vmware by Broadcom

Tanzu Packages

Tanzu Packages latest

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

https://techdocs.broadcom.com/

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

Contents

Installing and Using VMware Tanzu Packages

VMware Tanzu packages are service add-ons for Tanzu products that distributed as Carvel packages to standardize and simplify how you deploy and manage popular services with Tanzu.

You can install packages to:

- Make services available to apps hosted on Kubernetes workload clusters, to help developers
- Install platform intelligence or other services, to help platform operators

You can use the Tanzu CLI to install and manage packages on workload clusters deployed by Tanzu Kubernetes Grid (TKG) or clusters that run Tanzu Application Platform (TAP).

Installing and Using VMware Tanzu Packages is applicable only to Tanzu Kubernetes Grid with management clusters. If you are using TKGS with vSphere Supervisor, see Installing Standard Packages on TKG Service Clusters.

The sections below list CLI-managed packages that you can deploy to workload clusters and the package repositories where they are published.

Auto-managed packages not covered here: TKG and other products also use Carvel packages internally as *auto-managed packages* that are installed and upgraded automatically. These auto-managed packages are not installed by the user and do not typically require user attention. This publication covers CLI-managed packages, not auto-managed packages.

To learn more about how auto-managed packages work in TKG, see Packages in *About Tanzu Kubernetes Grid*.

Overview

CLI-managed packages extend Kubernetes clusters created by TKG. After creating a cluster, you can install packages from the `tanzu-standard` package repository or from package repositories that you add to the cluster. TKG manages the lifecycle of the `tanzu-standard` package repository.

VMware supports the packages listed below when deployed to TKG clusters. Custom package configurations, modifications, and updates for non-TKG clusters is not supported.

Packages and Package Repositories

A *CLI-managed package* is an optional component of a Kubernetes cluster that you can install and manage with the Tanzu Command Line Interface (CLI). These packages are installed after cluster creation.

CLI-managed packages are distributed via package repositories. To install and manage package repositories and the packages they contain, you use the `tanzu package` plugin of the Tanzu CLI. For information about how to use the `tanzu package` plugin, see Install and Manage Packages.

Tanzu Standard Package Repository

The Tanzu Standard package repository contains CLI-managed packages that platform operators install for their own use or to provide services that platform operators and application developers can use.

For a list of packages that the Tanzu Standard repository contains and how to install them in clusters, see Tanzu Standard Repository Packages in this documentation.

In Tanzu Kubernetes Grid (TKG) clusters, you install and manage packages using the Tanzu CLI or Kubectl, as described in this publication.

To associate TKG versions with versions of the Tanzu Standard package repository that they work with, see TKG, Tanzu CLI Plugin, and Tanzu Standard Package Repo Versions in the About TKG documentation.

To install packages from the Tanzu Mission Control (TMC)console see Install a Package in the TMC documentation.

Tanzu Application Platform Repository

The Tanzu Application Platform (TAP) repository contains CLI-managed packages useful for developers, such as Tanzu Application Platform, Tanzu Build Service, Cloud Native Runtimes, Application Accelerator for VMware Tanzu, and Cartographer.

For a list of packages that the TAP repository contains and how to install them in workload clusters, see the topics that cover package installation under Install Tanzu Application Platform in the Tanzu Application Platform documentation. In particular, to install the packages that constitute TAP, see Install Tanzu Application Platform package and profiles.

Package Versions

Package version compatibility on workload clusters is based on their Kubernetes version, not the TKG version of their standalone management cluster.

You can run different versions of the CLI-managed packages in different workload clusters. In a workload cluster, you can run either the latest supported version of a CLI-managed package or the versions that are released with the last two TKG releases.

Applying Overlays to Packages

To customize CLI-managed packages, you can apply `ytt` overlays. For information about how to download and install `ytt`, see Install the Carvel Tools in the Tanzu Kubernetes Grid documentation.

Tanzu Standard Repository Package Support

VMware provides the following support for the optional packages that are provided in the VMware Tanzu Standard Repository:

- VMware provides installation and upgrade validation for the packages that are included in the optional VMware Tanzu Standard Repository when they are deployed on Tanzu Kubernetes Grid. This validation is limited to the installation and upgrade of the package but includes any available updates required to address CVEs. Any bug fixes, feature enhancements, and security fixes are provided in new package versions when they are available in the upstream package project.
- VMware does not provide Runtime Level Support for the components provided by the Tanzu Standard Repository. The debugging of configuration, performance related issues, or debugging and fixing the package itself is not provided by VMware.

Tanzu Packages as a Deployment Option

Tanzu packages are one of several options for deploying services in Tanzu:

Deployment option	Deploys to	How deployed	Documentation
Tanzu packages	Tanzu-managed K8s clusters	TMC, Tanzu CLI, Kubectl	See above
Software catalog	K8s clusters	Tanzu Platform	Managing your Software Catalog in VMware Tanzu Platform hub
Container images	K8s clusters	Helm CLI	VMware Tanzu Application Catalog
Ops Manager tiles	Tanzu Application Service	Tanzu Ops Manager	Adding and deleting products from Tanzu Operations Manager

Tanzu Standard Repository Release Notes

The Tanzu Standard repository contains services packaged in Carvel format that you can deploy to Tanzu Kubernetes Grid (TKG) clusters. Each datestamped repository version contains a set of package versions that are all compatible with the same Kubernetes versions.

For information about VMware support for these packages, see Tanzu Standard Repository Package Support.

As of v2.5, TKG does not support clusters on AWS or Azure. See the End of Support for TKG Management and Workload Clusters on AWS and Azure in the *Tanzu Kubernetes Grid v2.5 Release Notes*.

Tanzu Standard v2025.1.27

This version of the datestamped Tanzu Standard repo is compatible with Tanzu Kubernetes Grid v2.5.2 and contains updates to the Prometheus and Whereabouts packages. The v2024.8.21 standard package is still supported and is still the default package for TKG 2.5.2. However, you should install v2025.1.27 if you use Prometheus and Whereabouts on your clusters.

The package versions in this release are compatible with Kubernetes minor versions v1.30, v1.29, v1.28, v1.27, and v1.26.

- **prometheus**
  - v2.45.0+vmware.2-tkg.2 release notes
- **whereabouts**
  - v0.7.0+vmware.2-tkg.2 release notes

These new versions update the default tolerations for the Prometheus and Whereabouts node-exporter to `NoSchedule` and `NoExecute` respectively:

```
tolerations:
- effect: NoSchedule
  operator: Exists
- effect: NoExecute
  operator: Exists
```

These tolerations match those of the Multus daemonSet.

For instructions about how to update Prometheus and Whereabouts from the versions provided in v2024.8.21 to the new versions in v2025.1.27, see Install Prometheus in Workload Clusters Deployed by a Standalone Management Cluster and Install Multus with Whereabouts in Workload Clusters Deployed by a Standalone Management Cluster.

Tanzu Standard v2024.8.21

This version of the datestamped Tanzu Standard repo is compatible with Tanzu Kubernetes Grid v2.5.2. The package versions in this release are compatible with Kubernetes minor versions v1.30, v1.29, v1.28, v1.27, and v1.26. This release contains the following component versions:

- **cert-manager**
  - v1.12.10+vmware.2-tkg.2-cert-manager release notes
- **contour**
  - v1.29.1+vmware.1-tkg.1 release notes
  - v1.28.5+vmware.1-tkg.1
  - v1.27.4+vmware.1-tkg.1
- **external-dns**
  - v0.14.2+vmware.1-tkg.1 release notes
  - v0.13.6+vmware.2-tkg.1
  - v0.12.2+vmware.8-tkg.1
- **fluent-bit**
  - v2.1.6+vmware.2-tkg.1 release notes
- **grafana**
  - v10.0.1+vmware.2-tkg.1 release notes
- **prometheus**
  - v2.45.0+vmware.2-tkg.1 release notes
- **harbor**
  - v2.10.3+vmware.1-tkg.1 release notes
  - v2.9.1+vmware.1-tkg.1
- **multus-cni**
  - v4.0.1+vmware.2-tkg.3 release notes
- **whereabouts**
  - v0.7.0+vmware.2-tkg.1 release notes
- **fluxcd-source-controller**
  - v1.1.2+vmware.6-tkg.1 release notes
  - v0.24.4+vmware.3-tkg.1
  - v0.33.0+vmware.3-tkg.1
  - v0.24.4+vmware.2-tkg.3
- **fluxcd-kustomize-controller**
  - v1.1.1+vmware.2-tkg.1 release notes
  - v0.32.0+vmware.3-tkg.1
  - v0.24.4+vmware.3-tkg.1

- **fluxcd-helm-controller**
  - v0.36.2+vmware.2-tkg.1 release notes
  - v0.28.1+vmware.3-tkg.1
  - v0.36.2+vmware.2-tkg.1-tkg.5
- **external-csi-snapshot-webhook**
  - v6.1.0+vmware.1-tkg.7 release notes
- **cluster-autoscaler**
  - v1.30.0+vmware.2-tkg.1 release notes
  - v1.29.0+vmware.2-tkg.1
  - v1.28.0+vmware.1-tkg.2
- **vsphere-pv-csi-webhook**
  - v3.1.0+vmware.1-tkg.4 release notes

Tanzu Standard v2024.7.2

Component versions packaged in this datestamped Tanzu Standard repo:

- **cert-manager**
  - v1.12.10+vmware.1-tkg.1 release notes
- **contour**
  - v1.28.2+vmware.1-tkg.1 release notes
  - v1.27.1+vmware.1-tkg.1
  - v1.26.2+vmware.1-tkg.1
- **external-dns**
  - v0.13.6+vmware.1-tkg.1 release notes
  - v0.12.2+vmware.7-tkg.1
  - v0.11.0+vmware.1-tkg.8
  - v0.10.0+vmware.1-tkg.8
- **fluent-bit**
  - v2.2.3+vmware.1-tkg.1 release notes
- **grafana**
  - v10.0.1+vmware.1-tkg.2 release notes
- **prometheus**
  - v2.45.0+vmware.1-tkg.2 release notes
- **harbor**
  - v2.9.1+vmware.1-tkg.1 release notes
- **multus-cni**
  - v4.0.1+vmware.2-tkg.1 release notes

- **whereabouts**
  - v0.6.3+vmware.1-tkg.2 release notes
- **fluxcd-source-controller**
  - v1.1.2+vmware.5-tkg.1 release notes
  - v1.1.2+vmware.1-tkg.1
  - v0.36.1+vmware.2-tkg.2
  - v0.33.0+vmware.2-tkg.3
  - v0.24.4+vmware.2-tkg.3
- **fluxcd-kustomize-controller**
  - v1.1.1+vmware.1-tkg.1 release notes
  - v0.32.0+vmware.1-tkg.4
  - v0.24.4+vmware.1-tkg.5
- **fluxcd-helm-controller**
  - v0.36.2+vmware.1-tkg.1 release notes
  - v0.28.1+vmware.1-tkg.4
  - v0.21.0+vmware.1-tkg.5
- **external-csi-snapshot-webhook**
  - v6.1.0+vmware.1-tkg.6 release notes
- **cluster-autoscaler**
  - v1.30.0+vmware.1-tkg.1 release notes
- **vsphere-pv-csi-webhook**
  - v3.1.0+vmware.1-tkg.3 release notes

Tanzu Standard v2024.6.27

Component versions packaged in this datestamped Tanzu Standard repo:

- **cert-manager**
  - v1.12.10+vmware.1-tkg.1 release notes
- **contour**
  - v1.28.2+vmware.1-tkg.1 release notes
  - v1.27.1+vmware.1-tkg.1
  - v1.26.2+vmware.1-tkg.1
- **external-dns**
  - v0.13.6+vmware.1-tkg.1 release notes
  - v0.12.2+vmware.7-tkg.1
  - v0.11.0+vmware.1-tkg.8
  - v0.10.0+vmware.1-tkg.8

- **fluent-bit**
  - v2.1.6+vmware.1-tkg.2 release notes
- **grafana**
  - v10.0.1+vmware.1-tkg.2 release notes
- **prometheus**
  - v2.45.0+vmware.1-tkg.2 release notes
- **harbor**
  - v2.9.1+vmware.1-tkg.1 release notes
- **multus-cni**
  - v4.0.1+vmware.2-tkg.1 release notes
- **whereabouts**
  - v0.6.3+vmware.1-tkg.2 release notes
- **fluxcd-source-controller**
  - v1.1.2+vmware.5-tkg.1 release notes
  - v1.1.2+vmware.1-tkg.1
  - v0.36.1+vmware.2-tkg.2
  - v0.33.0+vmware.2-tkg.3
  - v0.24.4+vmware.2-tkg.3
- **fluxcd-kustomize-controller**
  - v1.1.1+vmware.1-tkg.1 release notes
  - v0.32.0+vmware.1-tkg.4
  - v0.24.4+vmware.1-tkg.5
- **fluxcd-helm-controller**
  - v0.36.2+vmware.1-tkg.1 release notes
  - v0.28.1+vmware.1-tkg.4
  - v0.21.0+vmware.1-tkg.5
- **external-csi-snapshot-webhook**
  - v6.1.0+vmware.1-tkg.6 release notes
- **cluster-autoscaler**
  - v1.29.0+vmware.1-tkg.1 release notes
- **vsphere-pv-csi-webhook**
  - v3.1.0+vmware.1-tkg.3 release notes

Tanzu Standard v2024.5.14

Component versions packaged in this datestamped Tanzu Standard repo:

- **cert-manager**
  - v1.12.10+vmware.1-tkg.1 release notes

- **contour**
  - v1.28.2+vmware.1-tkg.1 release notes
  - v1.27.1+vmware.1-tkg.1
  - v1.26.2+vmware.1-tkg.1
- **external-dns**
  - v0.13.6+vmware.1-tkg.1 release notes
  - v0.12.2+vmware.7-tkg.1
  - v0.11.0+vmware.1-tkg.8
  - v0.10.0+vmware.1-tkg.8
- **fluent-bit**
  - v2.1.6+vmware.1-tkg.2 release notes
- **grafana**
  - v10.0.1+vmware.1-tkg.2 release notes
- **prometheus**
  - v2.45.0+vmware.1-tkg.2 release notes
- **harbor**
  - v2.9.1+vmware.1-tkg.1 release notes
- **multus-cni**
  - v4.0.1+vmware.2-tkg.1 release notes
- **whereabouts**
  - v0.6.3+vmware.1-tkg.1 release notes
- **fluxcd-source-controller**
  - v1.1.2+vmware.5-tkg.1 release notes
  - v1.1.2+vmware.1-tkg.1
  - v0.36.1+vmware.2-tkg.2
  - v0.33.0+vmware.2-tkg.3
  - v0.24.4+vmware.2-tkg.3
- **fluxcd-kustomize-controller**
  - v1.1.1+vmware.1-tkg.1 release notes
  - v0.32.0+vmware.1-tkg.4
  - v0.24.4+vmware.1-tkg.5
- **fluxcd-helm-controller**
  - v0.36.2+vmware.1-tkg.1 release notes
  - v0.28.1+vmware.1-tkg.4
  - v0.21.0+vmware.1-tkg.5
- **external-csi-snapshot-webhook**

    - v6.1.0+vmware.1-tkg.5 release notes
- **cluster-autoscaler**
    - v1.28.0+vmware.1-tkg.1 release notes
- **vsphere-pv-csi-webhook**
    - v3.1.0+vmware.1-tkg.3 release notes

Tanzu Standard v2024.4.12

Supported TKr Versions

Package version compatibility on workload clusters is based on their Kubernetes version, not the TKG version of their standalone management cluster.

Package versions in the Tanzu Standard repository are compatible with Tanzu Kubernetes releases (TKrs) for Kubernetes minor versions v1.28, v1.27, and v1.26 as follows:

Package	Package Version	Kubernetes v1.28 TKrs	Kubernetes v1.27 TKrs	Kubernetes v1.26 TKrs
Cert Manager `cert-manager`	1.12.2+vmware.2-tkg.2	✔	✔	✔
Cluster Autoscaler `cluster-autoscaler`	1.27.2+vmware.1-tkg.3	✔	✔	✔
	1.26.2+vmware.1-tkg.3	✔	✔	✔
	1.25.1+vmware.1-tkg.3	✔	✔	✔
Contour `contour`	1.28.2+vmware.1-tkg.1	✔	✔	✔
	1.27.1+vmware.1-tkg.1	✔	✔	✔
	1.26.2+vmware.1-tkg.1	✔	✔	✔
External CSI Snapshot Validation Webhook `external-csi-snapshot-webhook`	6.1.0+vmware.1-tkg.5	✔	✔	✔
External DNS `external-dns`	0.13.6+vmware.1-tkg.1	✔	✔	✔
	0.12.2+vmware.7-tkg.1	✔	✔	✔
	0.11.0+vmware.1-tkg.8	✔	✔	✔
	0.10.0+vmware.1-tkg.8	✔	✔	✔
Fluent Bit `fluent-bit`	2.1.6+vmware.1-tkg.2	✔	✔	✔
FluxCD Helm Controller `fluxcd-helm-controller`	0.28.1+vmware.1-tkg.4	✔	✔	✔
	0.21.0+vmware.1-tkg.5	✔	✔	✔
FluxCD Kustomize Controller `fluxcd-source-controller`	0.32.0+vmware.1-tkg.4	✔	✔	✔
	0.24.4+vmware.1-tkg.5	✔	✔	✔

Package	Package Version	Kubernetes v1.28 TKrs	Kubernetes v1.27 TKrs	Kubernetes v1.26 TKrs
FluxCD Source Controller `fluxcd-source-controller`	0.36.1+vmware.2-tkg.2	✔	✔	✔
	0.33.0+vmware.2-tkg.3	✔	✔	✔
	0.24.4+vmware.2-tkg.3	✔	✔	✔
Grafana `grafana`	10.0.1+vmware.1-tkg.2	✔	✔	✔
Harbor `harbor`	2.9.1+vmware.1-tkg.1	✔	✔	✔
Multus CNI `multus-cni`	4.0.1+vmware.2-tkg.1	✔	✔	✔
Prometheus `prometheus`	2.45.0+vmware.1-tkg.2	✔	✔	✔
vSphere PV CSI Webhook `vsphere-pv-csi-webhook`	3.1.0+vmware.1-tkg.3	✔	✔	✔
Whereabouts `whereabouts`	0.6.3+vmware.1-tkg.1	✔	✔	✔
	0.5.4+vmware.1-tkg.1	✔	✔	✔
	0.5.1+vmware.2-tkg.1	✔	✔	✔

cert-manager v1.12.2

What's New

- See the cert-manager v1.12.2 release notes

Supported Versions

TKG version	jetstack_cert-manager version	vmware cert-manager package version	Kubernetes version compatibility
2.5, 2.4	v1.12.2	v1.12.2+vmware.2-tkg.2	1.21-1.27

Image Versions

cert-manager v1.12.2 contains following component image versions:

- quay.io/jetstack/cert-manager-cainjector:v1.12.2
- quay.io/jetstack/cert-manager-controller:v1.12.2
- quay.io/jetstack/cert-manager-webhook:v1.12.2
- quay.io/jetstack/cert-manager-acmesolver:v1.12.2

Deprecations

Cert Manager package versions 1.9.x and prior are not supported on TKrs released with TKG v2.4. See Supported Releases in the cert-manager documentation.

Cluster Autoscaler v1.27.2

What's New

- New package for Tanzu Standard v2024.4.12 package repository.
- Changelogs for versions included:
  - Cluster Autoscaler v1.27.2
  - Cluster Autoscaler v1.26.2
  - Cluster Autoscaler v1.25.1

Contour v1.28.2

What's New

- Update Envoy to v1.29.2
- Disable Envoy removing TE header

Included versions:

- Contour v1.28.2
  - See the release notes for v1.28.2:
    - CHANGELOG-v1.28.2.md
- Contour v1.27.1
  - See the release notes for v1.27.1:
    - CHANGELOG-v1.27.1.md
- Contour v1.26.2
  - See the release notes for v1.26.2:
    - CHANGELOG-v1.26.2.md

See the Contour Compatibility Matrix.

External DNS v0.13.6

What's New

- See the release notes for the following versions of External DNS included in the Tanzu Standard v2024.4.12 repository:
- v0.13.6
- v0.12.2
- v0.11.0
- v0.10.0

Fluent-bit v2.1.6

What's New

- No version change since Tanzu Standard repository v2023.11.21; see the Fluent Bit v2.1.6 release notes.

Image versions

The Fluent Bit package v2.1.6+vmware.1-tkg.2 contains following component image versions:

- fluent/fluent-bit:2.1.6

FluxCD controllers

What's New

- **Source controller**: `fluxcd-source-controller` version is unchanged since v2023.9.19, released with TKG v2.4.0; see the fluxcd-source-controller v0.36.1 release notes.
- **Helm controller** and **Kustomize controller**: `fluxcd-helm-controller` and `fluxcd-kustomize-controller` versions are unchanged since v2023.7.13, released with TKG v2.3.0.

Grafana v10.0.1

What's New

- No version change since Tanzu Standard repository v2023.11.21; see the Grafana v10.0.1 release notes
- See the `k8s-sidecar` v1.24.6 release notes

Image versions

Grafana package v10.0.1+vmware.1-tkg.2 contains following component image versions:

- grafana/grafana:10.0.1
- kiwigrid/k8s-sidecar:1.24.6

Harbor v2.9.1

What's New

- See Harbor v2.9.1 release notes.
- For changes, see the `git compare` for v2.8.4 and v2.9.1.

Component Versions

Harbor v2.9.1 contains the following component image versions:

- harbor-core:v2.9.1

- harbor-db:v2.9.1
- harbor-exporter:v2.9.1
- harbor-jobservice:v2.9.1
- harbor-portal:v2.9.1
- harbor-registryctl:v2.9.1
- registry-photon:v2.9.1
- notary-server-photon:v2.9.1
- notary-signer-photon:v2.9.1
- redis-photon:v2.9.1
- trivy-adapter-photon:v2.9.1

Multus-CNI v4.0.1

- No change; same version used since TKG v2.3.

Prometheus v2.45.0

What's New

- No version change since Tanzu Standard repository v2023.11.21; see the Prometheus v2.45.0 release notes

Component Versions

Prometheus v2.45.0 contains the following component image versions:

- prom/prometheus:v2.45.0
- prom/alertmanager:v0.25.0
- prom/pushgateway:v1.6.0
- jimmidyson/configmap-reload:v0.11.0
- bitnami/kube-state-metrics:2.9.2
- quay.io/prometheus/node-exporter:v1.6.0

Deprecations

- Prometheus package versions `v2.37.0+vmware.2-tkg.1` and prior are not supported on TKrs released with TKG v2.4. Those package versions used Pod Security Policies, which were removed in Kubernetes v1.25.

Whereabouts v0.6.3

What's New

- Add overlapping ranges check to network_name feature

- Denormalize IP name before checking if pod is alive
- Rechecking pending Pods
- For more details, see Whereabouts v0.6.3.

Tanzu Standard v2024.2.1

Supported TKr Versions

Package version compatibility on workload clusters is based on their Kubernetes version, not the TKG version of their standalone management cluster.

Package versions in the Tanzu Standard repository are compatible with Tanzu Kubernetes releases (TKrs) for Kubernetes minor versions v1.28, v1.27, and v1.26 as follows:

Package	Package Version	Kubernetes v1.28 TKrs	Kubernetes v1.27 TKrs	Kubernetes v1.26 TKrs
Cert Manager `cert-manager`	1.12.2+vmware.2-tkg.1	✔	✔	✔
Cluster Autoscaler `cluster-autoscaler`	1.27.2+vmware.1-tkg.3	✔	✔	✔
	1.26.2+vmware.1-tkg.3	✔	✔	✔
	1.25.1+vmware.1-tkg.3	✔	✔	✔
Contour `contour`	1.26.1+vmware.1-tkg.1	✔	✔	✔
	1.25.3+vmware.1-tkg.1	✔	✔	✔
	1.24.6+vmware.1-tkg.1	✔	✔	✔
External CSI Snapshot Validation Webhook `external-csi-snapshot-webhook`	6.1.0+vmware.1-tkg.3	✔	✔	✔
External DNS `external-dns`	0.13.6+vmware.1-tkg.1	✔	✔	✔
	0.12.2+vmware.7-tkg.1	✔	✔	✔
	0.11.0+vmware.1-tkg.8	✔	✔	✔
	0.10.0+vmware.1-tkg.8	✔	✔	✔
Fluent Bit `fluent-bit`	2.1.6+vmware.1-tkg.2	✔	✔	✔
FluxCD Helm Controller `fluxcd-helm-controller`	0.28.1+vmware.1-tkg.4	✔	✔	✔
	0.21.0+vmware.1-tkg.5	✔	✔	✔
FluxCD Kustomize Controller `fluxcd-source-controller`	0.32.0+vmware.1-tkg.4	✔	✔	✔
	0.24.4+vmware.1-tkg.5	✔	✔	✔

Package	Package Version	Kubernetes v1.28 TKrs	Kubernetes v1.27 TKrs	Kubernetes v1.26 TKrs
FluxCD Source Controller `fluxcd-source-controller`	0.36.1+vmware.2-tkg.2	✔	✔	✔
	0.33.0+vmware.2-tkg.3	✔	✔	✔
	0.24.4+vmware.2-tkg.3	✔	✔	✔
Grafana `grafana`	10.0.1+vmware.1-tkg.2	✔	✔	✔
Harbor `harbor`	2.9.1+vmware.1-tkg.1	✔	✔	✔
Multus CNI `multus-cni`	4.0.1+vmware.2-tkg.1	✔	✔	✔
Prometheus `prometheus`	2.45.0+vmware.1-tkg.2	✔	✔	✔
vSphere PV CSI Webhook `vsphere-pv-csi-webhook`	3.1.0+vmware.1-tkg.3	✔	✔	✔
Whereabouts `whereabouts`	0.6.1+vmware.3-tkg.1	✔	✔	✔
	0.5.4+vmware.1-tkg.1	✔	✔	✔
	0.5.1+vmware.2-tkg.1	✔	✔	✔

cert-manager v1.12.2

What's New

- See the cert-manager v1.12.2 release notes

Supported Versions

TKG version	jetstack_cert-manager version	vmware cert-manager package version	Kubernetes version compatibility
2.5, 2.4	v1.12.2	v1.12.2+vmware.2-tkg.2	1.21-1.27

Image Versions

cert-manager v1.11.1 contains following component image versions:

- quay.io/jetstack/cert-manager-cainjector:v1.12.2
- quay.io/jetstack/cert-manager-controller:v1.12.2
- quay.io/jetstack/cert-manager-webhook:v1.12.2
- quay.io/jetstack/cert-manager-acmesolver:v1.12.2

Deprecations

Cert Manager package versions 1.9.x and prior are not supported on TKrs released with TKG v2.4. See Supported Releases in the cert-manager documentation.

Cluster Autoscaler v1.27.2

What's New

- New package for Tanzu Standard v2024.2.1 package repository.
- Changelogs for versions included:
  - Cluster Autoscaler v1.27.2
  - Cluster Autoscaler v1.26.2
  - Cluster Autoscaler v1.25.1

Contour v1.26.1

What's New

- Support for Gateway API with TKG v2.5 as described in Gateway API support.

Included versions:

- Contour v1.26.1
  - See the release notes for v1.26.1:
    - CHANGELOG-v1.26.1.md
- Contour v1.25.3
  - See the release notes for v1.25.0-3:
    - CHANGELOG-v1.25.0.md
    - CHANGELOG-v1.25.1.md
    - CHANGELOG-v1.25.2.md
    - CHANGELOG-v1.25.3.md
    - The Carvel package now supports TLS being terminated by the load balancer rather than Envoy, and forwarding the unencrypted traffic to Envoy???s insecure listening port, by setting `envoy.service.loadBalancerTLSTermination: true`.
- Contour v1.24.6
  - See the release notes for v1.24.6:
    - CHANGELOG-v1.24.6.md

See the Contour Compatibility Matrix.

Deprecations

- The `envoy.service.loadBalancerIP` configuration field is deprecated in the Contour v1.25.2 package schema. Use cloud provider-specific `Service` annotations instead.

External DNS v0.13.6

What's New

- See the release notes for the following versions of External DNS included in the Tanzu Standard v2024.2.1 repository:
- v0.13.6
- v0.12.2
- v0.11.0
- v0.10.0

Fluent-bit v2.1.6

What's New

- No version change since Tanzu Standard repository v2023.11.21; see the Fluent Bit v2.1.6 release notes.

Image versions

The Fluent Bit package v2.1.6+vmware.1-tkg.2 contains following component image versions:

- fluent/fluent-bit:2.1.6

FluxCD controllers

What's New

- **Source controller**: `fluxcd-source-controller` version is unchanged since v2023.9.19, released with TKG v2.4.0; see the fluxcd-source-controller v0.36.1 release notes.
- **Helm controller** and **Kustomize controller**: `fluxcd-helm-controller` and `fluxcd-kustomize-controller` versions are unchanged since v2023.7.13, released with TKG v2.3.0.

Grafana v10.0.1

What's New

- No version change since Tanzu Standard repository v2023.11.21; see the Grafana v10.0.1 release notes
- See the `k8s-sidecar` v1.24.6 release notes

Image versions

Grafana package v10.0.1+vmware.1-tkg.2 contains following component image versions:

- grafana/grafana:10.0.1
- kiwigrid/k8s-sidecar:1.24.6

Harbor v2.9.1

What's New

- See Harbor v2.9.1 release notes.
- For changes, see the `git compare` for v2.8.4 and v2.9.1.

Component Versions

Harbor v2.9.1 contains the following component image versions:

- harbor-core:v2.9.1
- harbor-db:v2.9.1
- harbor-exporter:v2.9.1
- harbor-jobservice:v2.9.1
- harbor-portal:v2.9.1
- harbor-registryctl:v2.9.1
- registry-photon:v2.9.1
- notary-server-photon:v2.9.1
- notary-signer-photon:v2.9.1
- redis-photon:v2.9.1
- trivy-adapter-photon:v2.9.1

Multus-CNI v4.0.1

- No change; same version used since TKG v2.3.

Prometheus v2.45.0

What's New

- No version change since Tanzu Standard repository v2023.11.21; see the Prometheus v2.45.0 release notes

Component Versions

Prometheus v2.45.0 contains the following component image versions:

- prom/prometheus:v2.45.0
- prom/alertmanager:v0.25.0
- prom/pushgateway:v1.6.0
- jimmidyson/configmap-reload:v0.11.0
- bitnami/kube-state-metrics:2.9.2
- quay.io/prometheus/node-exporter:v1.6.0

Deprecations

- Prometheus package versions `v2.37.0+vmware.2-tkg.1` and prior are not supported on TKrs released with TKG v2.4. Those package versions used Pod Security Policies, which were removed in Kubernetes v1.25.

Whereabouts v0.6.1

What's New

- No change; same version used since TKG v2.3.0.

Tanzu Standard v2023.11.21

Supported TKr Versions

Package version compatibility on workload clusters is based on their Kubernetes version, not the TKG version of their standalone management cluster.

Package versions in the Tanzu Standard repository are compatible with Tanzu Kubernetes releases (TKrs) for Kubernetes minor versions v1.27, v1.26, and v1.25 as follows:

Package	Package Version	Kubernetes v1.27 TKrs	Kubernetes v1.26 TKrs	Kubernetes v1.25 TKrs
Cert Manager `cert-manager`	1.12.2+vmware.1-tkg.1	✔	✔	✔
	1.11.1+vmware.1-tkg.1	✔	✔	✔
Cluster Autoscaler `cluster-autoscaler`	1.27.2+vmware.1-tkg.3	✔	✔	✔
	1.26.2+vmware.1-tkg.2	✔	✔	✔
	1.25.1+vmware.1-tkg.2	✔	✔	✔
Contour `contour`	1.25.2+vmware.1-tkg.1	✔	✔	✔
	1.24.5+vmware.1	✔	✔	✔
External CSI Snapshot Validation Webhook `external-csi-snapshot-webhook`	6.1.0+vmware.1-tkg.3	✔	✔	✔
External DNS `external-dns`	0.13.4+vmware.2-tkg.2	✔	✔	✔
Fluent Bit `fluent-bit`	2.1.6+vmware.1-tkg.1	✔	✔	✔
FluxCD Helm Controller `fluxcd-helm-controller`	0.28.1+vmware.1-tkg.3	✔	✔	✔
FluxCD Kustomize Controller `fluxcd-source-controller`	0.32.0+vmware.1-tkg.3	✔	✔	✔

Package	Package Version	Kubernetes v1.27 TKrs	Kubernetes v1.26 TKrs	Kubernetes v1.25 TKrs
FluxCD Source Controller `fluxcd-source-controller`	0.36.1+vmware.2	✔	✔	✔
	0.33.0+vmware.2	✔	✔	✔
Grafana `grafana`	10.0.1+vmware.1-tkg.1	✔	✔	✔
Harbor `harbor`	2.8.4+vmware.1-tkg.1	✔	✔	✔
	2.8.2+vmware.2-tkg.1	✔	✔	✔
Multus CNI `multus-cni`	4.0.1+vmware.1-tkg.2	✔	✔	✔
	3.8.0+vmware.3-tkg.1	✔	✔	✔
Prometheus `prometheus`	2.45.0+vmware.1-tkg.1	✔	✔	✔
	2.37.0+vmware.3-tkg.1	✔	✔	✔
	2.36.2+vmware.1-tkg.1	✖	✖	✔
vSphere PV CSI Webhook `vsphere-pv-csi-webhook`		✔*	✔	✔*
Whereabouts `whereabouts`	0.6.1+vmware.2-tkg.2	✔	✔	✔
	0.5.4+vmware.2-tkg.1	✔	✔	✔

* Expected compatibility

cert-manager v1.12.2

What's New

- See the cert-manager v1.12.2 release notes

Supported Versions

TKG version	jetstack_cert-manager version	vmware cert-manager package version	Kubernetes version compatibility
2.4	v1.12.2	v1.12.2+vmware.1-tkg.1	1.21-1.27

Image Versions

cert-manager v1.11.1 contains following component image versions:

- quay.io/jetstack/cert-manager-cainjector:v1.12.2
- quay.io/jetstack/cert-manager-controller:v1.12.2
- quay.io/jetstack/cert-manager-webhook:v1.12.2
- quay.io/jetstack/cert-manager-acmesolver:v1.12.2

Deprecations

Cert Manager package versions 1.9.x and prior are not supported on TKrs released with TKG v2.4. See Supported Releases in the cert-manager documentation.

