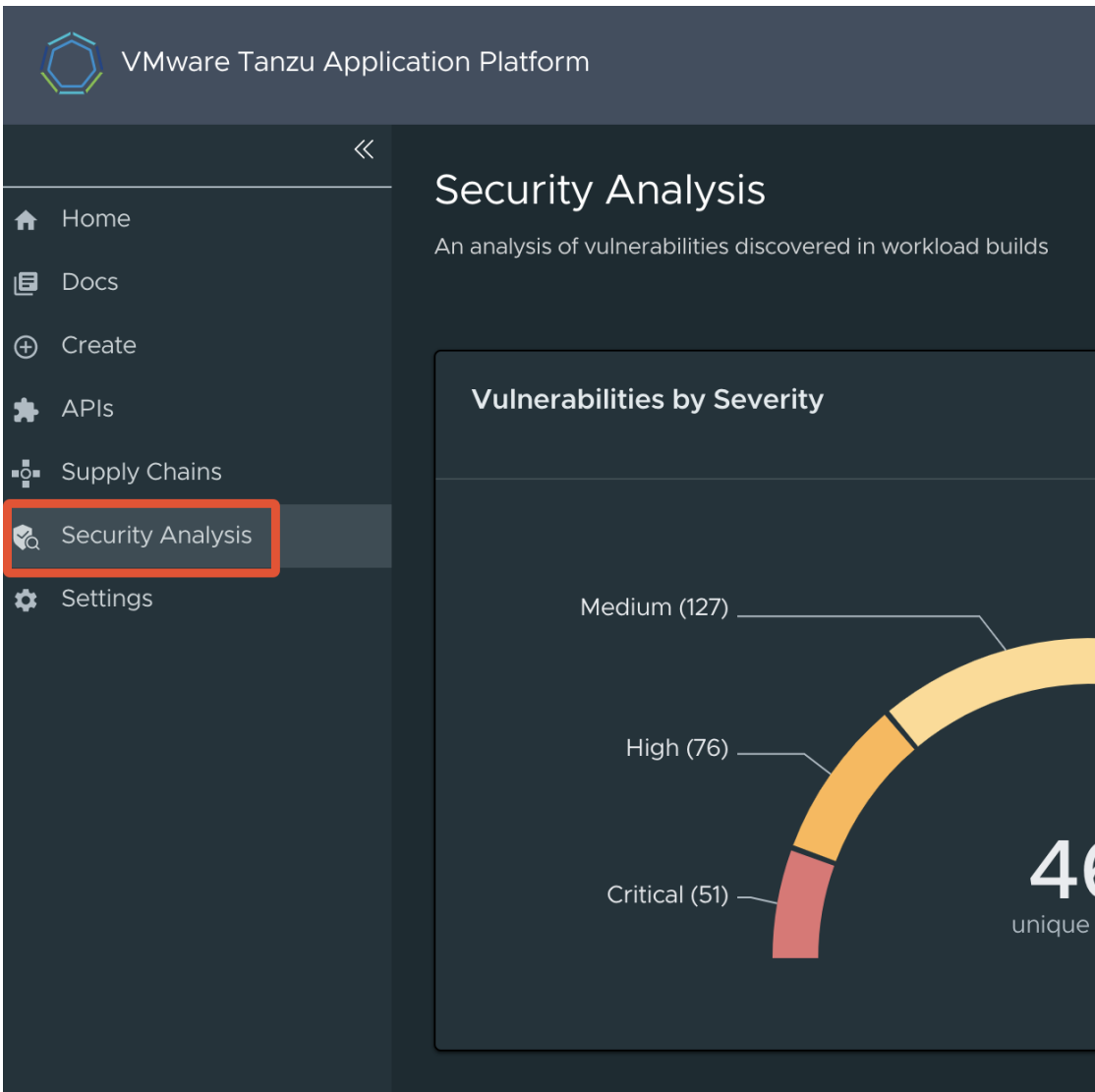
### Installing and configuring

The Security Analysis plug-in is installed by default. It is tightly coupled with the [Supply Chain Choreographer plug-in](#). After installing and configuring the Supply Chain Choreographer GUI plug-in, there is no additional configuration needed for the Security Analysis plug-in.

The Security Analysis plug-in is part of the Tanzu Application Platform Full and View profiles.

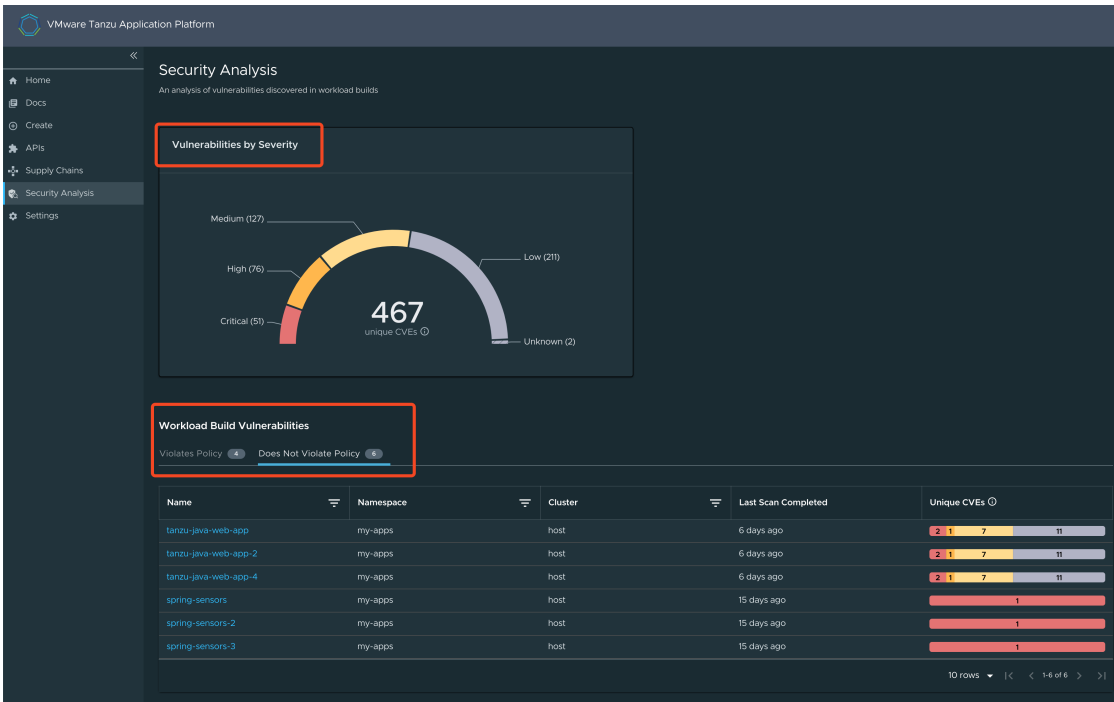
### Accessing the plug-in

The Security Analysis plug-in is always accessible from the left navigation pane. Click the **Security Analysis** button to open the **Security Analysis** dashboard.



## Viewing vulnerability data

The **Security Analysis** dashboard provides a summary of all vulnerabilities across all clusters for single-cluster and multicluster deployments.



The **Vulnerabilities by Severity** widget quickly counts the number of critical, high, medium, low, and unknown severity CVEs, based on the CVSS severity rating of each CVE.

It includes a sum of all workloads' source and image scan vulnerabilities. For example, if CVE-123 exists in the latest source scans and image scans of Workload ABC and Workload DEF, it is counted four times.

**Note**

The sum includes any CVEs on the allowlist (ignoreCVEs).

The **Workload Build Vulnerabilities** tables, with the **Violates Policy** tab and **Does Not Violate** tab, separate workloads based on the scan policy. For more information, see [Enforce compliance policy using Open Policy Agent](#) The Unique CVEs column uses the same sum logic as described earlier, but for individual workloads.

The sum of a workload's CVEs might not match the [Supply Chain Choreographer's Vulnerability Scan Results](#). The data on this dashboard is based on `kubectl describe` for `SourceScan` and `ImageScan`. The data on the Supply Chain Choreographer's Vulnerability Scan Results is based on Metadata Store data.

Only vulnerability scans associated with a Cartographer workload appear. Use [tanzu insight](#) to view results for non-workload scan results.

## Viewing CVE and package details

The Security Analysis plug-in has a **CVE** page and a **Package** page. These are accessed by clicking on a workload name, which opens the Supply Chain Choreographer plug-in. Clicking on the CVE or Package name opens the **CVE** or **Package** page, respectively.

## Workload Build Vulnerabilities

Violates Policy **4** Does Not Violate Policy **6**

Name	Namespaces
tanzu-java-web-app	my-apps
tanzu-java-web-app-2	my-apps
tanzu-java-web-app-4	my-apps

The **CVE** page contains basic information about the vulnerability and includes a table with all affected packages and versions.

The **Package** page contains basic information about a package and includes a table with all CVEs and the affected package versions.

## Supply Chain Choreographer in Tanzu Application Platform GUI

This topic tells you about Supply Chain Choreographer in Tanzu Application Platform GUI (commonly called TAP GUI).

### Overview

The Supply Chain Choreographer (SCC) plug-in enables you to visualize the execution of a workload by using any of the installed Out-of-the-Box supply chains. For more information about the Out-of-the-Box (OOTB) supply chains that are available in Tanzu Application Platform, see [Supply Chain Choreographer for Tanzu](#).

### Prerequisites

To use Supply Chain Choreographer in Tanzu Application Platform GUI you must have:

- One of the following installed on your cluster:
  - [Tanzu Application Platform Full profile](#)
  - [Tanzu Application Platform View profile](#)
  - [Tanzu Application Platform GUI package](#) and a metadata store package
- One of the following installed on the target cluster where you want to deploy your workload:
  - [Tanzu Application Platform Run profile](#)
  - [Tanzu Application Platform Full profile](#)

For more information, see [Overview of multicluster Tanzu Application Platform](#)

## Enable CVE scan results

To enable CVE scan results:

1. Obtain the [read-write token](#), which is created by default when installing Tanzu Application Platform. Alternatively, [create an additional read-write service account](#).
2. Add this proxy configuration to the `tap-gui:` section of `tap-values.yaml`:

```
tap_gui:
  app_config:
    proxy:
      /metadata-store:
        target: https://metadata-store-app.metadata-store:8443/api/v1
        changeOrigin: true
        secure: false
        headers:
          Authorization: "Bearer ACCESS-TOKEN"
          X-Custom-Source: project-star
```

Where `ACCESS-TOKEN` is the token you obtained after creating a read-write service account.



### Important

The `Authorization` value must start with the word `Bearer`.

## Enable GitOps Pull Request Flow

To enable the supply chain box-and-line diagram to show **Approve a Request** in the **Config Writer** stage, set up for GitOps and pull requests. For more information, see [GitOps vs. RegistryOps](#).

## Supply Chain Visibility

Before using the Supply Chain Visibility (SCV) plug-in to visualize a workload, you must create a workload.

The workload must have the `app.kubernetes.io/part-of` label specified, whether you manually create the workload or use one supplied with the OOTB supply chains.

Use the left sidebar navigation to access your workload and visualize it in the supply chain that is installed on your cluster.

The example workload described in this topic is named `tanzu-java-web-app`.

WORKLOADS			
Name	Namespace	Owner	Supply Chain
<a href="#">spring-petclinic</a>	dev	default-team	source-to-url
<a href="#">tanzu-java-web-app</a>	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

Supply Chain: tanzu-java-web-app

Errors: 0 Warnings: 0

te-provider → source-tester → source-scanner → image-builder → image-scanner → config-provider → app-config → config-write

Stage Detail: image-builder

20 hours ago

Overview

Blob URL

<http://source-controller.flux-system.svc.cluster.local/gitrepository/dev/tanzu-java-web-app/13dc15254d5b2ecc9b21243d2532b7bc3fba195d.tar.gz>

SubPath

Build Tool ClusterBuilder/default

Latest Build

ID	3
Age	20 hours ago
Reason	CONFIG
Docker Pull	Copy Image Tag

Latest Builds

ID	Age	Reason	Docker Pull
3	20 hours ago	CONFIG	Copy Image Tag
2	20 hours ago	CONFIG	Copy Image Tag
1	21 hours ago	CONFIG	Copy Image Tag

5 rows | 1-3 of 3

When a workload is deployed to a cluster that has the `deliverable` package installed, a new section appears in the supply chain that shows **Pull Config** boxes and **Delivery** boxes.

When you have a **Pull Request** configured in your environment, access the merge request from the supply chain by clicking **APPROVE A REQUEST**. This button is displayed after you click **Config Writer** in the supply chain diagram.

tanzu-java-web-app-pr-flow

Supply Chain: source-test-scan-to-url

Supply Chain Cluster: gke\_dap\_ops\_us-west2\_interfaces-test

Delivery Cluster: gke\_dap\_ops\_us-west2\_interfaces-test

Errors: 0 Warnings: 0

Service Bindings → Api Descriptors → Config Writer → Pull Config → Delivery

Stage Detail: Config Writer

11 days ago

Overview

Name tanzu-java-web-app-pr-flow-config-writer-pull-requester

Kind Runnable

Version carto.run/v1alpha1

Namespace my-apps

Action Needed

APPROVE A REQUEST

## View Vulnerability Scan Results

Click the **Source Scan** stage or **Image Scan** stage to view vulnerability source scans and image scans for workload builds. The data is from [Supply Chain Security Tools - Store](#).

CVE issues represent any vulnerabilities associated with a package or version found in the source code or image, including vulnerabilities from past scans.



**Note**

For example, the `log4shell` package is found in image ABC on 1 January without any CVEs. On 15 January, the `log4j` CVE issue is found while scanning image DEF. If a user returns to the **Image Scan** stage for image ABC, the `log4j` CVE issue appears and is associated with the `log4shell` package.

## Overview of enabling TLS for Tanzu Application Platform GUI

Many users want inbound traffic to Tanzu Application Platform GUI (commonly called TAP GUI) to be properly encrypted. These topics tell you how to enable TLS encryption either with an existing certificate or by using the included cert-manager instance.

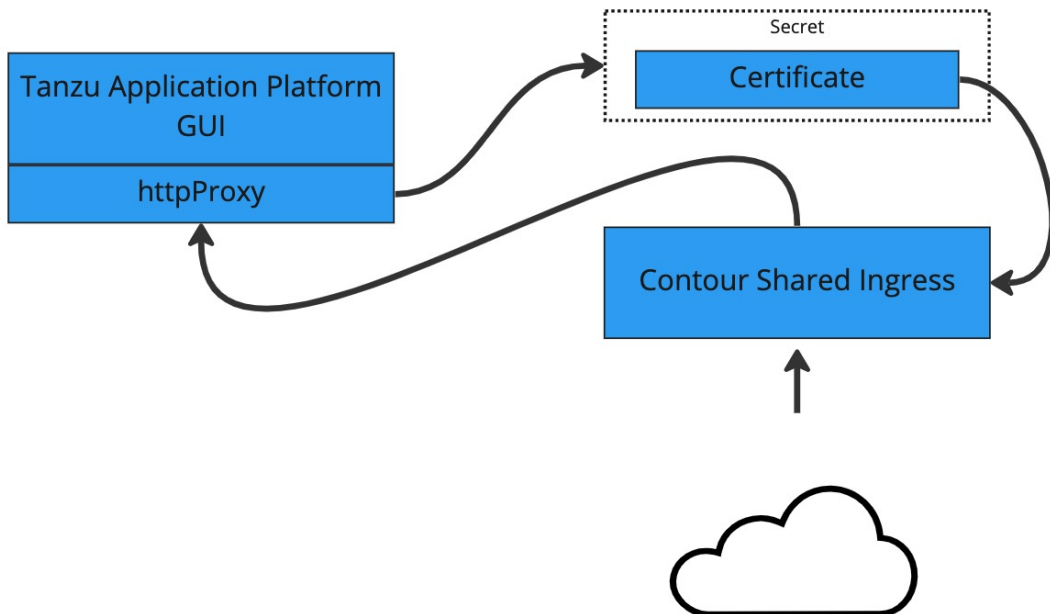
### Concepts

The two key concepts are certificate delegation and the relationship between cert-manager, certificates, and ClusterIssuers.

