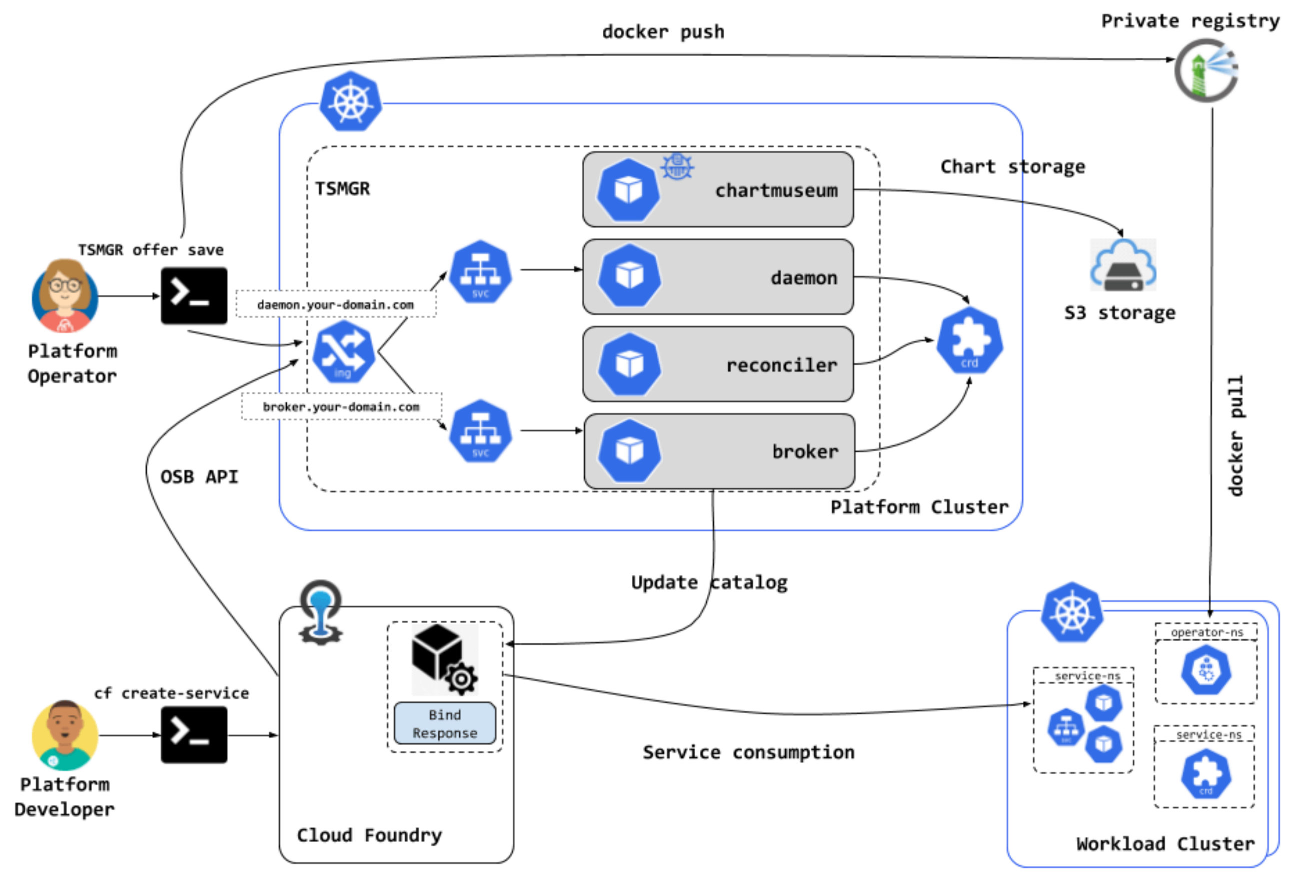
TSMGR Conceptual Overview

Contents

This topic describes Tanzu Service Manager (TSMGR) components, architecture, and technical details.

Overview

The following diagram describes how TSMGR deployed on Kubernetes communicates with Tanzu Kubernetes Grid and other workload clusters:



Components

TSMGR contains the following components:

- TSMGR Daemon
- TSMGR Broker

TSMGR Daemon

The TSMGR daemon provides the API that the TSMGR Command Line Interface (CLI) communicates with. When a platform operator adds a service offering using the TSMGR CLI command `tsmgr offer save`:

1. The daemon validates the offer.

2. The daemon stores the Helm charts in ChartMuseum. ChartMuseum uses S3 Storage.
3. The daemon stores the offer in the Kubernetes cluster where TSMGR is installed.
4. The broker updates the Cloud Foundry catalog.