Contour v1.25.2

What's New

Included versions:

- Contour v1.25.2
  - See the release notes for v1.25.0-2:
    - CHANGELOG-v1.25.0.md
    - CHANGELOG-v1.25.1.md
    - CHANGELOG-v1.25.2.md
    - The Carvel package now supports TLS being terminated by the load balancer rather than Envoy, and forwarding the unencrypted traffic to Envoy???s insecure listening port, by setting `envoy.service.loadBalancerTLSTermination: true`.
- Contour v1.24.5
  - See the release notes for v1.24.5:
    - CHANGELOG-v1.24.5.md

Supported Versions

- Contour v1.25.2 is supported on Kubernetes v1.25-v1.27.
- Contour v1.24.5 is supported on Kubernetes v1.24-v1.26.

See the Contour Compatibility Matrix.

Deprecations

- The `envoy.service.loadBalancerIP` configuration field is deprecated in the Contour v1.25.2 package schema. Use cloud provider-specific `Service` annotations instead.

External DNS v0.13.4

What's New

- No change; same version used since TKG v2.3. See External DNS v0.13.4.

Fluent-bit v2.1.6

What's New

- See the Fluent Bit v2.1.6 release notes.

Image versions

The Fluent Bit package v2.1.6+vmware.1-tkg.1 contains following component image versions:

- fluent/fluent-bit:2.1.6

FluxCD controllers

What's New

- **Source controller**: `fluxcd-source-controller` version is unchanged since v2023.9.19, released with TKG v2.4.0; see the fluxcd-source-controller v0.36.1 release notes.
- **Helm controller** and **Kustomize controller**: `fluxcd-helm-controller` and `fluxcd-kustomize-controller` versions are unchanged since v2023.7.13, released with TKG v2.3.0.

See the following Fluxcd controller package release notes:

- **Fluxcd-helm-controller**
  - fluxcd-helm-controller v0.21.0 release notes
  - fluxcd-helm-controller v0.28.1 release notes
- **Fluxcd-kustomize-controller**
  - fluxcd-kustomize-controller v0.24.4 release notes
  - fluxcd-kustomize-controller v0.32.0 release notes
- **Fluxcd-source-controller**
  - fluxcd-source-controller v0.24.4 release notes
  - fluxcd-source-controller v0.33.0 release notes

Grafana v10.0.1

What's New

- See the Grafana v10.0.1 release notes
- See the `k8s-sidecar` v1.24.6 release notes

Image versions

Grafana package v10.0.1+vmware.1-tkg.2 contains following component image versions:

- grafana/grafana:10.0.1
- kiwigrid/k8s-sidecar:1.24.6

Harbor v2.8.4

What's New

- See Harbor v2.8.4 release notes and Harbor v2.8.0 release notes

- New for Harbor v2.8:
    - Supports OCI Distribution Spec v1.1.0-rc1
    - Supports sending webhook payloads with CloudEvents format
    - Enhanced Jobservice Dashboard
    - More new features; for all changes, see the `git compare` for v2.7.1 and v2.8.4.

Supported Versions

TKG version	Harbor version	Compatible Kubernetes versions
2.4.1, 2.4.0	2.8.4	v1.25.10, v1.26.5, v1.27.2

Component Versions

Harbor v2.8.4 contains the following component image versions:

- harbor-core:v2.8.4
- harbor-db:v2.8.4
- harbor-exporter:v2.8.4
- harbor-jobservice:v2.8.4
- harbor-portal:v2.8.4
- harbor-registryctl:v2.8.4
- registry-photon:v2.8.4
- notary-server-photon:v2.8.4
- notary-signer-photon:v2.8.4
- redis-photon:v2.8.4
- trivy-adapter-photon:v2.8.4

Multus-CNI v4.0.1

- No change; same version used since TKG v2.3. See Multus CNI v4.0.1.

Prometheus v2.45.0

What's New

- See the Prometheus v2.45.0 release notes

Component Versions

Prometheus v2.45.0 contains the following component image versions:

- prom/prometheus:v2.45.0

- prom/alertmanager:v0.25.0
- prom/pushgateway:v1.6.0
- jimmidyson/configmap-reload:v0.11.0
- bitnami/kube-state-metrics:2.9.2
- quay.io/prometheus/node-exporter:v1.6.0

Deprecations

- Prometheus package versions `v2.37.0+vmware.2-tkg.1` and prior are not supported on TKrs released with TKG v2.4. Those package versions used Pod Security Policies, which were removed in Kubernetes v1.25.

Whereabouts v0.6.1

What's New

- No change; same version used since TKG v2.3.0. See Whereabouts v0.6.1.

Tanzu Standard v2023.10.16

These release notes apply to packages in the Tanzu Standard v2023.10.16 package repository, which is released around the same date as and compatible with Tanzu Kubernetes Grid (TKG) v2.3.1 and Kubernetes v1.24-1.26.

Package version compatibility on workload clusters is based on their Kubernetes version, not the TKG version of their standalone management cluster.

For information about VMware support for these packages, see Tanzu Standard Repository Package Support.

Cert-manager v1.11.1

What's New

- See the cert-manager v1.11.1 release notes
- The `vmware_cert-manager` package includes the `acmesolver` component from upstream `jetstack_cert-manager`.

Supported Versions

TKG version	jetstack_cert-manager version	VMware cert-manager package version	Kubernetes version compatibility
v2.3.1, v2.3.0	v1.11.1	v1.11.1+vmware.1	v1.21-1.27

Component Versions

cert manager v1.11.1 contains following component image versions:

- quay.io/jetstack/cert-manager-cainjector:v1.11.1
- quay.io/jetstack/cert-manager-controller:v1.11.1
- quay.io/jetstack/cert-manager-webhook:v1.11.1
- quay.io/jetstack/cert-manager-acmesolver:v1.11.1

Deprecations

The following cert manager versions are deprecated in TKG v2.3:

- v1.5.3
- v1.7.2
- v1.10.2

Contour v1.24.5

What's New

- See the release notes for Contour v1.24.5:
  - *CHANGELOG-v1.24.5.md*
- You can configure Envoy to install as a Deployment with a specified number of replicas rather than as a DaemonSet (default), using data values like the following:

```
envoy:
  workload:
    type: Deployment
    replicas: 3
```

- You can specify resource requests or limits for each container within the Contour and Envoy workloads, using data values like the following:

```
contour:
  resources:
    contour:
      requests:
        cpu: 250m
        memory: 128Mi
      limits:
        cpu: 1

        memory: 512Mi
envoy:
  resources:
    envoy:
# include requests and limits as desired
    shutdown-manager:
# include requests and limits as desired
```

- `data-values` file configuration values are verified. Specifying an unsupported value in the data values results in an error.

Supported Versions

TKG version	contour version	VMware contour package version	Kubernetes version compatibility
v2.3.1	v1.24.5	v1.24.5+vmware.1-tkg.1	v1.24-1.26
v2.3.0	v1.24.4	v1.24.4+vmware.1-tkg.1	v1.24-1.26

Supported Kubernetes Versions

Contour v1.24.4 is supported on Kubernetes v1.24-v1.26. See the Contour Compatibility Matrix.

Deprecations

- All versions of Contour prior to v1.24.4 have been removed from Tanzu Standard repository v2023.7.13.
- Contour package `data-values` files no longer accept `null` values. For any configuration field with a value set to `null`, you should omit the value entirely.

External DNS v0.13.4

What's New

- See the External DNS v0.13.4 release notes
- New `createNamespace` configuration field. Set to `true` to create the namespace that `external-dns` components are installed in. If set to false, package components install into an existing namespace.

Supported Versions

TKG version	external-dns version	VMware external-dns package version	Kubernetes version compatibility
v2.3.1, v2.3.0	v0.13.4	v0.13.4+vmware.2-tkg.2	v1.24-1.26

Fluent-bit v2.1.2

What's New

- See the Fluent Bit v2.1.2 release notes.

Supported Versions

TKG version	fluent-bit version	VMware fluent-bit package version	Kubernetes version compatibility
v2.3.1, v2.3.0	v2.1.2	v2.1.2+vmware.2-tkg.1	v1.24-1.26

Limitations / Known Issues

- Does not support AWS credential environment variables in the fluent-bit package ConfigMap for accessing AWS S3.
    - AWS credentials support is planned for a future release.

Fluxcd controllers

What's New

See the following Fluxcd controller package release notes:

- **Fluxcd-helm-controller**
    - fluxcd-helm-controller v0.21.0 release notes
    - fluxcd-helm-controller v0.28.1 release notes
- **Fluxcd-kustomize-controller**
    - fluxcd-kustomize-controller v0.24.4 release notes
    - fluxcd-kustomize-controller v0.32.0 release notes
- **Fluxcd-source-controller**
    - fluxcd-source-controller v0.24.4 release notes
    - fluxcd-source-controller v0.33.0 release notes

Supported Versions

TKG version	FluxCD controller	VMware package version	Kubernetes version compatibility
v2.3.1, v2.3.0	fluxcd-helm-controller	v0.21.0+vmware.1-tkg.1	v1.24-1.26
v2.3.1, v2.3.0	fluxcd-kustomize-controller	v0.24.4+vmware.1-tkg.1	v1.24-1.26
v2.3.1, v2.3.0	fluxcd-source-controller	v0.33.0+vmware.2-tkg.1	v1.24-1.26

Grafana v9.5.1

What's New

- See the Grafana v9.5.1 release notes

Supported Versions

TKG version	grafana version	VMware grafana package version	Kubernetes version compatibility
v2.3.1	v9.5.1	v9.5.1+vmware.2-tkg.3	v1.24-1.26
v2.3.0	v9.5.1	v9.5.1+vmware.2-tkg.1	v1.24-1.26

Component Versions

Grafana v9.5.1 contains the following component image versions:

- grafana/grafana:9.5.1
- kiwigrid/k8s-sidecar:1.22.0

Harbor v2.8.4

What's New

- See Harbor v2.8.4 release notes
- Due to CVEs, TKG v2.3 compatibility with the following Harbor versions has been removed:
  - v2.2.3_vmware.1-tkg.1
  - v2.2.3_vmware.1-tkg.2
  - v2.3.3_vmware.1-tkg.1
  - v2.5.3_vmware.1-tkg.1
  - v2.7.1_vmware.1-tkg.1

Supported Versions

TKG version	harbor version	VMware harbor package version	Kubernetes version compatibility
v2.3.1	v2.8.4	v2.8.4+vmware.1-tkg.1	v1.24-1.26
v2.3.0	v2.8.2	v2.8.2+vmware.2-tkg.1	v1.24-1.26

Component Versions

Harbor v2.8.4 contains the following component image versions:

- harbor-core:v2.8.4
- harbor-db:v2.8.4
- harbor-exporter:v2.8.4
- harbor-jobservice:v2.8.4
- harbor-portal:v2.8.4
- harbor-registryctl:v2.8.4
- registry-photon:v2.8.4
- notary-server-photon:v2.8.4
- notary-signer-photon:v2.8.4
- redis-photon:v2.8.4
- trivy-adapter-photon:v2.8.4

Multus-CNI v4.0.1

What's New

- Introduces a thick-plugin deployment and architecture. See Multus-CNI v4.0.1 new feature
- Changes default values as follows:

```
namespace: kube-system
#! DaemonSet related configuration
daemonset:
  resources:
    limits:
      cpu: 100m
      memory: 50Mi
    requests:
      cpu: 100m
      memory: 50Mi
configmap:
  cniVersion: 0.3.1
  multusConfigFile: auto
```

Supported Versions

TKG version	multus-cni version	VMware multus-cni package version	Kubernetes version compatibility
v2.3.1, v2.3.0	v2.1.2	v4.0.1+vmware.1-tkg.2	v1.24-1.26

Prometheus v2.43.0

What's New

- See the Prometheus v2.43.0 release notes

Supported Versions

TKG version	prometheus version	VMware prometheus package version	Kubernetes version compatibility
v2.3.1	v2.43.0	v2.43.0+vmware.4-tkg.1	v1.24-1.26
v2.3.0	v2.43.0	v2.43.0+vmware.1-tkg.1	v1.24-1.26

Component Versions

Prometheus v2.43.0 contains the following component image versions:

- prom/prometheus:v2.43.0
- prom/alertmanager:v0.25.0
- prom/pushgateway:v1.5.1
- jimmidyson/configmap-reload:v0.8.0
- bitnami/kube-state-metrics:2.8.2
- quay.io/prometheus/node-exporter:v1.5.0

Whereabouts v0.6.1

What's New

- Supports `ipRanges` for configuring dual-stack IP assigning; see Example IPv6 Config in the Whereabouts repo README.

Supported Versions

TKG version	whereabouts version	VMware whereabouts package version	Kubernetes version compatibility
v2.3.1, v2.3.0	v0.6.1	v0.6.1+vmware.2-tkg.1	v1.24-1.26

Tanzu Standard v2023.9.19

These release notes apply to packages in the Tanzu Standard v2023.9.19 package repository, which is released around the same date as and compatible with Tanzu Kubernetes Grid (TKG) v2.4.0 and Kubernetes v1.25-1.27.

Package version compatibility on workload clusters is based on their Kubernetes version, not the TKG version of their standalone management cluster.

For information about VMware support for these packages, see Tanzu Standard Repository Package Support.

Supported TKr Versions

Package versions in the Tanzu Standard repository are compatible with Tanzu Kubernetes releases (TKrs) for Kubernetes minor versions v1.27, v1.26, and v1.25 as follows:

Package	Package Version	Kubernetes v1.27 TKrs	Kubernetes v1.26 TKrs	Kubernetes v1.25 TKrs
Cert Manager `cert-manager`	1.12.2+vmware.1-tkg.1	✔	✔	✔
	1.11.1+vmware.1-tkg.1	✔	✔	✔
Contour `contour`	1.25.2+vmware.1-tkg.1	✔	✔	✔
	1.24.5+vmware.1	✔	✔	✔
External DNS `external-dns`	0.13.4+vmware.2-tkg.2	✔	✔	✔
Fluent Bit `fluent-bit`	2.1.6+vmware.1-tkg.1	✔	✔	✔
FluxCD Helm Controller `fluxcd-helm-controller`	0.28.1+vmware.1-tkg.3	✔	✔	✔
FluxCD Kustomize Controller `fluxcd-source-controller`	0.32.0+vmware.1-tkg.3	✔	✔	✔

Package	Package Version	Kubernetes v1.27 TKrs	Kubernetes v1.26 TKrs	Kubernetes v1.25 TKrs
FluxCD Source Controller `fluxcd-source-controller`	0.36.1+vmware.2	✔	✔	✔
	0.33.0+vmware.2	✔	✔	✔
Grafana `grafana`	10.0.1+vmware.1-tkg.1	✔	✔	✔
Harbor `harbor`	2.8.4+vmware.1-tkg.1	✔	✔	✔
	2.8.2+vmware.2-tkg.1	✔	✔	✔
Multus CNI `multus-cni`	4.0.1+vmware.1-tkg.2	✔	✔	✔
	3.8.0+vmware.3-tkg.1	✔	✔	✔
Prometheus `prometheus`	2.45.0+vmware.1-tkg.1	✔	✔	✔
	2.37.0+vmware.3-tkg.1	✔	✔	✔
	2.36.2+vmware.1-tkg.1	✖	✖	✔
Snapshot Validation Webhook `snapshot-validation-webhook`		✔*	✔	✔*
vSphere PV CSI Webhook `vsphere-pv-csi-webhook`		✔*	✔	✔*
Whereabouts `whereabouts`	0.6.1+vmware.2-tkg.2	✔	✔	✔
	0.5.4+vmware.2-tkg.1	✔	✔	✔

* Expected compatibility

cert-manager v1.12.2

What's New

- See the cert-manager v1.12.2 release notes

Supported Versions

TKG version	jetstack_cert-manager version	vmware cert-manager package version	Kubernetes version compatibility
2.4	v1.12.2	v1.12.2+vmware.1-tkg.1	1.21-1.27

Image Versions

cert-manager v1.11.1 contains following component image versions:

- quay.io/jetstack/cert-manager-cainjector:v1.12.2
- quay.io/jetstack/cert-manager-controller:v1.12.2

- quay.io/jetstack/cert-manager-webhook:v1.12.2
- quay.io/jetstack/cert-manager-acmesolver:v1.12.2

Deprecations

Cert Manager package versions 1.9.x and prior are not supported on TKrs released with TKG v2.4.0. See Supported Releases in the cert-manager documentation.

Contour v1.25.2

What's New

Included versions:

- Contour v1.25.2
  - See the release notes for v1.25.0-2:
    - CHANGELOG-v1.25.0.md
    - CHANGELOG-v1.25.1.md
    - CHANGELOG-v1.25.2.md
    - The Carvel package now supports TLS being terminated by the load balancer rather than Envoy, and forwarding the unencrypted traffic to Envoy???s insecure listening port, by setting `envoy.service.loadBalancerTLSTermination: true`.
- Contour v1.24.5
  - See the release notes for v1.24.5:
    - CHANGELOG-v1.24.5.md

Supported Versions

- Contour v1.25.2 is supported on Kubernetes v1.25-v1.27.
- Contour v1.24.5 is supported on Kubernetes v1.24-v1.26.

See the Contour Compatibility Matrix.

Deprecations

- Contour 1.24.4 has been removed from Tanzu Standard repository v2023.9.19 due to CVEs.
- The `envoy.service.loadBalancerIP` configuration field is deprecated in the Contour v1.25.2 package schema. Use cloud provider-specific `Service` annotations instead.

External DNS v0.13.4

What's New

- No change; same version as used for TKG v2.3.0.

Fluent-bit v2.1.6

What's New

- See the Fluent Bit v2.1.6 release notes.

Image versions

The Fluent Bit package v2.1.6+vmware.1-tkg.1 contains following component image versions:

- fluent/fluent-bit:2.1.6

FluxCD controllers

What's New

- **Source controller**: Tanzu Standard package repository v2023.9.19 has a new version of `fluxcd-source-controller`; see the fluxcd-source-controller v0.36.1 release notes.
- **Helm controller** and **Kustomize controller**: `fluxcd-helm-controller` and `fluxcd-kustomize-controller` versions are unchanged since v2023.7.13, released with TKG v2.3.

Grafana v10.0.1

What's New

- See the Grafana v10.0.1 release notes
- See the `k8s-sidecar` v1.24.6 release notes

Image versions

Grafana package v10.0.1+vmware.1-tkg.2 contains following component image versions:

- grafana/grafana:10.0.1
- kiwigrid/k8s-sidecar:1.24.6

Harbor v2.8.4

What's New

- See Harbor v2.8.4 release notes and Harbor v2.8.0 release notes
- New for Harbor v2.8:
    - Supports OCI Distribution Spec v1.1.0-rc1
    - Supports sending webhook payloads with CloudEvents format
    - Enhanced Jobservice Dashboard
    - More new features; for all changes, see the `git compare` for v2.7.1 and v2.8.4.

Supported Versions

TKG version	Harbor version	Compatible Kubernetes versions
2.4.0	2.8.4	v1.25.10, v1.26.5, v1.27.2

Component Versions

Harbor v2.8.4 contains the following component image versions:

- harbor-core:v2.8.4
- harbor-db:v2.8.4
- harbor-exporter:v2.8.4
- harbor-jobservice:v2.8.4
- harbor-portal:v2.8.4
- harbor-registryctl:v2.8.4
- registry-photon:v2.8.4
- notary-server-photon:v2.8.4
- notary-signer-photon:v2.8.4
- redis-photon:v2.8.4
- trivy-adapter-photon:v2.8.4

Deprecations

The following Harbor package versions have been removed from Tanzu Standard repository v2023.9.19 due to CVEs:

- v2.2.3_vmware.1-tkg.1
- v2.2.3_vmware.1-tkg.2
- v2.3.3_vmware.1-tkg.1
- v2.5.3_vmware.1-tkg.1
- v2.7.1_vmware.1-tkg.1

Multus-CNI v4.0.1

- No change; same version as used for TKG v2.3.0.

Prometheus v2.45.0

What's New

- See the Prometheus v2.45.0 release notes

Component Versions

Prometheus v2.45.0 contains the following component image versions:

- prom/prometheus:v2.45.0
- prom/alertmanager:v0.25.0
- prom/pushgateway:v1.6.0
- jimmidyson/configmap-reload:v0.11.0
- bitnami/kube-state-metrics:2.9.2
- quay.io/prometheus/node-exporter:v1.6.0

Deprecations

- Prometheus package versions `v2.37.0+vmware.2-tkg.1` and prior are not supported on TKrs released with TKG v2.4.0. Those package versions used Pod Security Policies, which were removed in Kubernetes v1.25.

Whereabouts v0.6.1

What's New

- No change; same version as used for TKG v2.3.0. See Whereabouts v0.6.1.

Tanzu Standard Repository Packages

This topic lists packages in the Tanzu Standard package repository that you can install into Tanzu Kubernetes Grid (TKG) workload clusters to add capabilities to the clusters.

This documentation is applicable only to Tanzu Kubernetes Grid with management clusters. If you are using TKGS with vSphere Supervisor, see Installing Standard Packages on TKG Service Clusters.

Tanzu Standard Repository Contents

The table below lists packages in the `tanzu-standard` package repository that you can install in workload clusters after they have been prepared as described in Prepare to Install Tanzu Packages. On TKG with a standalone management cluster, you can also install some packages on a shared services cluster that serves multiple workload clusters.

The table also lists package dependencies that you must install first.

For information about how to use the `tanzu package` plugin, see Packages in *Installing and Managing Packages with the Tanzu CLI*.

For information about VMware support for these packages, see Tanzu Standard Repository Package Support.

Package	Function	Install with Standalone MC	Dependencies
Cert Manager `cert-manager`	Certificate management	Install Cert Manager (Standalone MC)*	Cert Manager is a prerequisite for many other Tanzu packages and should usually be the first package you install.
Contour `contour`	Container networking	Install Contour (Standalone MC)*	Requires `cert-manager`
External DNS `external-dns`	Container registry	Install External DNS (Standalone MC)*	Requires `cert-manager`. Requires `external-dns` if you want to create DNS records for Contour HTTPProxy resources.
Fluent bit `fluent-bit`	Log forwarding	Install Fluent Bit (Standalone MC)	None

FluxCD Controllers `flux-source-controller` `helm-controller` `kustomize-controller`	Continuous delivery (CD)	Flux installation	`helm-controller` and `kustomize-controller` require `flux-source-controller`
Grafana `grafana`	Monitoring	Install Grafana (Standalone MC)	Requires `cert-manager`, `contour`, and `prometheus`
Harbor `harbor`	Image registry	Install Harbor (Standalone MC)*	Requires `cert-manager` Requires `contour` or NSX ALB for ingress. Recommends `external-dns` on infrastructure platforms with load balancing, especially in production or other environments in which Harbor availability is important.
Multus CNI `multus-cni`	Container networking	Install Multus	Requires `cert-manager`
Prometheus `prometheus`	Monitoring	Install Prometheus (Standalone MC)	Requires `cert-manager`
Whereabouts `whereabouts`	Container networking	Install Multus with Whereabouts	Requires `multus-cni`

*With a standalone management cluster, Cert Manager, Contour, External DNS, and Harbor can be deployed to a shared services cluster as well as to workload clusters.

FluxCD Package Notes

FluxCD packages do not have dedicated pages in VMware documentation, but see FluxCD controllers for more information.

Each FluxCD package installs a Kubernetes controller:

- **Flux Source Controller** retrieves artifacts from external sources such as Git, OCI, Helm repositories, and S3-compatible buckets.
- **Flux Helm Controller** lets you declaratively manage Helm chart releases defined with Kubernetes manifests.
- **Flux Kustomize Controller** lets you assemble and run continuous delivery pipelines for infrastructure and workloads defined with Kubernetes manifests.

Tanzu CLI Package Commands

This topic describes Tanzu CLI commands for installing and managing Tanzu packages and package repositories.

The `tanzu package` CLI plugin is intended only for CLI-managed packages. Do not use the commands provided in this topic to install and manage auto-managed packages. Their lifecycle is managed automatically by Tanzu Kubernetes Grid. For more information about auto-managed packages, see Auto-Managed Packages.

Overview

Tanzu CLI Package commands fall into two categories:

- `tanzu package repository` commands operate on package repositories as described in Package Repository Commands below, for example to:
    - List all package repositories available for or added to the target cluster
    - Get the details of an available or added package repository
    - Add, update, or delete a package repository
- `tanzu package` commands operate on packages as described in Package Commands below, for example to:
    - List all packages available to or installed in the target cluster
    - Get the details of an available or installed package
    - Install, update, or delete a package

For more information about `tanzu package` commands, see tanzu package.

Package Namespaces and System Namespaces

You install package repositories and packages to the namespace of your choice. If you do not set the `--namespace` option, the Tanzu CLI targets the `default` namespace.

The components of the packaged services run in a system namespace that is separate from where packages themselves are deployed. For example, Contour and Envoy components run in the namespace `tanzu-system-ingress` and Harbor components run in `tanzu-system-registry`.

Do not install packages to any `tanzu-system-` namespace.

Preparing to Install CLI-Managed Packages

Before using the Tanzu CLI to install packages from the `standard` package repository in a Tanzu Kubernetes Grid (TKG) workload cluster:

1. Follow the instructions and links in Prepare to Install Tanzu Packages to install the Tanzu CLI, Kubectl, and `imgpkg` on a Linux workstation, and add the `tanzu-standard` package repository to the target workload cluster.
2. Review Tanzu Standard Repository Packages to learn about available packages.
3. Install the `cert-manager` package in the workload cluster first, before installing other packages. See Install Cert Manager for Certificate Management.

Package Repository Commands

The sections below describe how to list, add, update, and delete package repositories.

List Package Repositories

The `tanzu package repository list` command lists all package repositories that are available in the target cluster. This list includes package repositories that have been added to the target cluster by running the `tanzu package repository add` command. To add a package repository to your cluster, see Add a Package Repository below.

- To list package repositories across all namespaces in the target cluster, run:

  ```
  tanzu package repository list -A
  ```

- To list package repositories in a specific namespace, run:

  ```
  tanzu package repository list -n NAMESPACE
  ```

  Where `NAMESPACE` is the target namespace. The `--namespace`, or `-n`, option is required if you want to list package repositories in a namespace other than `default`.

  To list package repositories in the `default` namespace, you can also run:

  ```
  tanzu package repository list
  ```

Get the Details of a Package Repository

The `tanzu package repository get` command retrieves the details of a package repository.

To retrieve the details of a package repository, run:

```
tanzu package repository get REPOSITORY-NAME -n REPOSITORY-NAMESPACE
```

Where:

- `REPOSITORY-NAME` is the name of the package repository in the target cluster.
- `REPOSITORY-NAMESPACE` is the namespace of the package repository in the target cluster.

Add a Package Repository

The `tanzu package repository add` command adds a package repository.

To add a package repository to the target cluster, run:

```
tanzu package repository add REPOSITORY-NAME --url REPOSITORY-URL -n REPOSITORY-NAMESP
ACE
```

Where:

- `REPOSITORY-NAME` is a name you choose for the package repository.
- `REPOSITORY-URL` is the OCI registry URL of the package repository.
    - See List Package Repositories to obtain this value from the Tanzu CLI, or in Tanzu Mission Control see the **Addons** > **Repositories** list in the **Cluster** pane.
    - If you omit a package repository tag at the end of the URL, the system uses the tag from the latest Tanzu Kubernetes Grid release.
    - This URL cannot be under `projects.packages.broadcom.com/tce`.
- `REPOSITORY-NAMESPACE` is the target namespace for the package repository. If this option is not specified, the Tanzu CLI adds the package repository to the `default` namespace.

For example, to add the `standard` package repository, which contains CLI-managed packages included in Tanzu Kubernetes Grid, run the command below. The target `--namespace` for the `standard` package repository is `tkg-system`.

```
tanzu package repository add tanzu-standard --url projects.packages.broadcom.com/tkg/p
ackages/standard/repo:v2024.4.12 --namespace tkg-system
```

Update a Package Repository

The `tanzu package repository update` command updates a package repository with a new version published at a URL.

To update a package repository in the target cluster, run:

```
tanzu package repository update REPOSITORY-NAME --url REPOSITORY-URL -n REPOSITORY-NAM
ESPACE
```

Where:

- `REPOSITORY-NAME` is the name of the package repository in the cluster.
- `REPOSITORY-URL` is the new URL of the package repository. If you do not specify a package repository tag in the URL, the system uses the package repository tag from the latest Tanzu Kubernetes Grid release.
- `REPOSITORY-NAMESPACE` is the namespace of the package repository in the cluster.

Delete a Package Repository

The `tanzu package repository delete` command removes a package repository. To delete a package repository from the target cluster, run:

```
tanzu package repository delete REPOSITORY-NAME -n REPOSITORY-NAMESPACE
```

Where:

- `REPOSITORY-NAME` is the name of the package repository.
- `REPOSITORY-NAMESPACE` is the namespace of the package repository.

Package Commands

The sections below describe how to list, install, update, and delete packages.

List Available Packages

The `tanzu package available list` command lists all available packages and package versions.

List available packages:

- To list available packages across all namespaces in the target cluster, run:

  ```
  tanzu package available list -A
  ```

- To list available packages in a specific namespace, run:

  ```
  tanzu package available list -n NAMESPACE
  ```

  Where `NAMESPACE` is the namespace from which you want to retrieve the list of available packages. To list available packages in the `default` namespace, you can also run:

  ```
  tanzu package available list
  ```

List available package versions:

- To list available package versions for a package across all namespaces in the target cluster, run:

  ```
  tanzu package available list AVAILABLE-PACKAGE-NAME -A
  ```

  Where `AVAILABLE-PACKAGE-NAME` is the package name that you retrieved by running the `tanzu package available list` command.

- To list available package versions for a package in a specific namespace, run:

  ```
  tanzu package available list AVAILABLE-PACKAGE-NAME -n AVAILABLE-PACKAGE-NAMESP
  ACE
  ```

  Where:

  - `AVAILABLE-PACKAGE-NAME` is the package name that you retrieved by running the `tanzu package available list` command.
  - `AVAILABLE-PACKAGE-NAMESPACE` is the namespace of the available package.

  To list available package versions for a package in the `default` namespace, you can also run:

```
tanzu package available list AVAILABLE-PACKAGE-NAME
```

Get the Details of an Available Package

The `tanzu package available get` command retrieves the details of an available package.

To retrieve the details of an available package, run:

```
tanzu package available get AVAILABLE-PACKAGE-NAME -n AVAILABLE-PACKAGE-NAMESPACE
```

Or:

```
tanzu package available get AVAILABLE-PACKAGE-NAME/AVAILABLE-PACKAGE-VERSION -n AVAILA
BLE-PACKAGE-NAMESPACE
```

Where:

- `AVAILABLE-PACKAGE-NAME` is the name of the available package. You can retrieve this name by running the `tanzu package available list` command.
- `AVAILABLE-PACKAGE-VERSION` is the version of the available package. You can retrieve the list of available package versions by running the `tanzu package available list AVAILABLE-PACKAGE-NAME` command.
- `AVAILABLE-PACKAGE-NAMESPACE` is the namespace of the available package.

To retrieve the default configuration of an available package, use the `--default-values-file-output` flag of the `tanzu package available get` command:

```
tanzu package available get AVAILABLE-PACKAGE-NAME/AVAILABLE-PACKAGE-VERSION -n AVAILA
BLE-PACKAGE-NAMESPACE --default-values-file-output FILE-PATH
```

To retrieve the values schema for an available package, including the default value for each key, use the `--values-schema` flag of the `tanzu package available get` command. This retrieves the `valuesSchema` section from the `Package` Kubernetes API resource for the available package. You can set the output format, `-o`, for the values schema to `yaml`, `json`, or `table`.

```
tanzu package available get AVAILABLE-PACKAGE-NAME/AVAILABLE-PACKAGE-VERSION -n AVAILA
BLE-PACKAGE-NAMESPACE --values-schema
```

List Installed Packages

The `tanzu package installed list` command lists all packages that are currently installed in the target cluster.

To list installed packages across all namespaces in a cluster, run:

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

To list installed packages in a specific namespace, run:

```
tanzu package installed list -n NAMESPACE
```

Where `NAMESPACE` is the namespace from which you want to retrieve the list of installed packages.

Get the Details of an Installed Package

The `tanzu package installed get` command retrieves the details of an installed package.

To retrieve the details of an installed package, run:

```
tanzu package installed get INSTALLED-PACKAGE-NAME -n INSTALLED-PACKAGE-NAMESPACE
```

Where:

- `INSTALLED-PACKAGE-NAME` is the name of the installed package.
- `INSTALLED-PACKAGE-NAMESPACE` is the namespace in which the package is installed.

If you want to save the current configuration of an installed package to a file, run:

```
tanzu package installed get INSTALLED-PACKAGE-NAME -n INSTALLED-PACKAGE-NAMESPACE --va
lues-file-output FILE-PATH
```

Where `FILE-PATH` is the path to the file. For example, `values.yaml`.

Install a Package

The `tanzu package install` command installs a CLI-managed package.

To install a CLI-managed package in the target cluster:

1. If you have not already done so, add the package repository that contains the package. See Add a Package Repository above.
2. Note the name and the version of the package that you want to install. To see the package name and version, run the `tanzu package available list` command. See List Available Packages above.
3. Install the package:

   ```
   tanzu package install PACKAGE-NAME -p AVAILABLE-PACKAGE-NAME -v AVAILABLE-PACKA
   GE-VERSION --values-file PACKAGE-CONFIGURATION-FILE -n TARGET-NAMESPACE
   ```

   Where:

   - `PACKAGE-NAME` is a name you choose for the package.
   - `AVAILABLE-PACKAGE-NAME` is the package name you noted above.
   - `AVAILABLE-PACKAGE-VERSION` is the package version you noted above.
   - `PACKAGE-CONFIGURATION-FILE` is the configuration file that you prepared for the package. If this option is not specified, the package is installed with the default configuration.
   - `TARGET-NAMESPACE` is the namespace in which to install the package (`PackageInstall`), package app (`App`), and any other Kubernetes resources that describe the package. For example, the `my-packages` or `tanzu-cli-managed-packages` namespace.

- If the `-n` flag is not specified, the Tanzu CLI uses the `default` namespace. Do not install the contents of the package, such as pods and services, into this namespace. The namespace for the package contents is set separately, in the package configuration.
- The specified namespace must already exist, for example from running `kubectl create namespace my-packages`.

To see the full list of supported flags for this command, use the `--help` option.

4. To confirm that the package has been installed, run the `tanzu package installed list` command. See List Installed Packages above.

Update a Package

The `tanzu package installed update` command updates the version and configuration of a CLI-managed package.

Before updating the version of a CLI-managed package, you typically update its repository as described in Update a Package Repository.