### Certificate delegation

Tanzu Application Platform GUI uses the established shared Contour ingress for TLS termination.

This enables you to store the certificate in a Kubernetes `secret` and then pass that `secret` and `namespace` to the `httpProxy` that was created during installation. To do this, see [Configuring a TLS certificate by using an existing certificate](#).



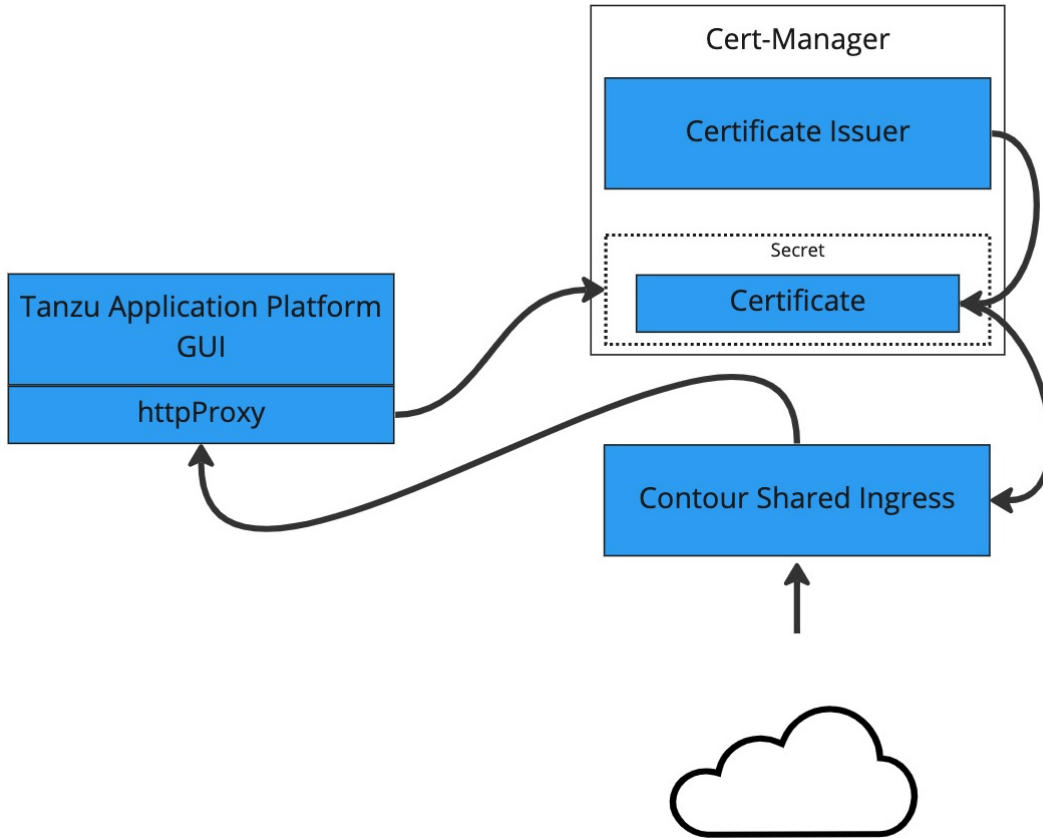
### cert-manager, certificates, and ClusterIssuers



Tanzu Application Platform GUI can also use the `cert-manager` package that is installed when the profile was installed.

This tool allows cert-manager to automatically acquire a certificate from a `clusterIssuer` entity.

This external entity can be an external certificate authority, such as Let's Encrypt, or a self-signed certificate.



## Guides

The following topics describe different ways to configure TLS:

- [Configuring a TLS certificate by using an existing certificate](#)
- [Configuring a TLS certificate by using a self-signed certificate](#)
- [Configuring a TLS certificate by using cert-manager and a ClusterIssuer](#)

## Overview of enabling TLS for Tanzu Application Platform GUI

Many users want inbound traffic to Tanzu Application Platform GUI (commonly called TAP GUI) to be properly encrypted. These topics tell you how to enable TLS encryption either with an existing certificate or by using the included cert-manager instance.

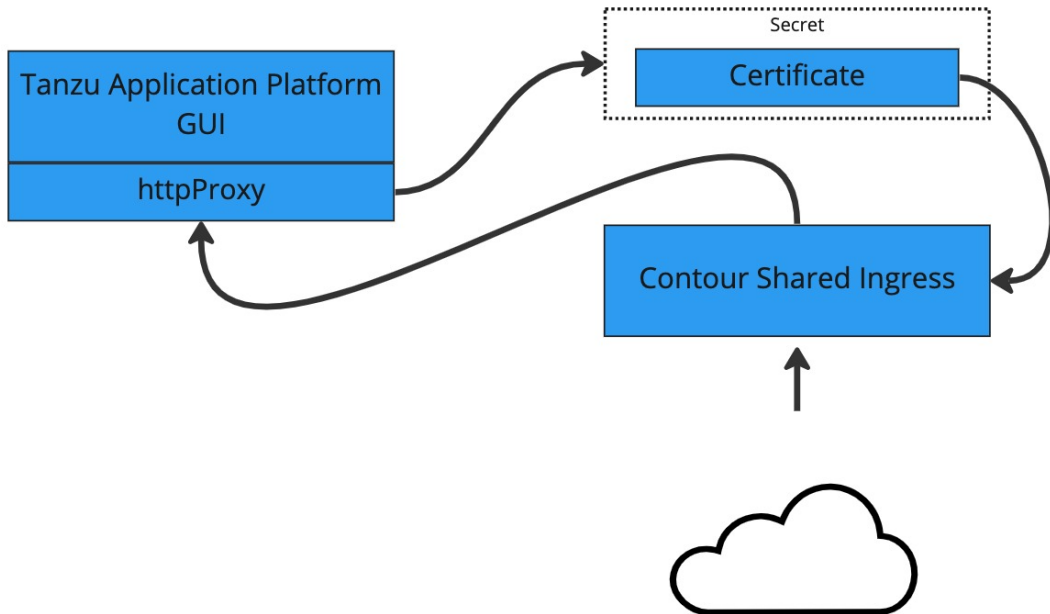
## Concepts

The two key concepts are certificate delegation and the relationship between cert-manager, certificates, and ClusterIssuers.

## Certificate delegation

Tanzu Application Platform GUI uses the established shared Contour ingress for TLS termination.

This enables you to store the certificate in a Kubernetes `secret` and then pass that `secret` and `namespace` to the `httpProxy` that was created during installation. To do this, see [Configuring a TLS certificate by using an existing certificate](#).

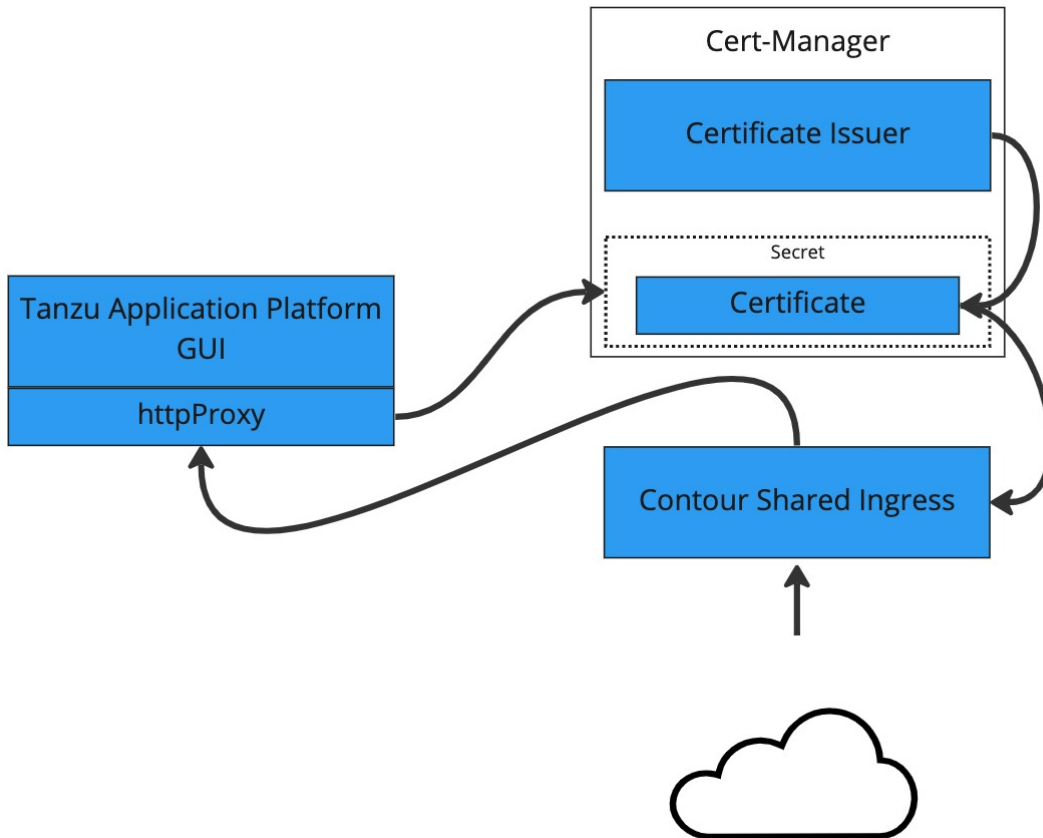


## cert-manager, certificates, and ClusterIssuers

Tanzu Application Platform GUI can also use the `cert-manager` package that is installed when the profile was installed.

This tool allows `cert-manager` to automatically acquire a certificate from a `clusterIssuer` entity.

This external entity can be an external certificate authority, such as Let's Encrypt, or a self-signed certificate.



## Guides

The following topics describe different ways to configure TLS:

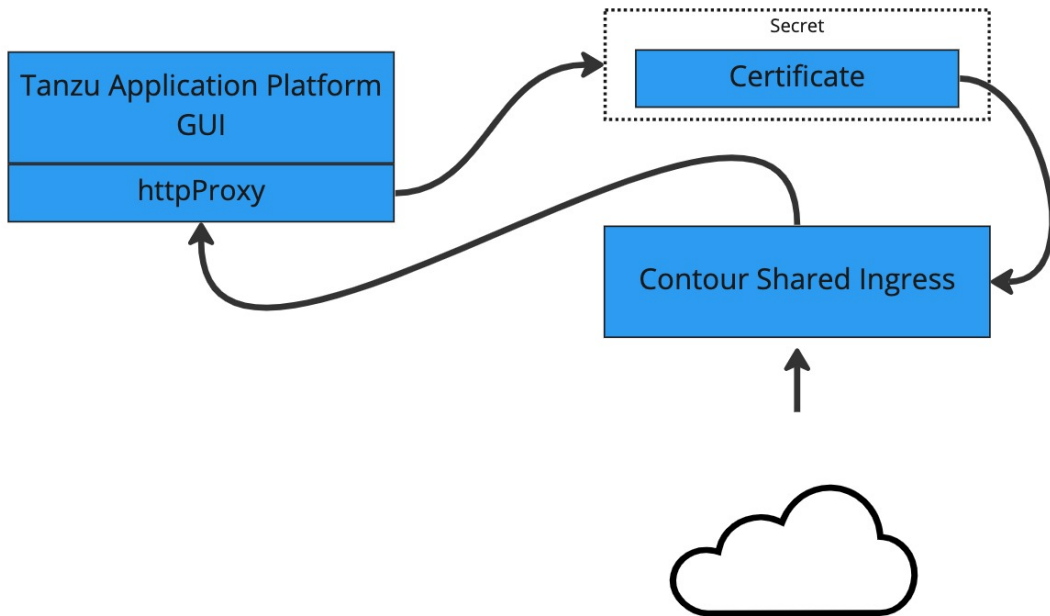
- [Configuring a TLS certificate by using an existing certificate](#)
- [Configuring a TLS certificate by using a self-signed certificate](#)
- [Configuring a TLS certificate by using cert-manager and a ClusterIssuer](#)

## Configure a TLS certificate by using an existing certificate

This topic tells you how to use the certificate information from your external certificate authority to encrypt inbound traffic to Tanzu Application Platform GUI (commonly called TAP GUI).

## Prerequisites

Your certificate authority gave you a certificate file, of the form `CERTIFICATE-FILE-NAME.crt`, and a signing key, of the form `KEY-FILE-NAME.key`. Ensure that these files are present on the host from which you run the CLI commands.



## Procedure

To configure Tanzu Application Platform GUI with an existing certificate:

1. Create the Kubernetes secret by running:

```
kubectl create secret tls tap-gui-cert --key="KEY-FILE-NAME.key" --cert="CERTIFICATE-FILE-NAME.crt" -n tap-gui
```

Where:

- `KEY-FILE-NAME` is the name of the `key` file that your certificate issuer gave you
  - `CERTIFICATE-FILE-NAME` is the name of the `cert` file that your certificate issuer gave you
2. Configure Tanzu Application Platform GUI to use the newly created secret. Do so by editing the `tap-values.yaml` file that you used during installation to include the following under the `tap-gui` section:
    - A top-level `tls` key with subkeys for `namespace` and `secretName`
    - A namespace referring to the namespace used earlier
    - A secret name referring to the `secretName` value defined earlier

Example:

```
tap_gui:
  tls:
    namespace: tap-gui
    secretName: tap-gui-cert
# Additional configuration below this line as needed
```

3. Update the Tanzu Application Platform package with the new values in `tap-values.yaml` by running:

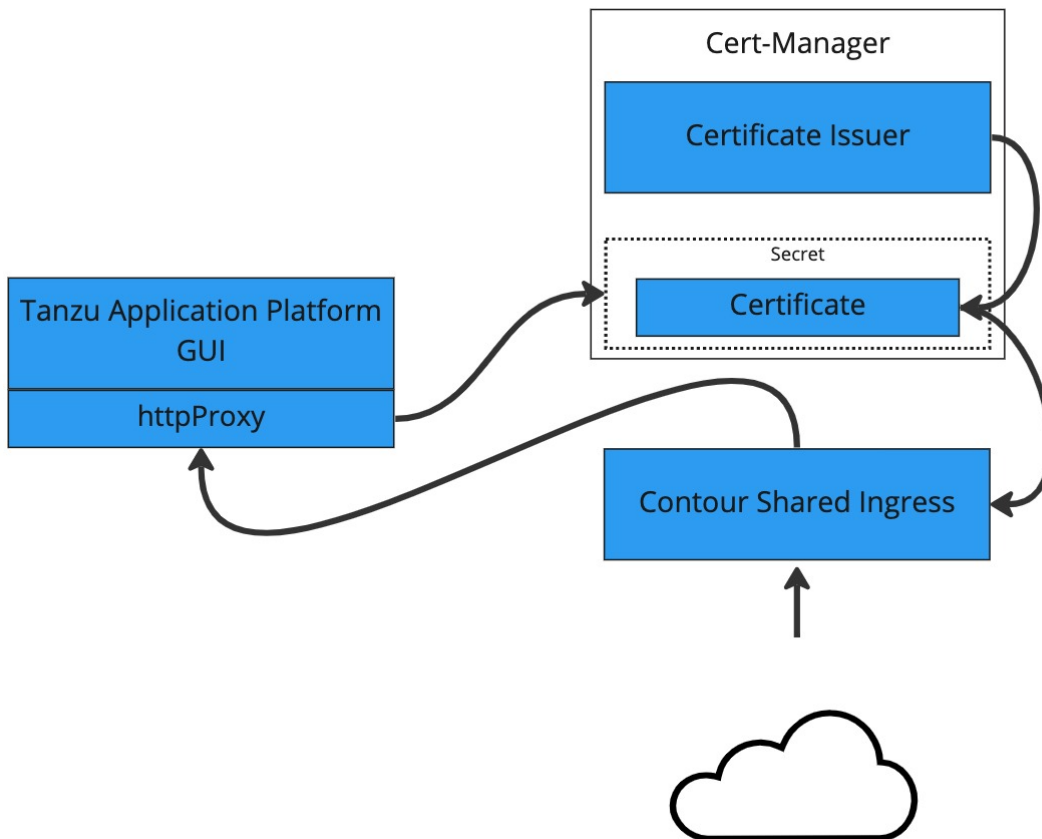
```
tanzu package installed update tap -p tap.tanzu.vmware.com -v TAP-VERSION --values-file tap-values.yaml -n tap-install
```

Where `TAP-VERSION` is the version number that matches the values you used when you installed your profile.