For more information about ChartMuseum, see the chartmuseum repository on GitHub.

TSMGR Broker

The TSMGR broker is an Open Service Broker that creates on-demand service instances on a Kubernetes cluster.

When a Tanzu Kubernetes Grid Integrated Edition (TKGI) developer runs `cf create-service`:

1. Cloud Controller sends that CLI request to the broker.
2. The broker deploys the Helm chart needed for the service.

For more information about Open Service Brokers, see Open Service Broker API.

For more information about the `cf create-service` command, see Creating Service Instances in the Cloud Foundry documentation.

TSMGR Resources

TSMGR uses the following external resources:

- S3-Compatible Bucket
- Kubernetes Clusters

S3-Compatible Bucket

TSMGR uses an external S3-compatible bucket to store the state of ChartMuseum. For instructions for configuring an external S3-compatible bucket for TSMGR, see Configuring External Storage.

Kubernetes Clusters

TSMGR is backed by at least one default Kubernetes cluster.

You can use different service offering plans to configure different types of clusters. For example, you could configure a large default cluster for more ephemeral service instances used in development. You could also configure another cluster with workload specialization for specific service plans.

For instructions for configuring a default Kubernetes cluster, see Managing Kubernetes Clusters for TSMGR.

TSMGR Life Cycle

This section outlines what happens when a TKGI developer uses the Cloud Foundry Command Line Interface (cf CLI) to create, bind, update, and delete service instances with TSMGR.

For information about the cf CLI, see Managing Service Instances with the cf CLI in the Cloud Foundry documentation.

Create

When a TKGI developer runs `cf create-service`, TSMGR deploys Helm charts to the Kubernetes cluster.

If your service offering has multiple charts, TSMGR loops through the list of charts and does the following for each chart:

1. Checks the scope of the chart. A chart can have either a `cluster` or `namespace` scope. See Cluster and Namespace Scopes below.
2. Retrieves the Kubernetes cluster defined in the specific plan. If no cluster is defined, TSMGR uses the default cluster.
3. Deploys the chart to the Kubernetes cluster. The chart is deployed in a namespace defined by the chart scope.
4. Waits for the deployment to succeed. TSMGR does not deploy subsequent charts until the previous chart is fully installed.
5. Stores state about the instance in custom resources in the Kubernetes cluster where TSMGR is installed.

For instructions about offering multiple charts in a single offering, see (Optional) Offer Multiple Charts in a Single Offering.

If TSMGR is configured with a private container registry, when TSMGR deploys a chart, it also does the following:

1. Changes the `global.imageRegistry` reference in the chart to pull from the configured private container registry.
2. Adds a secret containing the private container-registry credentials to the namespace. This secret enables TSMGR to pull the image.

For instructions about configuring a private container registry, see Load Container Images into a Private Container Registry.

Cluster and Namespace Scopes

When TSMGR checks the scope for a Helm chart, TSMGR creates a namespace based on the following:

- **If the chart has the cluster scope:** TSMGR creates a namespace named `TSMGR-CHART`, where `CHART` is the name of the chart.
- **If a chart has the namespace scope:** TSMGR creates a namespace named `TSMGR-GUID`, where `GUID` is the `service-instance-guid` for the service instance.

> **Note**
>
> If a chart has the `cluster` scope and the namespace for the chart is already present, TSMGR does not add the chart.

TSMGR does not internally track the identifiers for namespaces. It uses the predictable namespace names to find resources in the Kubernetes cluster.

Bind

When a TKGI developer runs `cf bind-service`, TSMGR retrieves the `services` and `secrets` credentials for the service offering.

- **If the Helm chart for the service offering includes a bind template:** The values of *services* and *secrets* are processed by the template and the resulting credentials are used for the bind. For instructions about creating a bind template, see (Optional) Create Binding Template for App Consumption.
- **If the chart does not include a bind template:** The *services* and *secrets* values are used directly as credentials.

Update

When a TKGI developer runs `cf update-service`, TSMGR triggers the Helm upgrade process. For information about the Helm upgrade process, see the Helm Upgrade in the Helm documentation.

You can use the `cf update-service` command to add or modify configuration parameters using the `-c` flag. The update uses existing values, which means you do not need to resend configuration parameters. The update also does not upgrade the Helm chart to a newer version.

Delete

When a TKGI developer runs `cf delete-service`, TSMGR does the following:

- Deletes each Helm chart with the `namespace` scope.
- Deletes the namespace for each chart with the `namespace` scope.
- Deletes the chart and namespace if the instance is the last instance of a `cluster` scope chart.