To update both the version and configuration of a CLI-managed package, run:

```
tanzu package installed update INSTALLED-PACKAGE-NAME -v TARGET-PACKAGE-VERSION --valu
es-file PACKAGE-CONFIGURATION-FILE -n INSTALLED-PACKAGE-NAMESPACE
```

Where:

- `INSTALLED-PACKAGE-NAME` is the name that you chose for the package. To see the name of the package, you can run the `tanzu package installed list` command. For more information, see List Installed Packages.
- (Optional) `TARGET-PACKAGE-VERSION` is the version that you want to update the package to. Not required if you are updating the package configuration, not the version.
- (Optional) `PACKAGE-CONFIGURATION-FILE` is the path to the `.yaml` file that contains the updated package configuration. Not required if you are updating the package version, not the configuration. To retrieve the current configuration, see Get the Details of an Installed Package.
- `INSTALLED-PACKAGE-NAMESPACE` is the namespace in which the package is installed.

The `tanzu package installed delete` command deletes a CLI-managed package.

To delete a CLI-managed package from the target cluster, run:

```
tanzu package installed delete INSTALLED-PACKAGE-NAME -n INSTALLED-PACKAGE-NAMESPACE
```

Where:

- `INSTALLED-PACKAGE-NAME` is the name of the package that you want to delete. To see the name of the package, you can run the `tanzu package installed list` command. For more information, see List Installed Packages.
- `INSTALLED-PACKAGE-NAMESPACE` is the namespace in which the package is installed. If you do not specify a namespace, `default` namespace is used.

Prepare to Install Tanzu Packages

This topic explains how to prepare Tanzu Kubernetes Grid (TKG) workload clusters before installing Tanzu packages on them. Tanzu packages are packaged services that extend workload cluster functionality.

Before you can install Tanzu packages to a workload cluster, you need to:

1. Install the Carvel `imgpkg` tool on your workstation.
2. Add a package repository to the target cluster.

The sections below describe these steps.

Prerequisites

Before preparing to install Tanzu packages to a workload cluster as described below, you need:

- A Linux client or jump host on which the Tanzu CLI and Kubectl are installed. See Install the Tanzu CLI and Kubernetes CLI for Use with Standalone Management Clusters
- A running workload cluster as described in Creating Workload Clusters.
- The target workload cluster has sufficient storage to run the packages you install, as described in Storage Requirements for Tanzu Packages below.

Install Carvel `imgpkg`

The Carvel `imgpkg` command (https://carvel.dev/imgpkg/) lets you browse package repositories.

To install Carvel `imgpkg`:

1. Install `imgpkg`.

   ```
   wget -O- https://carvel.dev/install.sh > install.sh
   sudo bash install.sh
   ```

2. Verify installation:

   ```
   imgpkg version
   imgpkg version 0.37.1
   ```

Add the Package Repository to the Cluster

Before you can install Tanzu packages to a cluster, you need to add the package repository from which the cluster downloads the package images.

The commands below refer to the Tanzu Standard package repository, which is distributed by VMware and contains the packages described in this publication. The Tanzu Standard repository is public, so you do not need to log in.

To add the package repository to a TKG workload cluster:

1. Log in to the target cluster with both the Tanzu CLI and Kubectl.
2. List the repository versions:

   ```
   imgpkg tag list -i projects.packages.broadcom.com/tkg/packages/standard/repo
   ```

   This command returns the available TKG package repository versions, for example:

   ```
   Tags

   Name
   ...
   v2023.10.16
   v2023.11.21
   v2023.7.13
   v2023.7.13_update.1
   v2023.7.13_update.2
   v2023.7.31_update.1
   v2023.9.19
   v2023.9.19_update.1
   v2024.2.1
   v2024.2.1_tmc.1
   v2024.4.12
   v2024.4.19
   v2024.5.14
   v2024.5.16
   v2024.6.27
   v2024.7.11
   v2024.7.2
   v2024.8.21
   v2025.1.27

   44 tags

   Succeeded
   ```

3. Add the package repository as follows, depending on your TKG deployment option:

   Run the `tanzu package repository add` command as described in Add a Package Repository.

4. Use Tanzu CLI or Kubectl commands to confirm that the repository object is created:
   - Tanzu CLI:

     ```
     tanzu package repository list -A
     ```

     This command returns output similar to the following:

     ```
     NAMESPACE   NAME            SOURCE
     STATUS
     ```

```
tkg-system  tanzu-standard (imgpkg) projects.packages.broadcom.com/tkg/pa
ckages/standard/repo:v2025.1.27   Reconcile succeeded
```

- kubectl:

```
kubectl get packagerepositories -A
```

This command returns output similar to the following:

```
NAMESPACE    NAME              AGE   DESCRIPTION
tkg-system   tanzu-standard    24s   Reconcile succeeded
```

5. Use Tanzu CLI or Kubectl commands to list the packages in the repository:
   - Tanzu CLI:

```
tanzu package repository list -A
```

This command returns a list of all available packages and their versions in the repository

```
tkg-system   cert-manager.tanzu.vmware.com.1.7.2+vmware.3-tkg.1    cert-m
anager.tanzu.vmware.com    1.7.2+vmware.3-tkg.1    23h8m10s
tkg-system   cert-manager.tanzu.vmware.com.1.7.2+vmware.3-tkg.3    cert-m
anager.tanzu.vmware.com    1.7.2+vmware.3-tkg.3    23h8m10s
...
---
tkg-system   contour.tanzu.vmware.com.1.28.5+vmware.1-tkg.1    contour.ta
nzu.vmware.com   1.28.5+vmware.1-tkg.1   23h8m10s
tkg-system   contour.tanzu.vmware.com.1.29.1+vmware.1-tkg.1    contour.ta
nzu.vmware.com   1.29.1+vmware.1-tkg.1   23h8m10s
```

   - kubectl:

```
kubectl -n tkg-system get packages
```

This `kubectl` command returns a list of all available packages and their versions in the repository by querying the packages CRD. Refer to this list to determine which package versions you want to install.

```
kubectl -n tkg-system get packages
NAME                                                                PACK
AGEMETADATA NAME                              VERSION                 AGE
cert-manager.tanzu.vmware.com.1.1.0+vmware.1-tkg.2                    cert
-manager.tanzu.vmware.com                     1.1.0+vmware.1-tkg.2    4h39m1
3s
cert-manager.tanzu.vmware.com.1.1.0+vmware.2-tkg.1                    cert
-manager.tanzu.vmware.com                     1.1.0+vmware.2-tkg.1    4h39m1
3s
...
---
contour.tanzu.vmware.com.1.17.1+vmware.1-tkg.1                        cont
our.tanzu.vmware.com                          1.17.1+vmware.1-tkg.1   4h39m1
2s
contour.tanzu.vmware.com.1.17.2+vmware.1-tkg.2                        cont
our.tanzu.vmware.com                          1.17.2+vmware.1-tkg.2   4h39m1
```

```
2s
...
```

Storage Requirements for Tanzu Packages

The TKG workload cluster where you deploy a Tanzu package should be provisioned with a default storage class. Specifically, the Prometheus and Grafana packages require a default storage class.

Component	Tanzu Package	Default Storage Size
Grafana	Grafana	8 Gi
Prometheus Server	Prometheus	8 Gi
Alertmanager	Prometheus	8 Gi
Harbor	Harbor Registry	Varies by PVC

To adjust the storage limit for the vSphere Namespace where the TKG cluster is provisioned:

1. Using the vSphere Client, log in to vCenter Server.
2. Select the vSphere Namespace where the target Tanzu Kubernetes cluster is provisioned.
3. Select **Configure** > **Resource Limits**.
4. Click **Edit**.
5. Adjust the **Storage** limit so that it is larger than the total size of the persistent volume claims required for the Prometheus and Grafana extensions.

Next Step

Proceed with installing the Tanzu Packages, starting with Cert Manager. See Install Cert Manager for Certificate Management.

cert-manager: Certificate Management

This topic gives an overview of the Cert Manager package, which you can install in Tanzu Kubernetes Grid (TKG) workload clusters to provide certificate management services for the cluster.

Cert Manager provides certificate management for the TKG cluster.

Cert Manager is a prerequisite for many other Tanzu packages and is usually the first package you install. See Install Cert Manager in Workload Clusters Deployed by a Standalone Management Cluster

Cert Manager Components

The cert-manager package installs on the cluster the containers listed in the table. For more information, see https://cert-manager.io/. The containers are pulled from the VMware public registry specified in the Package Repository.

Container	Resource Type	Replicas	Description
cert-manager	Deployment	1	Controller for certificate and issuer resources
cert-manager-cainjector	Deployment	1	Controller to inject CA certificates into webhooks, API services and CRDs
cert-manager-webhook	Deployment	1	Webhook for validating, mutating, defaulting and converting cert-manager API resources
cert-manager-acmesolver	Pod	1 per ACME challenge, short-lived	Solver for an ACME challenge which gets deployed dynamically per challenge

Cert Manager Package Configuration Parameters

You can customize your cert-manager installation by editing the default values in the cert-manager package configuration file.

The table below contains information about the values that you can customize in the `cert-manager-data-values.yaml` file and how they can be used to modify the default installation of cert-manager when deployed into a workload cluster.

Parameter	Description	Type	Default
`issuers` *	An array of bootstrapped, self-signed `ClusterIssuer` to be created by the package installation	array of objects	`[]`
`kubernetes_distribution` *	The distribution of Kubernetes, used to determine if distribution-specific configurations need to be applied. Options are an empty string `""` and `openshift`. If running on an Openshift cluster, this must be set to openshift. When set to openshift, a Role and RoleBinding are created to associate cert-manager's components with the appropriate Openshift Security Context Constraint resource.	string	none

Parameter	Description	Type	Default
`kubernetes_version` *	The version of Kubernetes being used, for enabling version-specific behaviors. Accept any valid major.minor.patch version of Kubernetes. This field is optional. Currently only has effect when kubernetes_distribution is set to openshift.	semantic version string	none
`namespace`	The namespace where cert-manager pods run, distinct from where the packages are deployed. This is also known as cert-manager cluster resource namespace. ACME solver pods will run in the namespace of its certificate.	string	`cert-manager`

* new parameter in `cert-manager.tanzu.vmware.com/v1.12.10+vmware.2-tkg.2`.

Examples for `issuers`:

- Include a single, self-signed `ClusterIssuer`

```
issuers:
- name: my-self-signed
  self_signed: {}
```

- Include a single, self-signed `ClusterIssuer` with specific private key configuration

```
issuers:
- name: my-self-signed
  self_signed:
    private_key:
      algorithm: RSA
      size: 8192
      encoding: PKCS8
```

Install Cert Manager (Standalone Management Cluster)

This topic explains how to install cert-manager into a workload cluster in Tanzu Kubernetes Grid (TKG). cert-manager installs automatically in a standalone management cluster.

This documentation is applicable only to Tanzu Kubernetes Grid with management clusters. If you are using TKGS with vSphere Supervisor, see Installing Standard Packages on TKG Service Clusters.

This topic applies to workload clusters running on vSphere, Amazon Web Services (AWS), and Azure.

As of v2.5, TKG does not support clusters on AWS or Azure. See the End of Support for TKG Management and Workload Clusters on AWS and Azure in the *Tanzu Kubernetes Grid v2.5 Release Notes*.

Prepare the Workload Cluster for cert-manager Installation

To prepare the cluster:

1. Get the admin credentials of the workload cluster into which you want to deploy cert-manager. For example:

   ```
   tanzu cluster kubeconfig get my-cluster --admin
   ```

2. Set the context of kubectl to the cluster. For example:

   ```
   kubectl config use-context my-cluster-admin@my-cluster
   ```

Install Cert Manager

To install cert-manager:

1. If you are installing cert-manager to a single-node cluster as described in Single-Node Clusters on vSphere, patch the `cert-manager` package annotations to prevent a conflict between the `cert-manager` installed as a core package on single-node clusters and the `cert-manager` in the Tanzu `standard` repo:

   ```
   kubectl annotate --overwrite package cert-manager.tanzu.vmware.com.v1.12.10+vmw
   are.2-tkg.2 tkg.tanzu.vmware.com/package-repo='standard'
   ```

2. If the cluster does not have a package repository with the cert-manager package installed, such as the `tanzu-standard` repository, install one:

   ```
   tanzu package repository add PACKAGE-REPO-NAME --url PACKAGE-REPO-ENDPOINT --na
   mespace tkg-system
   ```

   Where:

   - `PACKAGE-REPO-NAME` is the name of the package repository, such as `tanzu-standard` or the name of a private image registry configured with `ADDITIONAL_IMAGE_REGISTRY` variables.
   - `PACKAGE-REPO-ENDPOINT` is the URL of the package repository.
     - For the TKG v2.5.2 release, the `tanzu-standard` URL is `projects.packages.broadcom.com/tkg/packages/standard/repo:v2025.1.27`. See List Package Repositories to obtain this value from the Tanzu CLI, or in Tanzu Mission Control see the **Addons** > **Repositories** list in the **Cluster** pane.

3. Confirm that the `cert-manager` package is available in your workload cluster:

   ```
   tanzu package available list -A
   ```

4. Retrieve the version of the available package:

   ```
   tanzu package available list cert-manager.tanzu.vmware.com -A
   ```

5. Install the cert-manager package:

   ```
   tanzu package install cert-manager --package cert-manager.tanzu.vmware.com --na
   mespace TARGET-NAMESPACE --version AVAILABLE-PACKAGE-VERSION
   ```

Where:

- `TARGET-NAMESPACE` is the namespace in which you want to install the cert-manager package. For example, the `my-packages` or `tanzu-cli-managed-packages` namespace.
  - If the `--namespace` flag is not specified, the Tanzu CLI installs the package in the `default` namespace.
  - The specified namespace must already exist, for example from running `kubectl create namespace my-packages`.
- `AVAILABLE-PACKAGE-VERSION` is the version that you retrieved above.

For example:

```
tanzu package install cert-manager --package cert-manager.tanzu.vmware.com --na
mespace my-packages --version v1.12.10+vmware.2-tkg.2
```

6. Confirm that the `cert-manager` package has been installed:

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

The `cert-manager` package and `cert-manager` app are installed in the namespace that you specify when running the `tanzu package install` command.

7. Confirm that the `cert-manager` app has been successfully reconciled in your `TARGET-NAMESPACE`. For example:

```
kubectl get apps -A
NAMESPACE     NAME              DESCRIPTION           SINCE-DEPLOY   AGE
my-packages   cert-manager      Reconcile succeeded   3m2s           3m12s
...
```

If the status is not `Reconcile Succeeded`, view the full status details of the `cert-manager` app. Viewing the full status can help you to troubleshoot the problem.

```
kubectl get app cert-manager --namespace TARGET-NAMESPACE -o yaml
```

Where `TARGET-NAMESPACE` is the namespace in which you installed the package. If troubleshooting does not help you solve the problem, you must uninstall the package before installing it again:

```
tanzu package installed delete cert-manager --namespace TARGET-NAMESPACE
```

8. Confirm that the `cert-manager-` pods are running:

```
kubectl get pods -A
```

The cert-manager pods and any other resources associated with the cert-manager component are created in the `cert-manager` namespace.

Contour: Ingress Control

This topic gives an overview of the Contour package, which you can install in Tanzu Kubernetes Grid (TKG) workload clusters to provide ingress control services for the cluster.

Contour is a Kubernetes ingress controller that uses the Envoy reverse HTTP proxy. Contour with Envoy is commonly used with other packages, such as External DNS, Prometheus, and Harbor.

The Contour package includes the Contour ingress controller and the Envoy reverse HTTP proxy. See Install Contour in Workload Clusters Deployed by a Standalone Management Cluster.

Contour Components

The Contour package installs on the cluster the two containers listed in the table. For more information, see https://projectcontour.io/. The containers are pulled from the VMware public registry specified in the Package Repository.

Container	Resource Type	Replicas	Description
Envoy	DaemonSet	one per node	High performance reverse proxy
Contour	Deployment	2	Management and configuration server for Envoy

Contour Data Values

Below is an example `contour-data-values.yaml`.

The only customization is that the Envoy service is of type LoadBalancer (the default is NodePort). This means that the Envoy service will be accessible from outside of the cluster for ingress.

```
infrastructure_provider: vsphere
namespace: tanzu-system-ingress
contour:
 configFileContents: {}
 useProxyProtocol: false
 replicas: 2
 pspNames: "vmware-system-restricted"
 logLevel: info
envoy:
 service:
   type: LoadBalancer
   annotations: {}
   externalTrafficPolicy: Cluster
   disableWait: false
 hostPorts:
   enable: true
   http: 80
   https: 443
 hostNetwork: false
```

```
 terminationGracePeriodSeconds: 300
 logLevel: info
certificates:
 duration: 8760h
 renewBefore: 360h
```

Contour Package Configuration Parameters

You can customize your configuration by editing the default values in the Contour package configuration file.

The table below contains information about the values that you can customize in the `contour-data-values.yaml` file and how they can be used to modify the default behavior of Contour when deployed into a workload cluster.

If you reconfigure your Contour settings after the initial deployment, you must follow the steps in Update a Running Contour Deployment to apply the new configuration to the cluster.

Parameter	Description	Type	Default
`infrastructure_provider`	The underlying infrastructure provider. Valid values are `vsphere`, `aws`, and `azure`.	string	`vsphere`
`kubernetes_distribution`	The distribution of Kubernetes, used to determine if distribution-specific configurations need to be applied. Options are empty and openshift. If running on an Openshift cluster, this must be set to openshift. When set to openshift, a Role and RoleBinding are created to associate Contour's controllers with the appropriate Openshift Security Context Constraint resource.	string	none
`kubernetes_version`	The version of Kubernetes being used, for enabling version-specific behaviors. Accept any valid major.minor.patch version of Kubernetes. This field is optional. Currently only has effect when kubernetes_distribution is set to openshift.	semantic version string	none
`namespace`	The namespace where Contour and Envoy pods run, distinct from where the packages are deployed.	string	`tanzu-system-ingress`
`registry_secret_names`	The names of the placeholder secrets that will contain registry credentials to pull the Contour and Envoy images.	array of strings	`["contour-reg-creds"]`
`contour.configFileContents`	The YAML contents of the Contour config file. For more information, see the Contour Config File Contents section below.	YAML	none
`contour.replicas`	How many Contour pod replicas to have.	integer	`2`
`contour.useProxyProtocol`	Whether to enable `PROXY` protocol for all Envoy listeners.	boolean	`false`
`contour.logLevel`	The Contour log level. Valid values are `info` and `debug`.	string	`info`
`contour.pspNames`	A comma-separated list of pod security policies (PSPs) to apply to the Contour pods.	comma-separated string	`vmware-system-restricted`

Parameter	Description	Type	Default
`contour.resources.contour.limits.cpu`	CPU limit to apply to the contour container in the contour deployment.	string	none
`contour.resources.contour.limits.memory`	Memory limit to apply to the contour container in the contour deployment.	string	none
`contour.resources.contour.requests.cpu`	CPU request to apply to the contour container in the contour deployment.	string	none
`contour.resources.contour.requests.memory`	Memory request to apply to the contour container in the contour deployment.	string	none
`envoy.workload.type`	The type of Kubernetes workload Envoy is deployed as. Options are `Deployment` or `DaemonSet`.	string	`DaemonSet`
`envoy.workload.replicas`	The number of Envoy replicas to deploy when `envoy.workload.type` is set to `Deployment`.	integer	`2`
`envoy.workload.resources.envoy.limits.cpu`	CPU limit to apply to the envoy container in the envoy workload.	string	none
`envoy.workload.resources.envoy.limits.memory`	Memory limit to apply to the envoy container in the envoy workload.	string	none
`envoy.workload.resources.envoy.requests.cpu`	CPU request to apply to the envoy container in the envoy workload.	string	none
`envoy.workload.resources.envoy.requests.memory`	Memory request to apply to the envoy container in the envoy workload.	string	none
`envoy.workload.resources.shutdownManager.limits.cpu`	CPU limit to apply to the shutdown-manager container in the envoy workload.	string	none
`envoy.workload.resources.shutdownManager.limits.memory`	Memory limit to apply to the shutdown-manager container in the envoy workload.	string	none
`envoy.workload.resources.shutdownManager.requests.cpu`	CPU request to apply to the shutdown-manager container in the envoy workload.	string	none
`envoy.workload.resources.shutdownManager.limits.memory`	Memory request to apply to the shutdown-manager container in the envoy workload.	string	none

Parameter	Description	Type	Default
envoy.service.type	The type of Kubernetes service to provision for Envoy. Valid values are LoadBalancer, NodePort, and ClusterIP. If not specified, a NodePort service will be used for vsphere and a LoadBalancer for all other infrastructure providers.	string	none
envoy.service.loadBalancerIP	The desired load balancer IP for Envoy service. This setting is ignored if envoy.service.type is not set to LoadBalancer. This field configures the Service.Spec.LoadBalancerIP field which is deprecated as of Kubernetes 1.24. Users are encouraged to use cloud-provider specific Service annotations instead.	IP address string	none
envoy.service.externalTrafficPolicy	The external traffic policy for the Envoy service. Valid values are Local and Cluster.	string	Local
envoy.service.annotations	Annotations to set on the Envoy service.	map of string to string	none
envoy.service.nodePorts.http	If envoy.service.type == NodePort, the node port number to expose Envoy's HTTP listener on. If not specified, a node port will be auto-assigned by Kubernetes. If loadBalancerTLSTermination is true, this value will be ignored, as the http port entry will be removed.	integer	none
envoy.service.nodePorts.https	If envoy.service.type == NodePort, the node port number to expose Envoy's HTTPS listener on. If not specified, a node port will be auto-assigned by Kubernetes.	integer	none
envoy.service.aws.LBType	If infrastructure_provider == aws, the type of AWS load balancer to use. Valid values are classic and nlb. If not using aws, this value is ignored.	string	classic
envoy.service.loadBalancerTLSTermination *	When true, forwards traffic from 443 on the LoadBalancer to 8080 on the Envoy pod when terminating TLS at the LoadBalancer. Removes the http port entry on the Envoy Service.	boolean	false
envoy.hostPorts.enable	Whether to enable host ports for the Envoy pods. If false, envoy.hostPorts.http and envoy.hostPorts.https are ignored.	boolean	true
envoy.hostPorts.http	If envoy.hostPorts.enable == true, the host port number to expose Envoy's HTTP listener on.	integer	80
envoy.hostPorts.https	If envoy.hostPorts.enable == true, the host port number to expose Envoy's HTTPS listener on.	integer	443
envoy.hostNetwork	Whether to enable host networking for the Envoy pods.	boolean	false
envoy.terminationGracePeriodSeconds	The termination grace period, in seconds, for the Envoy pods.	integer	300
envoy.logLevel	The Envoy log level. Valid values are trace, debug, info, warn, error, critical, and off.	string	info
envoy.pspNames	A comma-separated list of pod security policies (PSPs) to apply to the Envoy pods.	comma-separated string	none

Parameter	Description	Type	Default
`certificates.duration`	How long the certificates should be valid for. **Deprecated**: Use `caDuration` and `leafDuration` instead. For backwards compatibility, `duration` takes precedence over `caDuration` and `leafDuration`.	time.Duration string	`8760h`
`certificates.renewBefore`	How long before expiration the certificates should be renewed. **Deprecated**: Use `caRenewBefore` and `leafRenewBefore` instead. For backwards compatibility , `renewBefore` takes precedence over `caRenewBefore` and `leafRenewBefore`.	time.Duration string	`360h`
`certificates.caDuration` **	How long the CA certificate for securing communication between Contour and Envoy should be valid for.	time.Duration string	`8760h`
`certificates.caRenewBefore` **	How long before expiration of the CA certificate for securing communication between Contour and Envoy should be renewed.	time.Duration string	`720h`
`certificates.leafDuration` **	How long the leaf certificates for securing communication between Contour and Envoy should be valid for. The leaf certificates are the certificates signed by the CA certificate.	time.Duration string	`720h`
`certificates.leafRenewBefore` **	How long before expiration of the leaf certificates for securing communication between Contour and Envoy should be renewed. The leaf certificates are the certificates signed by the CA certificate. It is recommended to set this to a value that is at least the leaf duration minus the ca certificate renew before, so the leaf certificates can be rotated every CA renew cycle.	time.Duration string	`360h`

* new parameter in Contour v1.25.2 and above

** new parameter in Contour v1.28.2 and above

Contour Config File Contents

As described above, the package configuration field `contour.configFileContents` can be used to specify the desired content for the Contour config file. The Contour package will use the contents of the `contour.configFileContents` field to create a ConfigMap which is mounted into the Contour pods as a volume. The format and exhaustive list of options for this config file are provided in the open-source Contour documentation.

For example, to customize the Contour config file to require TLS 1.3, use a data values file like the following:

```
contour:
  configFileContents:
    tls:
      minimum-protocol-version: 1.3
      maximum-protocol-version: 1.3
```

Some of the commonly used Contour config file settings are described below for convenience:

Parameter	Description	Type	Default
`server.xds-server-type`	XDS Server type to use: Supported Values: contour or envoy	string	none
`tls.minimum-protocol-version`	Minimum TLS version that Contour will negotiate	string	`1.2`
`tls.fallback-certificate.name`	Name of secret containing fallback certificate for requests that don't match SNI defined for a vhost	string	none
`tls.fallback-certificate.namespace`	Namespace of secret containing fallback certificate	string	none
`tls.envoy-client-certificate.name`	Name of the secret to use as client certificate, private key for TLS connection to backend service	string	none
`tls.envoy-client-certificate.namespace`	Namespace of the secret to use as client certificate, private key for TLS connection to backend service	string	none
`leaderelection.configmap-name`	Name of configmap to be used for contour leaderelection	string	`leader-elect`
`leaderelection.configmap-namespace`	Namespace of contour leaderelection configmap	string	`tanzu-system-ingress`
`disablePermitInsecure`	Disables HTTPProxy permitInsecure field	boolean	`false`
`accesslog-format`	Access log format	string	`envoy`
`json-fields`	Fields that will be logged	array of strings	envoy package doc no ne
`default-http-versions`	HTTP versions that Contour should program Envoy to serve	array of strings	`["HTTP/1.1", "HTTP/2"]`
`timeouts.request-timeout`	The timeout for an entire request to be received from a client	time.Duration	none (timeout is disabled)
`timeouts.connection-idle-timeout`	The time to wait before terminating an idle connection	time.Duration	`60s`
`timeouts.stream-idle-timeout`	The time to wait before terminating an request or stream with no activity	time.Duration	`5m`
`timeouts.max-connection-duration`	The time to wait before terminating an connection irrespective of activity or not	time.Duration	none (timeout is disabled)
`timeouts.connection-shutdown-grace-period`	The time to wait between sending an initial and final GOAWAY	time.Duration	`5s`
`cluster.dns-lookup-family`	dns-lookup-family to use for upstream requests to externalName type services from an HTTPProxy route. Supported Values: `auto`, `v4`, `v6`	string	none
`debug`	Turn on contour debugging	boolean	`false`
`ingress-status-address`	The address to set on status of every Ingress resource	string	none

Gateway API support

Gateway API is a project maintained by SIG-NETWORK that represents next-generation APIs for service networking in Kubernetes. The API is comprised of a set of CRDs, intended to be expressive and extensible, with role-oriented interfaces. For more information, see API Overview in the *Kubernetes Gateway API* documentation.

In TKG 2.5.1, the Standard release channel Gateway API CRDs from version v0.8.1 are available in management and workload clusters.

Contour package version v1.26.1 and later supports configuring Contour to reconcile Gateway API resources as follows:

1. Define and create `GatewayClass` and `Gateway` objects for Contour:

```
kind: GatewayClass
apiVersion: gateway.networking.k8s.io/v1beta1
metadata:
  name: example
spec:
  controllerName: projectcontour.io/gateway-controller
```

```
kind: Gateway
apiVersion: gateway.networking.k8s.io/v1beta1
metadata:
  name: contour
  namespace: tanzu-system-ingress
spec:
  gatewayClassName: example
  listeners:
  - name: http
    protocol: HTTP
    port: 80
    allowedRoutes:
      namespaces:
        from: All
```

The Contour package sets up the Gateway API in static provisioning mode and does not support more than two listener ports, `80` for HTTP and `443` for HTTPS.

2. Update your Contour package `data-values.yaml` file as described in Update a Running Contour Deployment to add the new `Gateway` object and enable attaching routes to it. For example, with the `Gateway` object named `contour` as defined above:

```
contour:
  configFileContents:
    gateway:
      gatewayRef:
        name: contour
        namespace: tanzu-system-ingress
```

3. Run `kubectl describe` on the `Gateway` object until you confirm that:
    - Its status is listed as `Accepted: True` and `Programmed: True`

- It has an IP address

4. Define and create an `HTTPRoute` object with `parentRefs` set to attach to the `Gateway` object and `backendRefs` set to name the Gateway API:

```
apiVersion: gateway.networking.k8s.io/v1beta1
kind: HTTPRoute
metadata:
  name: example
  namespace: tanzu-system-ingress
spec:
  parentRefs:
  - name: contour
  rules:
  - matches:
    - path:
        type: PathPrefix
        value: /
    backendRefs:
    - name: backend
      port: 8080
```

5. Run `kubectl describe` on the `HTTPRoute` object until you confirm that its status is `Accepted: True`.
6. To test the API, run `curl` or point a browser to the `Gateway` address with the API name, for example `curl http://10.186.139.192:8080/backend`.

For how to use Gateway API for advanced routing and request/response modification, see HTTP routing in the *Kubernetes Gateway API* documentation.

Route Timeout for File Downloads

By default, Envoy has a 15-second timeout for backend services to return a response. If you are using Contour for file transfer, or for other services that are slow to respond, you may need to adjust this value.

To set a custom response timeout, configure your HTTPProxy like the following:

```
apiVersion: projectcontour.io/v1
kind: HTTPProxy
metadata:
  name: httpproxy-custom-response-timeout
spec:
  virtualhost:
    fqdn: timeout.bar.com
  routes:
  - conditions:
    - prefix: /
    services:
    - name: s1
      port: 80
    timeoutPolicy:
      response: 5m
```

If you are using an Ingress resource instead, you can add the `projectcontour.io/response-timeout` annotation like the following:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-custom-response-timeout
  annotations:
    projectcontour.io/response-timeout: 5m
spec:
  rules:
  - host: timeout.bar.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: s1
            port:
              number: 80
```

See the open-source Contour documentation for HTTPProxy response timeouts and Ingress annotations for more information.

xDS mTLS Certificate Rotation Settings

By default, the xDS communication between the Contour controller and Envoy instances is secured via mTLS. The Contour package generates `cert-manager` Certificate objects, and `cert-manager` creates and rotates Secret objects for each component.

Prior to Contour 1.28.2, the validity periods for the generated CA certificate and client certificates were configured by the Contour package fields `certificates.duration` and `certificates.renewBefore`. The default values set certificates as valid for 1 year and rotated 30 days before expiry.

As of Contour 1.28.2, these fields have been deprecated and replaced by individual configuration fields for CA certificate and client certificate validity: `certificates.caDuration`, `certificates.caRenewBefore`, `certificates.leafDuration`, and `certificates.leafRenewBefore`. By default the generated CA is valid for 1 year and rotated 30 days before expiration. Leaf certificates are valid for 30 days and rotated 15 days before expiration.

To configure these fields with different values from the defaults, it is recommended to set `leafRenewBefore` to a value that is at least the `leaf.Duration` minus the CA `certificates.caRenewBefore`. This means that Contour and Envoy client certificates will be rotated more frequently than the CA, and ensures that stale CA certificate data does not end up in Secrets supplied to Contour and Envoy, which can cause connection issues and dropped traffic when Envoy instances restart or attempt to connect to the Contour controller.

The deprecated fields take precedence, for backwards compatibility, but it is recommended to use the newer fields to provide more control over CA and leaf certificates.

Install Contour (Standalone Management Cluster)

This topic explains how to deploy Contour into a workload cluster in Tanzu Kubernetes Grid.

This documentation is applicable only to Tanzu Kubernetes Grid with management clusters. If you are using TKGS with vSphere Supervisor, see Installing Standard Packages on TKG Service Clusters.

Contour is a Kubernetes ingress controller that uses the Envoy edge and service proxy. Tanzu Kubernetes Grid includes signed binaries for Contour and Envoy, which you can deploy into workload clusters to provide ingress control services in those clusters.

You deploy Contour and Envoy directly into workload clusters. Deploying Contour is a prerequisite if you want to deploy the Prometheus, Grafana, and Harbor packages.

For general information about ingress control, see Ingress Controllers in the Kubernetes documentation.

Prerequisites

- A bootstrap machine with the following installed:
  - Tanzu CLI, Tanzu CLI plugins, and `kubectl`, as described in Install the Tanzu CLI and Kubernetes CLI for Use with Standalone Management Clusters.
  - yq v4.5 or later.
- You have deployed at least one workload cluster. For instructions, see Creating Workload Clusters.

In this release of Tanzu Kubernetes Grid, the provided implementation of Contour and Envoy assumes that you use self-signed certificates.

Prepare the Workload Cluster for Contour Deployment

To prepare the cluster:

1. Get the `admin` credentials of the workload cluster into which you want to deploy Contour. For example:

   ```
   tanzu cluster kubeconfig get my-cluster --admin
   ```

   In the example above, `my-cluster` is the name of the cluster.

2. Set the context of `kubectl` to the cluster. For example:

   ```
   kubectl config use-context my-cluster-admin@my-cluster
   ```

3. If the cluster does not have a package repository with the Contour package installed, such as the `tanzu-standard` repository, install one:

   ```
   tanzu package repository add PACKAGE-REPO-NAME --url PACKAGE-REPO-ENDPOINT --na
   mespace tkg-system
   ```

   Where:

- `PACKAGE-REPO-NAME` is the name of the package repository, such as `tanzu-standard` or the name of a private image registry configured with `ADDITIONAL_IMAGE_REGISTRY` variables.
- `PACKAGE-REPO-ENDPOINT` is the URL of the package repository.
  - For the TKG v2.5.2 release, the `tanzu-standard` URL is `projects.packages.broadcom.com/tkg/packages/standard/repo:v2025.1.27`. See List Package Repositories to obtain this value from the Tanzu CLI, or in Tanzu Mission Control see the **Addons** > **Repositories** list in the **Cluster** pane.

4. If you have not already done so, install cert-manager in the cluster. For instructions, see Install Cert Manager for Certificate Management.
5. Proceed to Deploy Contour into the Workload Cluster below.

Deploy Contour into the Workload Cluster

After you have set up the cluster, you must first create the configuration file that is used when you install the Contour package and then install the package.

As of v2.5, TKG does not support clusters on AWS or Azure. See the End of Support for TKG Management and Workload Clusters on AWS and Azure in the *Tanzu Kubernetes Grid v2.5 Release Notes*.

1. Create a configuration file for the Contour package by retrieving the default configuration of the package:

   ```
   tanzu package available get contour.tanzu.vmware.com/PACKAGE-VERSION --default-values-file-output FILE-PATH
   ```

   Where `PACKAGE-VERSION` is the version of the Contour package that you want to install and `FILE-PATH` is the location to which you want to save the configuration file, for example, `contour-data-values.yaml`.

2. Configure the following in the `contour-data-values.yaml` file:

   vSphere
   : This file configures the Contour package on vSphere.

   ```
   ---
   infrastructure_provider: vsphere
   namespace: tanzu-system-ingress
   contour:
    configFileContents: {}
    useProxyProtocol: false
    replicas: 2
    pspNames: "vmware-system-restricted"
    logLevel: info
   envoy:
    service:
      type: NodePort
   ```

```
    annotations: {}
    externalTrafficPolicy: Cluster
    disableWait: false
  hostPorts:
    enable: true
    http: 80
    https: 443
  hostNetwork: false
  terminationGracePeriodSeconds: 300
  logLevel: info
certificates:
  duration: 8760h
  renewBefore: 360h
```

AWS

This file configures the Contour package on AWS.