## Configure a TLS certificate by using a self-signed certificate

This topic tells you how to use cert-manager to create a self-signed certificate issuer and then generate a certificate for Tanzu Application Platform GUI to use based on that issuer.

Some browsers and corporate policies do not allow you to visit webpages that have self-signed certificates. You might need to navigate through a series of error messages to visit the page.



## Prerequisite

Install a Tanzu Application Platform profile that includes cert-manager. Verify you did this by running the following command to detect the cert-manager namespace:

```
kubectl get ns
```

## Procedure

To configure a self-signed TLS certificate for Tanzu Application Platform GUI:

1. Create a `certificate.yaml` file that defines an issuer and a certificate. For example:

```
apiVersion: cert-manager.io/v1
kind: Issuer
```

```

metadata:
  name: ca-issuer
  namespace: tap-gui
spec:
  selfSigned: {}
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: tap-gui-cert
  namespace: tap-gui
spec:
  secretName: tap-gui-cert
  dnsNames:
  - tap-gui.INGRESS-DOMAIN
  issuerRef:
    name: ca-issuer

```

Where `INGRESS-DOMAIN` is your domain value that matches the values you used when you installed the profile.

2. Add the issuer and certificate to your cluster by running:

```
kubectl apply -f certificate.yaml
```

3. Configure Tanzu Application Platform GUI to use the newly created certificate. Update the `tap-values.yaml` file used during installation to include the following under the `tap-gui` section:

- o A top-level `tls` key with subkeys for `namespace` and `secretName`
- o A namespace referring to the namespace containing the `Certificate` object mentioned earlier
- o A secret name referring to the `secretName` value defined in your `Certificate` resource earlier

Example:

```

tap_gui:
  tls:
    namespace: tap-gui
    secretName: tap-gui-cert
# Additional configuration below this line as needed

```

4. Update the Tanzu Application Platform package with the new values in `tap-values.yaml`:

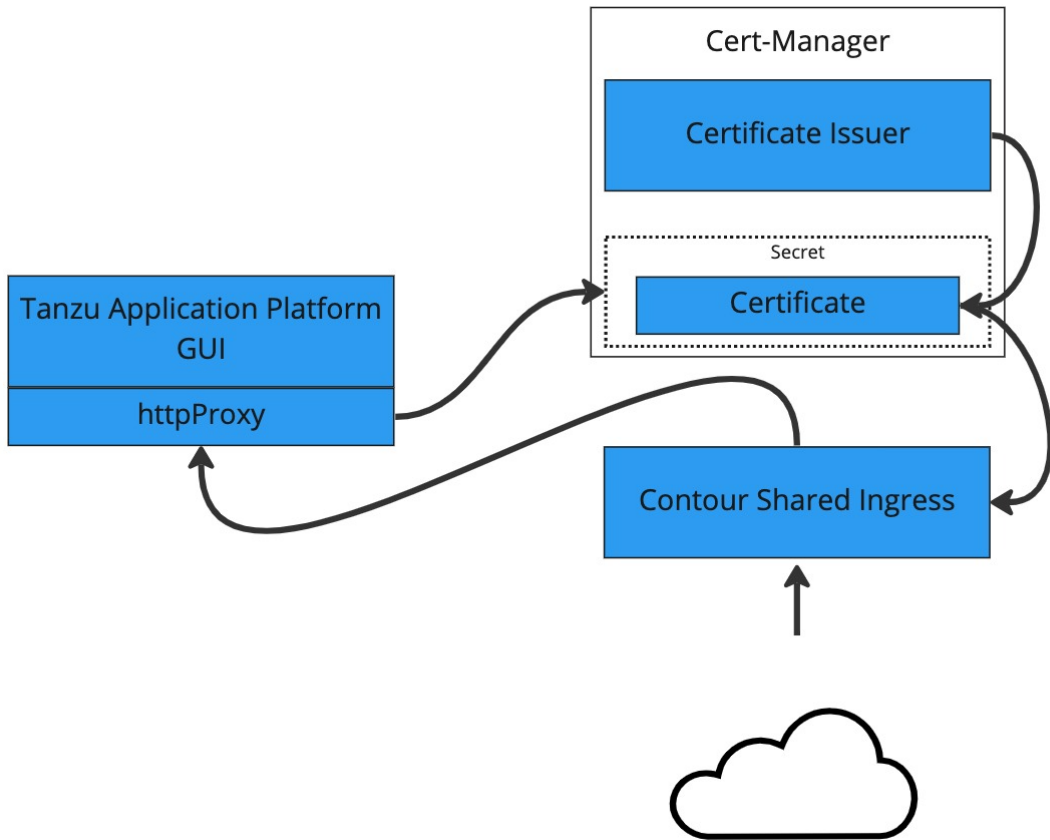
```
tanzu package installed update tap -p tap.tanzu.vmware.com -v TAP-VERSION --values-file tap-values.yaml -n tap-install
```

Where `TAP-VERSION` is the version that matches the values you used when you installed the profile.

## Configure a TLS certificate by using cert-manager and a ClusterIssuer

This topic tells you how to use cert-manager to create a certificate issuer and then generate a certificate for Tanzu Application Platform GUI (commonly called TAP GUI) to use based on that issuer.

This topic uses the free certificate issuer [Let's Encrypt](#). You can use other certificate issuers compatible with cert-manager in a similar fashion.



## Prerequisites

Fulfil these prerequisites:

- Install a Tanzu Application Platform profile that includes cert-manager. Verify you did this by running the following command to detect the cert-manager namespace:

```
kubectl get ns
```

- Obtain a domain name that you control or own and have proof that you control or own it. In most cases, this domain name is the one you used for the `INGRESS-DOMAIN` values when you installed Tanzu Application Platform and Tanzu Application Platform GUI.
- If cert-manager cannot perform the challenge to verify your domain's compatibility, you must do so manually. For more information, see [How It Works](#) and [Getting Started](#) in the Let's Encrypt documentation.
- Ensure that your domain name is pointed at the shared Contour ingress for the installation. Find the IP address by running:

```
kubectl -n tanzu-system-ingress get services envoy -o jsonpath='{.status.loadBalancer.ingress[0].ip}'
```

## Procedure

To configure a self-signed TLS certificate for Tanzu Application Platform GUI:

1. Create a `certificate.yaml` file that defines an issuer and a certificate. For example:

```

apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-http01-issuer
  namespace: cert-manager
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    email: EMAIL-ADDRESS
    privateKeySecretRef:
      name: letsencrypt-http01-issuer
    solvers:
      - http01:
          ingress:
            class: contour
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  namespace: cert-manager
  name: tap-gui
spec:
  commonName: tap-gui.INGRESS-DOMAIN
  dnsNames:
    - tap-gui.INGRESS-DOMAIN
  issuerRef:
    name: letsencrypt-http01-issuer
    kind: ClusterIssuer
  secretName: tap-gui

```

Where:

- `EMAIL-ADDRESS` is the email address that Let's Encrypt shows as responsible for this certificate
- `INGRESS-DOMAIN` is your domain value that matches the values you used when you installed the profile

2. Add the issuer and certificate to your cluster by running:

```
kubectl apply -f certificate.yaml
```

By applying the certificate, cert-manager attempts to perform an HTTP01 challenge by creating an Ingress resource specifically for the challenge. This is automatically removed from your cluster after the challenge is completed. For more information about how this works, and when it might not, see the [cert-manager documentation](#).

3. Validate the certificate was created and is ready by running:

```
kubectl get certs -n cert-manager
```

Wait a few moments for this to take place, if need be.

4. Configure Tanzu Application Platform GUI to use the newly created certificate. To do so, update the `tap-values.yaml` file that you used during installation to include the following items under the `tap-gui` section:
  - A top-level `tls` key with subkeys for `namespace` and `secretName`
  - A namespace referring to the namespace containing the `Certificate` object from earlier
  - A secret name referring to the `secretName` value defined in your `Certificate` resource earlier



Example:

```
tap_gui:
  tls:
    namespace: cert-manager
    secretName: tap-gui
# Additional configuration below this line as needed
```

5. Update the Tanzu Application Platform package with the new values in `tap-values.yaml` by running:

```
tanzu package installed update tap -p tap.tanzu.vmware.com -v TAP-VERSION --values-file tap-values.yaml -n tap-install
```

Where `TAP-VERSION` is the version that matches the values you used when you installed the profile.

## Upgrade Tanzu Application Platform GUI

This topic tells you how to upgrade Tanzu Application Platform GUI (commonly called TAP GUI) outside of a Tanzu Application Platform profile installation. If you installed Tanzu Application Platform through a profile, see [Upgrading Tanzu Application Platform](#) instead.

### Considerations

As part of the upgrade, Tanzu Application Platform updates its container with the new version.

As a result, if you installed Tanzu Application Platform GUI without the support of a backing [database](#), you lose your in-memory data for any manual component registrations when the container restarts. While the update is pulling the new pod from the registry, users might experience a short UI interruption and might need to re-authenticate because the in-memory session data is rebuilt.

### Upgrade within a Tanzu Application Platform profile

If you installed Tanzu Application Platform GUI as part of a Tanzu Application Platform profile, see [Upgrading Tanzu Application Platform](#).

### Upgrade Tanzu Application Platform GUI individually

These steps only apply to installing Tanzu Application Platform GUI individually, not as part of a Tanzu Application Platform profile.

To upgrade Tanzu Application Platform GUI outside of a Tanzu Application Platform profile:

1. Ensure that your repository has access to the new version of the package by running:

```
tanzu package available list tap-gui.tanzu.vmware.com -n tap-install
```

For example:

```
$ tanzu package available list tap-gui.tanzu.vmware.com -n tap-install
- Retrieving package versions for tap-gui.tanzu.vmware.com...
NAME                                VERSION  RELEASED-AT
tap-gui.tanzu.vmware.com            1.0.1    2021-12-22 17:45:51 +0000 UTC
tap-gui.tanzu.vmware.com            1.0.2    2022-01-25 01:57:19 +0000 UTC
```

2. Perform the package upgrade by using the targeted package update version. Run:

```
tanzu package installed update tap-gui -p tap-gui.tanzu.vmware.com -v VERSION
--values-file \
TAP-GUI-VALUES.yaml -n tap-install
```

Where:

- `VERSION` is the target version of Tanzu Application Platform GUI that you want.
- `TAP-GUI-VALUES` is the configuration values file that contains the configuration used when you installed Tanzu Application Platform GUI.

3. Verify that you upgraded your application by running:

```
tanzu package installed get tap-gui -n tap-install
```

## Troubleshoot Tanzu Application Platform GUI

This topic tells you how to troubleshoot issues encountered when installing Tanzu Application Platform GUI (commonly called TAP GUI). The topic is divided into sections:

- [General issues](#)
- [Runtime Resources tab issues](#)
- [Accelerators page issues](#)
- [Security Analysis plug-in issues](#)
- [Supply Chain Choreographer plug-in issues](#)

### General issues

The following are general issues:

#### Tanzu Developer Portal reports that the port range is not valid

##### Symptom

You provided a full URL in a `backend.reading.allow` entry, as in this example `tap-values.yaml` snippet:

```
tap_gui:
  app_config:
    backend:
      reading:
        allow:
          - host: http://gitlab.example.com/some-group/some-repo/-/blob/main/catalog-info.yaml
```

and you see the following error message:

```
Backend failed to start up, Error: Port range is not valid: //gitlab.example.com/some-group/some-repo/-/blob/main/catalog-info.yaml
```

##### Cause

Tanzu Application Platform GUI expects a host name to be passed into the field `backend.reading.allow[].host`.

##### Solution

Edit your `tap-values.yaml` file as in the following example:

```
tap_gui:
  app_config:
    backend:
      reading:
        allow:
          - host: gitlab.example.com
            paths: ['/some-group/some-repo/']
```

## Tanzu Application Platform GUI does not load the catalog

### Symptom

You are able to visit Tanzu Application Platform GUI, but it does not load the catalog and you see the following error message.

```
> Error: Could not fetch catalog entities.
> TypeError: Failed to fetch
```

When viewing your network tab you see that your browser has not downloaded mixed content. This might look different on different browsers.

**Chrome**  
In the **Status** column you see **(blocked:mixed-content)**

Name	Status	Type	Initiator	Size	Time
6329-c4812cf.chunk.js	200	script	load script:41	799 B	457 ms
entity-facets?facet=kind	(blocked:mixed-content)	fetch	index.esm.js:1420	0 B	0 ms
entity-facets?facet=metadata.tags	(blocked:mixed-content)	fetch	index.esm.js:1420	0 B	0 ms

**Firefox**  
In the **Transferred** column you see **Mixed Block**

Status	Method	Domain	File	Initiator	Type	Transferred
✓	GET	tap-gui.34.173.223.35.ng.ik	entity-filter/index.component	entity-filter/index.component	fetch	Mixed Block
✓	GET	tap-gui.34.173.223.35.ng.ik	entity-facets?facet=metadata.tags	fetch	fetch	Mixed Block
✓	GET	tap-gui.34.173.223.35.ng.ik	entity-filter/index.component	fetch	fetch	Mixed Block

### Cause

As of Tanzu Application Platform v1.4, Tanzu Application Platform GUI provides TLS connections by default. Because of this, if you visit a Tanzu Application Platform GUI site your connection is automatically upgraded to https.

You might have manually set the fields `app.baseUrl`, `backend.baseUrl`, and `backend.cors.origin` in your `tap-values.yaml` file. Tanzu Application Platform GUI uses the `baseUrl` to determine how to create links to fetch from its APIs. The combination of these two factors causes your browser to attempt to fetch mixed content.

### Solution

The solution is to delete these fields or update your values in `tap-values.yaml` to reflect that your Tanzu Application Platform GUI instance is serving https, as in the following example:

```
tap_gui:
  app_config:
    app:
      baseUrl: https://tap-gui.INGRESS-DOMAIN/
    backend:
      baseUrl: https://tap-gui.INGRESS-DOMAIN/
      cors:
        origin: https://tap-gui.INGRESS-DOMAIN/
```

Where `INGRESS-DOMAIN` is the ingress domain you have configured for Tanzu Application Platform.

The installer determines acceptable values based on your `tap_gui.ingressDomain` or `shared.ingress_domain` and the TLS status of the installation.

## Updating a supply chain causes an error (Can not create edge...)

### Symptom

Updating a supply chain causes an error (Can not create edge...) when an existing workload is clicked in the Workloads table and that supply chain is no longer present.

### Solution

Recreate the same workload to execute through the new or updated supply chain.

## Catalog not found

### Symptom

When you pull up Tanzu 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:
  - o Change the Tanzu Application Platform profile file if installed by using a profile.
  - o 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 issues

Here are some common troubleshooting steps for errors presented in the **Runtime Resources** tab.

### Error communicating with Tanzu Application Platform web server

#### Symptom

When accessing the **Runtime Resource Visibility** tab, the system displays `Error communicating with TAP GUI back end.`

#### Causes

- An interrupted Internet connection
- Error with the back end service

#### Solution

1. Confirm that you have Internet access.
2. Confirm that the back-end service is running correctly.
3. Confirm the cluster configuration is correct.

### No data available

#### Symptom

When accessing the **Runtime Resource Visibility** tab, the system displays

```
One or more resources are missing. This could be due to a label mismatch. \
Please make sure your resources have the label(s) "LABEL_SELECTOR".
```

#### Cause

No communications error has occurred, but no resources were found.

#### Solution

Confirm that you are using the correct label:

1. Verify the [Component definition](#) includes the annotation `backstage.io/kubernetes-label-selector`.
2. Confirm your Kubernetes resources correspond to that label drop-down menu.

### Errors retrieving resources

## Symptom

When opening the **Runtime Resource Visibility** tab, the system displays `One or more resources might be missing because of cluster query errors.`

The reported errors might not indicate a real problem. A build cluster might not have runtime CRDs installed, such as Knative Service, and a run cluster might not have build CRDs installed, such as a Cartographer workload. In these cases, 403 and 404 errors might be false positives.

You might receive the following error messages:

- `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 issues

Here are some common troubleshooting steps for errors displayed on the **Accelerators** page.

### No accelerators

#### Symptom

When the `app_config.backend.reading.allow` section is configured in the `tap-values.yaml` file during the `tap-gui` package installation, there are no accelerators on the **Accelerators** page.

#### Cause

This section in `tap-values.yaml` overrides the default configuration that gives Tanzu 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
```

## Security Analysis plug-in issues

Here are troubleshooting steps for errors affecting the **Security Analysis** page.

### Empty dashboard after upgrading from Tanzu Application Platform v1.3

#### Symptom

After upgrading to Tanzu Application Platform v1.4 from v1.3, the Security Analysis GUI dashboard appears empty.

#### Cause

From Tanzu Application Platform v1.4, the dashboard displays information from the Metadata Store. Previously, the Security Analysis GUI dashboard polled the Kubernetes clusters for information.

#### Solution

Repopulate the dashboard by running new source scans and image scans. To do so, do one of the following actions:

- Trigger a workload to run with a new commit to the source code
- Delete the relevant SourceScan or ImageScan on the Kubernetes cluster

## Supply Chain Choreographer plug-in issues

These are troubleshooting steps for the [Supply Chain Choreographer plug-in](#).

### An error occurred while loading data from the Metadata Store

#### Symptom

In the Supply Chain Choreographer plug-in, you see the error message `An error occurred while loading data from the Metadata Store`.



tanzu-java-web-app-scan2

Supply Chain: source-test-scan-to-url Errors: 1

Supply Chain Cluster: tkg-build-airgap-fuji

Delivery Cluster: tkg-run-airgap-fuji-config

**Stage Detail: Image Scanner**  
2 hours ago

<b>Overview</b>		<b>Policy</b>	
Registry	harbor-airgap.dapdaws.net	Name	scan-policy
Image	harbor-airgap.dapdaws.net/tap/workloads/tanzu-java-web-app-scan2-my-apps	UID	07e2e602-3c2b-4ad5-a0b4-e7a13ebb2158
Digest	sha256:81c064e7bb23d30ff66840c0c64f74a45de5c1ef58d219ee6d2f17a6ecc61ed	Generation	1
Scan Template	tanzu-java-web-app-scan2	Details	No Violations Found
UID	35ba48d6-1d12-419b-84e1-217be6bfff296		
Generation	1		

ⓘ An error occurred while loading data from the Metadata Store.

CVE ID	Severity	Package	Version	Description
(Empty table body)				

### Cause

There are multiple potential causes. The most common cause is `tap-values.yaml` missing the configuration that enables Tanzu Application Platform GUI to communicate with Supply Chain Security Tools - Store.

### Solution

See [Supply Chain Choreographer - Enable CVE scan results](#) for the necessary configuration to add to `tap-values.yaml`. After adding the configuration, update your Tanzu Application Platform deployment or Tanzu Application Platform GUI deployment with the new values.

## Overview of Tanzu Application Platform Telemetry

Tanzu Application Platform Telemetry (commonly known as TAP Telemetry) is a set of objects that collect data about the usage of Tanzu Application Platform (commonly known as TAP) and send it back to VMware for product improvements.

A benefit of remaining enrolled in telemetry and identifying your company during Tanzu Application Platform installation is that VMware can provide your organization with usage reports about Tanzu Application Platform.

For more information about enrolling in telemetry reports, see [Tanzu Application Platform usage reports](#).

For more information about how to install the telemetry component, see [Install Tanzu Application Platform Telemetry](#).

## Tanzu Application Platform usage reports

VMware offers the option to enroll in a usage reporting program that offers a summary of usage of your Tanzu Application Platform. You can enroll in the program by providing the Entitlement Account Number (EAN). An EAN is a unique ID assigned to all VMware customers. VMware uses EAN to identify data about Tanzu Application Platform. See [Locate the Entitlement Account number for new orders](#) for more details.

After locating the EAN, pass the number under the telemetry header in the `tap-values.yaml` file as a value for the `customer_entitlement_account_number` key.

```
tap_telemetry:
  customer_entitlement_account_number: "CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER"
```

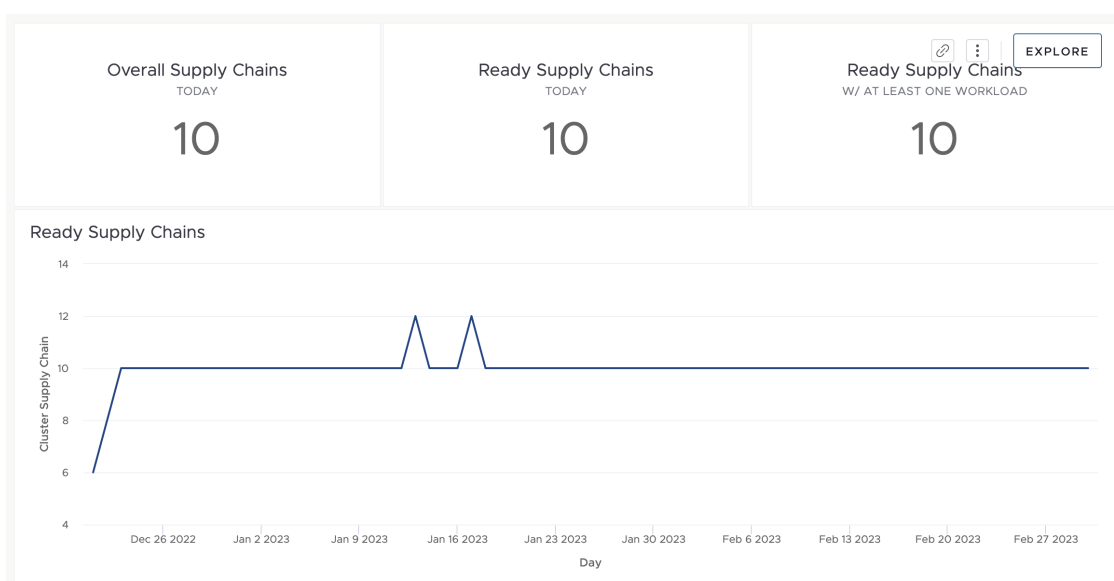
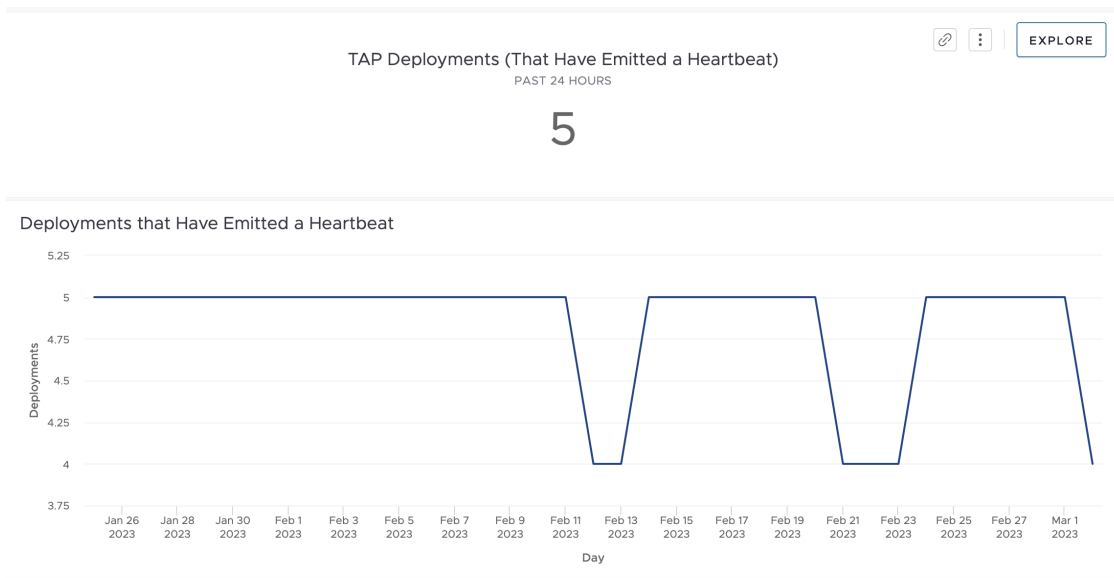
You must repeat the process for each Tanzu Application Platform Cluster included in the telemetry report. For more information, see [Full profile](#).

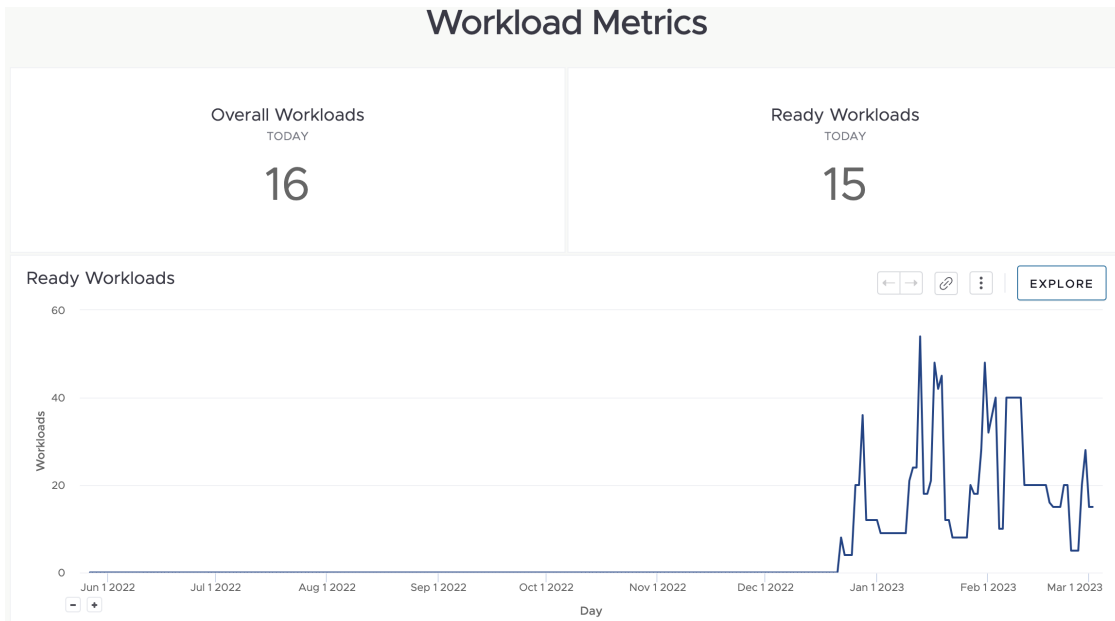
After enrollment, alert your VMware account team that you have configured the EAN field and want telemetry reports. This allows VMware to identify who the newly added EAN belongs to.

**Note**

Usage report is only supported for non-airgapped deployments of Tanzu Application Platform and the Cluster must participate in Tanzu Application Platform telemetry. You are enrolled in telemetry by default. You can opt out of telemetry collection by following the instructions in [Opt out of telemetry collection](#).

The following screenshots show the sample telemetry reports.





## Overview of Tanzu Application Platform Telemetry

Tanzu Application Platform Telemetry (commonly known as TAP Telemetry) is a set of objects that collect data about the usage of Tanzu Application Platform (commonly known as TAP) and send it back to VMware for product improvements.

A benefit of remaining enrolled in telemetry and identifying your company during Tanzu Application Platform installation is that VMware can provide your organization with usage reports about Tanzu Application Platform.

For more information about enrolling in telemetry reports, see [Tanzu Application Platform usage reports](#).

For more information about how to install the telemetry component, see [Install Tanzu Application Platform Telemetry](#).

## Tanzu Application Platform usage reports

VMware offers the option to enroll in a usage reporting program that offers a summary of usage of your Tanzu Application Platform. You can enroll in the program by providing the Entitlement Account Number (EAN). An EAN is a unique ID assigned to all VMware customers. VMware uses EAN to identify data about Tanzu Application Platform. See [Locate the Entitlement Account number for new orders](#) for more details.

After locating the EAN, pass the number under the telemetry header in the `tap-values.yaml` file as a value for the `customer_entitlement_account_number` key.