```
---
infrastructure_provider: aws
namespace: tanzu-system-ingress
contour:
  configFileContents: {}
  useProxyProtocol: false
  replicas: 2
  pspNames: "vmware-system-restricted"
  logLevel: info
envoy:
  service:
    type: LoadBalancer
    annotations: {}
    externalTrafficPolicy: Cluster
    aws:
      LBType: classic
    disableWait: false
  hostPorts:
    enable: true
    http: 80
    https: 443
  hostNetwork: false
  terminationGracePeriodSeconds: 300
  logLevel: info
certificates:
  duration: 8760h
  renewBefore: 360h
```

Azure

This file configures the Contour package on Azure.

```
---
infrastructure_provider: azure
namespace: tanzu-system-ingress
contour:
  configFileContents: {}
  useProxyProtocol: false
  replicas: 2
  pspNames: "vmware-system-restricted"
  logLevel: info
envoy:
```

```
 service:
   type: LoadBalancer
   annotations: {}
   externalTrafficPolicy: Cluster
   disableWait: false
 hostPorts:
   enable: true
   http: 80
   https: 443
 hostNetwork: false
 terminationGracePeriodSeconds: 300
 logLevel: info
certificates:
 duration: 8760h
 renewBefore: 360h
```

TKG v2.5 does not support clusters on AWS or Azure. See the End of Support for TKG Management and Workload Clusters on AWS and Azure in the *Tanzu Kubernetes Grid v2.5 Release Notes*.

1. If you are installing Contour to a vSphere cluster that uses NSX ALB as a load balancer service provider, modify the `contour-default-values.yaml` file to set `envoy.service.type` to `LoadBalancer`:

   ```
   [...]
   envoy:
    service:
      type: LoadBalancer
   ```

2. If you are installing Contour to an internet-restricted AWS environment, modify the `contour-data-values.yaml` file to add the following annotation to the Envoy service:

   ```
   infrastructure_provider: aws
   [...]
   envoy:
    service:
      annotations:
        service.beta.kubernetes.io/aws-load-balancer-internal: "true"
   ```

3. (Optional) Modify the `contour-data-values.yaml` file if needed.

   See Contour Configuration Parameters for a full list of available parameters.

   For example, the Contour package deploys two Contour replicas by default, but the number of replicas is configurable. You set this number in the `contour.replicas` value in `contour-data-values.yaml`. In most cases, you do not need to modify the `contour-data-values.yaml` file.

   You can also retrieve these values by running the below command against your target cluster:

   ```
   tanzu package available get contour.tanzu.vmware.com/AVAILABLE-VERSION --values
   -schema
   ```

Where `AVAILABLE-VERSION` is the version of the Contour package. The `--values-schema` flag retrieves the `valuesSchema` section from the `Package` API resource for the Contour package. You can set the output format, `--output`, for the values schema to `yaml`, `json`, or `table`. For more information, see Packages in *Install and Manage Packages*.

For example:

```
tanzu package available get contour.tanzu.vmware.com/v1.29.1+vmware.1-tkg.1 --v
alues-schema
```

4. If your `contour-data-values.yaml` file contains comments, remove them:

   ```
   yq -i eval '... comments=""' contour-data-values.yaml
   ```

5. Install the Contour package:

   1. Retrieve the name of the available package:

      ```
      tanzu package available list -A
      ```

   2. Retrieve the version of the available package:

      ```
      tanzu package available list contour.tanzu.vmware.com -A
      ```

   3. Install the package:

      ```
      tanzu package install contour \
      --package contour.tanzu.vmware.com \
      --version AVAILABLE-PACKAGE-VERSION \
      --values-file contour-data-values.yaml \
      --namespace TARGET-NAMESPACE
      ```

      Where:

      - `TARGET-NAMESPACE` is the namespace in which you want to install the Contour package. For example, the `my-packages` or `tanzu-cli-managed-packages` namespace.
        - If the `--namespace` flag is not specified, the Tanzu CLI uses the `default` namespace. The Contour and Envoy pods and any other resources associated with the Contour component are created in the `tanzu-system-ingress` namespace; do not install the Contour package into this namespace.
        - The specified namespace must already exist, for example from running `kubectl create namespace my-packages`.
      - `AVAILABLE-PACKAGE-VERSION` is the version that you retrieved above.

      For example:

      ```
      tanzu package install contour \
      --package contour.tanzu.vmware.com \
      --version v1.29.1+vmware.1-tkg.1 \
      ```

```
--values-file contour-data-values.yaml \
--namespace my-packages
```

6. Confirm that the `contour` package has been installed:

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

   For example:

   ```
   tanzu package installed list -A
   - Retrieving installed packages...
     NAME            PACKAGE-NAME                      PACKAGE-VERSION
   STATUS               NAMESPACE
     cert-manager    cert-manager.tanzu.vmware.com    1.12.2+vmware.1-tkg.1
   Reconcile succeeded  my-packages
     contour         contour.tanzu.vmware.com         v1.29.1+vmware.1-tkg.1
   Reconcile succeeded  my-packages
     antrea          antrea.tanzu.vmware.com
   Reconcile succeeded  tkg-system
     [...]
   ```

   To see more details about the package, you can also run:

   ```
   tanzu package installed get contour --namespace PACKAGE-NAMESPACE
   ```

   Where `PACKAGE-NAMESPACE` is the namespace in which the `contour` package is installed.

   For example:

   ```
   tanzu package installed get contour --namespace my-packages
   \ Retrieving installation details for contour...
   NAME:                    contour
   PACKAGE-NAME:            contour.tanzu.vmware.com
   PACKAGE-VERSION:         v1.29.1+vmware.1-tkg.1
   STATUS:                  Reconcile succeeded
   CONDITIONS:              [{ReconcileSucceeded True  }]
   USEFUL-ERROR-MESSAGE:
   ```

7. Confirm that the `contour` app has been successfully reconciled in your `PACKAGE-NAMESPACE`:

   ```
   kubectl get apps -A
   ```

   For example:

   ```
   NAMESPACE     NAME            DESCRIPTION           SINCE-DEPLOY   AGE
   my-packages   cert-manager    Reconcile succeeded   78s            3h5m
   my-packages   contour         Reconcile succeeded   57s            6m3s
   tkg-system    antrea          Reconcile succeeded   45s            3h18m
   [...]
   ```

   If the status is not `Reconcile Succeeded`, view the full status details of the `contour` app. Viewing the full status can help you troubleshoot the problem.

   ```
   kubectl get app contour --namespace PACKAGE-NAMESPACE -o yaml
   ```

Where `PACKAGE-NAMESPACE` is the namespace in which you installed the package. If troubleshooting does not help you solve the problem, you must uninstall the package before installing it again:

```
tanzu package installed delete contour --namespace PACKAGE-NAMESPACE
```

8. Confirm that Contour and Envoy pods are running in the `tanzu-system-ingress` namespace:

   ```
   kubectl get pods -A
   ```

   For example:

   ```
   kubectl get pods -A
   NAMESPACE              NAME
   READY   STATUS    RESTARTS   AGE
   [...]
   tanzu-system-ingress   contour-5dc6fc667c-c4w8k
   1/1     Running   0          14m
   tanzu-system-ingress   contour-5dc6fc667c-jnqwn
   1/1     Running   0          14m
   tanzu-system-ingress   envoy-mgfll
   2/2     Running   0          14m
   [...]
   ```

9. If you deployed Contour to AWS or Azure, confirm that a load balancer has been created for the Envoy service:

   ```
   kubectl get svc envoy -n tanzu-system-ingress -o jsonpath='{.status.loadBalance
   r.ingress[0].hostname}'
   ```

   On AWS, the loadbalancer has a name similar to `aabaaad4dfc8e4a808a70a7cbf7d9249-1201421080.us-west-2.elb.amazonaws.com`. On Azure, it will be an IP address similar to `20.54.226.44`.

Access the Envoy Administration Interface Remotely

After you have deployed Contour into a cluster, you can use the embedded Envoy administration interface to retrieve data about your deployments.

For information about the Envoy administration interface, see Administration Interface in the Envoy documentation.

1. Obtain the name of the Envoy pod:

   ```
   ENVOY_POD=$(kubectl -n tanzu-system-ingress get pod -l app=envoy -o name | head
   -1)
   ```

2. Forward the Envoy pod to port 9001 on your bootstrap machine:

   ```
   kubectl -n tanzu-system-ingress port-forward $ENVOY_POD 9001
   ```

3. From your bootstrap machine, retrieve information from your Contour deployment by sending `curl` queries to the Envoy administration endpoints listed in Accessing the Envoy Administration

Interface. For example, use the `/config_dump` endpoint to retrieve the currently loaded configuration:

```
curl http://localhost:9001/config_dump
```

Visualize the Internal Contour Directed Acyclic Graph (DAG)

When you have started running workloads in your cluster, you can visualize the traffic information that Contour exposes in the form of a directed acyclic graph (DAG).

1. Install Graphviz if it is not already installed. This package provides the `dot` command that creates the DAG image file.
2. Obtain the name of a Contour pod:

   ```
   CONTOUR_POD=$(kubectl -n tanzu-system-ingress get pod -l app=contour -o name |
   head -1)
   ```

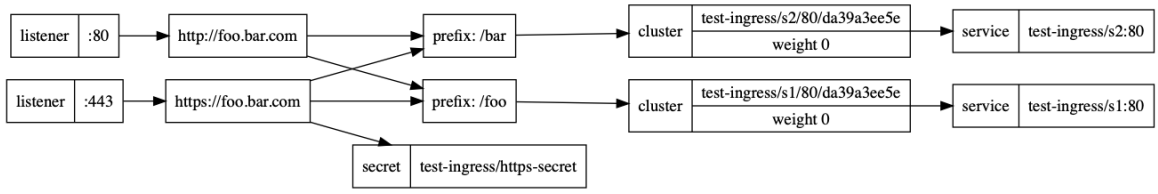
3. Forward port 6060 on the Contour pod:

   ```
   kubectl -n tanzu-system-ingress port-forward $CONTOUR_POD 6060
   ```

4. Open a new terminal window and download and save the DAG as a `*.png` file. The below command requires you to install `dot` on your system if it is not present already.

   ```
   curl localhost:6060/debug/dag | dot -T png > contour-dag.png
   ```

5. Open `contour-dag.png` to view the graph.



Update a Running Contour Deployment

If you need to make changes to the configuration of the Contour package after deployment, follow the steps below to update your deployed Contour package:

1. Update the Contour configuration in the `contour-data-values.yaml` file. For example, you can change the number of Contour replicas by setting `contour.replicas` to a new value.

   See Contour Configuration Parameters for a full list of available parameters.

2. Update the installed package:

   ```
   tanzu package installed update contour \
   --version INSTALLED-PACKAGE-VERSION \
   --values-file contour-data-values.yaml \
   --namespace INSTALLED-PACKAGE-NAMESPACE
   ```

Where:

- `INSTALLED-PACKAGE-VERSION` is the version of the installed Contour package.
- `INSTALLED-PACKAGE-NAMESPACE` is the namespace in which the Contour package is installed.

For example:

```
tanzu package installed update contour \
--version v1.29.1+vmware.1-tkg.1 \
--values-file contour-data-values.yaml \
--namespace my-packages
```

The Contour package will be reconciled using the new value or values that you added. It can take up to five minutes for `kapp-controller` to apply the changes.

For more information about the `tanzu package installed update` command, see Update a Package in *Install and Manage Packages*. You can use this command to update the version or the configuration of an installed package.

External DNS: Service Discovery

This topic gives an overview of the External DNS package, which you can install in Tanzu Kubernetes Grid (TKG) workload clusters to provide service discovery services for the cluster.

External DNS allows for DNS records to be created automatically for Kubernetes services with an ingress component such as Contour with Envoy. The External DNS package is validated with the following DNS providers: AWS (Route 53), Azure DNS, and RFC2136-compliant DNS servers (such as BIND).

You can use ExternalDNS to expose Kubernetes services for DNS lookup. You need to interface the ExternalDNS component with a supported DNS provider. The ExternalDNS package has been validated with the following providers:

- AWS (Route 53)
- Azure DNS
- RFC2136 (BIND)

See Install External DNS in Workload Clusters Deployed by a Standalone Management Cluster.

As of v2.5, TKG does not support clusters on AWS or Azure. See the End of Support for TKG Management and Workload Clusters on AWS and Azure in the *Tanzu Kubernetes Grid v2.5 Release Notes*.

External DNS Components, Configuration, Data Values

The following sections describe External DNS components and show how you can configure the External DNS package.

ExternalDNS Components

The ExternalDNS package installs on the cluster the container listed in the table. For more information, see https://github.com/kubernetes-sigs/external-dns. The package pulls the container from the VMware public registry specified in Package Repository.

Container	Resource Type	Replicas	Description
ExternalDNS	DaemonSet	6	Expose Kubernetes services for DNS lookup

ExternalDNS Data Values

ExternalDNS synchronizes exposed Kubernetes Services and Ingresses with DNS providers.

The following example can be use for RFC2136-compliant DNS provider (such as BIND).

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: dns-sa
  namespace: tkg-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: dns-role-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: dns-sa
    namespace: tkg-system
---
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: dns
  namespace: tkg-system
spec:
  serviceAccountName: dns-sa
  packageRef:
    refName: dns.tanzu.vmware.com
    versionSelection:
      constraints: v0.14.2+vmware.1-tkg.1
  values:
  - secretRef:
      name: dns-data-values
---
apiVersion: v1
kind: Secret
metadata:
  name: dns-data-values
  namespace: tkg-system
stringData:
  values.yml: |
    ---
    namespace: service-discovery
    dns:
      pspNames: "vmware-system-restricted"
      deployment:
        args:
        - --txt-owner-id=k8s
        - --provider=rfc2136
        - --rfc2136-host=192.168.0.1 #! IP of RFC2136 compatible dns server
        - --rfc2136-port=53
        - --rfc2136-zone=my-zone.example.org #! zone where services are deployed
        - --rfc2136-tsig-secret=REPLACE_ME_WITH_TSIG_SECRET #! TSIG key authorized to
update the DNS server
        - --rfc2136-tsig-secret-alg=hmac-sha256
        - --rfc2136-tsig-keyname=externaldns-key
        - --rfc2136-tsig-axfr
        - --source=service
        - --source=contour-httpproxy #! export contour HTTPProxy objs
```

```
        - --domain-filter=my-zone.example.org #! zone where services are deployed
```

The following example can be used for AWS DNS provider (Route 53).

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: dns-sa
  namespace: tkg-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: dns-role-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: dns-sa
    namespace: tkg-system
---
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: dns
  namespace: tkg-system
spec:
  serviceAccountName: dns-sa
  packageRef:
    refName: dns.tanzu.vmware.com
    versionSelection:
      constraints: v0.14.2+vmware.1-tkg.1
  values:
  - secretRef:
      name: dns-data-values
---
apiVersion: v1
kind: Secret
metadata:
  name: dns-data-values
  namespace: tkg-system
stringData:
  values.yml: |
    ---
    namespace: service-discovery
    dns:
      pspNames: "vmware-system-restricted"
      deployment:
        args:
        - --source=service
        - --source=ingress
        - --source=contour-httpproxy #! configure external-dns to read Contour HTTPPro
xy resources
        - --domain-filter=my-zone.example.org #! zone where services are deployed
        - --provider=aws
        - --policy=upsert-only #! would prevent ExternalDNS from deleting any records,
```

```
omit to enable full synchronization
        - --aws-zone-type=public #! only look at public hosted zones (valid values are
public, private or no value for both)
        - --aws-prefer-cname
        - --registry=txt
        - --txt-owner-id=REPLACE_ME_WITH_ROUTE_53_HOSTED_ZONE_ID #! Route53 hosted zon
e identifier for my-zone.example.org
        - --txt-prefix=txt #! disambiguates TXT records from CNAME records
        env:
          - name: AWS_ACCESS_KEY_ID
            valueFrom:
              secretKeyRef:
                name: route53-credentials #! Kubernetes secret for route53 credentials
                key: aws_access_key_id
          - name: AWS_SECRET_ACCESS_KEY
            valueFrom:
              secretKeyRef:
                name: route53-credentials #! Kubernetes secret for route53 credentials
                key: aws_secret_access_key
```

The following example can be used for an Azure DNS provider.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: dns-sa
  namespace: tkg-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: dns-role-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: dns-sa
    namespace: tkg-system
---
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: dns
  namespace: tkg-system
spec:
  serviceAccountName: dns-sa
  packageRef:
    refName: dns.tanzu.vmware.com
    versionSelection:
      constraints: v0.14.2+vmware.1-tkg.1
  values:
  - secretRef:
      name: dns-data-values
---
apiVersion: v1
kind: Secret
metadata:
```

```
  name: dns-data-values
  namespace: tkg-system
stringData:
  values.yml: |
    ---
    namespace: service-discovery
    dns:
      pspNames: "vmware-system-restricted"
      deployment:
        args:
        - --provider=azure
        - --source=service
        - --source=ingress
        - --source=contour-httpproxy #! configure external-dns to read Contour HTTPPro
xy resources
        - --domain-filter=my-zone.example.org #! zone where services are deployed
        - --azure-resource-group=my-resource-group #! Azure resource group
        volumeMounts:
        - name: azure-config-file
          mountPath: /etc/kubernetes
          readOnly: true
        #@overlay/replace
        volumes:
        - name: azure-config-file
          secret:
            secretName: azure-config-file
```

ExternalDNS Configuration Parameters

The table lists and describes the available configuration parameters for ExternalDNS. Refer to the site https://github.com/kubernetes-sigs/external-dns#running-externaldns for additional guidance.

Parameter	Description	Type	Default
externalDns.namespace	Namespace where external-dns will be deployed	string	tanzu-system-service-discovery
externalDns.image.repository	Repository containing external-dns image	string	projects.packages.broadcom.com/tkg
externalDns.image.name	Name of external-dns	string	external-dns
externalDns.image.tag	ExternalDNS image tag	string	v0.7.4_vmware.1
externalDns.image.pullPolicy	ExternalDNS image pull policy	string	IfNotPresent
externalDns.deployment.annotations	Annotations on the external-dns deployment	map<string,string>	{}
externalDns.deployment.args	Arguments passed via command-line to external-dns	list<string>	[] (Mandatory parameter)
externalDns.deployment.env	Environment variables to pass to external-dns	list<string>	[]
externalDns.deployment.securityContext	Security context of the external-dns container	SecurityContext	{}

Parameter	Description	Type	Default
externalDns.deployment.volumeMounts	Volume mounts of the external-dns container	list<VolumeMount>	[]
externalDns.deployment.volumes	Volumes of the external-dns pod	list<Volume>	[]

Example Configmap

The following example configmap defines a Kerberos configuration that ExternalDNS can interface with. Custom entries include the domain/realm name and the kdc/admin_server addresses.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: krb.conf
  namespace: tanzu-system-service-discovery
data:
  krb5.conf: |
    [logging]
    default = FILE:/var/log/krb5libs.log
    kdc = FILE:/var/log/krb5kdc.log
    admin_server = FILE:/var/log/kadmind.log

    [libdefaults]
    dns_lookup_realm = false
    ticket_lifetime = 24h
    renew_lifetime = 7d
    forwardable = true
    rdns = false
    pkinit_anchors = /etc/pki/tls/certs/ca-bundle.crt
    default_ccache_name = KEYRING:persistent:%{uid}

    default_realm = CORP.ACME

    [realms]
    CORP.ACME = {
      kdc = controlcenter.corp.acme
      admin_server = controlcenter.corp.acme
    }

    [domain_realm]
    corp.acme = CORP.ACME
    .corp.acme = CORP.ACME
```

Install External DNS (Standalone Management Cluster)

This topic explains how to deploy ExternalDNS into a workload cluster in Tanzu Kubernetes Grid.

This documentation is applicable only to Tanzu Kubernetes Grid with management clusters. If you are using TKGS with vSphere Supervisor, see Installing Standard Packages on TKG Service Clusters.

The ExternalDNS service publishes DNS records for applications to DNS servers, using a declarative, Kubernetes-native interface. It is packaged as a CLI-managed package in Tanzu Kubernetes Grid.

In environments where Harbor is deployed in a shared services cluster with load balancing (AWS, Azure, and vSphere with NSX Advanced Load Balancer), ExternalDNS may be used to publish a DNS hostname for the Harbor service. This provides access to Harbor from other clusters. For more information, see Harbor Registry and ExternalDNS.

As of v2.5, TKG does not support clusters on AWS or Azure. See the End of Support for TKG Management and Workload Clusters on AWS and Azure in the *Tanzu Kubernetes Grid v2.5 Release Notes*.

Prerequisites

- A bootstrap machine with the following installed:
  - Tanzu CLI, Tanzu CLI plugins, and `kubectl`.
  - yq v4.5 or later.
- You have deployed a management cluster on vSphere, Amazon Web Services (AWS), or Azure, in either an Internet-connected or Internet-restricted environment. If you are using Tanzu Kubernetes Grid in an Internet-restricted environment, you performed the procedure in Prepare to Deploy Management Clusters to an Internet-Restricted Environment before you deployed the management cluster.
- You have logged in to the Tanzu CLI with the `tanzu context use` command.
- You have determined the Fully Qualified Domain Names (FQDNs) for the services you want to expose with ExternalDNS to your DNS server.

Prepare the Cluster for ExternalDNS Deployment

The ExternalDNS service must be deployed into the same cluster as the services for which it will export DNS records.

- To install Harbor, follow the prerequisites and procedure in Install Harbor for Service Registry.
- ExternalDNS supports creating records for both Kubernetes Services and Contour HTTPProxy resources. If you want to create records for Contour HTTPProxy resources in a workload cluster, you must install the Contour package in the cluster. For instructions, see Install Contour for Ingress Control. The Contour package is also required by Harbor.

Prepare the Configuration File for the ExternalDNS Package

The ExternalDNS package has been validated with AWS (Route 53), Azure DNS, and RFC2136 (BIND). The configuration provided exports records for both Contour HTTPProxy resources and Kubernetes `Services` of type `LoadBalancer`.

The External DNS community maintains integrations with many DNS providers at varying levels of stability. Except where noted, VMware does not guarantee support for integrating the ExternalDNS package with specific providers.

AWS (Route 53)

To prepare the configuration file to deploy the ExternalDNS package on AWS,

1. Create a hosted zone within Route 53 with the domain that your services will be using.
2. Record the **Hosted zone ID**. You will use this ID later, when configuring ExternalDNS.
3. Create an IAM policy for ExternalDNS that allows ExternalDNS to update Route 53. In the AWS Console, go to the IAM dashboard and under **Access Management**, navigate to the **Policies** screen. Click **Create Policy** and switch to the JSON tab. Paste in the following policy. If needed, you may fine-tune the policy to allow updates to the hosted zone that you just created. Complete the wizard.

```
{
 "Version": "2012-10-17",
 "Statement": [
   {
     "Effect": "Allow",
     "Action": [
       "route53:ChangeResourceRecordSets"
     ],
     "Resource": [
       "arn:aws:route53:::hostedzone/*"
     ]
   },
   {
     "Effect": "Allow",
     "Action": [
       "route53:ListHostedZones",
       "route53:ListResourceRecordSets"
     ],
     "Resource": [
       "*"
     ]
   }
 ]
}
```

4. Create an IAM user for ExternalDNS with the policy that you created above. In the AWS Console, go to the **Users** screen and click **Add users**. Provide a name for the IAM user and ensure **Programmatic access** is enabled. On the **Permissions** screen of the wizard, click **Attach existing policies directly** and select the policy that you created in the previous step.
5. Continue to the final page of the wizard. Record the **Access key ID** and **Secret access key**. To make these Route 53 credentials available to ExternalDNS, you create a Kubernetes secret in the namespace where ExternalDNS will be running.
    1. Set the context of `kubectl` to the cluster where you are deploying ExternalDNS. For example:

```
kubectl config use-context tkg-services-admin@tkg-services
```

2. Create the Kubernetes secret:

   ```
   kubectl -n tanzu-system-service-discovery create secret generic rou
   te53-credentials \
   --from-literal=aws_access_key_id=YOUR-ACCESS-KEY-ID \
   --from-literal=aws_secret_access_key=YOUR-SECRET-ACCESS-KEY
   ```

   Where `YOUR-ACCESS-KEY-ID` and `YOUR-SECRET-ACCESS-KEY` are the credentials that you recorded above.

6. Create a configuration file for the ExternalDNS package by retrieving the default configuration of the package:

   ```
   tanzu package available get external-dns.tanzu.vmware.com/PACKAGE-VERSION
   --default-values-file-output FILE-PATH
   ```

   Where `PACKAGE-VERSION` is the version of the ExternalDNS package that you want to install and `FILE-PATH` is the location to which you want to save the configuration file, for example, `external-dns-data-values.yaml`.

7. Configure the following settings in the `external-dns-data-values.yaml` file. This file configures the ExternalDNS package.

   ```
   ---

   # Namespace in which to deploy ExternalDNS pods.
   namespace: tanzu-system-service-discovery

   # Deployment-related configuration.
   deployment:
   args:
     - --source=service
     - --source=ingress
     - --source=contour-httpproxy # Provide this to enable Contour HTTPProxy
   support. Must have Contour installed or ExternalDNS will fail.
     - --domain-filter=DOMAIN # Makes ExternalDNS see only the hosted zones m
   atching provided domain, omit to process all available hosted zones.
     - --policy=upsert-only # Prevents ExternalDNS from deleting any records,
   omit to enable full synchronization.
     - --registry=txt
     - --txt-owner-id=HOSTED-ZONE-ID
     - --txt-prefix=txt # Disambiguates TXT records from CNAME records.
     - --provider=aws
     - --aws-zone-type=public # Looks only at public hosted zones. Valid valu
   es are public, private, or no value for both.
     - --aws-prefer-cname
   env:
     - name: AWS_ACCESS_KEY_ID
       valueFrom:
       secretKeyRef:
         name: route53-credentials
         key: aws_access_key_id
     - name: AWS_SECRET_ACCESS_KEY
       valueFrom:
       secretKeyRef:
         name: route53-credentials
         key: aws_secret_access_key
   ```

```
securityContext: {}
volumeMounts: []
volumes: []
```

Replace the placeholders in the `external-dns-data-values.yaml` file with your values. For more configuration options, see the ExternalDNS documentation.

Before setting any additional configuration options in the `external-dns-data-values.yaml` file, review the values schema of the ExternalDNS package. To retrieve the values schema, run:

```
tanzu package available get external-dns.tanzu.vmware.com/AVAILABLE-VERSIO
N --values-schema
```

Where `AVAILABLE-VERSION` is the version of the ExternalDNS package. The `--values-schema` flag retrieves the `valuesSchema` section from the `Package` API resource for the ExternalDNS package. You can set the output format, `--output`, for the values schema to `yaml`, `json`, or `table`. For more information, see Packages in *Install and Manage Packages*.

RFC2136 (BIND) Server

The RFC2136 provider allows you to use any RFC2136-compatible DNS server as a provider for ExternalDNS, such as BIND.

1. Request or create a TSIG key for your server. This key must be authorized to update and transfer the zone you want to update. The key looks similar to the following:

```
key "externaldns-key" {
algorithm hmac-sha256;
secret "/2avn5M4ndEztbDqy66lfQ+PjRZta9UXLtToW6NV5nM=";
};
```

If you are managing your own DNS server, then you can create a TSIG key using the `tsig-keygen -a hmac-sha256 externaldns` command. Copy the output to your DNS servers configuration. For example, for BIND, you add the key to the `named.conf` file and configure the zone with the `allow-transfer` and `update-policy` fields. For example:

```
key "externaldns-key" {
algorithm hmac-sha256;
secret "/2avn5M4ndEztbDqy66lfQ+PjRZta9UXLtToW6NV5nM=";
};
zone "k8s.example.org" {
type master;
file "/etc/bind/zones/k8s.zone";
      allow-transfer {
      key "externaldns-key";
      };
      update-policy {
      grant externaldns-key zonesub ANY;
      };
};
```

The above assumes that you also have a zone file that might look similar to the following:

```
$TTL 60 ; 1 minute
@         IN SOA  k8s.example.org.  root.k8s.example.org. (
```

```
                    16  ; serial
                    60  ; refresh (1 minute)
                    60  ; retry (1 minute)
                    60  ; expire (1 minute)
                    60  ; minimum (1 minute)
                    )
              NS    ns.k8s.example.org.
ns             A    1.2.3.4
```

2. Create a configuration file for the ExternalDNS package by retrieving the default configuration of the package:

```
tanzu package available get external-dns.tanzu.vmware.com/PACKAGE-VERSION
--default-values-file-output FILE-PATH
```

   Where `PACKAGE-VERSION` is the version of the ExternalDNS package that you want to install and `FILE-PATH` is the location to which you want to save the configuration file, for example, `external-dns-data-values.yaml`.

3. Configure the following settings in the `external-dns-data-values.yaml` file. This file configures the ExternalDNS package.

```
---

# Namespace in which to deploy ExternalDNS pods.
namespace: tanzu-system-service-discovery

# Deployment-related configuration.
deployment:
args:
   - --source=service
   - --source=ingress
   - --source=contour-httpproxy # Provide this to enable Contour HTTPProxy
support. Must have Contour installed or ExternalDNS will fail.
   - --domain-filter=DOMAIN # For example, k8s.example.org. Makes External
DNS see only the hosted zones matching provided domain, omit to process al
l available hosted zones.
   - --policy=upsert-only # Prevents ExternalDNS from deleting any record
s, omit to enable full synchronization.
   - --registry=txt
   - --txt-owner-id=k8s
   - --txt-prefix=external-dns- # Disambiguates TXT records from CNAME rec
ords.
   - --provider=rfc2136
   - --rfc2136-host=IP-ADDRESS-OF-RFC2136-DNS-SERVER
   - --rfc2136-port=53
   - --rfc2136-zone=DNS-ZONE # For example, k8s.example.org.
   - --rfc2136-tsig-secret=TSIG-SECRET-FROM-STEP-1
   - --rfc2136-tsig-secret-alg=hmac-sha256
   - --rfc2136-tsig-keyname=TSIG-KEY-NAME # For example, externaldns-key.
   - --rfc2136-tsig-axfr
env: []
securityContext: {}
volumeMounts: []
volumes: []
```

Replace the placeholders in the `external-dns-data-values.yaml` file with your values. For more configuration options, see the ExternalDNS documentation.

Before setting any additional configuration options in the `external-dns-data-values.yaml` file, review the values schema of the ExternalDNS package. To retrieve the values schema, run:

```
tanzu package available get external-dns.tanzu.vmware.com/AVAILABLE-VERSIO
N --values-schema
```

Where `AVAILABLE-VERSION` is the version of the ExternalDNS package. The `--values-schema` flag retrieves the `valuesSchema` section from the `Package` API resource for the ExternalDNS package. You can set the output format, `--output`, for the values schema to `yaml`, `json`, or `table`. For more information, see Packages in *Install and Manage Packages*.

Azure

To prepare the configuration file to deploy the ExternalDNS package on Azure,

1. Log in to the `az` CLI:

   ```
   az login
   ```

2. Set your subscription:

   ```
   az account set -s SUBSCRIPTION-ID-GUID
   ```

3. Create a service principal:

   ```
   az ad sp create-for-rbac -n SERVICE-PRINCIPAL-NAME
   ```

   The JSON output of the command looks similar to the following:

   ```
   {
     "appId": "a72a7cfd-7cb0-4b02-b130-03ee87e6ca89",
     "displayName": "foo",
     "name": "http://foo",
     "password": "515c55da-f909-4e17-9f52-236ffe1d3033",
     "tenant": "b35138ca-3ced-4b4a-14d6-cd83d9ea62f0"
   }
   ```

4. Assign permissions to the service principal:

   1. Retrieve the ID of the resource group:

      ```
      az group show --name RESOURCE-GROUP --query id
      ```

   2. Assign the reader role to the service principal for the resource group scope. You will need the `appId` from the output of the `az ad sp create-for-rbac` command above.

      ```
      az role assignment create --role "Reader" --assignee APP-ID-GUID --
      scope RESOURCE-GROUP-RESOURCE-ID
      ```

   3. Retrieve the ID of the DNS zone:

```
az network dns zone show --name DNS-ZONE-NAME -g RESOURCE-GROUP-NAM
E --query id
```

4. Assign the contributor role to the service principal for the DNS zone scope:

   ```
   az role assignment create --role "Contributor" --assignee APP-ID-GU
   ID --scope DNS-ZONE-RESOURCE-ID
   ```

5. To connect the ExternalDNS service to the Azure DNS service, you create a configuration file named `azure.json` on your local machine with contents that look like the following:

   ```
   {
    "tenantId": "01234abc-de56-ff78-abc1-234567890def",
    "subscriptionId": "01234abc-de56-ff78-abc1-234567890def",
    "resourceGroup": "MyDnsResourceGroup",
    "aadClientId": "01234abc-de56-ff78-abc1-234567890def",
    "aadClientSecret": "uKiuXeiwui4jo9quae9o"
   }
   ```

   Replace the values in the example above with you own values as follows:

   - To retrieve `tenantId`, you can run the `az account show --query "tenantId"` command.
   - To retrieve `subscriptionId`, you can run the `az account show --query "id"` command.
   - `resourceGroup` is the name of the resource group that your DNS zone is within.
   - `aadClientId` is the `appId` from the output of the service principal.
   - `aadClientSecret` is the password from the output of the service principal.

6. To make your Azure credentials available to ExternalDNS, create a Kubernetes secret in the namespace where ExternalDNS will be running:

   1. Set the context of `kubectl` to the cluster where you are deploying ExternalDNS. For example:

      ```
      kubectl config use-context tkg-services-admin@tkg-services
      ```

   2. Create the secret, using the `azure.json` configuration file from the previous step:

      ```
      kubectl -n tanzu-system-service-discovery create secret generic azu
      re-config-file --from-file=azure.json
      ```

7. Create a configuration file for the ExternalDNS package by retrieving the default configuration of the package:

   ```
   tanzu package available get external-dns.tanzu.vmware.com/PACKAGE-VERSION
   --default-values-file-output FILE-PATH
   ```

   Where `PACKAGE-VERSION` is the version of the ExternalDNS package that you want to install and `FILE-PATH` is the location to which you want to save the configuration file, for example, `external-dns-data-values.yaml`.

8. Configure the following settings in the `external-dns-data-values.yaml` file. This file configures the ExternalDNS package.