```
tap_telemetry:
  customer_entitlement_account_number: "CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER"
```

You must repeat the process for each Tanzu Application Platform Cluster included in the telemetry report. For more information, see [Full profile](#).

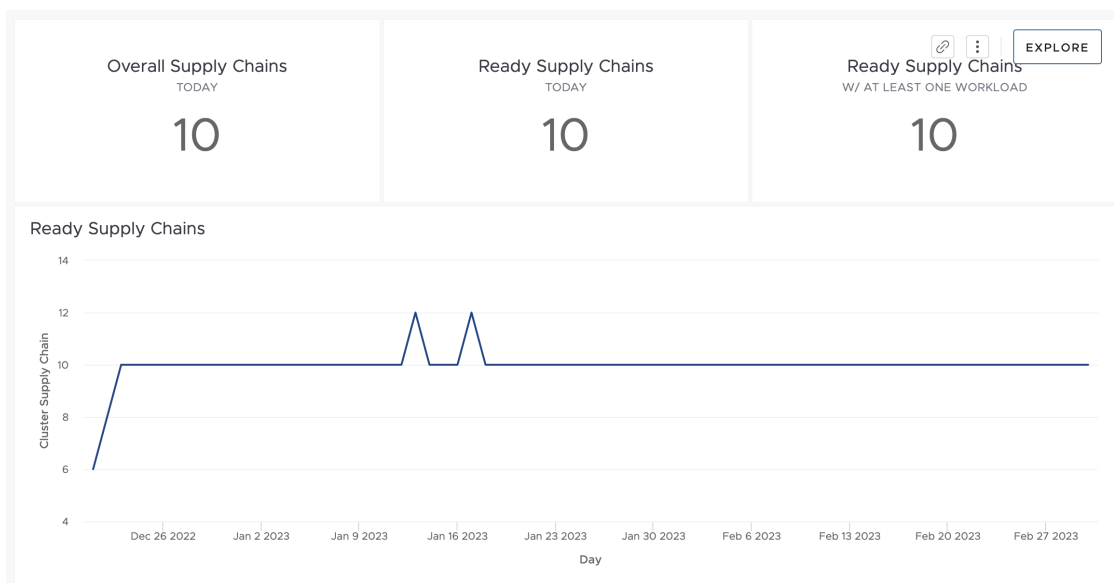
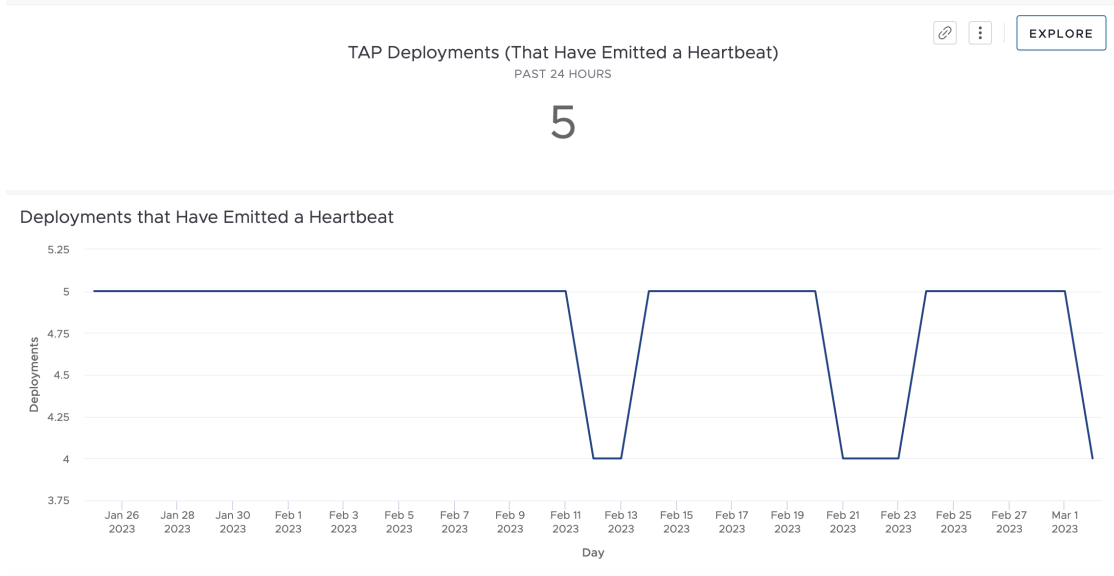
After enrollment, alert your VMware account team that you have configured the EAN field and want telemetry reports. This allows VMware to identify who the newly added EAN belongs to.

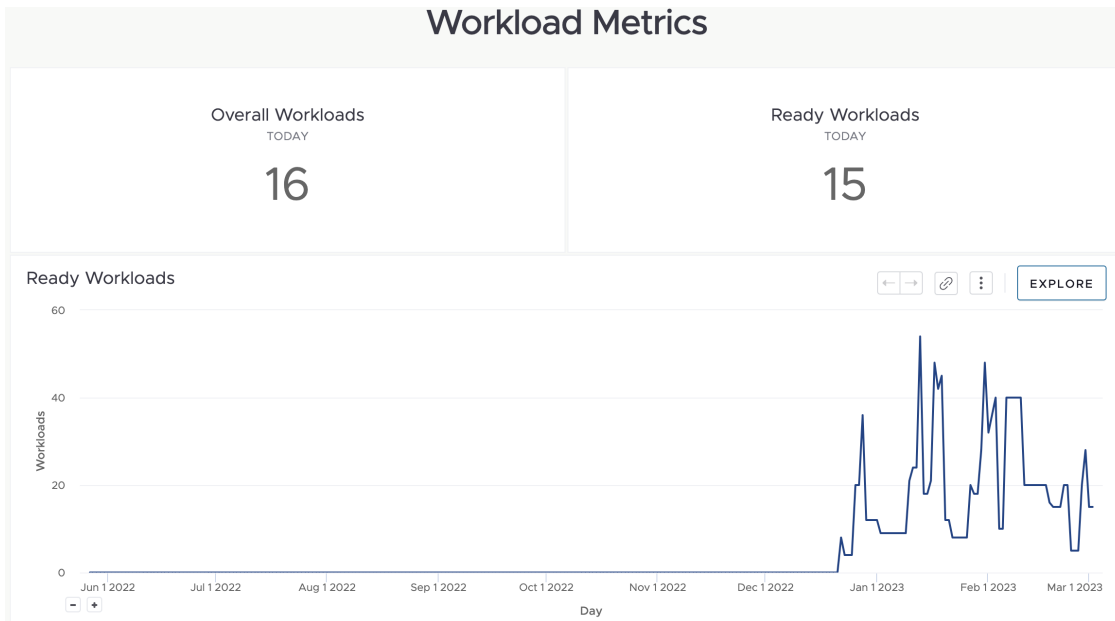


### Note

Usage report is only supported for non-airgapped deployments of Tanzu Application Platform and the Cluster must participate in Tanzu Application Platform telemetry. You are enrolled in telemetry by default. You can opt out of telemetry collection by following the instructions in [Opt out of telemetry collection](#).

The following screenshots show the sample telemetry reports.





## Install Tanzu Application Platform Telemetry

This topic tells you how to install Tanzu Application Platform Telemetry from the Tanzu Application Platform (commonly known as TAP) package repository.



### Note

Follow the steps in this topic if you do not want to use a profile to install Telemetry. For more information about profiles, see [Components and installation profiles](#).

## Prerequisites

Before installing Tap Telemetry:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see [Prerequisites](#).
- Install cert-manager on the cluster. For more information, see the [cert-manager documentation](#).
- See [Deployment Details and Configuration](#) to review what resources will be deployed.

## Install

To install Tanzu Application Platform Telemetry:

1. List version information for the package by running:

```
tanzu package available list tap-telemetry.tanzu.vmware.com --namespace tap-ins tall
```

For example:

```
$ tanzu package available list tap-telemetry.tanzu.vmware.com --namespace tap-install
- Retrieving package versions for tap-telemetry.tanzu.vmware.com...
```

NAME	VERSION	RELEASED-AT
tap-telemetry.tanzu.vmware.com	0.3.1	

- (Optional) List all the available deployment configuration options:

```
tanzu package available get tap-telemetry.tanzu.vmware.com/VERSION --values-schema -n tap-install
```

Where `VERSION` is the your package version number. For example, `0.3.1`.

For example:

```
$ tanzu package available get tap-telemetry.tanzu.vmware.com/0.3.1 --values-schema -n tap-install
| Retrieving package details for tap-telemetry.tanzu.vmware.com/0.3.1...
KEY                                DEFAULT  TYPE      DESCRIPTION
kubernetes_distribution            string   Kubernetes platform flavor where the tap-telemetry is being installed on. Accepted values are ['', 'open shift']
customer_entitlement_account_number string   Account number used to distinguish data by customer.
installed_for_vmware_internal_use  string   Indication of if the deployment is for vmware internal user. Accepted values are ['true', 'false']
```

- (Optional) Modify the deployment configurations by creating a configuration YAML with the desired custom configuration values. For example, if you want to provide your Customer Entitlement Number, create a `tap-telemetry-values.yaml` and configure the `customer_entitlement_account_number` property:

```
---
customer_entitlement_account_number: "12345"
```

See [Deployment details and configuration](#) for more information about the configuration options.

- Install the package by running:

```
tanzu package install tap-telmetry \
  --package-name tap-telemetry.tanzu.vmware.com \
  --version VERSION \
  --namespace tap-install \
  --values-file tap-telemetry-values.yaml
```

Where:

- `--values-file` is an optional flag. Only use it to customize the deployment configuration.
- `VERSION` is the package version number. For example, `0.3.1`.

For example:

```
$ tanzu package install tap-telmetry \
  --package-name tap-telemetry.tanzu.vmware.com \
  --version 0.3.1 \
  --namespace tap-install \
  --values-file tap-telemetry-values.yaml

Installing package 'tap-telemetry.tanzu.vmware.com'
Getting package metadata for 'tap-telemetry.tanzu.vmware.com'
Creating service account 'tap-telemetry-tap-install-sa'
Creating cluster admin role 'tap-telemetry-tap-install-cluster-role'
Creating cluster role binding 'tap-telemetry-tap-install-cluster-rolebinding'
```

```

Creating secret 'tap-telemetry-tap-install-values'
Creating package resource
Waiting for 'PackageInstall' reconciliation for 'tap-telemetry'
'PackageInstall' resource install status: Reconciling
'PackageInstall' resource install status: ReconcileSucceeded
'PackageInstall' resource successfully reconciled

Added installed package 'tap-telemetry'

```

## Deployment details and configurations of Tanzu Application Platform Telemetry

Use this topic to learn the deployment details and configurations of your Tanzu Application Platform Telemetry (commonly known as TAP Telemetry).

### What is deployed

The installation creates the following in your Kubernetes cluster:

- A deployment.
- A pod.
- A namespace `tap-telemetry`.
- A service account with read-write privileges named `informer`, and a corresponding secret for the service account. This secret is bound to a ClusterRole named `tap-telemetry-admin`.
- A Role `tap-telemetry-informer` to retrieve the deployment ID, which is sent as sender ID in heartbeat metrics.
- A RoleBinding `tap-telemetry-informer-admin` that binds the `informer` service account to the `tap-telemetry-informer` role.
- A ClusterRole `tap-telemetry-admin` that has access to each Tanzu Application Platform component to gather information from.
- A ClusterRoleBinding `tap-telemetry-informer-admin` that binds the `informer` service account to the `tap-telemetry-informer` cluster role.

### Deployment configuration

`customer_entitlement_account_number` is the unique identifier to differentiate between the data from your cluster and the data from other clusters. You can configure this property in your `tap-telemetry-values.yaml`:

```
customer_entitlement_account_number: "12345"
```

It creates a config map named `vmware-telemetry-identifiers` in the `vmware-system-telemetry` namespace, which is used internally to log your information.

Repeat these steps for the Build, Run, and View Cluster. For more information, see [Install multicluster Tanzu Application Platform profiles](#).

### Overview of Tanzu Build Service

This topic provides you with an overview of VMware Tanzu Build Service in Tanzu Application Platform (commonly known as TAP).

## Overview

Tanzu Build Service automates container creation, management, and governance at enterprise scale. Tanzu Build Service uses the open-source [Cloud Native Buildpacks](#) project to turn application source code into container images. It executes reproducible builds aligned with modern container standards and keeps images up to date.

For more information about Tanzu Build Service, see the [Tanzu Build Service documentation](#). For more information about Tanzu Buildpacks and their configuration, see the [Tanzu Buildpack documentation](#).

Tanzu Application Platform 1.4 includes Tanzu Build Service 1.9.

## Overview of Tanzu Build Service

This topic provides you with an overview of VMware Tanzu Build Service in Tanzu Application Platform (commonly known as TAP).

### Overview

Tanzu Build Service automates container creation, management, and governance at enterprise scale. Tanzu Build Service uses the open-source [Cloud Native Buildpacks](#) project to turn application source code into container images. It executes reproducible builds aligned with modern container standards and keeps images up to date.

For more information about Tanzu Build Service, see the [Tanzu Build Service documentation](#). For more information about Tanzu Buildpacks and their configuration, see the [Tanzu Buildpack documentation](#).

Tanzu Application Platform 1.4 includes Tanzu Build Service 1.9.

## Install Tanzu Build Service

This topic describes how to install Tanzu Build Service from the Tanzu Application Platform (commonly known as TAP) package repository by using the Tanzu CLI.

## Before you begin

Use this topic if you do not want to use a Tanzu Application Platform profile that includes Tanzu Build Service. The Full, Iterate, and Build profiles include Tanzu Build Service. For more information about profiles, see [Components and installation profiles](#).

The following procedure might not include some configurations required for your environment. For advanced information about installing Tanzu Build Service, see the [Tanzu Build Service documentation](#).

## Prerequisites

Before installing Tanzu Build Service:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see [Prerequisites](#).
- You must have access to a Docker registry that Tanzu Build Service can use to create builder images. Approximately 10 GB of registry space is required when using the `full` dependencies.
- Your Docker registry must be accessible with user name and password credentials.



## Deprecated Features

- **Automatic dependency updates:** For more information, see [Configure automatic dependency updates](#).
- **The Cloud Native Buildpack Bill of Materials (CNB BOM) format:** For more information, see [Deactivate the CNB BOM format](#).

## Install the Tanzu Build Service package

To install Tanzu Build Service by using the Tanzu CLI:

1. Get the latest version of the Tanzu Build Service package by running:

```
tanzu package available list buildservice.tanzu.vmware.com --namespace tap-inst
all
```

2. Gather the values schema by running:

```
tanzu package available get buildservice.tanzu.vmware.com/VERSION --values-sche
ma --namespace tap-install
```

Where `VERSION` is the version of the Tanzu Build Service package you retrieved in the previous step.

3. Create a `tbs-values.yaml` file using the following template:

```
---
kp_default_repository: "REPO-NAME"
kp_default_repository_username: "REPO-USERNAME"
kp_default_repository_password: "REPO-PASSWORD"
```

Where:

- `REPO-NAME` is a writable repository in your registry. Tanzu Build Service dependencies are written to this location. Examples:
  - Harbor has the form `"my-harbor.io/my-project/build-service"`.
  - Docker Hub has the form `"my-dockerhub-user/build-service"` or `"index.docker.io/my-user/build-service"`.
  - Google Cloud Registry has the form `"gcr.io/my-project/build-service"`.
- `REPO-USERNAME` and `REPO-PASSWORD` are the username and password for the user that can write to `REPO-NAME`. For Google Cloud Registry, use `_json_key` as the username and the contents of the service account JSON file for the password.



### Note

If you do not want to use plaintext for these credentials, you can configure them by using a secret reference or by using AWS IAM authentication. For more information, see [Use Secret References for registry credentials](#) or [Use AWS IAM authentication for registry credentials](#).

4. If you are running on Openshift, add `kubernetes_distribution: openshift` to your `tbs-values.yaml` file.

- (Optional) Under the `ca_cert_data` key in the `tbs-values.yaml` file, provide a PEM-encoded CA certificate for Tanzu Build Service. This certificate is used for accessing the container image registry and is also provided to the build process.



#### Note

If `shared.ca_cert_data` is configured in the `tap-values.yaml` file, Tanzu Build Service inherits that value.

Configuring `ca_cert_data` key in the `tbs-values.yaml` file adds the CA certificates at build time. To add CA certificates to the built image, see [Configure custom CA certificates for a single workload using service bindings](#).

For example:

```
---
kp_default_repository: "REPO-NAME"
kp_default_repository_username: "REPO-USERNAME"
kp_default_repository_password: "REPO-PASSWORD"
ca_cert_data: |
  -----BEGIN CERTIFICATE-----
  ...
  -----END CERTIFICATE-----
```

- (Optional) Tanzu Build Service is bootstrapped with the `lite` set of dependencies. To configure `full` dependencies, add the key-value pair `exclude_dependencies: true` to your `tbs-values.yaml` file. This is to exclude the default `lite` dependencies from the installation. For example:

```
---
kp_default_repository: "REPO-NAME"
kp_default_repository_username: "REPO-USERNAME"
kp_default_repository_password: "REPO-PASSWORD"
exclude_dependencies: true
```

For more information about the differences between `full` and `lite` dependencies, see [About lite and full dependencies](#).

- Install the Tanzu Build Service package by running:

```
tanzu package install tbs -p buildservice.tanzu.vmware.com -v VERSION -n tap-in
stall -f tbs-values.yaml
```

Where `VERSION` is the version of the Tanzu Build Service package you retrieved earlier.

For example:

```
$ tanzu package install tbs -p buildservice.tanzu.vmware.com -v VERSION -n tap-
install -f tbs-values.yaml

| Installing package 'buildservice.tanzu.vmware.com'
| Getting namespace 'tap-install'
| Getting package metadata for 'buildservice.tanzu.vmware.com'
| Creating service account 'tbs-tap-install-sa'
| Creating cluster admin role 'tbs-tap-install-cluster-role'
| Creating cluster role binding 'tbs-tap-install-cluster-rolebinding'
| Creating secret 'tbs-tap-install-values'
- Creating package resource
- Package install status: Reconciling
```

```
Added installed package 'tbs' in namespace 'tap-install'
```

- (Optional) Verify the cluster builders that the Tanzu Build Service installation created by running:

```
tanzu package installed get tbs -n tap-install
```

- If you configured `full` dependencies in your `tbs-values.yaml` file, install the `full` dependencies by following the procedure in [Install full dependencies](#).

## (Optional) Alternatives to plaintext registry credentials

Tanzu Build Service requires credentials for the `kp_default_repository` and the Tanzu Network registry.

You can apply them directly in-line in plaintext in the `tbs-values.yaml` or `tap-values.yaml` configuration by using the `kp_default_repository_username`, `kp_default_repository_password`, `tanzunet_username`, and `tanzunet_password` fields.

If you do not want credentials saved in plaintext, you can use existing secrets or IAM roles by using [secret references](#) or [AWS IAM authentication](#) in your `tbs-values.yaml` or `tap-values.yaml`.

### Use Secret references for registry credentials

You might not want to install Tanzu Build Service with passwords saved in plaintext in the `tbs-values.yaml`.

To store these credentials in `Secrets` and reference them in the `tbs-values.yaml`:

- Using the Tanzu CLI, create a secret of type `kubernetes.io/dockerconfigjson` containing credentials for the writable repository in your registry (`kp_default_repository`):

```
tanzu secret registry add kp-default-repository-creds \
  --username "${USERNAME}" \
  --password "${PASSWORD}" \
  --server "${SERVER-NAME}" \
  --namespace tap-install
```

Where:

- `USERNAME` and `PASSWORD` are the user name and password for the user that can write to the `kp_default_repository`. For Google Cloud Registry, use `_json_key` as the user name, and the contents of the service account JSON file for the password.
- `SERVER-NAME` is the host name of the registry server for the `kp_default_repository`.

Examples:

- Harbor has the form `server: "my-harbor.io"`.
- Docker Hub has the form `server: "index.docker.io"`.
- Google Cloud Registry has the form `server: "gcr.io"`.

- Use the following alternative configuration for `tbs-values.yaml`:



#### Note

if you are installing Tanzu Build Service as part of a Tanzu Application Platform profile, you configure this in your `tap-values.yaml` file under the `buildservice` section.