```
---

# Namespace in which to deploy ExternalDNS.
namespace: tanzu-system-service-discovery

# Deployment-related configuration.
deployment:
 args:
   - --source=service
   - --source=ingress
   - --source=contour-httpproxy # Provide this to enable Contour HTTPProxy
support. Must have Contour installed or ExternalDNS will fail.
   - --domain-filter=DOMAIN # For example, k8s.example.org. Makes External
DNS see only the hosted zones matching provided domain, omit to process al
l available hosted zones.
   - --policy=upsert-only # Prevents ExternalDNS from deleting any record
s, omit to enable full synchronization.
   - --registry=txt
   - --txt-prefix=externaldns- # Disambiguates TXT records from CNAME reco
rds.
   - --provider=azure
   - --azure-resource-group=RESOURCE-GROUP # Azure resource group.
 env: []
 securityContext: {}
 volumeMounts:
   - name: azure-config-file
     mountPath: /etc/kubernetes
     readOnly: true
 volumes:
   - name: azure-config-file
     secret:
       secretName: azure-config-file
```

Replace the placeholders in the `external-dns-data-values.yaml` file with your values. For more configuration options, see the ExternalDNS documentation.

Before setting any additional configuration options in the `external-dns-data-values.yaml` file, review the values schema of the ExternalDNS package. To retrieve the values schema, run:

```
tanzu package available get external-dns.tanzu.vmware.com/AVAILABLE-VERSIO
N --values-schema
```

Where `AVAILABLE-VERSION` is the version of the ExternalDNS package. The `--values-schema` flag retrieves the `valuesSchema` section from the `Package` API resource for the ExternalDNS package. You can set the output format, `--output`, for the values schema to `yaml`, `json`, or `table`. For more information, see Packages in *Install and Manage Packages*.

Install the ExternalDNS Package

1. Set the context of `kubectl` to the cluster where you are deploying ExternalDNS. For example:

   ```
   kubectl config use-context tkg-services-admin@tkg-services
   ```

2. If the cluster does not have a package repository with the ExternalDNS package installed, such as the `tanzu-standard` repository, install one:

   ```
   tanzu package repository add PACKAGE-REPO-NAME --url PACKAGE-REPO-ENDPOINT --na
   mespace tkg-system
   ```

   Where:

   - `PACKAGE-REPO-NAME` is the name of the package repository, such as `tanzu-standard` or the name of a private image registry configured with `ADDITIONAL_IMAGE_REGISTRY` variables.
   - `PACKAGE-REPO-ENDPOINT` is the URL of the package repository.
     - For the TKG v2.5.2 release, the `tanzu-standard` URL is `projects.packages.broadcom.com/tkg/packages/standard/repo:v2025.1.27`. See List Package Repositories to obtain this value from the Tanzu CLI, or in Tanzu Mission Control see the **Addons** > **Repositories** list in the **Cluster** pane.

3. Retrieve the name of the ExternalDNS package:

   ```
   tanzu package available list -A
   ```

4. Retrieve the version of the ExternalDNS package:

   ```
   tanzu package available list external-dns.tanzu.vmware.com -A
   ```

5. If your `external-dns-data-values.yaml` file contains comments, remove them before installing the package:

   ```
   yq -i eval '... comments="" ' external-dns-data-values.yaml
   ```

6. Install the package:

   ```
   tanzu package install external-dns \
   --package external-dns.tanzu.vmware.com \
   --version AVAILABLE-PACKAGE-VERSION \
   --values-file external-dns-data-values.yaml \
   --namespace TARGET-NAMESPACE
   ```

   Where:

   - `TARGET-NAMESPACE` is the namespace in which you want to install the ExternalDNS package. For example, the `my-packages` or `tanzu-cli-managed-packages` namespace.
     - If the `--namespace` flag is not specified, the Tanzu CLI installs the package in the `default` namespace. The ExternalDNS pods and any other resources associated with the ExternalDNS component are created in the `tanzu-system-service-`

`discovery` namespace; do not install the ExternalDNS package into this namespace.

- The specified namespace must already exist, for example from running `kubectl create namespace my-packages`.

- `AVAILABLE-PACKAGE-VERSION` is the version that you retrieved above.

For example:

```
tanzu package install external-dns \
--package external-dns.tanzu.vmware.com \
--version v0.14.2+vmware.1-tkg.1 \
--values-file external-dns-data-values.yaml \
--namespace my-packages
```

7. Confirm that the `external-dns` package has been installed:

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

   To see more details about the package, you can also run:

   ```
   tanzu package installed get external-dns --namespace PACKAGE-NAMESPACE
   ```

   Where `PACKAGE-NAMESPACE` is the namespace in which the `external-dns` package is installed.

8. Confirm that the `external-dns` app has been successfully reconciled in your `PACKAGE-NAMESPACE`:

   ```
   kubectl get apps -A
   ```

   If the status is not `Reconcile Succeeded`, view the full status details of the `external-dns` app. Viewing the full status can help you troubleshoot the problem.

   ```
   kubectl get app external-dns --namespace PACKAGE-NAMESPACE -o yaml
   ```

   Where `PACKAGE-NAMESPACE` is the namespace in which you installed the package. If troubleshooting does not help you solve the problem, you must uninstall the package before installing it again:

   ```
   tanzu package installed delete external-dns --namespace PACKAGE-NAMESPACE
   ```

9. Confirm that ExternalDNS pods are running in the `tanzu-system-service-discovery` namespace:

   ```
   kubectl get pods -A
   ```

Validating ExternalDNS

If configured with Contour, ExternalDNS will automatically watch the specified namespace for HTTPProxy resources and create DNS records for services with hostnames that match the configured domain filter.

ExternalDNS will also automatically watch for Kubernetes Services with the annotation `external-dns.alpha.kubernetes.io/hostname` and create DNS records for services whose annotations match the

configured domain filter.

For example, for a service with the annotation `external-dns.alpha.kubernetes.io/hostname: foo.k8s.example.org`, ExternalDNS will create a DNS record for `foo.k8s.example.org`. You can validate that the record exists by examining the zone that you created.

Update a Running ExternalDNS Deployment

If you need to make changes to the configuration of the ExternalDNS package after deployment, follow these steps to update your deployed ExternalDNS package.

1. Update the ExternalDNS configuration in `external-dns-data-values.yaml`.
2. Update the configuration of the installed package:

   ```
   tanzu package installed update external-dns \
   --version INSTALLED-PACKAGE-VERSION \
   --values-file external-dns-data-values.yaml \
   --namespace INSTALLED-PACKAGE-NAMESPACE
   ```

   Where:

   - `INSTALLED-PACKAGE-VERSION` is the version of the installed ExternalDNS package.
   - `INSTALLED-PACKAGE-NAMESPACE` is the namespace in which the ExternalDNS package is installed.

   For example:

   ```
   tanzu package installed update external-dns \
   --version v0.14.2+vmware.1-tkg.1 \
   --values-file external-dns-data-values.yaml \
   --namespace my-packages
   ```

The ExternalDNS package will be reconciled using the new value or values that you added. It can take up to five minutes for `kapp-controller` to apply the changes.

For more information about the `tanzu package installed update` command, see Update a Package in *Install and Manage Packages*. You can use this command to update the version or the configuration of an installed package.

Fluent Bit: Log Forwarding

This topic gives an overview of the Fluent Bit package, which you can install in Tanzu Kubernetes Grid (TKG) workload clusters to provide log forwarding services for the cluster.

Fluent Bit is a fast, lightweight log processor and forwarder that lets you collect application data and logs from different sources, unify them, and send them to multiple destinations.

You can use Fluent Bit as a log forwarder for logs from a TKG cluster. You need to have a logging management server deployed for storing and analyzing logs. You can use one of the supported logging servers.

- Syslog
- HTTP
- Elastic Search
- Kafka
- Splunk

See Install Fluent Bit in Workload Clusters Deployed by a Standalone Management Cluster.

Fluent Bit Components, Configuration, Data Values

The following sections describe Fluent Bit components and show how you can configure the Fluent Bit package.

Fluent Bit Components

The Fluent Bit package installs on the cluster the container listed in the table. For more information, see https://fluentbit.io/. The package pulls the container from the VMware public registry specified in Package Repository.

Container	Resource Type	Replicas	Description
Fluent Bit	DaemonSet	6	Log collector, aggregator, forwarder

Fluent Bit Data Values

The following example can be used for an HTTP logging endpoint.

```
...
    ---
    namespace: fluentbit-logging
    tkg:
      instance_name: "<TKG_INSTANCE_NAME>"
      cluster_name: "<CLUSTER_NAME>"
```

```
    fluentbit:
      pspNames: "vmware-system-restricted"
      output_plugin: "http"
      http:
        host: "<HTTP_HOST>"
        port: "<HTTP_PORT>"
        uri: "<URI>"
        header_key_value: "<HEADER_KEY_VALUE>"
        format: "json"
```

The following example can be used for Elastic Search.

```
...
    ---
    namespace: fluentbit-logging
    tkg:
      instance_name: "<TKG_INSTANCE_NAME>"
      cluster_name: "<CLUSTER_NAME>"
    fluentbit:
      pspNames: "vmware-system-restricted"
      output_plugin: "elasticsearch"
      elasticsearch:
        host: "<ELASTIC_SEARCH_HOST>"
        port: "<ELASTIC_SEARCH_PORT>"
```

The following example can be used for Kafka.

```
...
    ---
    namespace: fluentbit-logging
    tkg:
      instance_name: "<TKG_INSTANCE_NAME>"
      cluster_name: "<CLUSTER_NAME>"
    fluentbit:
      pspNames: "vmware-system-restricted"
      output_plugin: "kafka"
      kafka:
        broker_service_name: "<BROKER_SERVICE_NAME>"
        topic_name: "<TOPIC_NAME>"
```

The following example can be used for Splunk.

```
...
    ---
    namespace: fluentbit-logging
    tkg:
      instance_name: "<TKG_INSTANCE_NAME>"
      cluster_name: "<CLUSTER_NAME>"
    fluentbit:
      pspNames: "vmware-system-restricted"
      output_plugin: "splunk"
      splunk:
        host: "<SPLUNK_HOST>"
        port: "<SPLUNK_PORT>"
```

```
        token: "<SPLUNK_TOKEN>"
```

Fluent Bit Configuration Parameters (Standalone MC)

The table below lists configuration parameters of the Fluent Bit package and describes their default values. You can set the following configuration values in your `fluent-bit-data-values.yaml` file created in Deploy Fluent Bit on a Cluster.

Parameter	Description	Type	Default
`namespace`	Namespace where Fluent Bit will be deployed.	String	`tanzu-system-logging`
`fluent_bit.config.outputs`	For information about the configuration for Fluent Bit outputs, see the Fluent Bit documentation.	Multiline YAML	`Standard output`
`fluent_bit.config.parsers`	For information about the configuration for Fluent Bit parsers, see the Fluent Bit documentation.	Multiline YAML	`JSON parser`
`fluent_bit.config.plugins`	Content for Fluent Bit plugins configuration.	String	`<nil>`
`fluent_bit.config.service`	For information about the configuration for Fluent Bit service, see the Fluent Bit documentation.	Multiline YAML	Default Fluent Bit service config.
`fluent_bit.config.streams`	Content for Fluent Bit streams file.	String	`<nil>`
`fluent_bit.config.filters`	For information about the configuration for Fluent Bit filters, see the Fluent Bit documentation.	Multiline YAML	Default Kubernetes filter.
`fluent_bit.config.inputs`	For information about the configuration for Fluent Bit inputs, see the Fluent Bit documentation.	String	Ingest Kubernetes container logs using the `tail` plugin and ingest `systemd` logs from kubelet.
`fluent_bit.daemonset.resources`	For information about the configuration for Fluent Bit containers resource requirements, see the Fluent Bit documentation.	Multiline YAML	`<nil>`
`fluent_bit.daemonset.podAnnotations`	The Fluent Bit deployments pod annotations.	List	`<nil>`
`fluent_bit.daemonset.podLabels`	The Fluent Bit deployments pod labels.	List	`<nil>`
`fluent_bit.ipv6Primary`	If you are deploying Fluent Bit to an IPv6 cluster, set `ipv6Primary` to `true`.	Boolean	`false`

Install Fluent Bit (Standalone Management Cluster)

This topic explains how to deploy Fluent Bit into a standalone management or workload cluster in Tanzu Kubernetes Grid.

This documentation is applicable only to Tanzu Kubernetes Grid with management clusters. If you are using TKGS with vSphere Supervisor, see Installing Standard

Packages on TKG Service Clusters.

Fluent Bit is a lightweight log processor and forwarder. The Fluent Bit implementation provided in this release of Tanzu Kubernetes Grid allows you to collect logs from clusters running on vSphere 8, Amazon Web Services (AWS), and Azure. You can then forward them to a log storage provider such as Elastic Search, Kafka, Splunk, or an HTTP endpoint.

You can deploy Fluent Bit on any standalone management cluster or workload cluster from which you want to collect logs. First, you configure an output plugin on the cluster from which you want to gather logs, depending on the endpoint that you use. Then, you deploy Fluent Bit on the cluster.

Prerequisites

- A bootstrap machine with the following installed:
  - Tanzu CLI and `kubectl`, as described in Install the Tanzu CLI and Kubernetes CLI for Use with Standalone Management Clusters.
  - yq v4.5 or later.
- A standalone management cluster and workload cluster running on vSphere 8, AWS, or Azure. For information about deploying workload clusters, see Creating Workload Clusters.
- You have deployed one of the logging management backends for storing and analyzing logs that Fluent Bit supports, such as VMware Aria Operations for Logs. For information, see the Fluent Bit documentation.

As of v2.5, TKG does not support clusters on AWS or Azure. See the End of Support for TKG Management and Workload Clusters on AWS and Azure in the *Tanzu Kubernetes Grid v2.5 Release Notes*.

Prepare the Cluster for Fluent Bit Deployment

You must deploy Fluent Bit on each cluster from which you want to collect logs.

To prepare the cluster:

1. Get the admin credentials of the cluster into which you want to deploy Fluent Bit. For example:

   ```
   tanzu cluster kubeconfig get my-cluster --admin
   ```

2. Set the context of kubectl to the cluster. For example:

   ```
   kubectl config use-context my-cluster-admin@my-cluster
   ```

Deploy Fluent Bit on a Cluster

Follow the procedure below to deploy Fluent Bit on a cluster:

1. If the cluster does not have a package repository with the Fluent Bit package installed, such as the `tanzu-standard` repository, install one:

   ```
   tanzu package repository add PACKAGE-REPO-NAME --url PACKAGE-REPO-ENDPOINT --na
   mespace tkg-system
   ```

   Where:

   - `PACKAGE-REPO-NAME` is the name of the package repository, such as `tanzu-standard` or the name of a private image registry configured with `ADDITIONAL_IMAGE_REGISTRY` variables.
   - `PACKAGE-REPO-ENDPOINT` is the URL of the package repository.
     - For the TKG v2.5.2 release, the `tanzu-standard` URL is `projects.packages.broadcom.com/tkg/packages/standard/repo:v2025.1.27`. See List Package Repositories to obtain this value from the Tanzu CLI, or in Tanzu Mission Control see the **Addons** > **Repositories** list in the **Cluster** pane.
2. If you have not already done so, install cert-manager in the cluster. For instructions, see Install Cert Manager for Certificate Management.
3. Confirm that the Fluent Bit package is available in your cluster:

   ```
   tanzu package available list -A
   ```

4. Retrieve the version of the available package:

   ```
   tanzu package available list fluent-bit.tanzu.vmware.com -A
   ```

5. Prepare a configuration file for the Fluent Bit package:
   1. Retrieve the default configuration of the package:

      ```
      tanzu package available get fluent-bit.tanzu.vmware.com/PACKAGE-VERSION -
      -default-values-file-output FILE-PATH
      ```

      Where `PACKAGE-VERSION` is the version of the Fluent Bit package that you want to install and `FILE-PATH` is the location to which you want to save the configuration file, for example, `fluent-bit-data-values.yaml`.
   2. Based on the log management backend you want to configure, add one or more `[Output]` configurations to the `fluent_bit.config.outputs` key in the `fluent-bit-data-values.yaml` file.
   3. Review and if needed, modify other configuration parameters. See Fluent Bit Configuration Parameters (Standalone MC) for a full list of available parameters.
6. After you prepare your `fluent-bit-data-values.yaml` file, remove all comments in the file:

   ```
   yq -i eval '... comments="" ' fluent-bit-data-values.yaml
   ```

7. Run `tanzu package install` to deploy the package:

```
tanzu package install fluent-bit \
--package fluent-bit.tanzu.vmware.com \
--version AVAILABLE-PACKAGE-VERSION \
--values-file fluent-bit-data-values.yaml \
--namespace TARGET-NAMESPACE
```

Where:

- `TARGET-NAMESPACE` is the namespace in which you want to install the Fluent Bit package. For example, the `my-packages` namespace.
  - If the `--namespace` flag is not specified, the Tanzu CLI uses the `default` namespace. The Fluent Bit pods and any other resources associated with the Fluent Bit component are created in the namespace that you provided in `fluent-bit-data-values.yaml`; do not install the Fluent Bit package into this namespace.
  - The specified namespace must already exist, for example from running `kubectl create namespace my-packages`.
- `AVAILABLE-PACKAGE-VERSION` is the version that you retrieved by running `tanzu package available` above.

For example:

```
tanzu package install fluent-bit --package fluent-bit.tanzu.vmware.com --namesp
ace my-packages --version v2.1.6+vmware.2-tkg.1
```

Verify Fluent Bit Deployment

After you deploy Fluent Bit, you can verify that the deployment is successful:

1. Confirm that the fluent-bit package is installed. For example:

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

   The `fluent-bit` package and the `fluent-bit` app are installed in the namespace that you specify when running the `tanzu package install` command.

2. Confirm that the `fluent-bit` app is successfully reconciled in your `AVAILABLE-PACKAGE-NAMESPACE`:

   ```
   kubectl get apps -A
   ```

   If the status is not `Reconcile succeeded`, view the full status details of the `fluent-bit` app. Viewing the full status can help you troubleshoot the problem:

   ```
   kubectl get app fluent-bit --namespace AVAILABLE-PACKAGE-NAMESPACE -o yaml
   ```

   Where `AVAILABLE-PACKAGE-NAMESPACE` is the namespace in which you installed the package.

3. Confirm that the new services are running by listing all of the pods that are running in the cluster:

```
kubectl get pods -A
```

In the `tanzu-system-logging` namespace, you should see the `fluent-bit` service running in a pod:

```
NAMESPACE               NAME                     READY   STATUS    RESTARTS   A
GE
[...]
tanzu-system-logging    fluent-bit-zd2tn         1/1     Running   0          6
m23s
[...]
```

The Fluent Bit pods and any other resources associated with the Fluent Bit component are created in the namespace you provided in the `fluent-bit-data-values.yaml` file. If you are using the default namespace, these are created in the `tanzu-system-logging` namespace.

Update a Running Fluent Bit Deployment

To make changes to the configuration of the Fluent Bit package after deployment, update your deployed Fluent Bit package:

1. Update the Fluent Bit configuration in the `fluent-bit-data-values.yaml` file.
2. Update the installed package:

   ```
   tanzu package installed update fluent-bit \
   --version v2.1.6+vmware.2-tkg.1 \
   --values-file fluent-bit-data-values.yaml \
   --namespace my-packages
   ```

   Expected output:

   ```
   | Updating package 'fluent-bit'
   - Getting package install for 'fluent-bit'
   | Updating secret 'fluent-bit-my-packages-values'
   | Updating package install for 'fluent-bit'

    Updated package install 'fluent-bit' in namespace 'my-packages'
   ```

The Fluent Bit package is reconciled using the new value or values that you added. It can take up to five minutes for `kapp-controller` to apply the changes.

For information about updating, see Update a Package.

Delete a Fluent Bit Deployment

To remove the Fluent Bit package on your cluster, run:

```
tanzu package installed delete fluent-bit --namespace my-packages
```

For information about deleting, see Delete a Package.

Harbor: Container Registry

This topic gives an overview of the Harbor package, which you can install in Tanzu Kubernetes Grid (TKG) workload clusters to provide image registry services for the cluster.

Harbor is an open-source, enterprise-ready container registry system that provides an image repository, image vulnerability scanning, and project administration. See Install Harbor in Workload Clusters Deployed by a Standalone Management Cluster.

Harbor Components, Configuration, Data Values

The following sections describe Harbor components and show how you can configure the Harbor package.

Harbor Components

The Harbor package installs on the cluster the containers listed in the table. For more information, see https://goharbor.io/. The package pulls the containers from the VMware public registry specified in Package Repository.

Container	Resource Type	Replicas	Description
`harbor-core`	Deployment	1	Management and configuration server for Envoy
`harbor-database`	Pod	1	Postgres database
`harbor-jobservice`	Deployment	1	Harbor job service
`harbor-notary-server`	Deployment	1	Harbor notary service
`harbor-notary-signer`	Deployment	1	Harbor notary
`harbor-portal`	Deployment	1	Harbor web interface
`harbor-redis`	Pod	1	Harbor redis instance
`harbor-registry`	Deployment	2	Harbor container registry instance
`harbor-trivy`	Pod	1	Harbor image vulnerability scanner

Harbor Data Values

Below are example `harbor-data-values` for the secret in the `harbor.yaml` file provided with the installation.

Data Value	Description
`hostname: myharbordomain.com`	The FQDN for accessing Harbor admin UI and Registry service.

Data Value	Description
`harborAdminPassword: change-it`	The initial password for the Harbor admin account. This is applied only during installation. You can update it using the Harbor UI or API after installation.
`secretKey: 0123456789ABCDEF`	The secret key used for encryption. Must be a string of 16 chars.
`database.password: change-it`	The initial password of the postgres database.
`core.secret: change-it`	Secret is used when core server communicates with other components.
`xsrfKey: 0123456789ABCDEF0123456789ABCDEF`	The XSRF key. Must be a string of 32 chars.
`jobservice.secret: change-it`	Secret is used when job service communicates with other components.
`registry.secret: change-it`	Secret is used to secure the upload state from client and registry storage backend.
`persistence.persistentVolumeClaim.registry.storageClass: mystorageclass`	Specify the vSphere storage policy used to provision the volume.
`persistence.persistentVolumeClaim.jobservice.jobLog.storageClass: mystorageclass`	Specify the vSphere storage policy used to provision the volume.
`persistence.persistentVolumeClaim.database.size: size`	If you are performing a new installation of Harbor v2.10.3, increase the value from 1Gi to 10Gi, if you have sufficient available disk space.
`persistence.persistentVolumeClaim.database.storageClass: mystorageclass`	Specify the vSphere storage policy used to provision the volume.
`persistence.persistentVolumeClaim.redis.size: size`	If you are performing a new installation of Harbor v2.10.3, increase the value from 1Gi to 10Gi, if you have sufficient available disk space.
`persistence.persistentVolumeClaim.redis.storageClass: mystorageclass`	Specify the vSphere storage policy used to provision the volume.
`persistence.persistentVolumeClaim.registry.size: size`	If you are performing a new installation of Harbor v2.10.3, increase the value from 1Gi to 10Gi, if you have sufficient available disk space.
`persistence.persistentVolumeClaim.trivy.storageClass: mystorageclass`	Specify the vSphere storage policy used to provision the volume.

Harbor Configuration Parameters

The Harbor configuration is set in the `harbor-data-values.yaml` file. The table lists and describes the minimum required fields for deployment.

Property	Value	Description
hostname	FQDN	The FQDN that you have designated to access the Harbor UI and for referencing the registry in client applications. The domain should be configured in an external DNS server such that it resolves to the Envoy Service IP created by Contour.
tlsCertificate.tlsSecretLabels	{“managed-by”: “vmware-vRegistry”}	The certificate that Tanzu Kubernetes Grid uses to install the Harbor CA as a trusted root on Tanzu Kubernetes Grid clusters.

Property	Value	Description
persistence.persistentVolumeClaim.registry.storageClass	A storage policy name.	A storage class that is used for the Harbor registry PVCs.
persistence.persistentVolumeClaim.jobservice.jobLog.storageClass	A storage policy name.	A storage class that is used for the Harbor jobservice PVCs.
persistence.persistentVolumeClaim.database.storageClass	A storage policy name.	A storage class that is used for the Harbor database PVCs.
persistence.persistentVolumeClaim.redis.storageClass	A storage policy name.	A storage class that is used for the Harbor redis PVCs.
persistence.persistentVolumeClaim.trivy.storageClass	A storage policy name.	A storage class that is used for Harbor trivy PVCs.

Upgrading Harbor

When upgrading Harbor, VMware recommends only upgrading from N-1 or N-2 versions, to avoid database migration gaps.

Install Harbor (Standalone Management Cluster)

This topic explains how to deploy Harbor into a workload cluster or shared services cluster in Tanzu Kubernetes Grid.

- This documentation is applicable only to Tanzu Kubernetes Grid with management clusters. If you are using TKGS with vSphere Supervisor, see Installing Standard Packages on TKG Service Clusters.
- Notary and Chartmuseum are deprecated in Harbor v2.6 and are scheduled to be removed in a future release as noted in the Harbor v2.6.0 release notes. Users should switch to Sigstore Cosign for container signing and verification.
- When upgrading Harbor, VMware recommends only upgrading from N-1 or N-2 versions, to avoid database migration gaps.

Harbor

Harbor is an open-source, trusted, cloud-native container registry that stores, signs, and scans content. Tanzu Kubernetes Grid includes signed, packaged binaries for Harbor which you can deploy into a workload cluster to provide container registry services for that cluster. This Harbor package extends the open-source Docker distribution by adding functionalities usually required by users such as security and identity control and management.

Tanzu Kubernetes Grid includes signed binaries for Harbor, which you can deploy into:

- A workload cluster to provide container registry services for that clusters
- A shared services cluster, to provide container registry services for other workload clusters, in a deployment with a standalone management cluster.

When deployed as a shared service, Harbor is available to all of the workload clusters managed by the same standalone management cluster. To implement Harbor as a shared service, you deploy it into a special cluster that is dedicated to running shared services. Each management cluster can only have one shared service cluster.

Harbor Registry and ExternalDNS

VMware recommends installing ExternalDNS alongside the Harbor registry on infrastructures with load balancing, especially in production or other environments in which Harbor availability is important.

If the IP address to the ingress load balancer changes, ExternalDNS automatically picks up the change and re-maps the new address to the Harbor hostname. This precludes the need to manually re-map the address as described in Connect to the Harbor User Interface.

Prerequisites

- You have a running standalone management cluster.
- You have installed the Tanzu CLI and `kubectl`. For instructions, see Install the Tanzu CLI and Kubernetes CLI for Use with Standalone Management Clusters.
- You have installed yq v4.5 or later.
- You have installed imgpkg.
- You have logged in to the Tanzu CLI with the `tanzu context use` command.
- You have installed imgpkg. For information about how to download and install `imgpkg`, see Install the Carvel Tools.
- If you are deploying Harbor into a shared services cluster, the cluster has been created, as described in Create a Shared Services Cluster.

You cannot use Harbor's proxy cache feature for running Tanzu Kubernetes Grid v2.3 in an internet-restricted environment. You can still use a Harbor proxy cache to proxy images from prior versions of Tanzu Kubernetes Grid, and non-Tanzu images such as application images.

Prepare a Cluster for Harbor Deployment

To prepare a cluster for Harbor deployment:

1. Set the context of `kubectl` to the workload cluster or the shared services cluster. For example:

```
kubectl config use-context tkg-services-admin@tkg-services
```

2. If the cluster does not have a package repository with the Harbor package installed, such as the `tanzu-standard` repository, install one:

   ```
   tanzu package repository add PACKAGE-REPO-NAME --url PACKAGE-REPO-ENDPOINT --na
   mespace tkg-system
   ```

   Where:

   - `PACKAGE-REPO-NAME` is the name of the package repository, such as `tanzu-standard` or the name of a private image registry configured with `ADDITIONAL_IMAGE_REGISTRY` variables.
   - `PACKAGE-REPO-ENDPOINT` is the URL of the package repository.
     - For the TKG v2.5.2 release, the `tanzu-standard` URL is `projects.packages.broadcom.com/tkg/packages/standard/repo:v2025.1.27`. See List Package Repositories to obtain this value from the Tanzu CLI, or in Tanzu Mission Control see the **Addons** > **Repositories** list in the **Cluster** pane.
3. If you have not already done so, install the cert-manager and Contour packages. For instructions, see Install Contour for Ingress Control.
4. (Optional) Install the ExternalDNS package. For instructions, see Install ExternalDNS for Service Discovery.
5. Proceed to Deploy Harbor into a Cluster below.

Deploy Harbor into a Cluster

Follow this procedure to deploy Harbor into a workload or shared services cluster:

1. Confirm that the Harbor package is available in the cluster:

   ```
   tanzu package available list -A
   ```

2. Retrieve the version of the available package:

   ```
   tanzu package available list harbor.tanzu.vmware.com -A
   ```

3. Create a configuration file for the Harbor package by retrieving the default configuration of the package:

   ```
   tanzu package available get harbor.tanzu.vmware.com/PACKAGE-VERSION --default-v
   alues-file-output FILE-PATH
   ```

   Where `PACKAGE-VERSION` is the version of package as listed by `tanzu package available list`, for example, `v2.10.3+vmware.1-tkg.1`.

   The above method using `--default-values-file-output` only renders part of the configuration. To obtain a full configuration file for the Harbor package, use `imgpkg` to get it from the bundle. For example:

```
imgpkg pull -b projects.packages.broadcom.com/tkg/packages/standar
d/harbor:PACKAGE-VERSION -o /tmp/harbor-package-PACKAGE-VERSION
```

4. Set the mandatory passwords and secrets in the `harbor-data-values.yaml` file by doing one of the following:
    - To automatically generate random passwords and secrets, run:

      ```
      image_url=$(kubectl -n tkg-system get packages harbor.tanzu.vmware.com.PA
      CKAGE-VERSION -o jsonpath='{.spec.template.spec.fetch[0].imgpkgBundle.ima
      ge}')
      imgpkg pull -b $image_url -o /tmp/harbor-package-PACKAGE-VERSION

      cp /tmp/harbor-package-PACKAGE-VERSION/config/values.yaml harbor-data-val
      ues.yaml

      bash /tmp/harbor-package-PACKAGE-VERSION/config/scripts/generate-password
      s.sh harbor-data-values.yaml
      ```

      Where `PACKAGE-VERSION` is the version of the Harbor package that you want to install.

      For example, for the Harbor package v2.10.3, run:

      ```
      image_url=$(kubectl -n tkg-system get packages harbor.tanzu.vmware.com.v
      2.10.3+vmware.1-tkg.1 -o jsonpath='{.spec.template.spec.fetch[0].imgpkgBu
      ndle.image}')
      imgpkg pull -b $image_url -o /tmp/harbor-package-2.10.3
      bash /tmp/harbor-package-2.10.3/config/scripts/generate-passwords.sh harb
      or-data-values.yaml
      ```

    - To set your own passwords and secrets, update the following entries in the `harbor-data-values.yaml` file:
        - `harborAdminPassword`
        - `secretKey`
        - `database.password`
        - `core.secret`
        - `core.xsrfKey`
        - `jobservice.secret`
        - `registry.secret`
5. Specify other settings in the `harbor-data-values.yaml` file.
    - Set the `hostname` setting to the hostname you want to use to access Harbor. For example, `harbor.yourdomain.com`.
    - To use your own certificates, update the `tls.crt`, `tls.key`, and `ca.crt` settings with the contents of your certificate, key, and CA certificate. The certificate can be signed by a trusted authority or be self-signed. If you leave these blank, Tanzu Kubernetes Grid automatically generates a self-signed certificate.

- If you used the `generate-passwords.sh` script, optionally update the `harborAdminPassword` with something that is easier to remember.
- Non-empty values are required for the following:
  - `storageClass`: Under `persistence.persistentVolumeClaim`, for `registry`, `jobservice.jobLog`, `database`, `redis`, and `trivy`, set `storageClass` to a storage profile returned by `kubectl get sc`.

    With the `azure-file` storage class you cannot change filesystem permissions after the disk is mounted, because of an Azure issue described in "Could not change permissions" error while using Azure Files in the Azure documentation.

  - `pspNames`: Set `pspNames` to PSP values returned by `kubectl get psp`, for example, `"vmware-system-restricted,vmware-system-privileged"`.
- Optionally, update other `persistence` settings to specify how Harbor stores data.

  If you need to store a large quantity of container images in Harbor, set `persistence.persistentVolumeClaim.registry.size` to a larger number.

To see more information about the values in the `harbor-data-values.yaml` file, run the below command against your target cluster:

```
tanzu package available get harbor.tanzu.vmware.com/AVAILABLE-VERSION --values-
schema
```

Where `AVAILABLE-VERSION` is the version of the Harbor package. The `--values-schema` flag retrieves the `valuesSchema` section from the `Package` API resource for the Harbor package. You can set the output format, `--output`, for the values schema to `yaml`, `json`, or `table`.

For example:

```
tanzu package available get harbor.tanzu.vmware.com/v2.10.3+vmware.1-tkg.1 --va
lues-schema
```

6. Remove all comments in the `harbor-data-values.yaml` file:

```
yq -i eval '... comments=""' harbor-data-values.yaml
```

7. Install the package:

```
tanzu package install harbor \
--package harbor.tanzu.vmware.com \
--version AVAILABLE-PACKAGE-VERSION \
--values-file harbor-data-values.yaml \
--namespace TARGET-NAMESPACE
```

Where:

- `TARGET-NAMESPACE` is the namespace in which you want to install the Harbor package. For example, the `my-packages` or `tanzu-cli-managed-packages` namespace.
  - If the `--namespace` flag is not specified, the Tanzu CLI installs the package and its resources in the `default` namespace. The Harbor pods and any other resources associated with the Harbor component are created in the `tanzu-system-registry` namespace; do not install the Harbor package into this namespace.
  - The specified namespace must already exist, for example from running `kubectl create namespace my-packages`.
- `AVAILABLE-PACKAGE-VERSION` is the version that you retrieved above.

For example:

```
tanzu package install harbor \
--package harbor.tanzu.vmware.com \
--version v2.10.3+vmware.1-tkg.1 \
--values-file harbor-data-values.yaml \
--namespace my-packages
```

8. Confirm that the `harbor` package has been installed:

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

   To see more details about the package, you can also run:

   ```
   tanzu package installed get harbor --namespace PACKAGE-NAMESPACE
   ```

   Where `PACKAGE-NAMESPACE` is the namespace in which the `harbor` package is installed.

9. Confirm that the `harbor` app has been successfully reconciled in your `PACKAGE-NAMESPACE`:

   ```
   kubectl get apps -A
   ```

   If the status is not `Reconcile Succeeded`, view the full status details of the `harbor` app. Viewing the full status can help you troubleshoot the problem.

   ```
   kubectl get app harbor --namespace PACKAGE-NAMESPACE -o yaml
   ```

   Where `PACKAGE-NAMESPACE` is the namespace in which you installed the package. If troubleshooting does not help you solve the problem, you must uninstall the package before installing it again:

   ```
   tanzu package installed delete harbor --namespace PACKAGE-NAMESPACE
   ```

10. Confirm that the Harbor services are running by listing all of the pods in the cluster:

    ```
    kubectl get pods -A
    ```

    In the `tanzu-system-registry` namespace, you should see the `harbor core`, `database`, `jobservice`, `notary`, `portal`, `redis`, `registry`, and `trivy` services running in a pod with names similar to the following:

```
NAMESPACE                NAME                                  READY   STATUS
RESTARTS   AGE
[...]
tanzu-system-ingress     contour-6b568c9b88-h5s2r              1/1     Running
0          26m
tanzu-system-ingress     contour-6b568c9b88-mlg2r              1/1     Running
0          26m
tanzu-system-ingress     envoy-wfqdp                           2/2     Running
0          26m
tanzu-system-registry    harbor-core-557b58b65c-4kzhn          1/1     Running
0          23m
tanzu-system-registry    harbor-database-0                     1/1     Running
0          23m
tanzu-system-registry    harbor-jobservice-847b5c8756-t6kfs    1/1     Running
0          23m
tanzu-system-registry    harbor-notary-server-6b74b8dd56-d7swb 1/1     Running
2          23m
tanzu-system-registry    harbor-notary-signer-69d4669884-dglzm 1/1     Running
2          23m
tanzu-system-registry    harbor-portal-8f677757c-t4cbj         1/1     Running
0          23m
tanzu-system-registry    harbor-redis-0                        1/1     Running
0          23m
tanzu-system-registry    harbor-registry-85b96c7777-wsdnj      2/2     Running
0          23m
tanzu-system-registry    harbor-trivy-0                        1/1     Running
0          23m
[...]
```

11. Obtain the Harbor CA certificate from the `harbor-tls` secret in the `tanzu-system-registry` namespace:

    ```
    kubectl -n tanzu-system-registry get secret harbor-tls -o=jsonpath="{.data.ca\.
    crt}" | base64 -d
    ```

    Record the output for the following step

12. If the Harbor registry uses a self-signed CA, add it to workload clusters using the applicable procedure based on how the cluster was deployed:

    - **New clusters**: See Configure Clusters with Multiple Trusted Registries
    - **Existing clusters**: See Add Custom CA Certificate Trust to Existing Clusters

Save a copy of the `harbor-data-values.yaml` file to use as a back up, and as a reference for settings such as the secret key, password, stoarge size, and so on, when upgrading Harbor in the future.

Connect to the Harbor User Interface

The Harbor UI is exposed via the Envoy service load balancer that is running in the `tanzu-system-ingress` namespace in the cluster. To allow users to connect to the Harbor UI, you must map the address of the Envoy service load balancer to the hostname of the Harbor service, for example, `harbor.yourdomain.com`.

1. Obtain the address of the Envoy service load balancer.

   ```
   kubectl get svc envoy -n tanzu-system-ingress -o jsonpath='{.status.loadBalancer.ingress[0]}'
   ```

   On **vSphere without NSX Advanced Load Balancer (ALB)**, the Envoy service is exposed via `NodePort` instead of `LoadBalancer`, so the above output will be empty, and you can use the IP address of any worker node in the cluster instead. On **vSphere with NSX ALB**, the Envoy service has a Load Balancer IP address similar to `20.54.226.44`.

2. Map the address of the Envoy service load balancer to the hostname of the Harbor service. For clusters that are running on vSphere, you must add an IP to hostname mapping in `/etc/hosts` or add corresponding `A` records in your DNS server. For example, if the IP address is `10.93.9.100`, add the following to `/etc/hosts`:

   ```
   10.93.9.100 harbor.yourdomain.com notary.harbor.yourdomain.com
   ```

   On Windows machines, the equivalent to `/etc/hosts/` is `C:\Windows\System32\Drivers\etc\hosts`.

Users can now connect to the Harbor UI by navigating to `https://harbor.yourdomain.com` in a Web browser and log in as user `admin` with the `harborAdminPassword` that you configured in `harbor-data-values.yaml`.

Push and Pull Images to and from Harbor

Now that Harbor is set up, you can push images to it to make them available for your cluster to pull.

1. If Harbor uses a self-signed certificate, download the Harbor CA certificate from `https://harbor.yourdomain.com/api/v2.0/systeminfo/getcert` and install it on your local machine, so Docker can trust this CA certificate.
   - On Linux, save the certificate as `/etc/docker/certs.d/harbor.yourdomain.com/ca.crt`.
   - On macOS, follow this procedure.
   - On Windows, right-click the certificate file and select **Install Certificate**.
2. Log in to the Harbor registry with the user `admin`. When prompted, enter the `harborAdminPassword` that you set when you installed the Harbor package in the cluster.

   ```
   docker login harbor.yourdomain.com -u admin
   ```

3. Tag an existing image that you have already pulled locally, for example, `nginx:1.7.9`.

   ```
   docker tag nginx:1.7.9 harbor.yourdomain.com/library/nginx:1.7.9
   ```

4. Push the image to the Harbor registry.

   ```
   docker push harbor.yourdomain.com/library/nginx:1.7.9
   ```

5. Now you can pull the image from the Harbor registry on any machine where the Harbor CA certificate is installed.

   ```
   docker pull harbor.yourdomain.com/library/nginx:1.7.9
   ```

Update a Running Harbor Deployment

If you need to make changes to the configuration of the Harbor package after deployment, follow these steps to update your deployed Harbor package.

1. Update the Harbor configuration in `harbor-data-values.yaml`. For example, you can increase the amount of registry storage by updating the `persistence.persistentVolumeClaim.registry.size` value.
2. Update the configuration of the installed package:

   ```
   tanzu package installed update harbor \
   --version INSTALLED-PACKAGE-VERSION \
   --values-file harbor-data-values.yaml \
   --namespace INSTALLED-PACKAGE-NAMESPACE
   ```

   Where:

   - `INSTALLED-PACKAGE-VERSION` is the version of the installed Harbor package.
   - `INSTALLED-PACKAGE-NAMESPACE` is the namespace in which the Harbor package is installed.

   For example:

   ```
   tanzu package installed update harbor \
   --version v2.10.3+vmware.1-tkg.1 \
   --values-file harbor-data-values.yaml \
   --namespace my-packages
   ```

The Harbor package will be reconciled using the new value or values that you added. It can take up to five minutes for `kapp-controller` to apply the changes.

For more information about the `tanzu package installed update` command, see Update a Package in *Install and Manage Packages*. You can use this command to update the version and the configuration of an installed package.

Multus and Whereabouts: Container Networking

This topic gives an overview of the Multus and Whereabouts packages, and how to install them in Tanzu Kubernetes Grid (TKG) workload clusters deployed by a standalone management cluster to provide multiple container networking interface (CNI) and IP Address Management (IPAM) services for the cluster.

Multus lets you attach multiple network interfaces to single pods in a cluster, and Whereabouts lets you associate each network interface with a different address range.

- Multus CNI is an open-source container network interface plugin for Kubernetes that enables attaching multiple network interfaces to pods. See the Multus CNI repository for more information.
- Whereabouts is an open source IPAM CNI plugin you can use with Multus to assign IP addresses to pods cluster-wide dynamically. See the Whereabouts Overview for more information.

Installation: For instructions on how to install Multus and Whereabouts to implement multiple pod network interfaces, see:

- Install Multus in Workload Clusters Deployed by a Standalone Management Cluster
- Install Multus with Whereabouts on Workload Clusters Deployed by a Standalone Management Cluster

Install Multus on Workload Clusters

Multus CNI is a Container Network Interface (CNI) plugin for Kubernetes that lets you attach multiple network interfaces to a single pod and associate each with a different address range.

This documentation is applicable only to Tanzu Kubernetes Grid with management clusters. If you are using TKGS with vSphere Supervisor, see Installing Standard Packages on TKG Service Clusters.

This topic explains how to install the Multus package onto a workload cluster deployed by a standalone management cluster and use it to create pods with multiple network interfaces. For example, Antrea or Calico as the primary CNI, and a secondary interface such as macvlan or ipvlan, or SR-IOV or DPDK devices for hardware or accelerated interfaces.

Binaries for macvlan and ipvlan are already installed in the workload cluster node template.

Prerequisites

- A bootstrap machine with the following installed:

    - Tanzu CLI, Tanzu CLI plugins, and `kubectl`, as described in Install the Tanzu CLI and Kubernetes CLI for Use with Standalone Management Clusters.
    - yq v4.5 or later.
- A Tanzu Kubernetes Grid management cluster and workload cluster running on vSphere, Amazon Web Services (AWS), or Azure.
    - Multus CNI requires workload cluster worker nodes of size `large` or `extra-large`, as described in Predefined Node Sizes.

As of v2.5, TKG does not support clusters on AWS or Azure. See the End of Support for TKG Management and Workload Clusters on AWS and Azure in the *Tanzu Kubernetes Grid v2.5 Release Notes*.

Install the Multus CNI Package

Once the Multus CNI is installed in a cluster, it should not be deleted. See Deleting Multus Unsupported below.

To install the Multus CNI package on a workload cluster and configure the cluster to use it:

1. If the cluster does not have a package repository with the Multus CNI package installed, such as the `tanzu-standard` repository, install one:

    ```
    tanzu package repository add PACKAGE-REPO-NAME --url PACKAGE-REPO-ENDPOINT --na
    mespace tkg-system
    ```

    Where:

    - `PACKAGE-REPO-NAME` is the name of the package repository, such as `tanzu-standard` or the name of a private image registry configured with `ADDITIONAL_IMAGE_REGISTRY` variables.
    - `PACKAGE-REPO-ENDPOINT` is the URL of the package repository.
        - For the TKG v2.5.2 release, the `tanzu-standard` URL is `projects.packages.broadcom.com/tkg/packages/standard/repo:v2025.1.27`. See List Package Repositories to obtain this value from the Tanzu CLI, or in Tanzu Mission Control see the **Addons** > **Repositories** list in the **Cluster** pane.
2. (Optional) To configure Multus:
    1. Create a configuration file that retrieves the Multus parameters and deploys it as a Daemonset.

        ```
        tanzu package available get multus-cni.tanzu.vmware.com/PACKAGE-VERSION -
        -default-values-file-output FILE-PATH
        ```

Where `PACKAGE-VERSION` is the version of the Multus package that you want to install and `FILE-PATH` is the location to which you want to save the configuration file, for example, `multus-data-values.yaml`.

See `entrypoint.sh` parameters in the Multus CNI repository for information about the settings for the configuration file.

2. Run the `tanzu package available list` command to list the available versions of the Multus package, for example:

```
tanzu package available list multus-cni.tanzu.vmware.com -A
 NAME                          VERSION                RELEASED-AT
NAMESPACE
 multus-cni.tanzu.vmware.com v4.0.1+vmware.2-tkg.3 2021-06-04 18:00:00 +0
000 UTC tanzu-package-repo-global
 multus-cni.tanzu.vmware.com 3.8.0+vmware.1-tkg.1  2021-06-04 18:00:00 +0
000 UTC tanzu-package-repo-global
```

Make sure that your custom image registry can be reached if you are operating in a network-restricted environment.

3. Run the `tanzu package available get` command with `--values-schema` to see which field values can be set:

```
tanzu package available get multus-cni.tanzu.vmware.com/VERSION --values-
schema -o FORMAT
```

Where: - `VERSION` is a version listed in the `tanzu package available list` output - `FORMAT` is either `yaml` or `json`

4. Populate the `multus-data-values.yaml` configuration file with your desired field values.

3. Remove all comments from the `multus-data-values.yaml` file:

```
yq -i eval '... comments=""' multus-data-values.yaml
```

4. Run `tanzu package install` to install the package.

```
tanzu package install multus-cni --package multus-cni.tanzu.vmware.com --versio
n AVAILABLE-PACKAGE-VERSION --values-file multus-data-values.yaml --namespace T
ARGET-NAMESPACE
```

Where:

- `TARGET-NAMESPACE` is the namespace in which you want to install the Multus package. For example, the `my-packages` or `tanzu-cli-managed-packages` namespace.
  - If the `--namespace` flag is not specified, the Tanzu CLI installs the package in the `default` namespace.
  - The specified namespace must already exist, for example from running `kubectl create namespace my-packages`.

- `AVAILABLE-PACKAGE-VERSION` is the version that you retrieved above, for example `4.0.1+vmware.2-tkg.3`.

5. Run `tanzu package installed get` to check the status of the installed package.

```
tanzu package installed get multus-cni --namespace NAMESPACE
```

6. Create a custom resource definition (CRD) for `NetworkAttachmentDefinition` that defines the CNI configuration for network interfaces to be used by Multus CNI.
    1. Create a CRD specification. For example, this `multus-cni-crd.yaml` specifies a `NetworkAttachmentDefinition` named `macvlan-conf` that configures a `macvlan` CNI:

```
---
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: macvlan-conf
spec:
  config: '{
    "cniVersion": "0.3.0",
    "type": "macvlan",
    "master": "ens5",
    "mode": "bridge",
    "ipam": {
      "type": "host-local",
      "subnet": "192.168.1.0/24",
      "rangeStart": "192.168.1.200",
      "rangeEnd": "192.168.1.216",
      "routes": [
        { "dst": "0.0.0.0/0" }
        ],
      "gateway": "192.168.1.1"
    }
  }'
```

    2. Create the resource; for example `kubectl create -f multus-cni-crd.yaml`

7. Create a pod with the annotation `k8s.v1.cni.cncf.io/networks`, which takes a comma-delimited list of the names of `NetworkAttachmentDefinition` custom resource.
    1. Create the pod specification, for example `my-multi-cni-pod.yaml`:

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: macvlan-conf
spec:
  securityContext:
    runAsUser: 1000
    runAsGroup: 3000
    fsGroup: 2000
    seccompProfile:
      type: RuntimeDefault
    runAsNonRoot: true
```

```
  containers:
  - name: sample-pod
    image: harbor-repo.vmware.com/dockerhub-proxy-cache/library/busybox:
1.28
    command: [ "sh", "-c", "sleep 1h" ]
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop:
        - ALL
```

2. Create the pod; for example `kubectl create -f my-multi-cni-crd.yaml` creates the pod `sample-pod`.

Once the pod is created, it will have three network interfaces:

- `lo` the loopback interface
- `eth0` the default pod network managed by Antrea or Calico CNI
- `net1` the new interface created via the annotation `k8s.v1.cni.cncf.io/networks: macvlan-conf`.

The default network gets the name `eth0` and additional network pod interfaces get the name as `net1`, `net2`, and so on.

Validating Multus

Run `kubectl describe pod` on the pod, and confirm that the annotation `k8s.v1.cni.cncf.io/network-status` lists all network interfaces. For example:

```
$ kubectl describe pod sample-pod

Name:         sample-pod
Namespace:    default
Priority:     0
Node:         tcecluster-md-0-6476897f75-rl9vt/10.170.109.225
Start Time:   Thu, 27 May 2021 15:31:20 +0000
Labels:       <none>
Annotations:  k8s.v1.cni.cncf.io/network-status:
                [{
                    "name": "",
                    "interface": "eth0",
                    "ips": [
                        "100.96.1.80"
                    ],
                    "mac": "66:39:dc:63:50:a3",
                    "default": true,
                    "dns": {}
                },{
                    "name": "default/macvlan-conf",
                    "interface": "net1",
                    "ips": [
                        "192.168.1.201"
                    ],
```

```
                "mac": "02:77:cb:a0:60:e3",
                "dns": {}
            }]
        k8s.v1.cni.cncf.io/networks: macvlan-conf
```

Then run `kubectl exec sample-pod -- ip a show dev net1` to check if the target interface is up and running with IP listed in annotations above.

Deleting Multus Unsupported

Once the Multus CNI is installed in a cluster, it should not be deleted.

Deleting Multus does not uninstall the the Multus configuration file `/etc/cni/net.d/00-multus.conf` from the CNI scripts directory, which causes issues such as:

- Failure to create new pods. This is a known issue; see Issue #461 in the Multus repository.
- Failure to delete pods created with before Multus is deleted. Pods remain stuck with status `Terminating` with errors in `kubelet` log.

Install Multus with Whereabouts on Workload Clusters

Whereabouts is an IP Address Management (IPAM) CNI plugin that dynamically assigns IP addresses to pods across all the nodes in a cluster. Compared to other IPAM plugins such as `host-local`, which only assigns IP addresses to pods on the same node, `whereabouts` assigns IP addresses cluster-wide.

This documentation is applicable only to Tanzu Kubernetes Grid with management clusters. If you are using TKGS with vSphere Supervisor, see Installing Standard Packages on TKG Service Clusters.

In this topic, we will show how you can attach a secondary network interface to a pod with IP address assigned in the range you specified using `whereabouts`. For example, you will have Antrea or Calico as the primary CNI, a secondary interface created using macvlan or ipvlan, and Whereabouts as the IPAM CNI.

As of v2.5, TKG does not support clusters on AWS or Azure. See the End of Support for TKG Management and Workload Clusters on AWS and Azure in the *Tanzu Kubernetes Grid v2.5 Release Notes*.

Prerequisites

- A bootstrap machine with the following installed:
    - Multus CNI package to support multiple networks. See Install Multus in Workload Clusters Deployed by a Standalone Management Cluster for more information.
    - Tanzu CLI, Tanzu CLI plugins, and `kubectl`, as described in Install the Tanzu CLI and Kubernetes CLI for Use with Standalone Management Clusters.

    - yq v4.5 or later.
- A Tanzu Kubernetes Grid management cluster and a workload cluster running on vSphere, Amazon Web Services (AWS), or Azure.
    - Whereabouts requires workload cluster worker nodes of size `large` or `extra-large`, as described in Predefined Node Sizes.

Install updated Whereabouts version from Tanzu Standard v2025.1.27 (TKG 2.5.2 only)

Tanzu Standard v2025.1.27 includes a new version of the Whereabouts package. How you install it depends on whether or not you installed the previous version of Whereabouts on your clusters.

Whereabouts is already installed

If you installed Whereabouts with Tanzu Standard Packages v2024.8.21 on TKG 2.5.2 clusters, you must update Whereabouts to the version provided in the Tanzu Standard Packages v2025.1.27 release.

1. Add the new package repository in the running workload cluster by running:

    ```
    tanzu package repository add alternate-repo -n tkg-system --url projects.packag
    es.broadcom.com/tkg/packages/standard/repo:v2025.1.27
    ```

2. Check the available packages by running:

    ```
    tanzu -n tkg-system package available get whereabouts.tanzu.vmware.com
    ```

    You should see the new version of the Whereabouts package in the output:

    ```
    0.7.0+vmware.2-tkg.1  2025-01-27 13:44:33 +0000 UTC
    0.7.0+vmware.2-tkg.2  2025-01-27 13:44:33 +0000 UTC
    ```

3. If v0.7.0+vmware.2-tkg.1 is already installed on the cluster, before updating it, create a new YAML configuration named `whereabouts-config.yaml`, containing the new toleration settings:

    ```
    whereabouts:
      tolerations:
      - key: key1
        operator: Equal
        value: value1
        effect: NoSchedule
    ```

4. Update the Whereabouts package by using the configuration file by running:

    ```
    tanzu -n tkg-system package installed update whereabouts --values-file whereabo
    uts-config.yaml --version 0.7.0+vmware.2-tkg.2
    ```

Whereabouts is not already installed

If the version of Whereabouts from Tanzu Standard Packages v2024.8.21 is not already installed, and you require the default `NoExecute` toleration, run:

```
tanzu -n tkg-system package install whereabouts -p whereabouts.tanzu.vmware.com --vers
ion 0.7.0+vmware.2-tkg.1
```

Install the Whereabouts Package

To install Whereabouts package on a workload cluster and configure the cluster to use it:

1. Configure and install Whereabouts.
    1. Create a configuration file that retrieves the Whereabouts parameters.

        ```
        tanzu package available get whereabouts.tanzu.vmware.com/PACKAGE-VERSION
        --default-values-file-output FILE-PATH
        ```

        The namespace on which to deploy `whereabouts` components must be `kube-system`. Any customized namespace installation will fail.

        Where `PACKAGE-VERSION` is the version of the Whereabouts package that you want to install and `FILE-PATH` is the location to which you want to save the configuration file, for example, `whereabouts-data-values.yaml`. The above command will create a configuration file named `whereabouts-data-values.yaml` containing the default values.

    2. Run the `tanzu package available list` command to list the available versions of the Whereabouts package, for example:

        ```
        tanzu package available list whereabouts.tanzu.vmware.com -A
         NAME                           VERSION               RELEASED-AT
        NAMESPACE
         whereabouts.tanzu.vmware.com   0.6.3+vmware.2-tkg.1  2023-30-04 18:00:00
        +0000 UTC   tanzu-package-repo-global
        ```

        Make sure that your custom image registry can be reached if you are operating in a network-restricted environment.

    3. Run the `tanzu package available get` command with `--values-schema` to see which field values can be set:

        ```
        tanzu package available get whereabouts.tanzu.vmware.com/VERSION --values
        -schema -o FORMAT
        ```

        Where: - `VERSION` is a version listed in the `tanzu package available list` output - `FORMAT` is either `yaml` or `json`

    4. Populate the `whereabouts-data-values.yaml` configuration file with your desired field values.

2. Remove all comments from the `whereabouts-data-values.yaml` file:

```
yq -i eval '... comments=""' whereabouts-cni-default-values.yaml
```

3. Run `tanzu package install` to install the package.

```
tanzu package install whereabouts --package whereabouts.tanzu.vmware.com --vers
ion AVAILABLE-PACKAGE-VERSION --values-file whereabouts-data-values.yaml --name
space TARGET-NAMESPACE
```

   Where:

   - `TARGET-NAMESPACE` is the namespace in which you want to install the Whereabouts package. For example, the `my-packages` or `tanzu-cli-managed-packages` namespace.
     - If the `--namespace` flag is not specified, the Tanzu CLI installs the package in the `default` namespace.
     - The specified namespace must already exist, for example from running `kubectl create namespace my-packages`.
   - `AVAILABLE-PACKAGE-VERSION` is the version that you retrieved above, for example `0.6.3+vmware.2-tkg.1`.

4. Run `tanzu package installed get` to check the status of the installed package.

```
tanzu package installed get whereabouts -o <json|yaml|table>
```

5. Create a custom resource definition (CRD) for `NetworkAttachmentDefinition` that defines the CNI configuration for network interfaces to be used by Multus CNI with Whereabouts as the IPAM type.

   1. Create a CRD specification. For example, this `multus-cni-crd.yaml` specifies a `NetworkAttachmentDefinition` named `macvlan-conf` that configures a `macvlan` CNI and has `whereabouts` as the IPAM type.:

```
---
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: macvlan-conf
spec:
  config: '{
    "cniVersion": "0.3.1",
    "plugins": [
      {
      "type": "macvlan",
      "capabilities": { "ips": true },
      "master": "eth0",
      "mode": "bridge",
      "ipam": {
        "type": "whereabouts",
        "range": "192.168.20.0/24",
        "range_start": "192.168.20.10",
        "range_end": "192.168.20.100",
        "gateway": "192.168.20.1"
        }
```

```
      } ]
    }'
```

2. Create the resource; for example `kubectl create -f multus-cni-crd.yaml`

6. Create a pod with the annotation `k8s.v1.cni.cncf.io/networks` to specify the additional network to add.

    1. Create the pod specification; for example, `my-multi-cni-pod.yaml`:

    ```
    apiVersion: v1
    kind: Pod
    metadata:
      name: pod0
      annotations:
        k8s.v1.cni.cncf.io/networks: macvlan-conf
    spec:
      containers:
      - name: pod0
        command: ["/bin/ash", "-c", "trap : TERM INT; sleep infinity & wait"]
        image: docker.io/kserving/tools:latest
    ```

    2. Create the pod. For example, `kubectl create -f my-multi-cni-crd.yaml` creates the pod `pod0`.

    3. Once the pod is created, it will have three network interfaces:

        - `lo` the loopback interface
        - `eth0` the default pod network managed by Antrea or Calico CNI
        - `net1` the new interface created via the annotation `k8s.v1.cni.cncf.io/networks: macvlan-conf`.

        The default network gets the name `eth0` and additional network pod interfaces get the name as `net1`, `net2`, and so on.

    4. Run `kubectl describe pod` on `pod0`, and confirm that the annotation `k8s.v1.cni.cncf.io/network-status` lists all network interfaces.

        For example:

    ```
    $ kubectl describe pod pod0

    Name:         pod0
    Namespace:    default
    Priority:     0
    Node:         tcecluster-md-0-6476897f75-rl9vt/10.170.109.225
    Start Time:   Thu, 25 August 2022 15:31:20 +0000
    Labels:       <none>
    Annotations:  k8s.v1.cni.cncf.io/network-status:
                   [{
                        "name": "",
                        "interface": "eth0",
    ```

```
                    "ips": [
                        "100.96.1.80"
                    ],
                    "mac": "66:39:dc:63:50:a3",
                    "default": true,
                    "dns": {}
                },{
                    "name": "default/macvlan-conf",
                    "interface": "net1",
                    "ips": [
                        "192.168.20.11"
                    ],
                    "mac": "02:77:cb:a0:60:e3",
                    "dns": {}
                }]
            k8s.v1.cni.cncf.io/networks: macvlan-conf
```

Then run `kubectl exec sample-pod -- ip a show dev net1` to check if the target interface is up and running with IP listed in annotations above. Repeat this step to validate the configuration for other pods that you will create in the next step.

7. Create two additional pods, `pod1` on the same node and `pod2` on a different node. We will use these pods to verify network access within a single node and nodes across the cluster.

Validate Network Traffic between Pods Accross the Cluster

You can now check network access between pods in the same node and pods across the cluster.

1. Verify that the network access between pods on the same node works. For example, the following command verifies that `pod0` can reach `pod1` via its assigned IP address.

```
  kubectl exec -it pod0 -- ping -c 3 192.168.20.12
  PING 192.168.20.12 (192.168.20.12) 56(84) bytes of data.
  64 bytes from 192.168.20.12: icmp_seq=1 ttl=64 time=0.237 ms
  64 bytes from 192.168.20.12: icmp_seq=2 ttl=64 time=0.215 ms
  64 bytes from 192.168.20.12: icmp_seq=3 ttl=64 time=0.156 ms

--- 192.168.20.12 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2030ms
rtt min/avg/max/mdev = 0.156/0.202/0.237/0.037 ms
```

2. Verify that the network access between pods on different node works. For example, the following command verifies that `pod0` can reach `pod2` via its assigned IP address.

```
  kubectl exec -it pod0 -- ping -c 3 <192.168.20.13
  PING 192.168.20.13 (192.168.20.13) 56(84) bytes of data.
  64 bytes from 192.168.20.13: icmp_seq=1 ttl=64 time=0.799 ms
  64 bytes from 192.168.20.13: icmp_seq=2 ttl=64 time=0.626 ms
  64 bytes from 192.168.20.13: icmp_seq=3 ttl=64 time=0.655 ms

--- 192.168.20.13 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2030ms
rtt min/avg/max/mdev = 0.626/0.693/0.799/0.078 ms
```

Prometheus and Grafana: Monitoring

This topic gives an overview of the Prometheus and Grafana packages, and how you can install them in Tanzu Kubernetes Grid (TKG) workload clusters to provide monitoring services for the cluster.

- **Prometheus** is an open-source systems monitoring and alerting toolkit. It can collect metrics from target clusters at specified intervals, evaluate rule expressions, display the results, and trigger alerts if certain conditions arise. For more information about Prometheus, see the Prometheus Overview. The Tanzu Kubernetes Grid implementation of Prometheus includes Alert Manager, which you can configure to notify you when certain events occur.
- **Grafana** is open-source visualization and analytics software. It allows you to query, visualize, alert on, and explore your metrics no matter where they are stored. In other words, Grafana provides you with tools to turn your time-series database (TSDB) data into high-quality graphs and visualizations. For more information about Grafana, see What is Grafana?.

See Install Prometheus in Workload Clusters Deployed by a Standalone Management Cluster and Install Grafana in Workload Clusters Deployed by a Standalone Management Cluster.

As of v2.5, TKG does not support clusters on AWS or Azure. See the End of Support for TKG Management and Workload Clusters on AWS and Azure in the *Tanzu Kubernetes Grid v2.5 Release Notes*.

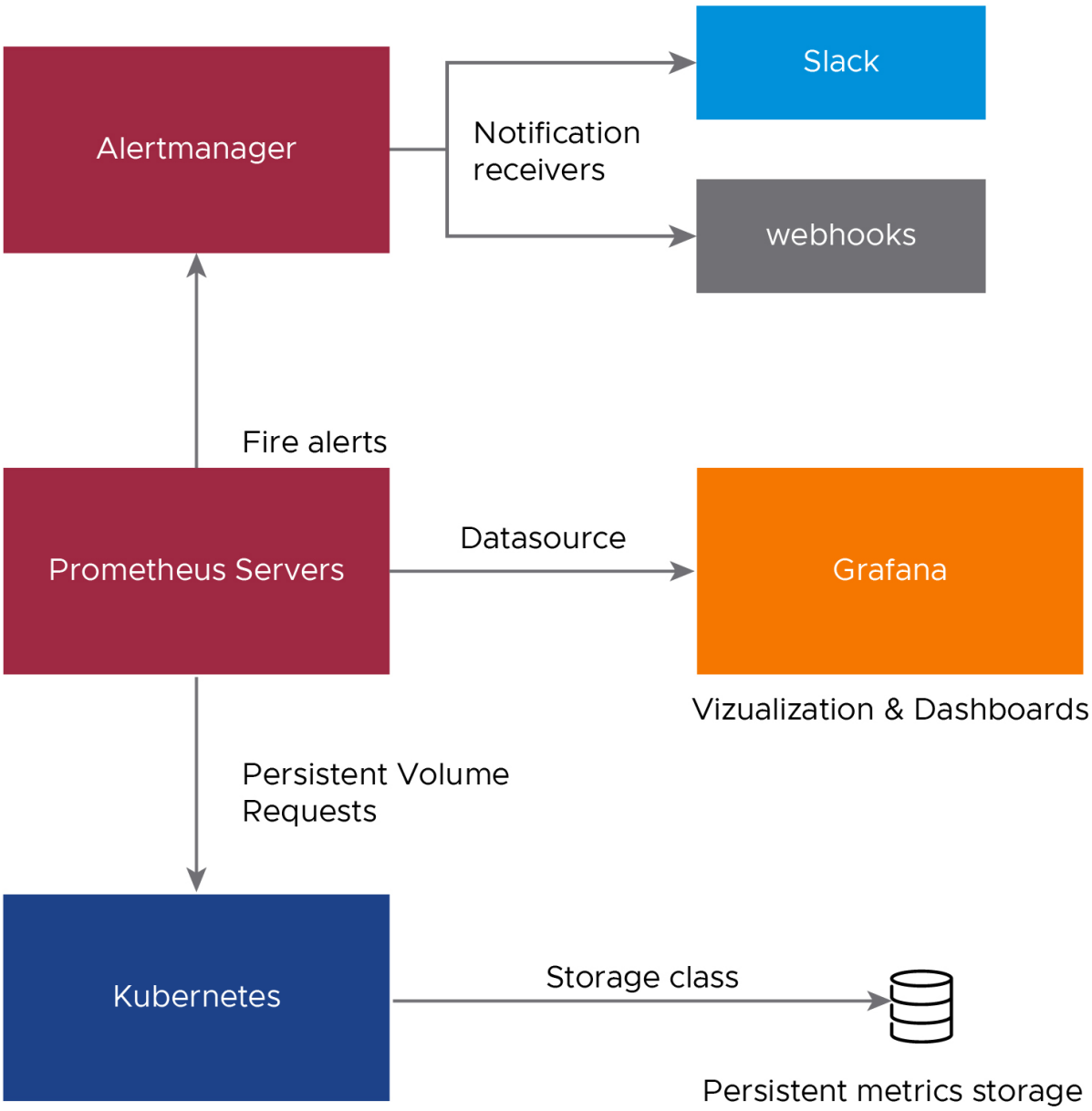
Prometheus, Alertmanager, and Grafana Components, Configuration, Data Values

Prometheus collects metrics from configured targets at given intervals, evaluates rule expressions, and displays the results.

Alertmanager triggers alerts if some condition is observed to be true.

Grafana lets you query, visualize, alert on, and explore your metrics no matter where they are stored.

The following diagram shows how the monitoring components on a cluster interact.



Prometheus Components

The Prometheus package installs on a TKG cluster the containers listed in the table. The package pulls the containers from the VMware public registry specified in Package Repository.

Container	Resource Type	Replicas	Description
`prometheus-alertmanager`	Deployment	1	Handles alerts sent by client applications such as the Prometheus server.
`prometheus-cadvisor`	DaemonSet	5	Analyzes and exposes resource usage and performance data from running containers
`prometheus-kube-state-metrics`	Deployment	1	Monitors node status and capacity, replica-set compliance, pod, job, and cronjob status, resource requests and limits.

Container	Resource Type	Replicas	Description
prometheus-node-exporter	DaemonSet	5	Exporter for hardware and OS metrics exposed by kernels.
prometheus-pushgateway	Deployment	1	Service that allows you to push metrics from jobs which cannot be scraped.
prometheus-server	Deployment	1	Provides core functionality, including scraping, rule processing, and alerting.

Prometheus Data Values

Below is an example `prometheus-data-values.yaml` file.