```

---
kp_default_repository: "KP-DEFAULT-REPOSITORY"
kp_default_repository_secret:
  name: kp-default-repository-creds
  namespace: tap-install

```

Where:

- `KP-DEFAULT-REPOSITORY` is a writable repository in your registry. Tanzu Build Service dependencies are written to this location. Examples:
  - Harbor has the form `"my-harbor.io/my-project/build-service"`.
  - Docker Hub has the form `"my-dockerhub-user/build-service"` or `"index.docker.io/my-user/build-service"`.
  - Google Cloud Registry has the form `"gcr.io/my-project/build-service"`.

3. To apply this configuration, continue the installation steps.

## Use AWS IAM authentication for registry credentials

Tanzu Build Service supports using AWS IAM roles to authenticate with Amazon Elastic Container Registry (ECR) on Amazon Elastic Kubernetes Service (EKS) clusters.

To use AWS IAM authentication:

1. Configure an AWS IAM role that has read and write access to the repository in the container image registry used when installing Tanzu Application Platform.
2. Use the following alternative configuration for `tbs-values.yaml`:



### Note

if you are installing Tanzu Build Service as part of a Tanzu Application Platform profile, you configure this in your `tap-values.yaml` file under the `buildservice` section.

```

---
kp_default_repository: "REPO-NAME"
kp_default_repository_aws_iam_role_arn: "IAM-ROLE-ARN"

```

Where:

- `REPO-NAME` is a writable repository in your registry. Tanzu Build Service dependencies are written to this location.
  - `IAM-ROLE-ARN` is the AWS IAM role Amazon Resource Name (ARN) for the role configured in the previous step. For example, `arn:aws:iam::xyz:role/my-install-role`.
3. The developer namespace requires configuration for Tanzu Application Platform to use AWS IAM authentication for ECR. Configure an AWS IAM role that has read and write access to the registry for storing workload images.
  4. Using the supply chain service account, add an annotation including the role ARN configured earlier by running:

```

kubectl annotate serviceaccount -n DEVELOPER-NAMESPACE SERVICE-ACCOUNT-NAME \
  eks.amazonaws.com/role-arn=IAM-ROLE-ARN

```

Where:

- `DEVELOPER-NAMESPACE` is the namespace where workloads are created.
- `SERVICE-ACCOUNT-NAME` is the supply chain service account. This is `default` if unset.
- `IAM-ROLE-ARN` is the AWS IAM role ARN for the role configured earlier. For example, `arn:aws:iam::xyz:role/my-developer-role`.

5. Apply this configuration by continuing the steps in [Install the Tanzu Build Service package](#).

## Install full dependencies

If you configured `full` dependencies in your `tbs-values.yaml` file, you must install the `full` dependencies package.

For a more information about `lite` and `full` dependencies, see [About lite and full dependencies](#).

To install `full` Tanzu Build Service dependencies:

1. If you have not done so already, add the key-value pair `exclude_dependencies: true` to your `tbs-values.yaml` file. For example:



### Note

if you are installing Tanzu Build Service as part of a Tanzu Application Platform profile, you configure this in your `tap-values.yaml` file under the `buildservice` section.

```
---
kp_default_repository: "REPO-NAME"
kp_default_repository_username: "REPO-USERNAME"
kp_default_repository_password: "REPO-PASSWORD"
exclude_dependencies: true
```

2. Get the latest version of the Tanzu Build Service package by running:

```
tanzu package available list buildservice.tanzu.vmware.com --namespace tap-inst
all
```

3. Relocate the Tanzu Build Service `full` dependencies package repository by running:

```
imgpkg copy -b registry.tanzu.vmware.com/tanzu-application-platform/full-tbs-de
ps-package-repo:VERSION \
--to-repo INSTALL-REGISTRY-HOSTNAME/TARGET-REPOSITORY/tbs-full-deps
```

Where:

- `VERSION` is the version of the Tanzu Build Service package you retrieved in the previous step.
- `INSTALL-REGISTRY-HOSTNAME` is your container image registry.
- `TARGET-REPOSITORY` is your target repository.

4. Add the Tanzu Build Service `full` dependencies package repository by running:

```
tanzu package repository add tbs-full-deps-repository \
--url INSTALL-REGISTRY-HOSTNAME/TARGET-REPOSITORY/tbs-full-deps:VERSION \
--namespace tap-install
```

Where:

- `VERSION` is the version of the Tanzu Build Service package you retrieved earlier.

- `INSTALL-REGISTRY-HOSTNAME` is your container image registry.
- `TARGET-REPOSITORY` is your target repository.

5. Install the `full` dependencies package by running:

```
tanzu package install full-tbs-deps -p full-tbs-deps.tanzu.vmware.com -v VERSION
N -n tap-install
```

Where `VERSION` is the version of the Tanzu Build Service package you retrieved earlier.

## (Optional) Configure automatic dependency updates



### Important

The automatic updates feature is being deprecated. The recommended way to patch dependencies is by upgrading Tanzu Application Platform to the latest patch version. For upgrade instructions, see [Upgrading Tanzu Application Platform](#).

You can configure Tanzu Build Service to update dependencies in the background as they are released. This enables workloads to keep up to date automatically. For more information about automatic dependency updates, see [About automatic dependency updates \(deprecated\)](#).

To configure automatic dependency updates, add the following to the contents of your `tbs-values.yaml`:



### Note

If you are installing Tanzu Build Service as part of a Tanzu Application Platform profile, you configure this in your `tap-values.yaml` file under the `buildservice` section.

```
tanzunet_username: TANZU-NET-USERNAME
tanzunet_password: TANZU-NET-PASSWORD
descriptor_name: DESCRIPTOR-NAME
enable_automatic_dependency_updates: true
```

Where:

- `TANZU-NET-USERNAME` and `TANZU-NET-PASSWORD` are the email address and password to log in to VMware Tanzu Network. You can also configure these credentials by using a secret reference. For more information, see [Use Secret references for registry credentials](#).
- `DESCRIPTOR-NAME` is the name of the descriptor to import. For more information, see [Descriptors](#). Available options are:
  - `lite` is the default if not set. It has a smaller footprint, which enables faster installations.
  - `full` is optimized to speed up builds and includes dependencies for all supported workload types.

## (Optional) Deactivate the CNB BOM format

The legacy CNB BOM format is deprecated, but is enabled by default in Tanzu Application Platform.

To manually deactivate the format, add `include_legacy_bom=false` to either the `tbs-values.yaml` file, or to the `tap-values.yaml` file under the `buildservice` section.

## Install Tanzu Build Service on an air-gapped environment

This topic describes how to install Tanzu Build Service on a Kubernetes cluster and registry that are air-gapped from external traffic.

### Before you begin

Use this topic if you do not want to use a Tanzu Application Platform profile that includes Tanzu Build Service.

The Full, Iterate, and Build profiles include Tanzu Build Service. For more information about profiles, see [Components and installation profiles](#).

### Prerequisites

Before installing Tanzu Build Service:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see [Prerequisites](#).
- You must have access to a Docker registry that Tanzu Build Service can use to create builder images. Approximately 10 GB of registry space is required when using the `full` dependencies.
- Your Docker registry must be accessible with user name and password credentials.

### Deprecated Features

**The Cloud Native Buildpack Bill of Materials (CNB BOM) format:** For more information, see [Deactivate the CNB BOM format](#).

## Install the Tanzu Build Service package

These steps assume that you have installed the Tanzu Application Platform packages in your air-gapped environment.

To install the Tanzu Build Service package on an air-gapped environment:

1. Get the latest version of the Tanzu Build Service package by running:

```
tanzu package available list buildservice.tanzu.vmware.com --namespace tap-inst
all
```

2. Gather the values schema by running:

```
tanzu package available get buildservice.tanzu.vmware.com/VERSION --values-sche
ma --namespace tap-install
```

Where `VERSION` is the version of the Tanzu Build Service package you retrieved in the previous step.

3. Create a `tbs-values.yaml` file. The required fields for an air-gapped installation are as follows:

```
---
kp_default_repository: REPO-NAME
kp_default_repository_username: REGISTRY-USERNAME
kp_default_repository_password: REGISTRY-PASSWORD
```

```
ca_cert_data: CA-CERT-CONTENTS
exclude_dependencies: true
```

Where:

- o `REPO-NAME` is the fully qualified path to a writeable repository in your internal registry. Tanzu Build Service dependencies are written to this location. For example:
  - For Harbor: `harbor.io/my-project/build-service`
  - For Artifactory: `artifactory.com/my-project/build-service`
- o `REPO-USERNAME` and `REPO-PASSWORD` are the user name and password for the user that can write to `REPO-NAME`.



#### Note

If you do not want to use plaintext for these credentials, you can instead configure these credentials by using a Secret reference. For more information, see [Use Secret references for registry credentials](#).

- o `CA-CERT-CONTENTS` are the contents of the PEM-encoded CA certificate for the internal registry.

#### 4. Install the package by running:

```
tanzu package install tbs -p buildservice.tanzu.vmware.com -v VERSION -n tap-in
stall -f tbs-values.yaml
```

Where `VERSION` is the version of the Tanzu Build Service package you retrieved earlier.

For example:

```
$ tanzu package install tbs -p buildservice.tanzu.vmware.com -v VERSION -n tap-
install -f tbs-values.yaml

| Installing package 'buildservice.tanzu.vmware.com'
| Getting namespace 'tap-install'
| Getting package metadata for 'buildservice.tanzu.vmware.com'
| Creating service account 'tbs-tap-install-sa'
| Creating cluster admin role 'tbs-tap-install-cluster-role'
| Creating cluster role binding 'tbs-tap-install-cluster-rolebinding'
| Creating secret 'tbs-tap-install-values'
- Creating package resource
- Package install status: Reconciling
Added installed package 'tbs' in namespace 'tap-install'
```

## Install the Tanzu Build Service dependencies

By default, Tanzu Build Service is installed with `lite` dependencies.

When installing Tanzu Build Service on an air-gapped environment, the `lite` dependencies cannot be used as they require Internet access. You must install the `full` dependencies.

To install `full` dependencies:

1. Relocate the Tanzu Build Service `full` dependencies package repository by running:

```
imgpkg copy -b registry.tanzu.vmware.com/tanzu-application-platform/full-tbs-de
ps-package-repo:VERSION \
  --to-tar=tbs-full-deps.tar
# move tbs-full-deps.tar to environment with registry access
```



```
imgpkg copy --tar tbs-full-deps.tar \
  --to-repo=INSTALL-REGISTRY-HOSTNAME/TARGET-REPOSITORY/tbs-full-deps
```

Where:

- `VERSION` is the version of the Tanzu Build Service package you retrieved earlier.
- `INSTALL-REGISTRY-HOSTNAME` is your container registry.
- `TARGET-REPOSITORY` is your target repository.

2. Add the Tanzu Build Service `full` dependencies package repository by running:

```
tanzu package repository add tbs-full-deps-repository \
  --url INSTALL-REGISTRY-HOSTNAME/TARGET-REPOSITORY/tbs-full-deps:VERSION \
  --namespace tap-install
```

Where:

- `INSTALL-REGISTRY-HOSTNAME` is your container registry.
- `TARGET-REPOSITORY` is your target repository.
- `VERSION` is the version of the Tanzu Build Service package you retrieved earlier.

3. Install the `full` dependencies package by running:

```
tanzu package install full-tbs-deps -p full-tbs-deps.tanzu.vmware.com -v VERSION
N -n tap-install
```

Where `VERSION` is the version of the Tanzu Build Service package you retrieved earlier.

## Configure Tanzu Build Service properties on a workload

This topic tells you how to configure your workload with Tanzu Build Service properties.

### Overview

Tanzu Build Service builds registry images from source code for Tanzu Application Platform. You can configure these build configurations by using a workload.

Tanzu Build Service is only applicable to the build process. Configurations, such as environment variables and service bindings, might require a different process for runtime.

### Configure build-time service bindings

You can configure build-time service bindings for Tanzu Build Service.

Tanzu Build Service supports using the Service Binding Specification for Kubernetes for application builds. For more information, see the [service binding specification for Kubernetes](#) in GitHub.

Service binding configuration is specific to the buildpack that is used to build the app. For more information about configuring buildpack service bindings for the buildpack you are using, see the [VMware Tanzu Buildpacks documentation](#).

To configure a service binding for a Tanzu Application Platform workload, follow these steps:

1. Create a YAML file named `service-binding-secret.yaml` for a secret as follows:

```
apiVersion: v1
kind: Secret
metadata:
  name: settings-xml
  namespace: DEVELOPER-NAMESPACE
```

```

type: service.binding/maven
stringData:
  type: maven
  provider: sample
  settings.xml: |
    MY-SETTINGS

```

Where: - `DEVELOPER-NAMESPACE` is the namespace where workloads are created. - `MY-SETTINGS` is the contents of your service bindings file.

2. Apply the YAML file by running:

```
kubectl apply -f service-binding-secret.yaml
```

3. Create the workload with `buildServiceBindings` configured by running:

```

tanzu apps workload create WORKLOAD-NAME \
  --param-yaml buildServiceBindings='[{"name": "settings-xml", "kind": "Secret"}]' \
  ...

```

Where `WORKLOAD-NAME` is the name of the workload you want to configure.

## Configure environment variables

If you have build-time environment variable dependencies, you can set environment variables that are available at build-time.

You can also configure buildpacks with environment variables. Buildpack configuration depends on the specific buildpack being used. For more information about configuring environment variables for the buildpack you are using, see the [VMware Tanzu Buildpacks documentation](#).

For example:

```

tanzu apps workload create WORKLOAD-NAME \
  --build-env "ENV_NAME=ENV_VALUE" \
  --build-env "BP_MAVEN_BUILD_ARGUMENTS=-Dmaven.test.skip=true"

```

Where `WORKLOAD-NAME` is the name of the workload you want to configure.

## Configure the service account

Using the Tanzu CLI, you can configure the service account used during builds. This service account is the one configured for the developer namespace. If unset, `default` is used.

To configure the service account used during builds, run:

```

tanzu apps workload create WORKLOAD-NAME \
  --param serviceAccount=SERVICE-ACCOUNT-NAME \

```

Where:

- `WORKLOAD-NAME` is the name of the workload you want to configure.
- `SERVICE-ACCOUNT-NAME` is the name of the service account you want to use during builds.

## Configure the cluster builder

To configure the ClusterBuilder used during builds:

1. View the available ClusterBuilds by running:

```
kubectl get clusterbuilder
```

2. Set the ClusterBuilder used during builds by running:

```
tanzu apps workload create WORKLOAD-NAME \
  --param clusterBuilder=CLUSTER-BUILDER-NAME \
```

Where:

- `WORKLOAD-NAME` is the name of the workload you want to configure.
- `CLUSTER-BUILDER-NAME` is the ClusterBuilder you want to use.

## Configure the workload container image registry

Using the Tanzu CLI, you can configure the registry where workload images are saved. The service account used for this workload must have read and write access to this registry location.

To configure the registry where workload images are saved, run:

```
tanzu apps workload create WORKLOAD-NAME \
  --param-yaml registry={"server": SERVER-NAME, "repository": REPO-NAME}
```

Where:

- `SERVER-NAME` is the host name of the registry server. Examples:
  - Harbor has the form `"my-harbor.io"`.
  - Docker Hub has the form `"index.docker.io"`.
  - Google Cloud Registry has the form `"gcr.io"`.
- `REPO-NAME` is where workload images are stored in the registry. Images are written to `SERVER-NAME/REPO-NAME/workload-name`. Examples:
  - Harbor has the form `"my-project/supply-chain"`.
  - Docker Hub has the form `"my-dockerhub-user"`.
  - Google Cloud Registry has the form `"my-project/supply-chain"`.

## Configure custom CA certificates for a single workload using service bindings

If the language family buildpack you are using includes the Paketo CA certificates buildpack, you can use a service binding to provide custom certificates during the build and run process. For more information about language family buildpacks, see the [Tanzu Buildpacks documentation](#).

To create a service binding to provide custom CA certificates for a workload:

1. Create a YAML file named `service-binding-ca-cert.yaml` for a secret as follows:

```
apiVersion: v1
kind: Secret
metadata:
  name: my-ca-certs
data:
  type: ca-certificates
  provider: sample
  CA-CERT-FILENAME: |
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
```

Where `CA-CERT-FILENAME` is the name of your PEM encoded CA certificate file. For example, `arbitrary-file-name.pem`.

2. Apply the YAML file by running:

```
kubectl apply -f service-binding-ca-cert.yaml
```

3. To build with the custom certificate, create the workload with `--param-yaml` `buildServiceBindings` flag:

```
tanzu apps workload create WORKLOAD-NAME \
  --param-yaml buildServiceBindings='[{"apiVersion": "v1", "kind": "Secret", "name": "my-ca-certs"}]' \
  ...
```

Where `WORKLOAD-NAME` is the name of the workload you want to create.

4. To deploy with the custom certificate, create the workload with the `--service-ref` flag:

```
tanzu apps workload create WORKLOAD-NAME \
  --service-ref my-ca-certs=v1:Secret:my-ca-certs \
  ...
```

Where `WORKLOAD-NAME` is the name of the workload you want to create.

## Using custom CA certificates for all workloads

To provide custom CA certificates to the build process for all workloads, see the optional step to add the `ca_cert_data` key [Install the Tanzu Build Service package](#).

## Create a signed container image with Tanzu Build Service

This topic tells you how to create a Tanzu Build Service image resource that builds a container image from source code signed with Cosign.

### Prerequisites

Before you can configure Tanzu Build Service to sign your image builds, you must:

- Install Tanzu Build Service. The Full, Iterate, and Build profiles include Tanzu Build Service by default. If you have not installed Tanzu Application Platform with one of these profiles, see [Installing Tanzu Build Service](#).
- Install Cosign. For instructions, see the [Cosign documentation](#).
- Have a [Builder](#) or [ClusterBuilder](#) resource configured.
- Have an [image](#) resource configured.
- Review the [kpack tutorial](#). This topic builds upon the steps in this tutorial.

## Configure Tanzu Build Service to sign your image builds

To configure Tanzu Build Service to sign your image builds:

1. Ensure that you are in a Kubernetes context where you are authenticated and authorized to create and edit secret and service account resources.
2. Generate a Cosign keypair and store it as a Kubernetes secret by running:

```
cosign generate-key-pair k8s://NAMESPACE/COSIGN-KEYPAIR-NAME
```

Where:

- `NAMESPACE` is the namespace to store the Kubernetes secret in.
- `COSIGN-KEYPAIR-NAME` is the name of the Kubernetes secret.

For example:

```
cosign generate-key-pair k8s://default/tutorial-cosign-key-pair
```

3. Enter a password for the private key. Enter any password you want. After the command has completed, you will see the following output:

```
Successfully created secret tutorial-cosign-key-pair in namespace default
Public key written to cosign.pub
```

You will also see a `cosign.pub` file in your current directory. Keep this file as you will need it to verify the signature of the images that are built.

4. If you are using [Docker Hub](#) or a registry that does not support OCI media types, add the annotation `kpack.io/cosign.docker-media-types: "1"` to the Cosign secret as follows:

```
apiVersion: v1
kind: Secret
type: Opaque
metadata:
  name: tutorial-cosign-key-pair
  namespace: default
  annotations:
    kpack.io/cosign.docker-media-types: "1"
data:
  cosign.key: PRIVATE-KEY-DATA
  cosign.password: COSIGN-PASSWORD
  cosign.pub: PUBLIC-KEY-DATA
```

For more information about configuring Cosign key pairs, see the [Tanzu Build Service documentation](#).

5. To enable Cosign signing, create or edit the service account resource that is referenced in the image resource so that it includes the Cosign keypair secret created earlier. The service account is in the same namespace as the image resource and is directly referenced by the image or default if there isn't one. The default is the default service account in the workload namespace.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: SERVICE-ACCOUNT-NAME
  namespace: default
secrets:
- name: REGISTRY-CREDENTIALS
- name: COSIGN-KEYPAIR-NAME
imagePullSecrets:
- name: REGISTRY-CREDENTIALS
```

Where:

- `SERVICE-ACCOUNT-NAME` is the name of your service account resource. For example, `tutorial-cosign-service-account`.

- `COSIGN-KEYPAIR-NAME` is the name of the Cosign keypair secret generated earlier. For example, `tutorial-cosign-key-pair`.
- `REGISTRY-CREDENTIALS` is the secret that provides credentials for the container registry where application container images are pushed to.

6. Apply the service account resource to the cluster by running:

```
kubectl apply -f cosign-service-account.yaml
```

7. Create an image resource file named `image-cosign.yaml`. For example:

```
apiVersion: kpack.io/v1alpha2
kind: Image
metadata:
  name: tutorial-cosign-image
  namespace: default
spec:
  tag: IMAGE-REGISTRY
  serviceAccountName: tutorial-cosign-service-account
  builder:
    name: my-builder
    kind: Builder
  source:
    git:
      url: https://github.com/spring-projects/spring-petclinic
      revision: 82cb521d636b282340378d80a6307a08e3d4a4c4
```

Where:

- `IMAGE-REGISTRY` with a writable repository in your registry. The secret referenced in the service account is a secret providing credentials for the registry where application container images are pushed to. For example:
    - Harbor has the form `"my-harbor.io/my-project/my-repo"`
    - Docker Hub has the form `"my-dockerhub-user/my-repo"` or `"index.docker.io/my-user/my-repo"`
    - Google Cloud Registry has the form `"gcr.io/my-project/my-repo"`
8. If you are using Out of the Box Supply Chains, edit the respective `ClusterImageTemplate` to enable signing in your supply chain. For more information, see [Authoring supply chains](#).



#### Important

VMware discourages referencing the service account using the `service_account` value when installing the Out of the Box Supply Chain. This is because it gives your run cluster access to the private signing key.

9. Apply the image resource to the cluster by running:

```
kubectl apply -f image-cosign.yaml
```

10. After the image resource finishes building, you can get the fully resolved and built OCI image by running:

```
kubectl -n default get image tutorial-cosign-image
```

Example output:

NAME	LATESTIMAGE	READY
tutorial-cosign-image	index.docker.io/your-project/app@sha256:6744b...	True

- Verify image signature by running:

```
cosign verify --key cosign.pub LATEST-IMAGE-WITH-DIGEST
```

Where `LATEST-IMAGE-WITH-DIGEST` is the value of `LATESTIMAGE` you retrieved in the previous step. For example: `index.docker.io/your-project/app@sha256:6744b...`

The expected output is similar to the following:

```
Verification for index.docker.io/your-project/app@sha256:6744b... --
The following checks were performed on each of these signatures:
- The cosign claims were validated
- The signatures were verified against the specified public key
- Any certificates were verified against the Fulcio roots.
```

- Configure Supply Chain Security Tools for VMware Tanzu - Policy Controller to ensure that only signed images are allowed in your cluster. For more information, see the [Supply Chain Security Tools for VMware Tanzu - Policy Controller](#) documentation.

## Tanzu Build Service Dependencies

This topic tells you about Tanzu Build Service dependencies.



### Important

Ubuntu Bionic will stop receiving support in April 2023. The Bionic stack for Tanzu Build Service is deprecated and will be removed in a future release. VMware recommends that you migrate builds to Jammy stacks. For how to migrate builds, see [Use Jammy stacks for a workload](#).

To build OCI images, Tanzu Build Service has the following dependencies: Cloud Native [Buildpacks](#), [Stacks](#), and [Lifecycle](#)s.

## How dependencies are installed

When Tanzu Application Platform is installed with Tanzu Build Service, it is bootstrapped with a set of dependencies. No extra configuration is required. Each version of Tanzu Application Platform and Tanzu Build Service contains new dependencies.

When Tanzu Application Platform is upgraded, new dependencies are installed which might cause workload images to rebuild. To ensure dependency compatibility, Tanzu Build Service only releases patches for dependencies in patch versions of Tanzu Application Platform. For upgrade instructions, see [Upgrade the full dependencies package](#).

To upgrade Tanzu Build Service dependencies outside of Tanzu Application Platform releases, use the `kpack` CLI. This enables you to consume new versions of buildpacks and stacks and remediate vulnerabilities more quickly. For more information, see [Updating Build Service Dependencies](#).

By default, Tanzu Build Service is installed with the `lite` set of dependencies, which are smaller-footprint and contain a subset of the buildpacks and stacks in the `full` set of dependencies. For a comparison of `lite` and `full` dependencies, see [Dependency comparison](#) later in this topic.

## View installed dependencies

To view the set of dependencies installed with Tanzu Build Service, inspect the status of the cluster builders by running:

```
kubectl get clusterbuilder -o yaml
```

Cluster builders contain stack and buildpack metadata.

## Bionic and Jammy stacks

Tanzu Application Platform v1.3 and later supports Ubuntu v22.04 (Jammy) based builds. Ubuntu Bionic will stop receiving support in April 2023. VMware recommends that you migrate builds to Jammy.

For more information about support for Jammy stacks, see [About lite and full dependencies](#) later in this topic.

## Use Jammy stacks for a workload

To use the Jammy stacks or migrate an existing workload, configure the workload with a Jammy builder by using the `param` flag, for example, `--param clusterBuilder=base-jammy`. For further instructions, see [Configure the cluster builder](#).



### Note

While upgrading apps to a newer stack, you might encounter the build platform erroneously reusing the old build cache. If you encounter this issue, delete and recreate the workload in Tanzu Application Platform, or delete and recreate the image in Tanzu Build Service.

## Default all workloads to Jammy stacks

By default, Tanzu Application Platform is installed with Bionic as the default stack.

To default all workloads to the Jammy stack, include the `stack_configuration: jammy-only` field under the `buildservice:` section in `tap-values.yaml`. This installs Tanzu Application Platform and Tanzu Build Service with no Bionic-based builders, and all workloads will be built with Jammy.



### Important

Only use this configuration if you are sure all workloads can be safely built with Jammy.

## About lite and full dependencies

Each version of Tanzu Application Platform is released with two types of Tanzu Build Service dependencies: `lite` and `full`. These dependencies consist of the buildpacks and stacks required for application builds. Each type serves different use cases. Both types are suitable for production workloads.

By default, Tanzu Build Service is installed with `lite` dependencies, which do not contain all buildpacks and stacks. To use all buildpacks and stacks, you must install the `full` dependencies. For instructions about installing `full` dependencies, see [Install full dependencies](#).

For a table comparing the differences between `full` and `lite` dependencies, see [Dependency comparison](#).



## Lite dependencies

The `lite` dependencies are the default set installed with Tanzu Build Service.

`lite` dependencies contain a smaller footprint to speed up installation time, but do not support all workload types. For example, `lite` dependencies do not contain the PHP buildpack and cannot be used to build PHP workloads.

### Lite dependencies: stacks

The `lite` dependencies contain the following stacks:

- `base` (ubuntu Bionic)
- `default` (identical to `base`)
- `base-jammy` (ubuntu Jammy)

For more information, see [Stacks](#) in the VMware Tanzu Buildpacks documentation.

### Lite dependencies: buildpacks

The `lite` dependencies contain the following buildpacks in Tanzu Application Platform v1.3:

Buildpack	Version	Supported Stacks
Java Buildpack for VMware Tanzu (Lite)	8.0.0	Bionic, Jammy
Java Native Image Buildpack for Tanzu (Lite)	6.39.0	Bionic, Jammy
.NET Core Buildpack for VMware Tanzu (Lite)	1.19.2	Bionic, Jammy
Node.js Buildpack for VMware Tanzu (Lite)	1.17.1	Bionic, Jammy
Python Buildpack for VMware Tanzu (Lite)	2.3.1	Bionic, Jammy
Go Buildpack for VMware Tanzu (Lite)	2.0.4	Bionic, Jammy
Web Servers Buildpack for VMware Tanzu (Lite)	0.4.1	Bionic, Jammy
Ruby Buildpack for VMware Tanzu (Lite)	1.5.0	Bionic, Jammy
Procfile Buildpack for VMware Tanzu (Lite)	5.4.0	Bionic, Jammy

And the following components:

Component	Version	Supported Stacks
CNB Lifecycle	0.14.3	Bionic, Jammy
Base Stack of Ubuntu Bionic for VMware Tanzu	1.2.34	Bionic
Base Stack of Ubuntu Jammy for VMware Tanzu	0.1.16	Jammy

## Full dependencies

The Tanzu Build Service `full` set of dependencies contain more buildpacks and stacks, which allows for more workload types.

The dependencies are pre-packaged, so builds do not have to download them from the Internet. This can speed up build times and allows builds to occur in air-gapped environments. Due to the larger footprint of `full`, installations might take longer.

The `full` dependencies are not installed with Tanzu Build Service by default, you must install them. For instructions for installing `full` dependencies, see [Install Tanzu Build Service with full dependencies](#).

## Full dependencies: stacks

The `full` dependencies contain the following stacks, which support different use cases:

- `base` (ubuntu Bionic)
- `default` (identical to `base`)
- `full` (ubuntu Bionic)
- `tiny` (ubuntu Bionic)
- `base-jammy` (ubuntu Jammy)
- `full-jammy` (ubuntu Jammy)
- `tiny-jammy` (ubuntu Jammy)

For more information, see [Stacks](#) in the VMware Tanzu Buildpacks documentation.

## Full dependencies: buildpacks

The `full` dependencies contain the following buildpacks in Tanzu Application Platform v1.3:

Buildpack	Version	Supported Stacks
Java Buildpack for VMware Tanzu	8.0.0	Bionic, Jammy
Java Native Image Buildpack for Tanzu	6.39.0	Bionic, Jammy
.NET Core Buildpack for VMware Tanzu	1.19.2	Bionic, Jammy
Node.js Buildpack for VMware Tanzu	1.17.1	Bionic, Jammy
Python Buildpack for VMware Tanzu	2.3.1	Bionic, Jammy
Ruby Buildpack for VMware Tanzu	1.5.0	Bionic, Jammy
Go Buildpack for VMware Tanzu	2.0.4	Bionic, Jammy
PHP Buildpack for VMware Tanzu	2.0.0	Bionic, Jammy
Web Servers Buildpack for VMware Tanzu	0.4.1	Bionic, Jammy
Profile Buildpack for VMware Tanzu	5.4.0	Bionic, Jammy

And the following components:

Component	Version	Supported Stacks
CNB Lifecycle	0.14.3	Bionic, Jammy
Tiny Stack of Ubuntu Bionic for VMware Tanzu	1.3.88	Bionic
Base Stack of Ubuntu Bionic for VMware Tanzu	1.2.34	Bionic
Full Stack of Ubuntu Bionic for VMware Tanzu	1.3.121	Bionic
Tiny Stack of Ubuntu Jammy for VMware Tanzu	0.1.17	Jammy
Base Stack of Ubuntu Jammy for VMware Tanzu	0.1.16	Jammy
Full Stack of Ubuntu Jammy for VMware Tanzu	0.1.38	Jammy

## Dependency comparison

The following table compares the contents of the `lite` and `full` dependencies.

	lite	full
Faster installation time	Yes	No
Dependencies pre-packaged (faster builds)	No	Yes
Supports air-gapped installation	No	Yes
Contains base stack	Yes	Yes
Contains full stack	No	Yes
Contains tiny stack	No	Yes
Contains Jammy stack	Yes	Yes
Supports Java workloads	Yes	Yes
Supports Node.js workloads	Yes	Yes
Supports Go workloads	Yes	Yes
Supports Python workloads	Yes	Yes
Supports Ruby workloads	No	Yes
Supports .NET Core workloads	Yes	Yes
Supports PHP workloads	No	Yes
Supports static workloads	Yes	Yes
Supports binary workloads	Yes	Yes
Supports web servers buildpack	Yes	Yes

## About automatic dependency updates (deprecated)



### Important

The automatic updates feature is being deprecated. The recommended way to patch dependencies is by upgrading Tanzu Application Platform to the latest patch version. For upgrade instructions, see [Upgrading Tanzu Application Platform](#).