Note the following:

- Ingress is enabled (ingress: enabled: true).
- Ingress is configured for URLs ending in /alertmanager/ (alertmanager_prefix:) and / (prometheus_prefix:).
- The FQDN for Prometheus is `prometheus.system.tanzu` (virtual_host_fqdn:).
- Supply your own custom certificate in the Ingress section (tls.crt, tls.key, ca.crt).
- The pvc for alertmanager is 2GiB. Supply the `storageClassName` for the default storage policy.
- The pvc for prometheus is 20GiB. Supply the `storageClassName` for the vSphere storage policy.

```
namespace: prometheus-monitoring
alertmanager:
  config:
    alertmanager_yml: |
      global: {}
      receivers:
      - name: default-receiver
      templates:
      - '/etc/alertmanager/templates/*.tmpl'
      route:
        group_interval: 5m
        group_wait: 10s
        receiver: default-receiver
        repeat_interval: 3h
  deployment:
    replicas: 1
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    updateStrategy: Recreate
  pvc:
    accessMode: ReadWriteOnce
    storage: 2Gi
    storageClassName: default
  service:
    port: 80
    targetPort: 9093
    type: ClusterIP
```

```
ingress:
  alertmanager_prefix: /alertmanager/
  alertmanagerServicePort: 80
  enabled: true
  prometheus_prefix: /
  prometheusServicePort: 80
  tlsCertificate:
    ca.crt: |
      -----BEGIN CERTIFICATE-----
      MIIFczCCA1ugAwIBAgIQTYJITQ3SZ4BBS9UzXfJIuTANBgkqhkiG9w0BAQsFADBM
      ...
      w0oGuTTBfxSMKs767N3G1q5tz0mwFpIqIQtXUSmaJ+9p7IkpWcThLnyYYo1IpWm/
      ZHtjzZMQVA==
      -----END CERTIFICATE-----
    tls.crt: |
      -----BEGIN CERTIFICATE-----
      MIIHxTCCBa2gAwIBAgITIgAAAAQnSpH7QfxTKAAAAAAABDANBgkqhkiG9w0BAQsF
      ...
      YYsIjp7/f+Pk1DjzWx8JIAbzItKLucDreAmmDXqk+DrBP9LYqtmjB0n7nSErgK8G
      sA3kGCJdOkI0kgF10gsinaouG2jVlwNOsw==
      -----END CERTIFICATE-----
    tls.key: |
      -----BEGIN PRIVATE KEY-----
      MIIJRAIBADANBgkqhkiG9w0BAQEFAASCCS4wggkqAgEAAoICAQDOGHT8I12KyQGS
      ...
      l1NzswracGQIzo03zk/X3Z6P2YOea4BkZ0Iwh34wOHJnTkfEeSx6y+oSFMcFRthT
      yfFCZUk/sVCc/C1a4VigczXftUGiRrTR
      -----END PRIVATE KEY-----
  virtual_host_fqdn: prometheus.system.tanzu
kube_state_metrics:
  deployment:
    replicas: 1
  service:
    port: 80
    targetPort: 8080
    telemetryPort: 81
    telemetryTargetPort: 8081
    type: ClusterIP
node_exporter:
  daemonset:
    hostNetwork: false
    updatestrategy: RollingUpdate
  service:
    port: 9100
    targetPort: 9100
    type: ClusterIP
prometheus:
  pspNames: "vmware-system-restricted"
  config:
    alerting_rules_yml: |
      {}
    alerts_yml: |
      {}
    prometheus_yml: |
      global:
        evaluation_interval: 1m
        scrape_interval: 1m
        scrape_timeout: 10s
      rule_files:
```

```
      - /etc/config/alerting_rules.yml
      - /etc/config/recording_rules.yml
      - /etc/config/alerts
      - /etc/config/rules
      scrape_configs:
      - job_name: 'prometheus'
        scrape_interval: 5s
        static_configs:
        - targets: ['localhost:9090']
      - job_name: 'kube-state-metrics'
        static_configs:
        - targets: ['prometheus-kube-state-metrics.prometheus.svc.cluster.local:8080']

      - job_name: 'node-exporter'
        static_configs:
        - targets: ['prometheus-node-exporter.prometheus.svc.cluster.local:9100']

      - job_name: 'kubernetes-pods'
        kubernetes_sd_configs:
        - role: pod
        relabel_configs:
        - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_scrape]
          action: keep
          regex: true
        - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_path]
          action: replace
          target_label: __metrics_path__
          regex: (.+)
        - source_labels: [__address__, __meta_kubernetes_pod_annotation_prometheus_io_
port]
          action: replace
          regex: ([^:]+)(?::\d+)?;(\d+)
          replacement: $1:$2
          target_label: __address__
        - action: labelmap
          regex: __meta_kubernetes_pod_label_(.+)
        - source_labels: [__meta_kubernetes_namespace]
          action: replace
          target_label: kubernetes_namespace
        - source_labels: [__meta_kubernetes_pod_name]
          action: replace
          target_label: kubernetes_pod_name
      - job_name: kubernetes-nodes-cadvisor
        kubernetes_sd_configs:
        - role: node
        relabel_configs:
        - action: labelmap
          regex: __meta_kubernetes_node_label_(.+)
        - replacement: kubernetes.default.svc:443
          target_label: __address__
        - regex: (.+)
          replacement: /api/v1/nodes/$1/proxy/metrics/cadvisor
          source_labels:
          - __meta_kubernetes_node_name
          target_label: __metrics_path__
        scheme: https
        tls_config:
          ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
          insecure_skip_verify: true
```

```
        bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      - job_name: kubernetes-apiservers
        kubernetes_sd_configs:
        - role: endpoints
        relabel_configs:
        - action: keep
          regex: default;kubernetes;https
          source_labels:
          - __meta_kubernetes_namespace
          - __meta_kubernetes_service_name
          - __meta_kubernetes_endpoint_port_name
        scheme: https
        tls_config:
          ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
          insecure_skip_verify: true
        bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      alerting:
        alertmanagers:
        - scheme: http
          static_configs:
          - targets:
            - alertmanager.prometheus.svc:80
        - kubernetes_sd_configs:
            - role: pod
          relabel_configs:
          - source_labels: [__meta_kubernetes_namespace]
            regex: default
            action: keep
          - source_labels: [__meta_kubernetes_pod_label_app]
            regex: prometheus
            action: keep
          - source_labels: [__meta_kubernetes_pod_label_component]
            regex: alertmanager
            action: keep
          - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_probe]
            regex: .*
            action: keep
          - source_labels: [__meta_kubernetes_pod_container_port_number]
            regex:
            action: drop
    recording_rules_yml: |
      groups:
        - name: kube-apiserver.rules
          interval: 3m
          rules:
          - expr: |2
              (
                (
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-ap
iservers",verb=~"LIST|GET"}[1d]))
                -
                (
                  (
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernet
es-apiservers",verb=~"LIST|GET",scope=~"resource|",le="0.1"}[1d]))
                    or
                    vector(0)
                  )
                  +
```

```
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes
-apiservers",verb=~"LIST|GET",scope="namespace",le="0.5"}[1d]))
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes
-apiservers",verb=~"LIST|GET",scope="cluster",le="5"}[1d]))
                  )
                )
                +
                # errors
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LI
ST|GET",code=~"5.."}[1d]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LIST
|GET"}[1d]))
            labels:
              verb: read
            record: apiserver_request:burnrate1d
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-ap
iservers",verb=~"LIST|GET"}[1h]))
                  -
                  (
                    (
                      sum(rate(apiserver_request_duration_seconds_bucket{job="kubernet
es-apiservers",verb=~"LIST|GET",scope=~"resource|",le="0.1"}[1h]))
                      or
                      vector(0)
                    )
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes
-apiservers",verb=~"LIST|GET",scope="namespace",le="0.5"}[1h]))
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes
-apiservers",verb=~"LIST|GET",scope="cluster",le="5"}[1h]))
                  )
                )
                +
                # errors
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LI
ST|GET",code=~"5.."}[1h]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LIST
|GET"}[1h]))
            labels:
              verb: read
            record: apiserver_request:burnrate1h
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-ap
iservers",verb=~"LIST|GET"}[2h]))
                  -
                  (
```

```
                    (
                      sum(rate(apiserver_request_duration_seconds_bucket{job="kubernet
es-apiservers",verb=~"LIST|GET",scope=~"resource|",le="0.1"}[2h]))
                      or
                      vector(0)
                    )
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes
-apiservers",verb=~"LIST|GET",scope="namespace",le="0.5"}[2h]))
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes
-apiservers",verb=~"LIST|GET",scope="cluster",le="5"}[2h]))
                  )
                )
                +
                # errors
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LI
ST|GET",code=~"5.."}[2h]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LIST
|GET"}[2h]))
            labels:
              verb: read
            record: apiserver_request:burnrate2h
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-ap
iservers",verb=~"LIST|GET"}[30m]))
                  -
                  (
                    (
                      sum(rate(apiserver_request_duration_seconds_bucket{job="kubernet
es-apiservers",verb=~"LIST|GET",scope=~"resource|",le="0.1"}[30m]))
                      or
                      vector(0)
                    )
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes
-apiservers",verb=~"LIST|GET",scope="namespace",le="0.5"}[30m]))
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes
-apiservers",verb=~"LIST|GET",scope="cluster",le="5"}[30m]))
                  )
                )
                +
                # errors
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LI
ST|GET",code=~"5.."}[30m]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LIST
|GET"}[30m]))
            labels:
              verb: read
            record: apiserver_request:burnrate30m
          - expr: |2
```

```
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-ap
iservers",verb=~"LIST|GET"}[3d]))
                  -
                  (
                    (
                      sum(rate(apiserver_request_duration_seconds_bucket{job="kubernet
es-apiservers",verb=~"LIST|GET",scope=~"resource|",le="0.1"}[3d]))
                      or
                      vector(0)
                    )
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes
-apiservers",verb=~"LIST|GET",scope="namespace",le="0.5"}[3d]))
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes
-apiservers",verb=~"LIST|GET",scope="cluster",le="5"}[3d]))
                  )
                )
                +
                # errors
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LI
ST|GET",code=~"5.."}[3d]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LIST
|GET"}[3d]))
            labels:
              verb: read
            record: apiserver_request:burnrate3d
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-ap
iservers",verb=~"LIST|GET"}[5m]))
                  -
                  (
                    (
                      sum(rate(apiserver_request_duration_seconds_bucket{job="kubernet
es-apiservers",verb=~"LIST|GET",scope=~"resource|",le="0.1"}[5m]))
                      or
                      vector(0)
                    )
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes
-apiservers",verb=~"LIST|GET",scope="namespace",le="0.5"}[5m]))
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes
-apiservers",verb=~"LIST|GET",scope="cluster",le="5"}[5m]))
                  )
                )
                +
                # errors
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LI
ST|GET",code=~"5.."}[5m]))
              )
```

```
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LIST
|GET"}[5m]))
            labels:
              verb: read
            record: apiserver_request:burnrate5m
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-ap
iservers",verb=~"LIST|GET"}[6h]))
                  -
                  (
                    (
                      sum(rate(apiserver_request_duration_seconds_bucket{job="kubernet
es-apiservers",verb=~"LIST|GET",scope=~"resource|",le="0.1"}[6h]))
                      or
                      vector(0)
                    )
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes
-apiservers",verb=~"LIST|GET",scope="namespace",le="0.5"}[6h]))
                    +
                    sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes
-apiservers",verb=~"LIST|GET",scope="cluster",le="5"}[6h]))
                  )
                )
                +
                # errors
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LI
ST|GET",code=~"5.."}[6h]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"LIST
|GET"}[6h]))
            labels:
              verb: read
            record: apiserver_request:burnrate6h
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-ap
iservers",verb=~"POST|PUT|PATCH|DELETE"}[1d]))
                  -
                  sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-a
piservers",verb=~"POST|PUT|PATCH|DELETE",le="1"}[1d]))
                )
                +
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"PO
ST|PUT|PATCH|DELETE",code=~"5.."}[1d]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"POST
|PUT|PATCH|DELETE"}[1d]))
            labels:
              verb: write
            record: apiserver_request:burnrate1d
```

```
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-ap
iservers",verb=~"POST|PUT|PATCH|DELETE"}[1h]))
                  -
                  sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-a
piservers",verb=~"POST|PUT|PATCH|DELETE",le="1"}[1h]))
                )
                +
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"PO
ST|PUT|PATCH|DELETE",code=~"5.."}[1h]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"POST
|PUT|PATCH|DELETE"}[1h]))
            labels:
              verb: write
            record: apiserver_request:burnrate1h
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-ap
iservers",verb=~"POST|PUT|PATCH|DELETE"}[2h]))
                  -
                  sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-a
piservers",verb=~"POST|PUT|PATCH|DELETE",le="1"}[2h]))
                )
                +
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"PO
ST|PUT|PATCH|DELETE",code=~"5.."}[2h]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"POST
|PUT|PATCH|DELETE"}[2h]))
            labels:
              verb: write
            record: apiserver_request:burnrate2h
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-ap
iservers",verb=~"POST|PUT|PATCH|DELETE"}[30m]))
                  -
                  sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-a
piservers",verb=~"POST|PUT|PATCH|DELETE",le="1"}[30m]))
                )
                +
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"PO
ST|PUT|PATCH|DELETE",code=~"5.."}[30m]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"POST
|PUT|PATCH|DELETE"}[30m]))
            labels:
              verb: write
```

```
            record: apiserver_request:burnrate30m
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-ap
iservers",verb=~"POST|PUT|PATCH|DELETE"}[3d]))
                  -
                  sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-a
piservers",verb=~"POST|PUT|PATCH|DELETE",le="1"}[3d]))
                )
                +
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"PO
ST|PUT|PATCH|DELETE",code=~"5.."}[3d]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"POST
|PUT|PATCH|DELETE"}[3d]))
            labels:
              verb: write
            record: apiserver_request:burnrate3d
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-ap
iservers",verb=~"POST|PUT|PATCH|DELETE"}[5m]))
                  -
                  sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-a
piservers",verb=~"POST|PUT|PATCH|DELETE",le="1"}[5m]))
                )
                +
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"PO
ST|PUT|PATCH|DELETE",code=~"5.."}[5m]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"POST
|PUT|PATCH|DELETE"}[5m]))
            labels:
              verb: write
            record: apiserver_request:burnrate5m
          - expr: |2
              (
                (
                  # too slow
                  sum(rate(apiserver_request_duration_seconds_count{job="kubernetes-ap
iservers",verb=~"POST|PUT|PATCH|DELETE"}[6h]))
                  -
                  sum(rate(apiserver_request_duration_seconds_bucket{job="kubernetes-a
piservers",verb=~"POST|PUT|PATCH|DELETE",le="1"}[6h]))
                )
                +
                sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"PO
ST|PUT|PATCH|DELETE",code=~"5.."}[6h]))
              )
              /
              sum(rate(apiserver_request_total{job="kubernetes-apiservers",verb=~"POST
|PUT|PATCH|DELETE"}[6h]))
            labels:
```

```
              verb: write
            record: apiserver_request:burnrate6h
          - expr: |
              sum by (code,resource) (rate(apiserver_request_total{job="kubernetes-api
servers",verb=~"LIST|GET"}[5m]))
            labels:
              verb: read
            record: code_resource:apiserver_request_total:rate5m
          - expr: |
              sum by (code,resource) (rate(apiserver_request_total{job="kubernetes-api
servers",verb=~"POST|PUT|PATCH|DELETE"}[5m]))
            labels:
              verb: write
            record: code_resource:apiserver_request_total:rate5m
          - expr: |
              histogram_quantile(0.99, sum by (le, resource) (rate(apiserver_request_d
uration_seconds_bucket{job="kubernetes-apiservers",verb=~"LIST|GET"}[5m]))) > 0
            labels:
              quantile: "0.99"
              verb: read
            record: cluster_quantile:apiserver_request_duration_seconds:histogram_quan
tile
          - expr: |
              histogram_quantile(0.99, sum by (le, resource) (rate(apiserver_request_d
uration_seconds_bucket{job="kubernetes-apiservers",verb=~"POST|PUT|PATCH|DELETE"}[5
m]))) > 0
            labels:
              quantile: "0.99"
              verb: write
            record: cluster_quantile:apiserver_request_duration_seconds:histogram_quan
tile
          - expr: |2
              sum(rate(apiserver_request_duration_seconds_sum{subresource!="log",verb!
~"LIST|WATCH|WATCHLIST|DELETECOLLECTION|PROXY|CONNECT"}[5m])) without(instance, pod)
              /
              sum(rate(apiserver_request_duration_seconds_count{subresource!="log",ver
b!~"LIST|WATCH|WATCHLIST|DELETECOLLECTION|PROXY|CONNECT"}[5m])) without(instance, pod)
            record: cluster:apiserver_request_duration_seconds:mean5m
          - expr: |
              histogram_quantile(0.99, sum(rate(apiserver_request_duration_seconds_buc
ket{job="kubernetes-apiservers",subresource!="log",verb!~"LIST|WATCH|WATCHLIST|DELETEC
OLLECTION|PROXY|CONNECT"}[5m])) without(instance, pod))
            labels:
              quantile: "0.99"
            record: cluster_quantile:apiserver_request_duration_seconds:histogram_quan
tile
          - expr: |
              histogram_quantile(0.9, sum(rate(apiserver_request_duration_seconds_buck
et{job="kubernetes-apiservers",subresource!="log",verb!~"LIST|WATCH|WATCHLIST|DELETECO
LLECTION|PROXY|CONNECT"}[5m])) without(instance, pod))
            labels:
              quantile: "0.9"
            record: cluster_quantile:apiserver_request_duration_seconds:histogram_quan
tile
          - expr: |
              histogram_quantile(0.5, sum(rate(apiserver_request_duration_seconds_buck
et{job="kubernetes-apiservers",subresource!="log",verb!~"LIST|WATCH|WATCHLIST|DELETECO
LLECTION|PROXY|CONNECT"}[5m])) without(instance, pod))
            labels:
```

```
              quantile: "0.5"
            record: cluster_quantile:apiserver_request_duration_seconds:histogram_quan
tile
        - interval: 3m
          name: kube-apiserver-availability.rules
          rules:
          - expr: |2
              1 - (
                (
                  # write too slow
                  sum(increase(apiserver_request_duration_seconds_count{verb=~"POST|PU
T|PATCH|DELETE"}[30d]))
                  -
                  sum(increase(apiserver_request_duration_seconds_bucket{verb=~"POST|P
UT|PATCH|DELETE",le="1"}[30d]))
                ) +
                (
                  # read too slow
                  sum(increase(apiserver_request_duration_seconds_count{verb=~"LIST|GE
T"}[30d]))
                  -
                  (
                    (
                      sum(increase(apiserver_request_duration_seconds_bucket{verb=~"LI
ST|GET",scope=~"resource|",le="0.1"}[30d]))
                      or
                      vector(0)
                    )
                    +
                    sum(increase(apiserver_request_duration_seconds_bucket{verb=~"LIST
|GET",scope="namespace",le="0.5"}[30d]))
                    +
                    sum(increase(apiserver_request_duration_seconds_bucket{verb=~"LIST
|GET",scope="cluster",le="5"}[30d]))
                  )
                ) +
                # errors
                sum(code:apiserver_request_total:increase30d{code=~"5.."} or vector
(0))
              )
              /
              sum(code:apiserver_request_total:increase30d)
            labels:
              verb: all
            record: apiserver_request:availability30d
          - expr: |2
              1 - (
                sum(increase(apiserver_request_duration_seconds_count{job="kubernetes-
apiservers",verb=~"LIST|GET"}[30d]))
                -
                (
                  # too slow
                  (
                    sum(increase(apiserver_request_duration_seconds_bucket{job="kubern
etes-apiservers",verb=~"LIST|GET",scope=~"resource|",le="0.1"}[30d]))
                    or
                    vector(0)
                  )
                  +
```

```
                  sum(increase(apiserver_request_duration_seconds_bucket{job="kubernet
es-apiservers",verb=~"LIST|GET",scope="namespace",le="0.5"}[30d]))
                  +
                  sum(increase(apiserver_request_duration_seconds_bucket{job="kubernet
es-apiservers",verb=~"LIST|GET",scope="cluster",le="5"}[30d]))
                )
                +
                # errors
                sum(code:apiserver_request_total:increase30d{verb="read",code=~"5.."}
or vector(0))
              )
              /
              sum(code:apiserver_request_total:increase30d{verb="read"})
            labels:
              verb: read
            record: apiserver_request:availability30d
          - expr: |2
              1 - (
                (
                  # too slow
                  sum(increase(apiserver_request_duration_seconds_count{verb=~"POST|PU
T|PATCH|DELETE"}[30d]))
                  -
                  sum(increase(apiserver_request_duration_seconds_bucket{verb=~"POST|P
UT|PATCH|DELETE",le="1"}[30d]))
                )
                +
                # errors
                sum(code:apiserver_request_total:increase30d{verb="write",code=~"5.."}
or vector(0))
              )
              /
              sum(code:apiserver_request_total:increase30d{verb="write"})
            labels:
              verb: write
            record: apiserver_request:availability30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-ap
iservers",verb="LIST",code=~"2.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-ap
iservers",verb="GET",code=~"2.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-ap
iservers",verb="POST",code=~"2.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-ap
iservers",verb="PUT",code=~"2.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-ap
iservers",verb="PATCH",code=~"2.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-ap
iservers",verb="DELETE",code=~"2.."}[30d]))
```

```
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-ap
iservers",verb="LIST",code=~"3.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-ap
iservers",verb="GET",code=~"3.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-ap
iservers",verb="POST",code=~"3.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-ap
iservers",verb="PUT",code=~"3.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-ap
iservers",verb="PATCH",code=~"3.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-ap
iservers",verb="DELETE",code=~"3.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-ap
iservers",verb="LIST",code=~"4.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-ap
iservers",verb="GET",code=~"4.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-ap
iservers",verb="POST",code=~"4.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-ap
iservers",verb="PUT",code=~"4.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-ap
iservers",verb="PATCH",code=~"4.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-ap
iservers",verb="DELETE",code=~"4.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-ap
iservers",verb="LIST",code=~"5.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-ap
iservers",verb="GET",code=~"5.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-ap
```

```
iservers",verb="POST",code=~"5.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-ap
iservers",verb="PUT",code=~"5.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-ap
iservers",verb="PATCH",code=~"5.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code, verb) (increase(apiserver_request_total{job="kubernetes-ap
iservers",verb="DELETE",code=~"5.."}[30d]))
            record: code_verb:apiserver_request_total:increase30d
          - expr: |
              sum by (code) (code_verb:apiserver_request_total:increase30d{verb=~"LIST
|GET"})
            labels:
              verb: read
            record: code:apiserver_request_total:increase30d
          - expr: |
              sum by (code) (code_verb:apiserver_request_total:increase30d{verb=~"POST
|PUT|PATCH|DELETE"})
            labels:
              verb: write
            record: code:apiserver_request_total:increase30d
    rules_yml: |
      {}
  deployment:
    configmapReload:
      containers:
        args:
          - --volume-dir=/etc/config
          - --webhook-url=http://127.0.0.1:9090/-/reload
    containers:
      args:
        - --storage.tsdb.retention.time=42d
        - --config.file=/etc/config/prometheus.yml
        - --storage.tsdb.path=/data
        - --web.console.libraries=/etc/prometheus/console_libraries
        - --web.console.templates=/etc/prometheus/consoles
        - --web.enable-lifecycle
    replicas: 1
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    updateStrategy: Recreate
  pvc:
    accessMode: ReadWriteOnce
    storage: 20Gi
    storageClassName: default
  service:
    port: 80
    targetPort: 9090
    type: ClusterIP
pushgateway:
  deployment:
    replicas: 1
  service:
```

```
    port: 9091
    targetPort: 9091
    type: ClusterIP
```

Prometheus Configuration Parameters (Standalone MC)

The following table lists configuration parameters of the Prometheus package and describes their default values.

You can set the following configuration values in your `prometheus-data-values.yaml` file.

To review and edit the Prometheus package's current configuration parameters and values, retrieve its values schema:

```
tanzu package available get prometheus.tanzu.vmware.com/2.45.0+vmware.2-tkg.2 -n AVAIL
ABLE-PACKAGE-NAMESPACE --values-schema
```

Parameter	Description	Type	Default
`namespace`	Namespace where Prometheus will be deployed.	String	`tanzu-system-monitoring`
`prometheus.deployment.replicas`	Number of Prometheus replicas.	String	`1`
`prometheus.deployment.containers.args`	Prometheus container arguments. You can configure this parameter to change retention time. For information about configuring Prometheus storage parameters, see the Prometheus documentation. **Note** Longer retention times require more storage capacity than shorter retention times. It might be necessary to increase the persistent volume claim size if you are significantly increasing the retention time.	List	*n/a*
`prometheus.deployment.containers.resources`	Prometheus container resource requests and limits.	Map	`{}`
`prometheus.deployment.podAnnotations`	The Prometheus deployments pod annotations.	Map	`{}`
`prometheus.deployment.podLabels`	The Prometheus deployments pod labels.	Map	`{}`
`prometheus.deployment.configMapReload.containers.args`	Configmap-reload container arguments.	List	*n/a*
`prometheus.deployment.configMapReload.containers.resources`	Configmap-reload container resource requests and limits.	Map	`{}`
`prometheus.service.type`	Type of service to expose Prometheus. Supported Values: `ClusterIP`.	String	`ClusterIP`

Parameter	Description	Type	Default
`prometheus.service.port`	Prometheus service port.	Integer	`80`
`prometheus.service.targetPort`	Prometheus service target port.	Integer	`9090`
`prometheus.service.labels`	Prometheus service labels.	Map	`{}`
`prometheus.service.annotations`	Prometheus service annotations.	Map	`{}`
`prometheus.pvc.annotations`	Storage class annotations.	Map	`{}`
`prometheus.pvc.storageClassName`	Storage class to use for persistent volume claim. By default this is null and default provisioner is used.	String	`null`
`prometheus.pvc.accessMode`	Define access mode for persistent volume claim. Supported values: `ReadWriteOnce`, `ReadOnlyMany`, `ReadWriteMany`.	String	`ReadWriteOnce`
`prometheus.pvc.storage`	Define storage size for persistent volume claim.	String	`150Gi`
`prometheus.config.prometheus_yml`	For information about the global Prometheus configuration, see the Prometheus documentation.	YAML file	`prometheus.yaml`
`prometheus.config.alerting_rules_yml`	For information about the Prometheus alerting rules, see the Prometheus documentation.	YAML file	`alerting_rules.yaml`
`prometheus.config.recording_rules_yml`	For information about the Prometheus recording rules, see the Prometheus documentation.	YAML file	`recording_rules.yaml`
`prometheus.config.alerts_yml`	Additional prometheus alerting rules are configured here.	YAML file	`alerts_yml.yaml`
`prometheus.config.rules_yml`	Additional prometheus recording rules are configured here.	YAML file	`rules_yml.yaml`
`alertmanager.deployment.replicas`	Number of alertmanager replicas.	Integer	`1`
`alertmanager.deployment.containers.resources`	Alertmanager container resource requests and limits.	Map	`{}`
`alertmanager.deployment.podAnnotations`	The Alertmanager deployments pod annotations.	Map	`{}`
`alertmanager.deployment.podLabels`	The Alertmanager deployments pod labels.	Map	`{}`
`alertmanager.service.type`	Type of service to expose Alertmanager. Supported Values: `ClusterIP`.	String	`ClusterIP`
`alertmanager.service.port`	Alertmanager service port.	Integer	`80`

Parameter	Description	Type	Default
`alertmanager.service.targetPort`	Alertmanager service target port.	Integer	`9093`
`alertmanager.service.labels`	Alertmanager service labels.	Map	`{}`
`alertmanager.service.annotations`	Alertmanager service annotations.	Map	`{}`
`alertmanager.pvc.annotations`	Storage class annotations.	Map	`{}`
`alertmanager.pvc.storageClassName`	Storage class to use for persistent volume claim. By default this is null and default provisioner is used.	String	`null`
`alertmanager.pvc.accessMode`	Define access mode for persistent volume claim. Supported values: `ReadWriteOnce`, `ReadOnlyMany`, `ReadWriteMany`.	String	`ReadWriteOnce`
`alertmanager.pvc.storage`	Define storage size for persistent volume claim.	String	`2Gi`
`alertmanager.config.alertmanager_yml`	For information about the global YAML configuration for Alert Manager, see the Prometheus documentation.	YAML file	`alertmanager_yml`
`kube_state_metrics.deployment.replicas`	Number of kube-state-metrics replicas.	Integer	`1`
`kube_state_metrics.deployment.containers.resources`	kube-state-metrics container resource requests and limits.	Map	`{}`
`kube_state_metrics.deployment.podAnnotations`	The kube-state-metrics deployments pod annotations.	Map	`{}`
`kube_state_metrics.deployment.podLabels`	The kube-state-metrics deployments pod labels.	Map	`{}`
`kube_state_metrics.service.type`	Type of service to expose kube-state-metrics. Supported Values: `ClusterIP`.	String	`ClusterIP`
`kube_state_metrics.service.port`	kube-state-metrics service port.	Integer	`80`
`kube_state_metrics.service.targetPort`	kube-state-metrics service target port.	Integer	`8080`
`kube_state_metrics.service.telemetryPort`	kube-state-metrics service telemetry port.	Integer	`81`
`kube_state_metrics.service.telemetryTargetPort`	kube-state-metrics service target telemetry port.	Integer	`8081`

Parameter	Description	Type	Default
`kube_state_metrics.service.labels`	kube-state-metrics service labels.	Map	`{}`
`kube_state_metrics.service.annotations`	kube-state-metrics service annotations.	Map	`{}`
`node_exporter.daemonset.replicas`	Number of node-exporter replicas.	Integer	`1`
`node_exporter.daemonset.containers.resources`	node-exporter container resource requests and limits.	Map	`{}`
`node_exporter.daemonset.hostNetwork`	Host networking requested for this pod.	boolean	`false`
`node_exporter.daemonset.podAnnotations`	The node-exporter deployments pod annotations.	Map	`{}`
`node_exporter.daemonset.podLabels`	The node-exporter deployments pod labels.	Map	`{}`
`node_exporter.service.type`	Type of service to expose node-exporter. Supported Values: `ClusterIP`.	String	`ClusterIP`
`node_exporter.service.port`	node-exporter service port.	Integer	`9100`
`node_exporter.service.targetPort`	node-exporter service target port.	Integer	`9100`
`node_exporter.service.labels`	node-exporter service labels.	Map	`{}`
`node_exporter.service.annotations`	node-exporter service annotations.	Map	`{}`
`pushgateway.deployment.replicas`	Number of pushgateway replicas.	Integer	`1`
`pushgateway.deployment.containers.resources`	pushgateway container resource requests and limits.	Map	`{}`
`pushgateway.deployment.podAnnotations`	The pushgateway deployments pod annotations.	Map	`{}`
`pushgateway.deployment.podLabels`	The pushgateway deployments pod labels.	Map	`{}`
`pushgateway.service.type`	Type of service to expose pushgateway. Supported Values: `ClusterIP`.	String	`ClusterIP`
`pushgateway.service.port`	pushgateway service port.	Integer	`9091`

Parameter	Description	Type	Default
`pushgateway.service.targetPort`	pushgateway service target port.	Integer	`9091`
`pushgateway.service.labels`	pushgateway service labels.	Map	`{}`
`pushgateway.service.annotations`	pushgateway service annotations.	Map	`{}`
`cadvisor.daemonset.replicas`	Number of cadvisor replicas.	Integer	`1`
`cadvisor.daemonset.containers.resources`	cadvisor container resource requests and limits.	Map	`{}`
`cadvisor.daemonset.podAnnotations`	The cadvisor deployments pod annotations.	Map	`{}`
`cadvisor.daemonset.podLabels`	The cadvisor deployments pod labels.	Map	`{}`
`ingress.enabled`	Activate/Deactivate ingress for prometheus and alertmanager.	Boolean	`false`
`ingress.virtual_host_fqdn`	Hostname for accessing promethues and alertmanager.	String	`prometheus.system.tanzu`
`ingress.prometheus_prefix`	Path prefix for prometheus.	String	`/`
`ingress.alertmanager_prefix`	Path prefix for alertmanager.	String	`/alertmanager/`
`ingress.prometheusServicePort`	Prometheus service port to proxy traffic to.	Integer	`80`
`ingress.alertmanagerServicePort`	Alertmanager service port to proxy traffic to.	Integer	`80`
`ingress.tlsCertificate.tls.crt`	Optional certificate for ingress if you want to use your own TLS certificate. A self signed certificate is generated by default. **Note** `tls.crt` is a key and not nested.	String	`Generated cert`
`ingress.tlsCertificate.tls.key`	Optional certificate private key for ingress if you want to use your own TLS certificate. **Note** `tls.key` is a key and not nested.	String	`Generated cert key`
`ingress.tlsCertificate.ca.crt`	Optional CA certificate. **Note** `ca.crt` is a key and not nested.	String	`CA certificate`

Prometheus Server Configuration Parameters

You can set the following fields in the Prometheus Server `ConfigMap`.

Parameter	Description	Type	Default
evaluation_interval	frequency to evaluate rules	duration	1m
scrape_interval	frequency to scrape targets	duration	1m
scrape_timeout	How long until a scrape request times out	duration	10s
rule_files	Rule files specifies a list of globs. Rules and alerts are read from all matching files	yaml file	??
scrape_configs	A list of scrape configurations.	list	??
job_name	The job name assigned to scraped metrics by default	string	??
kubernetes_sd_configs	List of Kubernetes service discovery configurations.	list	??
relabel_configs	List of target relabel configurations.	list	??
action	Action to perform based on regex matching.	string	??
regex	Regular expression against which the extracted value is matched.	string	??
source_labels	The source labels select values from existing labels.	string	??
scheme	Configures the protocol scheme used for requests.	string	??
tls_config	Configures the scrape request's TLS settings.	string	??
ca_file	CA certificate to validate API server certificate with.	filename	??
insecure_skip_verify	Disable validation of the server certificate.	boolean	??
bearer_token_file	Optional bearer token file authentication information.	filename	??
replacement	Replacement value against which a regex replace is performed if the regular expression matches.	string	??
target_label	Label to which the resulting value is written in a replace action.	string	??

Alert Manager Configuration Parameters

You can set the following fields in the Alert Manager `ConfigMap`.

Parameter	Description	Type	Default
resolve_timeout	ResolveTimeout is the default value used by alertmanager if the alert does not include EndsAt	duration	5m
smtp_smarthost	The SMTP host through which emails are sent.	duration	1m
slack_api_url	The Slack webhook URL.	string	global.slack_api_url

Parameter	Description	Type	Default
pagerduty_url	The pagerduty URL to send API requests to.	string	global.pagerduty_url
templates	Files from which custom notification template definitions are read	file path	??
group_by	group the alerts by label	string	??
group_interval	set time to wait before sending a notification about new alerts that are added to a group	duration	5m
group_wait	How long to initially wait to send a notification for a group of alerts	duration	30s
repeat_interval	How long to wait before sending a notification again if it has already been sent successfully for an alert	duration	4h
receivers	A list of notification receivers.	list	??
severity	Severity of the incident.	string	??
channel	The channel or user to send notifications to.	string	??
html	The HTML body of the email notification.	string	??
text	The text body of the email notification.	string	??
send_resolved	Whether or not to notify about resolved alerts.	filename	??
email_configs	Configurations for email integration	boolean	??

Prometheus Pod Annotations

Annotations on pods allow a fine control of the scraping process. These annotations must be part of the pod metadata. They will have no effect if set on other objects such as Services or DaemonSets.

Pod Annotation	Description
`prometheus.io/scrape`	The default configuration will scrape all pods and, if set to false, this annotation will exclude the pod from the scraping process.
`prometheus.io/path`	If the metrics path is not /metrics, define it with this annotation.
`prometheus.io/port`	Scrape the pod on the indicated port instead of the pod???s declared ports (default is a port-free target if none are declared).

The DaemonSet manifest below will instruct Prometheus to scrape all of its pods on port 9102.

```
apiVersion: apps/v1beta2 # for versions before 1.8.0 use extensions/v1beta1
kind: DaemonSet
metadata:
  name: fluentd-elasticsearch
  namespace: weave
  labels:
    app: fluentd-logging
```

```
spec:
  selector:
    matchLabels:
      name: fluentd-elasticsearch
  template:
    metadata:
      labels:
        name: fluentd-elasticsearch
      annotations:
        prometheus.io/scrape: 'true'
        prometheus.io/port: '9102'
    spec:
      containers:
      - name: fluentd-elasticsearch
        image: gcr.io/google-containers/fluentd-elasticsearch:1.20
```

Grafana Components

The Grafana package installs on the cluster the container listed in the table. For more information, see https://grafana.com/. The Grafana package pulls the container from the VMware public registry specified in Package Repository.

Container	Resource Type	Replicas	Description
Grafana	Deployment	2	Data visualization

Grafana Data Values

Below is an example `grafana-data-values.yaml` file with the following customizations:

- ingress is enabled (ingress: enabled: true)
- ingress is configured for URLs ending in / (prefix:)
- The FQDN for Grafana is grafana.system.tanzu (virtual_host_fqdn:)
- The pvc for grafana is 2GB and will be created under the default vSphere storageClass
- The secret: admin_user and admin_password are for the Grafana UI and both values must be base64 encoded (in the example, “admin” is used for each and base64 encoded; you should use secure credentials for your installation)

```
namespace: grafana-dashboard
grafana:
  pspNames: "vmware-system-restricted"
  deployment:
    replicas: 1
    updateStrategy: Recreate
  pvc:
    accessMode: ReadWriteOnce
    storage: 2Gi
    storageClassName: default
  secret:
    admin_user: YWRtaW4=
    admin_password: YWRtaW4=
    type: Opaque
  service:
```

```
    port: 80
    targetPort: 3000
    type: LoadBalancer
ingress:
  enabled: true
  prefix: /
  servicePort: 80
  virtual_host_fqdn: grafana.system.tanzu
```

Grafana Configuration Parameters (Standalone MC)

The following table lists configuration parameters of the Grafana package and describes their default values.

You can set the following configuration values in your `grafana-data-values.yaml` file.

To review and edit the Grafana package's current configuration parameters and values, retrieve its values schema:

```
tanzu package available get grafana.tanzu.vmware.com/10.0.1+vmware.2-tkg.1 -n AVAILABL
E-PACKAGE-NAMESPACE --values-schema
```

Parameter	Description	Type	Default
namespace	Namespace where Grafana will be deployed.	String	`tanzu-system-dashboards`
grafana.deployment.replicas	Number of Grafana replicas.	Integer	`1`
grafana.deployment.containers.resources	Grafana container resource requests and limits.	Map	`{}`
grafana.deployment.k8sSidecar.containers.resources	k8s-sidecar container resource requests and limits.	Map	`{}`
grafana.deployment.podAnnotations	The Grafana deployments pod annotations.	Map	`{}`
grafana.deployment.podLabels	The Grafana deployments pod labels.	Map	`{}`
grafana.service.type	Type of service to expose Grafana. Supported Values: `ClusterIP`, `NodePort`, `LoadBalancer`. (For vSphere set this to `NodePort`)	String	`LoadBalancer`
grafana.service.port	Grafana service port.	Integer	`80`
grafana.service.targetPort	Grafana service target port.	Integer	`9093`
grafana.service.labels	Grafana service labels.	Map	`{}`
grafana.service.annotations	Grafana service annotations.	Map	`{}`
grafana.config.grafana_ini	For information about Grafana configuration, see Grafana Configuration Defaults in GitHub.	Config file	`grafana.ini`

Parameter	Description	Type	Default
grafana.config.datasource_yaml	For information about datasource config, see the Grafana documentation.	String	`prometheus`
grafana.config.dashboardProvider_yaml	For information about dashboard provider config, see the Grafana documentation.	YAML file	`provider.yaml`
grafana.pvc.annotations	Storage class to use for persistent volume claim. By default this is null and default provisioner is used.	String	`null`
grafana.pvc.storageClassName	Storage class to use for persistent volume claim. By default this is null and default provisioner is used.	String	`null`
grafana.pvc.accessMode	Define access mode for persistent volume claim. Supported values: `ReadWriteOnce`, `ReadOnlyMany`, `ReadWriteMany`.	String	`ReadWriteOnce`
grafana.pvc.storage	Define storage size for persistent volume claim.	String	`2Gi`
grafana.secret.type	Secret type defined for Grafana dashboard.	String	`Opaque`
grafana.secret.admin_user	Base64-encoded username to access Grafana dashboard. Defaults to `YWRtaW4=`, which is equivalent to `admin` in plain text.	String	`YWRtaW4=`
grafana.secret.admin_password	Base64-encoded password to access Grafana dashboard. Defaults to `YWRtaW4=`, which is equivalent to `admin` in plain text.	String	`YWRtaW4=`
ingress.enabled	Activate/Deactivate ingress for grafana.	Boolean	`true`
ingress.virtual_host_fqdn	Hostname for accessing grafana.	String	`grafana.system.tanzu`
ingress.prefix	Path prefix for grafana.	String	`/`
ingress.servicePort	Grafana service port to proxy traffic to.	Integer	`80`
ingress.tlsCertificate.tls.crt	Optional certificate for ingress if you want to use your own TLS cert. A self signed certificate is generated by default. **Note** `tls.crt` is a key and not nested.	String	`Generated cert`
ingress.tlsCertificate.tls.key	Optional certificate private key for ingress if you want to use your own TLS certificate. **Note** `tls.key` is a key and not nested.	String	`Generated cert private key`
ingress.tlsCertificate.ca.crt	Optional CA certificate. **Note** `ca.crt` is a key and not nested.	String	`CA certificate`

Install Prometheus (Standalone Management Cluster)

This topic explains how to deploy Prometheus into a workload cluster. The procedures below apply to vSphere, Amazon Web Services (AWS), and Azure deployments.

- This documentation is applicable only to Tanzu Kubernetes Grid with management clusters. If you are using TKGS with vSphere Supervisor, see Installing Standard Packages on TKG Service Clusters.

- As of v2.5, TKG does not support clusters on AWS or Azure. See the End of Support for TKG Management and Workload Clusters on AWS and Azure in the *Tanzu Kubernetes Grid v2.5 Release Notes*.

Prometheus

Prometheus is an open-source systems monitoring and alerting toolkit. Tanzu Kubernetes Grid includes signed binaries for Prometheus that you can deploy on workload clusters to monitor cluster health and services.

Prerequisites

- A bootstrap machine with the following installed:
  - Tanzu CLI, Tanzu CLI plugins, and `kubectl`, as described in Install the Tanzu CLI and Kubernetes CLI for Use with Standalone Management Clusters.
  - yq v4.5 or later.
- A management cluster with the Tanzu Standard package repository installed.
- A workload cluster created by the management cluster above, with the following packages installed:
  - Prometheus; see Deploy Prometheus on Workload Clusters.
  - Contour; see Install Contour for Ingress Control.
  - cert-manager; see Install Cert Manager for Certificate Management.
- (Optional) If ingress for Prometheus is enabled, you must install the `cert-manager` and `contour` packages.

Support for IPv6 addresses in Tanzu Kubernetes Grid is limited; see Deploy Clusters on IPv6 (vSphere Only). If you are not deploying to an IPv6-only networking environment, you must provide IPv4 addresses in the following steps.

Install updated Prometheus version from Tanzu Standard v2025.1.27 (TKG 2.5.2 only)

Tanzu Standard v2025.1.27 includes a new version of the Prometheus package. How you install it depends on whether or not you installed the previous version of Prometheus on your clusters.

Prometheus is already installed

If you installed Prometheus with Tanzu Standard Packages v2024.8.21 on TKG 2.5.2 clusters, you must update Prometheus to the version provided in the Tanzu Standard Packages v2025.1.27 release.

1. Add the new package repository in the running workload cluster by running:

```
tanzu package repository add alternate-repo -n tkg-system --url projects.packag
es.broadcom.com/tkg/packages/standard/repo:v2025.1.27
```

2. Check the available packages by running:

```
tanzu -n tkg-system package available get prometheus.tanzu.vmware.com
```

You should see the new version of the Prometheus package in the output:

```
2.45.0+vmware.2-tkg.1  2023-03-21 18:00:00 +0000 UTC
2.45.0+vmware.2-tkg.2  2023-03-21 18:00:00 +0000 UTC
```

3. If v2.45.0+vmware.2-tkg.1 is already installed on the cluster, before updating it, create a new YAML configuration named `prom-config.yaml`, containing the new toleration settings:

```
node_exporter:
 daemonset:
   tolerations:
   - key: value2
     effect: NoSchedule
     operator: Exists
```

4. Update the Prometheus package by using the configuration file by running:

```
tanzu -n tkg-system package installed update prometheus \
--values-file prom-config.yaml --version 2.45.0+vmware.2-tkg.2
```

Prometheus is not already installed

If the version of Prometheus from Tanzu Standard Packages v2024.8.21 is not already installed, and you require the default `NoExecute` toleration, run:

```
tanzu -n tkg-system package install prometheus -p prometheus.tanzu.vmware.com --versio
n 2.45.0+vmware.2-tkg.2
```

Prepare the Workload Cluster for Prometheus Deployment

To prepare the cluster:

1. Get the admin credentials of the workload cluster into which you want to deploy Prometheus. For example:

```
tanzu cluster kubeconfig get my-cluster --admin
```

2. Set the context of kubectl to the cluster. For example:

```
kubectl config use-context my-cluster-admin@my-cluster
```

(Optional) Enable Ingress for Prometheus

To enable ingress, you can install the below optional packages:

1. Install Cert Manager. For information, see Install Cert Manager for Certificate Management.
2. Install Contour. For information, see Install Contour for Ingress control.

Continue to Deploy Prometheus into the Workload Cluster below.

Deploy Prometheus into the Workload Cluster

To install Prometheus:

1. If the cluster does not have a package repository with the Prometheus package installed, such as the `tanzu-standard` repository, install one:

   ```
   tanzu package repository add PACKAGE-REPO-NAME --url PACKAGE-REPO-ENDPOINT --na
   mespace tkg-system
   ```

   Where:

   - `PACKAGE-REPO-NAME` is the name of the package repository, such as `tanzu-standard` or the name of a private image registry configured with `ADDITIONAL_IMAGE_REGISTRY` variables.
   - `PACKAGE-REPO-ENDPOINT` is the URL of the package repository.
     - For the TKG v2.5.2 release, the `tanzu-standard` URL is `projects.packages.broadcom.com/tkg/packages/standard/repo:v2025.1.27`. See List Package Repositories to obtain this value from the Tanzu CLI, or in Tanzu Mission Control see the **Addons** > **Repositories** list in the **Cluster** pane.
2. Confirm that the Prometheus package is available in your workload cluster:

   ```
   tanzu package available list -A
   ```

3. Retrieve the version of the available package:

   ```
   tanzu package available list prometheus.tanzu.vmware.com -A
   | Retrieving package versions for prometheus.tanzu.vmware.com...
    NAME                           VERSION                        RELEASED-AT
   NAMESPACE
    prometheus.tanzu.vmware.com    2.45.0+vmware.2-tkg.2          2020-11-24T18:
   00:00Z  tanzu-package-repo-global
   ```

When you are ready to deploy Prometheus, you can:

- Deploy Prometheus with Default Configurations
- Deploy Prometheus with Custom Values

Deploy Prometheus with Default Configurations

After you confirm the package version and retrieve it, you can install the package.

1. Install the Prometheus package using its default values:

   ```
   tanzu package install prometheus \
   --package prometheus.tanzu.vmware.com \
   --version AVAILABLE-PACKAGE-VERSION \
   --namespace TARGET-NAMESPACE
   ```

   Where:

- `TARGET-NAMESPACE` is the namespace in which you want to install the Prometheus package. For example, the `my-packages` or `tanzu-cli-managed-packages` namespace.
  - If the `--namespace` flag is not specified, the Tanzu CLI uses the `default` namespace. The Prometheus pods and any other resources associated with the Prometheus component are created in the `tanzu-system-monitoring` namespace; do not install the Prometheus package into this namespace.
  - The specified namespace must already exist, for example from running `kubectl create namespace my-packages`.
- `AVAILABLE-PACKAGE-VERSION` is the version that you retrieved above, for example `2.45.0+vmware.2-tkg.2`.

For example:

```
tanzu package install prometheus --package prometheus.tanzu.vmware.com --namesp
ace my-packages --version 2.45.0+vmware.2-tkg.2
```

```
\ Installing package 'prometheus.tanzu.vmware.com'
| Getting package metadata for 'prometheus.tanzu.vmware.com'
| Creating service account 'prometheus-my-packages-sa'
| Creating cluster admin role 'prometheus-my-packages-cluster-role'
| Creating cluster role binding 'prometheus-my-packages-cluster-rolebinding'
- Creating package resource
\ Package install status: Reconciling

Added installed package 'prometheus' in namespace 'my-packages'
```

Continue to Verify Prometheus Deployment below.

Deploy Prometheus with Custom Values

To install the Prometheus package using user-provided values:

1. Create a configuration file. This file configures the Prometheus package.

   ```
   tanzu package available get prometheus.tanzu.vmware.com/PACKAGE-VERSION --defau
   lt-values-file-output FILE-PATH
   ```

   Where `PACKAGE-VERSION` is the version of the Prometheus package that you want to install and `FILE-PATH` is the location to which you want to save the configuration file, for example, `prometheus-data-values.yaml`. The above command creates a configuration file named `prometheus-data-values.yaml` containing the default values. Note that in the previous versions, this file was called `prometheus-data-values.yaml`.

   See Prometheus Configuration Parameters (Standalone MC) for a full list of available parameters.

2. After you make any changes needed to your `prometheus-data-values.yaml` file, remove all comments in it:

   ```
   yq -i eval '... comments=""' prometheus-data-values.yaml
   ```

3. Deploy the package:

```
tanzu package install prometheus \
--package prometheus.tanzu.vmware.com \
--version PACKAGE-VERSION \
--values-file prometheus-data-values.yaml \
--namespace TARGET-NAMESPACE
```

Where:

- `TARGET-NAMESPACE` is the namespace in which you want to install the Prometheus package, Prometheus package app, and any other Kubernetes resources that describe the package. For example, the `my-packages` or `tanzu-cli-managed-packages` namespace. If the `--namespace` flag is not specified, the Tanzu CLI uses the `default` namespace. The Prometheus pods and any other resources associated with the Prometheus component are created in the `tanzu-system-monitoring` namespace; do not install the Prometheus package into this namespace.
- `PACKAGE-VERSION` is the version that you retrieved above, for example `2.45.0+vmware.2-tkg.2`.

Continue to Verify Prometheus Deployment below.

Verify Prometheus Deployment

After you deploy Prometheus, you can verify that the deployment is successful:

1. Confirm that the Prometheus package is installed. For example:

```
tanzu package installed list -A
/ Retrieving installed packages...
NAME            PACKAGE-NAME                        PACKAGE-VERSION
STATUS                  NAMESPACE
cert-manager    cert-manager.tanzu.vmware.com       1.12.2+vmware.1-tkg.2
Reconcile succeeded     my-packages
prometheus      prometheus.tanzu.vmware.com         2.45.0+vmware.2-tkg.2
Reconcile succeeded     my-packages
antrea          antrea.tanzu.vmware.com
Reconcile succeeded     tkg-system
metrics-server  metrics-server.tanzu.vmware.com
Reconcile succeeded     tkg-system
vsphere-cpi     vsphere-cpi.tanzu.vmware.com
Reconcile succeeded     tkg-system
vsphere-csi     vsphere-csi.tanzu.vmware.com
Reconcile succeeded     tkg-system
```

   The `prometheus` package and the `prometheus` app are installed in the namespace that you specify when running the `tanzu package install` command.

2. Confirm that the `prometheus` app is successfully reconciled:

```
kubectl get apps -A
```

   For example:

```
NAMESPACE     NAME                                DESCRIPTION           SINCE-D
EPLOY   AGE
```

```
my-packages   cert-manager                          Reconcile succeeded   74s
29m
my-packages   prometheus                            Reconcile succeeded   20s
33m
tkg-system    antrea                                Reconcile succeeded   70s
3h43m
[...]
```

If the status is not `Reconcile succeeded`, view the full status details of the `prometheus` app. Viewing the full status can help you troubleshoot the problem:

```
kubectl get app prometheus --namespace PACKAGE-NAMESPACE -o yaml
```

Where `PACKAGE-NAMESPACE` is the namespace in which you installed the package.

3. Confirm that the new services are running by listing all of the pods that are running in the cluster:

```
kubectl get pods -A
```

In the `tanzu-system-monitoring` namespace, you should see the `prometheus`, `alertmanager`, `node_exporter`, `pushgateway`, `cadvisor` and `kube_state_metrics` services running in a pod:

```
NAMESPACE               NAME                                        READY   STATUS
RESTARTS   AGE
[...]
tanzu-system-monitoring   alertmanager-d6bb4d94d-7fgmb
1/1     Running   0          35m
tanzu-system-monitoring   prometheus-cadvisor-pgfck
1/1     Running   0          35m
tanzu-system-monitoring   prometheus-kube-state-metrics-868b5b749d-9w5f2
1/1     Running   0          35m
tanzu-system-monitoring   prometheus-node-exporter-97x6c
1/1     Running   0          35m
tanzu-system-monitoring   prometheus-node-exporter-dnrkk
1/1     Running   0          35m
tanzu-system-monitoring   prometheus-pushgateway-84cc9b85c6-tgmv6
1/1     Running   0          35m
tanzu-system-monitoring   prometheus-server-6479964fb6-kk9g2
2/2     Running   0          35m
[...]
```

The Prometheus pods and any other resources associated with the Prometheus component are created in the namespace you provided in `prometheus-data-values.yaml`. If you are using the default namespace, these are created in the `tanzu-system-monitoring` namespace.

Update a Running Prometheus Deployment

To make changes to the configuration of the Prometheus package after deployment, update your deployed Prometheus package:

1. Update the Prometheus configuration in the `prometheus-data-values.yaml` file.
2. Update the installed package:

```
tanzu package installed update prometheus \
--version 2.45.0+vmware.2-tkg.2 \
--values-file prometheus-data-values.yaml \
--namespace my-packages
```

Expected output:

```
| Updating package 'prometheus'
- Getting package install for 'prometheus'
| Updating secret 'prometheus-my-packages-values'
| Updating package install for 'prometheus'

 Updated package install 'prometheus' in namespace 'my-packages'
```

The Prometheus package is reconciled using the new value or values that you added. It can take up to five minutes for `kapp-controller` to apply the changes.

For information about updating, see Update a Package.

Delete a Prometheus Deployment

To remove the Prometheus package on your cluster, run:

```
tanzu package installed delete prometheus --namespace my-packages
```

For information about deleting, see Delete a Package.

Configure Notifications in Alert Manager

To configure notifications for Alert Manager, edit the `alertmanager.config.alertmanager_yml` section in your `prometheus-data-values.yaml` file.

For information about configuring notifications, such as Slack or Email, see Configuration in the Prometheus documentation.

Access the Prometheus Dashboard

By default, ingress is not enabled on Prometheus. This is because access to the Prometheus dashboard is not authenticated. To access the Prometheus dashboard:

1. Deploy Contour on the cluster.

   For information about deploying Contour, see Install Contour for Ingress Control.

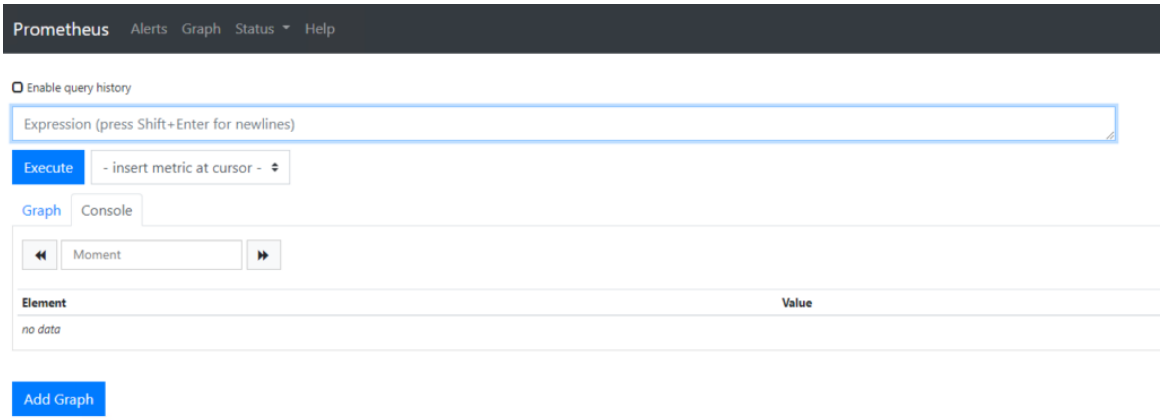
2. Copy the `ingress.enabled` section below into `prometheus-data-values.yaml`.

```
ingress:
  enabled: false
  virtual_host_fqdn: "prometheus.system.tanzu"
  prometheus_prefix: "/"
  alertmanager_prefix: "/alertmanager/"
  prometheusServicePort: 80
  alertmanagerServicePort: 80
  #! [Optional] The certificate for the ingress if you want to use your own TLS
```

```
certificate.
  #! We will issue the certificate by cert-manager when it's empty.
  tlsCertificate:
    #! [Required] the certificate
    tls.crt:
    #! [Required] the private key
    tls.key:
    #! [Optional] the CA certificate
    ca.crt:
```

3. Update `ingress.enabled` from `false` to `true`.
4. Create a DNS record to map `prometheus.system.tanzu` to the address of the Envoy load balancer.

   To obtain the address of the Envoy load balancer, see Install Contour for Ingress Control.
5. Access the Prometheus dashboard by navigating to `https://prometheus.system.tanzu` in a browser.



What to Do Next

The Prometheus package is now running and scraping data from your cluster. To visualize the data in Grafana dashboards, see Deploy Grafana on Workload Clusters.

Install Grafana (Standalone Management Cluster)

This topic explains how to deploy Grafana into a workload cluster. The procedures below apply to vSphere, Amazon Web Services (AWS), and Azure deployments.

Grafana

Grafana is open-source software that allows you to visualize and analyze metrics data collected by Prometheus on your clusters. Tanzu Kubernetes Grid includes a Grafana package that you can deploy on your workload clusters.

Prerequisites

- A bootstrap machine with the following installed:

    - Tanzu CLI, Tanzu CLI plugins, and `kubectl`, as described in Install the Tanzu CLI and Kubernetes CLI for Use with Standalone Management Clusters.
    - yq v4.5 or later.
- A standalone management cluster with the Tanzu Standard package repository installed.
- A workload cluster created by the management cluster above, with the following packages installed:
    - Prometheus; see Deploy Prometheus on Workload Clusters.
    - Contour; see Install Contour for Ingress Control.
    - cert-manager; see Install Cert Manager for Certificate Management.

As of v2.5, TKG does not support clusters on AWS or Azure. See the End of Support for TKG Management and Workload Clusters on AWS and Azure in the *Tanzu Kubernetes Grid v2.5 Release Notes*.

Important Support for IPv6 addresses in Tanzu Kubernetes Grid is limited; see Deploy Clusters on IPv6 (vSphere Only). If you are not deploying to an IPv6-only networking environment, you must provide IPv4 addresses in the following steps.

Prepare the Workload Cluster for Grafana Deployment

To prepare the cluster:

1. Get the admin credentials of the workload cluster into which you want to deploy Grafana. For example:

    ```
    tanzu cluster kubeconfig get my-cluster --admin
    ```

2. Set the context of kubectl to the cluster. For example:

    ```
    kubectl config use-context my-cluster-admin@my-cluster
    ```

3. Enable Ingress for Grafana: By default, Grafana has Ingress enabled. This requires you to install the following packages:
    - **cert-manager:** For information, see Install Cert Manager for Certificate Management.
    - **Contour:** For information, see Install Contour for Ingress Control.

Continue to Deploy Grafana on a Workload Cluster.

Deploy Grafana into the Workload Cluster

To deploy Grafana:

1. If the cluster does not have a package repository with the Grafana package installed, such as the `tanzu-standard` repository, install one:

    ```
    tanzu package repository add PACKAGE-REPO-NAME --url PACKAGE-REPO-ENDPOINT --na
    mespace tkg-system
    ```

Where:

- `PACKAGE-REPO-NAME` is the name of the package repository, such as `tanzu-standard` or the name of a private image registry configured with `ADDITIONAL_IMAGE_REGISTRY` variables.
- `PACKAGE-REPO-ENDPOINT` is the URL of the package repository.
  - For the TKG v2.5.2 release, the `tanzu-standard` URL is `projects.packages.broadcom.com/tkg/packages/standard/repo:v2025.1.27`. See List Package Repositories to obtain this value from the Tanzu CLI, or in Tanzu Mission Control see the **Addons** > **Repositories** list in the **Cluster** pane.

2. Confirm that the Grafana package is available in your workload cluster:

```
tanzu package available list -A
```

3. Retrieve the version of the available package:

```
tanzu package available list grafana.tanzu.vmware.com -A
| Retrieving package versions for grafana.tanzu.vmware.com...
 NAME                          VERSION                          RELEASED-AT
NAMESPACE
 grafana.tanzu.vmware.com      v10.0.1+vmware.2-tkg.1             2021-05-19T1
8:00:00Z  tanzu-package-repo-global
```

4. Create a configuration file for your Grafana configuration:

```
tanzu package available get grafana.tanzu.vmware.com/PACKAGE-VERSION --default-
values-file-output FILE-PATH
```

Where `PACKAGE-VERSION` is the version of the Grafana package that you want to install and `FILE-PATH` is the location to which you want to save the configuration file, for example, `grafana-data-values.yaml`.

See Grafana Configuration Parameters (Standalone MC) for a full list of available parameters.

5. Edit `grafana-data-values.yaml` and replace `secret.admin_password` with a Base64-encoded password. To generate a Base64-encoded password, run:

```
echo -n 'mypassword' | base64
```

You can also use the Base64 encoding tool at https://www.base64encode.org/ to encode your password. For example, using either method, `mypassword` results in the encoded password `bXlwYXNzd29yZA==`.

6. (Optional) Modify the Grafana datasource configuration in `grafana-data-values.yaml`. Grafana is configured with Prometheus as a default data source. If you have customized the Prometheus deployment namespace and it is not deployed in the default namespace, `tanzu-system-monitoring`, you need to change the Grafana datasource configuration in `grafana-data-values.yaml`. To change the datasource configuration, copy the section below into the position shown and modify `url` as needed.

```
#! The namespace in which to deploy grafana.
namespace: tanzu-system-dashboards

grafana:
  #! The grafana configuration.
  config:
    #! Refer to https://grafana.com/docs/grafana/latest/administration/provisio
ning/#example-data-source-config-file
    datasource_yaml: |-
      apiVersion: 1
      datasources:
        - name: Prometheus
          type: prometheus
          url: http://prometheus-server.<change-to-prometheus-namespace>.svc.cl
uster.local
          access: proxy
          isDefault: true
```

See Grafana Configuration Parameters (Standalone MC) for a full list of available parameters.

7. After you make any changes needed to your `grafana-data-values.yaml` file, remove all comments in it:

```
yq -i eval '... comments=""' grafana-data-values.yaml
```

8. Deploy the package:

```
tanzu package install grafana \
--package grafana.tanzu.vmware.com \
--version AVAILABLE-PACKAGE-VERSION \
--values-file grafana-data-values.yaml \
--namespace TARGET-NAMESPACE
```

Where:

- `TARGET-NAMESPACE` is the namespace in which you want to install the Grafana package. For example, the `my-packages` or `tanzu-user-managed-packages` namespace.
  - If the `--namespace` flag is not specified, the Tanzu CLI uses the `default` namespace. The Grafana pods and any other resources associated with the Grafana component are created in the namespace you set in `grafana-data-values.yaml`; do not install the Grafana package into this namespace.
  - The specified namespace must already exist, for example from running `kubectl create namespace my-packages`.
- `AVAILABLE-PACKAGE-VERSION` is the version that you retrieved above.

For example:

```
tanzu package install grafana --package grafana.tanzu.vmware.com --version v10.
0.1+vmware.2-tkg.1 --values-file grafana-data-values.yaml --namespace my-packag
es
```

```
- Installing package 'grafana.tanzu.vmware.com'
| Getting namespace 'my-packages'
```

```
| Getting package metadata for 'grafana.tanzu.vmware.com'
| Creating service account 'grafana-my-packages-sa'
| Creating cluster admin role 'grafana-my-packages-cluster-role'
| Creating cluster role binding 'grafana-my-packages-cluster-rolebinding'
| Creating secret 'grafana-my-packages-values'
- Creating package resource
\ Package install status: Reconciling

 Added installed package 'grafana' in namespace 'my-packages'
```

If the installation fails with `error: Secret in version "v1" cannot be handled as a Secret: illegal base64 data at input byte 4 (reason: BadRequest)`, see Secret not created when installing Grafana from default YAML file in *Troubleshooting Workload Cluster Issues*.

Verify Grafana Deployment

After you deploy Grafana, you can verify that the deployment is successful:

1. Confirm that the Grafana package is installed. For example:

```
tanzu package installed list -A
/ Retrieving installed packages...
NAME            PACKAGE-NAME                      PACKAGE-VERSION          STATU
S               NAMESPACE
cert-manager    cert-manager.tanzu.vmware.com     v1.12.10+vmware.2-tkg.2  Recon
cile succeeded  my-packages
contour         contour.tanzu.vmware.com          v1.29.1+vmware.1-tkg.1   Recon
cile succeeded  my-packages
grafana         grafana.tanzu.vmware.com          v10.0.1+vmware.2-tkg.1   Recon
cile succeeded  tkg-system
prometheus      prometheus.tanzu.vmware.com       2.45.0+vmware.2-tkg.2    Recon
cile succeeded  tkg-system
antrea          antrea.tanzu.vmware.com                                    Recon
cile succeeded  tkg-system
[...]
```

   The `grafana` package and the `grafana` app are installed in the namespace that you specify when running the `tanzu package install` command.

2. Confirm that the `grafana` app is successfully reconciled:

```
kubectl get apps -A
```

   For example:

```
NAMESPACE     NAME                          DESCRIPTION           SINCE-DEPLOY
AGE
my-packages   cert-manager                  Reconcile succeeded   37s
21h
my-packages   contour                       Reconcile succeeded   33s
129m
my-packages   grafana                       Reconcile succeeded   19s
```

```
6m56s
my-packages   prometheus                    Reconcile succeeded   46s
21h
tkg-system    antrea                        Reconcile succeeded   3m50s
24h
[...]
```

If the status is not `Reconcile succeeded`, view the full status details of the `grafana` app. Viewing the full status can help you troubleshoot the problem:

```
kubectl get app grafana --namespace PACKAGE-NAMESPACE -o yaml
```

Where `PACKAGE-NAMESPACE` is the namespace in which you installed the package.

3. Confirm that the new services are running by listing all of the pods that are running in the cluster:

   ```
   kubectl get pods -A
   ```

   In the `tanzu-system-dashboards` namespace, you should see the `grafana` service running in a pod:

   ```
   NAMESPACE               NAME                                    READY   STATUS
   RESTARTS   AGE
   [...]
   tanzu-system-dashboards   grafana-6865dbb4f5-pk2qg              2/2     Running
   0          7m7s
   [...]
   ```

The Grafana pods and any other resources associated with the Grafana component are created in the namespace you provided in `grafana-data-values.yaml`. If you are using the default namespace, these are created in the `tanzu-system-dashboards` namespace.

Update a Running Grafana Deployment

To make changes to the configuration of the Grafana package after deployment, update your deployed Grafana package:

1. Update the Grafana configuration in the `grafana-data-values.yaml` file.
2. Update the installed package:

   ```
   tanzu package installed update grafana \
   --version v10.0.1+vmware.2-tkg.1 \
   --values-file grafana-data-values.yaml \
   --namespace my-packages
   ```

   Expected output:

   ```
   | Updating package 'grafana'
   - Getting package install for 'grafana'
   | Updating secret 'grafana-my-packages-values'
   | Updating package install for 'grafana'

    Updated package install 'grafana' in namespace 'my-packages'
   ```

The Grafana package is reconciled using the new value or values that you added. It can take up to five minutes for `kapp-controller` to apply the changes.

For information about updating, see Update a Package.

Delete a Grafana Deployment

To remove the Grafana package on your cluster, run:

```
tanzu package installed delete grafana --namespace my-packages
```

For information about deleting, see Delete a Package.

Access the Grafana Dashboard

After Grafana is deployed, the grafana package creates a Contour HTTPProxy object with a Fully Qualified Domain Name (FQDN) of `grafana.system.tanzu`.

To use this FQDN to access the Grafana dashboard:

1. Create an entry in your local `/etc/hosts` file that points an IP address to this FQDN:
    - **AWS or Azure:** Use the IP address of the LoadBalancer for the Envoy service in the `tanzu-system-ingress` namespace.
    - **vSphere:** Use the IP address of a worker node.
2. Navigate to `https://grafana.system.tanzu`.

Because the site uses self-signed certificates, you might need to navigate through a browser-specific security warning before you are able to access the dashboard.