You can configure Tanzu Build Service to update dependencies in the background as they are released. This enables workloads to keep up to date automatically.

## Descriptors (deprecated)

Tanzu Build Service descriptors are curated sets of dependencies that include stacks and buildpacks. Descriptors are only used if Tanzu Build Service is configured for automatic dependency updates. Descriptors are imported into Tanzu Build Service to update the entire cluster.

Descriptors are continuously released on the [VMware Tanzu Network Build Service Dependencies](#) page to provide updated buildpack dependencies and updated stack images. This allows the use of dependencies that have patched CVEs. For more information about buildpacks and stacks, see the [VMware Tanzu Buildpacks documentation](#).

There are two types of descriptor, `lite` and `full`. The different descriptors can apply to different use cases and workload types. The differences between the `full` and `lite` descriptors are the same as the the differences between `full` and `lite` dependencies. For a comparison of the `lite` and `full` descriptors, see [About lite and full dependencies](#).

## Security context constraint for OpenShift

This topic tells you about running Tanzu Build Service on OpenShift clusters.

On OpenShift clusters Tanzu Build Service must run with a custom [Security Context Constraint](#) (SCC) to enable compliance. Tanzu Application Platform configures the following SCC for Tanzu Build Service when you configure the `kubernetes_distribution: openshift` key in the `tap-values.yaml` file.

```
---
kind: SecurityContextConstraints
apiVersion: security.openshift.io/v1
metadata:
  name: tbs-restricted-scc-with-seccomp
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: false
allowPrivilegedContainer: false
allowedCapabilities:
  - NET_BIND_SERVICE
defaultAddCapabilities: null
fsGroup:
  type: RunAsAny
groups: []
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities:
  - ALL
runAsUser:
  type: MustRunAsNonRoot
seLinuxContext:
  type: MustRunAs
seccompProfiles:
  - runtime/default
supplementalGroups:
  type: RunAsAny
users: []
volumes:
  - configMap
  - downwardAPI
  - emptyDir
  - persistentVolumeClaim
  - projected
  - secret
```

It also applies the following RBAC to allow Tanzu Build Service services to use the SCC:

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    apps.tanzu.vmware.com/aggregate-to-workload: "true"
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  name: system:tbs:scc:restricted-with-seccomp
rules:
  - apiGroups:
    - security.openshift.io
    resourceNames:
    - tbs-restricted-scc-with-seccomp
```

```

resources:
  - securitycontextconstraints
verbs:
  - use
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: system:tbs:scc:restricted-with-seccomp
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:tbs:scc:restricted-with-seccomp
subjects:
  - kind: ServiceAccount
    namespace: build-service
    name: dependency-updater-serviceaccount
  - kind: ServiceAccount
    namespace: build-service
    name: dependency-updater-controller-serviceaccount
  - kind: ServiceAccount
    namespace: build-service
    name: secret-syncer-service-account
  - kind: ServiceAccount
    namespace: build-service
    name: warmer-service-account
  - kind: ServiceAccount
    namespace: build-service
    name: build-service-daemonset-serviceaccount
  - kind: ServiceAccount
    namespace: cert-injection-webhook
    name: cert-injection-webhook-sa
  - kind: ServiceAccount
    namespace: kpack
    name: kp-default-repository-serviceaccount
  - kind: ServiceAccount
    namespace: kpack
    name: kpack-pull-lifecycle-serviceaccount
  - kind: ServiceAccount
    namespace: kpack
    name: controller
  - kind: ServiceAccount
    namespace: kpack
    name: webhook
  - kind: ServiceAccount
    namespace: stacks-operator-system
    name: controller-manager

```

## Troubleshoot Tanzu Build Service

This topic tells you how to troubleshoot Tanzu Build Service when used with Tanzu Application Platform (commonly known as TAP).

### Builds fail due to volume errors on EKS running Kubernetes v1.23

#### Symptom

After installing or upgrading Tanzu Application Platform on an Amazon Elastic Kubernetes Service (EKS) cluster running Kubernetes v1.23, build pods show:

```
'running PreBind plugin "VolumeBinding": binding volumes: timed out waiting
for the condition'
```

## Cause

This is due to the CSIMigrationAWS in this Kubernetes version, which requires users to install the [Amazon EBS CSI driver](#) to use AWS Elastic Block Store (EBS) volumes. For more information about EKS support for Kubernetes v1.23, see the [Amazon blog post](#).

Tanzu Application Platform uses the default storage class which uses EBS volumes by default on EKS.

## Solution

Follow the AWS documentation to install the [Amazon EBS CSI driver](#) before installing Tanzu Application Platform, or before upgrading to Kubernetes v1.23.

## Smart-warmer-image-fetcher reports ErrImagePull due to dockerd's layer depth limitation

### Symptom

When using dockerd as the cluster's container runtime, you might see the `smart-warmer-image-fetcher` pods report a status of `ErrImagePull`.

### Cause

This error might be due to dockerd's layer depth limitation, in which the maximum supported image layer depth is 125.

To verify that the `ErrImagePull` status is due to dockerd's maximum supported image layer depth, check for event messages containing the words `max depth exceeded`. For example:

```
$ kubectl get events -A | grep "max depth exceeded"
build-service          73s          Warning      Failed          pod/smart-warmer-image-f
etcher-wxtr8          Failed to pull image
  "harbor.somewhere.com/aws-repo/build-service:clusterbuilder-full@sha256:065bb361fd91
4a3970ad3dd93c603241e69cca214707feaa6
d8617019e20b65e": rpc error: code = Unknown desc = failed to register layer: max de
pth exceeded
```

### Solution

To work around this issue, configure your cluster to use containerd or CRI-O as its default container runtime. For instructions, refer to the following documentation for your Kubernetes cluster provider.

For AWS, see:

- The [Amazon blog](#)
- The [eksctl CLI documentation](#)

For AKS, see:

- The [Microsoft Azure documentation](#)
- The [Microsoft Azure blog](#)

For GKE, see:

- The [GKE documentation](#)

For OpenShift, see:

- The [Red Hat Hybrid Cloud blog](#)
- The [Red Hat Openshift documentation](#)

---

## Nodes fail due to “trying to send message larger than max” error

### Symptom

You see the following error, or similar, in a node status:

```
Warning ContainerGCFailed 119s (x2523 over 42h) kubelet rpc error: code = ResourceExhausted desc = grpc: trying to send message larger than max (16779959 vs. 16777216)
```

### Cause

This is due to the way that the container runtime interface (CRI) handles garbage collection for unused images and containers.

### Solution

Do not use Docker as the CRI because it is not supported. Some versions of EKS default to Docker as the runtime.

---

## Build platform uses the old build cache after upgrade to new stack

### Symptom

While upgrading apps to a newer stack, you might encounter the build platform erroneously reusing the old build cache.

### Solution

If you encounter this issue, delete and recreate the workload in Tanzu Application Platform, or delete and recreate the image in Tanzu Build Service.

## Create a GitHub build action (Alpha)

This topic tells you how to use a GitHub action to create a Tanzu Build Service build on a cluster.



#### Important

Alpha features are experimental and are not ready for production use. Configuration and behavior is likely to change, and functionality might be removed in a future release.

## Prerequisites

- Ensure that [Tanzu Application Platform](#) is installed.

## Procedure

### Developer namespace

1. Create a developer namespace where the build resource will be created.

```
kubectl create namespace DEVELOPER-NAMESPACE
```

2. Create a service account in the `DEVELOPER-NAMESPACE` that has access to the registry credentials. This service account name will be used in the action.

### Access to Kubernetes API server

The GitHub action talks directly to the Kubernetes API server, if you are running this on github.com with the default action runners, ensure that your API server is accessible from GitHub's [IP ranges](#). Alternatively, it might be possible to run the action on a custom runner within your firewall (with access to the Tanzu Application Platform cluster).

### Permissions Required

These are the minimum permissions required on the Tanzu Build Service cluster:

```
```bash
ClusterRole
├─ kpack.io
│   └─ clusterbuilders verbs=[get]
Role (DEVELOPER NAMESPACE)
├─ ''
│   └─ pods verbs=[get watch list] ✓
│   └─ pods/log verbs=[get] ✓
├─ kpack.io
│   └─ builds verbs=[get watch list create delete] ✓
...
```
```

This example contains the minimum required permissions:

```
```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: DEVELOPER_NAMESPACE
---
apiVersion: v1
kind: ServiceAccount
metadata:
  namespace: DEVELOPER_NAMESPACE
  name: github-actions
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: github-actions
subjects:
  - kind: ServiceAccount
    namespace: DEVELOPER_NAMESPACE
    name: github-actions
roleRef:
  kind: ClusterRole
  name: github-actions
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
```
```



```

kind: RoleBinding
metadata:
  name: github-actions
  namespace: DEVELOPER_NAMESPACE
subjects:
  - kind: ServiceAccount
    namespace: DEVELOPER_NAMESPACE
    name: github-actions
roleRef:
  kind: Role
  name: github-actions
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: github-actions
rules:
  - apiGroups: ['kpack.io']
    resources:
      - clusterbuilders
    verbs: ['get']
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: github-actions
  namespace: DEVELOPER_NAMESPACE
rules:
  - apiGroups: ['']
    resources: ['pods']
    verbs: ['get', 'watch', 'list']
  - apiGroups: ['']
    resources: ['pods/log']
    verbs: ['get']
  - apiGroups: ['kpack.io']
    resources:
      - builds
    verbs: ['get', 'watch', 'list', 'create', 'delete']
...

```

To access the values on Google Kubernetes Engine (steps might vary on other IaaS providers):

```

```console
DEV_NAMESPACE=DEVELOPER_NAMESPACE
SECRET=$(kubectl get sa github-actions -oyaml -n $DEV_NAMESPACE | yq '.secrets[0].name')

CA_CERT=$(kubectl get secret $SECRET -oyaml -n $DEV_NAMESPACE | yq '.data."ca.crt"')
NAMESPACE=$(kubectl get secret $SECRET -oyaml -n $DEV_NAMESPACE | yq '.data.namespace | base64 -d')
TOKEN=$(kubectl get secret $SECRET -oyaml -n $DEV_NAMESPACE | yq '.data.token | base64 -d')
SERVER=$(kubectl config view --minify | yq '.clusters[0].cluster.server')
...

```

Create the required secrets on the repository through [GitHub.com](https://github.com) or through the [gh](#) CLI:

```

```bash
gh secret set CA_CERT --app actions --body "$CA_CERT"
gh secret set NAMESPACE --app actions --body "$NAMESPACE"
gh secret set TOKEN --app actions --body "$TOKEN"
gh secret set SERVER --app actions --body "$SERVER"
...

```

## Use the action

1. To use the action in a workflow, run the following YAML:

```
- uses: vmware-tanzu/build-image-action@v1-alpha
  with:
    ## Authorization
    # Host of the API server
    server: `${{ secrets.SERVER }}`
    # CA Certificate of the API Server
    ca_cert: `${{ secrets.CA_CERT }}`
    # Service Account token to access Kubernetes
    token: `${{ secrets.TOKEN }}`
    # _(required)_ The namespace to create the build resource in
    namespace: `${{ secrets.NAMESPACE }}`

    ## Image configuration
    # _(required)_ Destination for the built image
    # Example: gcr.io/<my-project>/<my-image>
    destination: ''
    # Optional list of build time environment variables
    env: ''
    # Name of the service account in the namespace, defaults to `default`
    serviceAccountName: ''
    # Name of the cluster builder to use, defaults to `default`
    clusterBuilder: ''
    # Max active time that the pod can run for in seconds, defaults to 3600
    timeout:
```

For example:

```
- name: Build Image
  id: build
  uses: vmware-tanzu/build-image-action@v1-alpha
  with:
    # Authorization
    server: ${ secrets.SERVER }
    token: ${ secrets.TOKEN }
    ca_cert: ${ secrets.CA_CERT }
    namespace: ${ secrets.NAMESPACE }
    # Image configuration
    destination: gcr.io/project-id/name-for-image
    serviceAccountName: my-sa-that-has-access-to-reg-credentials
    env: |
      BP_JAVA_VERSION=17
```

2. The previous step should output the full name, including the SHA of the built image. To use the output in a subsequent step:

```
- name: Do something with image
  run:
    echo "${ steps.build.outputs.name }"
```

## Debugging

To run this action in “debug” mode, add a secret called `ACTIONS_STEP_DEBUG` with the value set to `true` as documented in the [GitHub Action Docs](#).

## Overview of Tekton

Tekton is a cloud-native, open-source framework for creating CI/CD systems. It allows developers to build, test, and deploy across cloud providers and on-premises systems. For more information about Tekton, see the [Tekton documentation](#).

## Overview of Tekton

Tekton is a cloud-native, open-source framework for creating CI/CD systems. It allows developers to build, test, and deploy across cloud providers and on-premises systems. For more information about Tekton, see the [Tekton documentation](#).

## Install Tekton

This topic tells you how to install Tekton Pipelines from the Tanzu Application Platform package repository.



### Note

Follow the steps in this topic if you do not want to use a profile to install Tekton Pipelines. For more information about profiles, see [Components and installation profiles](#).

## Prerequisites

Before installing Tekton Pipelines, complete all [prerequisites](#) to install Tanzu Application Platform.

## Install Tekton Pipelines

To install Tekton Pipelines:

1. See the Tekton Pipelines package versions available to install by running:

```
tanzu package available list -n tap-install tekton.tanzu.vmware.com
```

For example:

```
$ tanzu package available list -n tap-install tekton.tanzu.vmware.com
\ Retrieving package versions for tekton.tanzu.vmware.com...
NAME                VERSION  RELEASED-AT
tekton.tanzu.vmware.com 0.30.0   2021-11-18 17:05:37Z
```

2. Install Tekton Pipelines by running:

```
tanzu package install tekton-pipelines -n tap-install -p tekton.tanzu.vmware.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 your installed packages](#). Otherwise, perform the steps in this section for each namespace where you create Tekton Pipelines.

Service accounts that run Tekton workloads need access to the image pull secrets for the Tanzu package. This includes the `default` service account in a namespace, which is created automatically but is not associated with any image pull secrets. Without these credentials, PipelineRuns fail with a timeout and the pods report that they cannot pull images.

To configure a namespace to use Tekton Pipelines:

1. Create an image pull secret in the current namespace and fill it from the `tap-registry` secret. For more information, see [Relocate images to a registry](#).
2. Create an empty secret, and annotate it as a target of the secretgen controller, by running:

```

kubectl create secret generic pull-secret --from-literal=.dockerconfigjson={} -
-type=kubernetes.io/dockerconfigjson
kubectl annotate secret pull-secret secretgen.carvel.dev/image-pull-secret=""

```

3. After you create a `pull-secret` secret in the same namespace as the service account, add the secret to the service account by running:

```

kubectl patch serviceaccount default -p '{"imagePullSecrets": [{"name": "pull-s
ecret"}]}'

```

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.